TMA4300: Computer Intensive Statistical Methods - Exercise 1, Spring 2024

Yawar Mahmood

2024-02-11

Problem A: Stochastic simulation by the probability integral transform and bivariate techniques

Problem 1)

To use the probability integral transform, we first need to determine the cumulative distribution function (CDF) and its inverse (the quantile function). Let X follow a exponential distribution with parameter λ. Then the CDF and the quantile function are well-defined:

$$F(x) = p = 1 - e^{-\lambda x}, \quad x \geq 0$$

$$F^{-1}(p) = x = -\frac{1}{\lambda}\ln(1-p), \quad 0 < p < 1$$

where:

- $F(x)$ is the CDF
- λ is the rate parameter of the exponential distribution, $\lambda > 0$
- $F^{-1}(p)$ is the quantile function.
- p is a probability value

We can simplify the quantile function further. Since $(1-p) \sim U(0,1)$ and $p \sim U(0,1)$, then we can exchange $1-p$ with u

$$F^{-1}(u) = -\frac{1}{\lambda}\ln(u), \quad u \sim \text{Uniform}(0,1)$$

In the code chunk below, we sample u from the runif() function, and transform it using the quantile function given above.

```
generate_samples_from_exp <- function(lambda, n) {
  # Check if the rate parameter 'lambda' is positive
  if (lambda <= 0) {
    stop("Rate parameter lambda must be positive")
  }
```

```r
  # Generate n random numbers uniformly distributed between 0 and 1
  u <- runif(n)

  # Transform the uniformly distributed numbers to an exponential distribution
  return(-log(u) / lambda)
}
```

We want to check our code visually, if our generated samples align with the theoretical exponential function. We set $\lambda = 0.5$ and draw one million samples:

```r
# Generating one million samples, with lambda = 0.5
exp_samples <- generate_samples_from_exp(lambda = 0.5, n = 1000000)

# Plot
ggplot(data.frame(exp_samples), aes(x = exp_samples)) +
  geom_histogram(aes(y = ..density.., fill = "Sample Histogram"), binwidth = 0.1, color = "blue") +
  stat_function(fun = function(x) {0.5 * exp(-0.5 * x)}, aes(color = "Theoretical Distribution"), size =
  scale_fill_manual(name = "Legend", values = c("Sample Distribution" = "blue")) +
  scale_color_manual(name = "Legend", values = c("Theoretical Distribution" = "orange")) +
  labs(title = "Histogram of Samples from an Exponential Distribution",
       caption = "Figure 1: Histogram of one million samples from an exponential\n   distribution, with
       x = "Value",
       y = "Density") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5),
        plot.caption = element_text(hjust = 0.5),
        axis.title.x = element_text(hjust = 0.5),
        axis.title.y = element_text(hjust = 0.5),
        legend.title = element_blank())
```

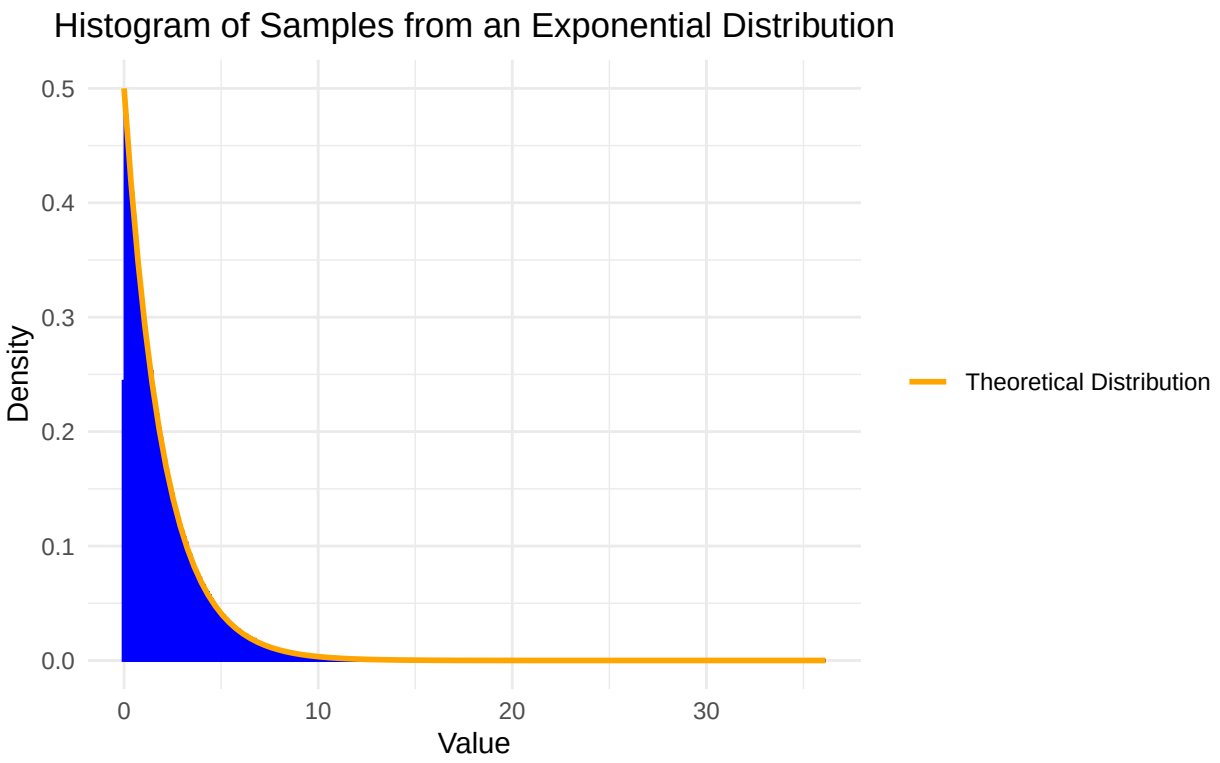

Figure 1: Histogram of one million samples from an exponential distribution, with rate 0.5, overlaid with the theoretical exponential density function.

Visually, our code seems to work, but we can further check the workings of the code by comparing the sample mean/variance with the theoretical values for these statistics. The theoretical values are simply given by:

$$\text{Mean} = \frac{1}{\lambda} = \frac{1}{0.5} = 2$$

$$\text{Variance} = \frac{1}{\lambda^2} = \frac{1}{0.5^2} = 4$$

And the simulated values are:

```
# Sample mean and variance
mean(exp_samples)
```

```
## [1] 1.999243
```

```
var(exp_samples)
```

```
## [1] 3.982012
```

Which seems to hold up with the theoretical values.

Problem 2)

Task a)

Consider the probability density function:

$$g(x) = \begin{cases} cx^{\alpha-1}, & \text{for } 0 < x < 1 \\ ce^{-x}, & \text{for } 1 \leq x \\ 0, & \text{otherwise} \end{cases}$$

where:

- c is a normalizing constant
- $\alpha \in (0, 1)$.

The CDF $G(x)$ is the integral of the PDF g over its domain:

$$G(x) = \int g(x)\, dx$$

As suggested by the PDF, we split this into two cases.
When $0 < x < 1$:

$$G(x) = \int_0^x cx^{\alpha-1} dx = c\frac{x^\alpha}{\alpha}$$

When $x \geq 1$:

$$G(x) = \int_0^1 cx^{\alpha-1} dx + \int_1^x ce^{-x} dx = c\left(\alpha^{-1} - e^{-x} + e^{-1}\right)$$

We can determine the value of c, since:

$$\int g(x) = 1 = \frac{c}{\alpha} + \frac{c}{e}$$

giving

$$c = \frac{\alpha e}{e + \alpha}$$

Putting it all together:

$$G(x) = p = \begin{cases} \frac{e}{e+\alpha}x^\alpha, & \text{for } 0 < x < 1 \\ \frac{e - \alpha e^{1-x} + \alpha}{e+\alpha}, & \text{for } 1 \leq x \\ 0, & \text{otherwise} \end{cases}$$

Solving for x gives the inverse:

$$G^{-1}(p) = x = \begin{cases} \left(\frac{(e+\alpha)p}{e}\right)^{\frac{1}{\alpha}}, & 0 < p < \frac{e}{\alpha+e} \\ 1 - \ln\left(\frac{e+\alpha-p(e+\alpha)}{\alpha}\right), & \frac{e}{\alpha+e} \leq p \leq 1 \\ 0, & \text{otherwise} \end{cases}$$

Task b)

As in problem 1) we use the inverse transform method to sample from g.

```r
# Generate samples from g
generate_samples_from_g <- function(alpha, n) {
  # Check if the alpha parameter is within the valid range (0, 1)
  if (alpha <= 0 || alpha >= 1) {
    stop("Alpha must be in the range (0, 1)")
  }

  # Generate n random numbers uniformly distributed between 0 and 1
  p <- runif(n)

  # Calculate the normalizing constant c
  c <- (alpha*exp(1))/(exp(1) + alpha)

  # Case 1: For values in the range 0 < x < 1
  samples_case_1 <- ((p*(exp(1)+alpha))/(exp(1)))^(1/alpha)
  # Case 2: For values in the range x >= 1
  samples_case_2 <- 1-log((exp(1)+alpha-p*(exp(1)+alpha))/(alpha))

  # Use a logical condition to choose the correct value from either case 1 or case 2
  samples <- ifelse(p < c / alpha, samples_case_1, samples_case_2)

  return(samples)
}

# Theoretical distribution function g(x)
g_function_theoretical <- function(x, alpha, c) {
  # Initialize result vector
  y <- numeric(length(x))

  # Apply the function piecewise
  y[x < 1] <- c * x[x < 1]^(alpha - 1)
  y[x >= 1] <- c * exp(-x[x >= 1])

  # Avoid calculating for x very close to 0 to avoid inf values
  y[x < 0.000000001] <- NA

  return(y)
}
```

We want to check our code visually, if our generated samples align with the theoretical distriburion of g. We set $\alpha = 0.5$ and draw one million samples:

```r
# Parameters
alpha_value <- 0.5
c_value <- (alpha_value * exp(1)) / (exp(1) + alpha_value)
n = 1000000

g_samples <- generate_samples_from_g(alpha_value, n)

# Plot
```

```r
ggplot(data.frame(x = g_samples), aes(x = x)) +
  geom_histogram(aes(y = ..density.., fill = "Sample Distribution"), binwidth = 0.1, color = "blue") +
  stat_function(fun = function(x) g_function_theoretical(x, alpha_value, c_value), aes(color = "Theoreti
  scale_fill_manual(name = "Legend", values = c("Sample Distribution" = "blue")) +
  scale_color_manual(name = "Legend", values = c("Theoretical Distribution" = "orange")) +
  labs(title = "Histogram of Samples from Custom Distribution g",
       caption = paste("Figure 2: Histogram of one million samples from a custom distribution g with alp
       x = "Value",
       y = "Density") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5),
        plot.caption = element_text(hjust = 0.5),
        axis.title.x = element_text(hjust = 0.5),
        axis.title.y = element_text(hjust = 0.5),
        legend.title = element_blank())
```

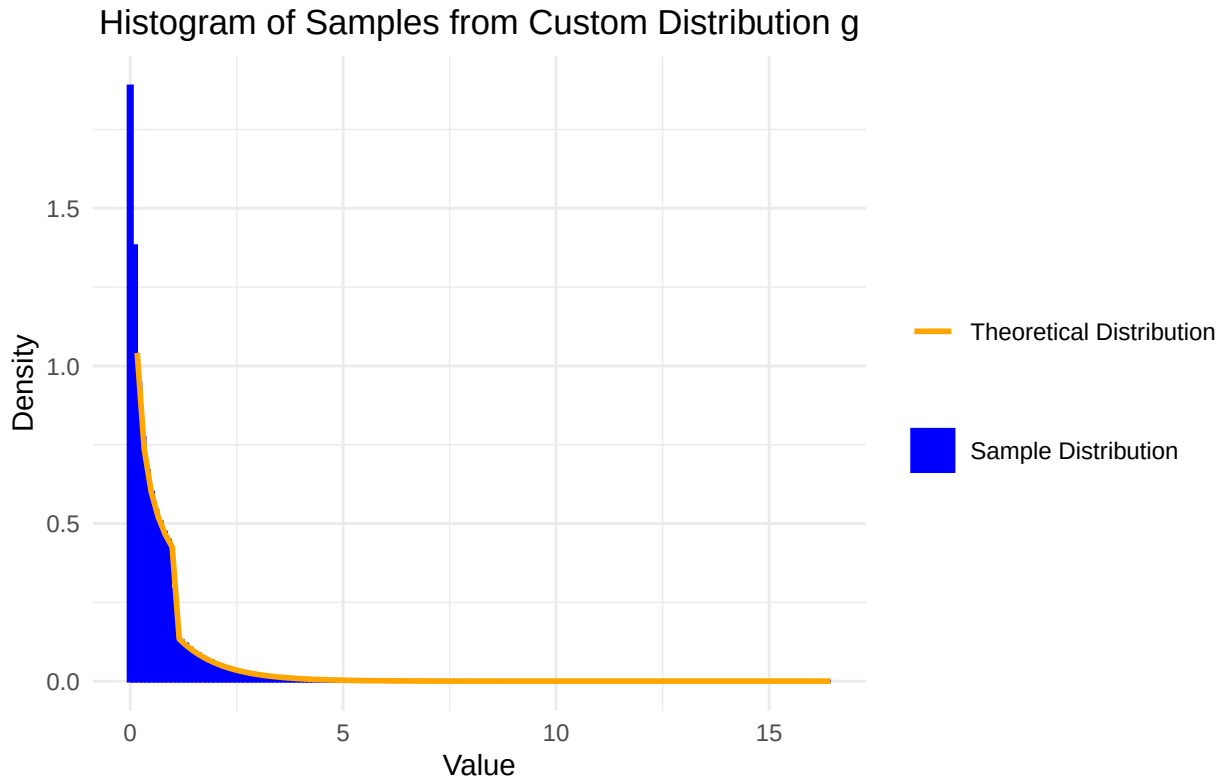

Figure 2: Histogram of one million samples from a custom distribution g with alpha = 0.5
Overlay with the theoretical distribution function.

Visually, our code seems to work, but we can further check the workings of the code by comparing the sample mean/variance with the theoretical values for these statistics. The theoretical values are given by solving the integrals:

```r
# Calculate (E[X])
mean_integral <- integrate(function(x) x * g_function_theoretical(x, alpha_value, c_value), lower = 0, u
mean_value <- mean_integral$value

# Calculate (E[X^2])
second_moment_integral <- integrate(function(x) x^2 * g_function_theoretical(x, alpha_value, c_value), l
```

```r
second_moment_value <- second_moment_integral$value

# Calculate the variance (Var[X] = E[X^2] - (E[X])^2)
variance_value <- second_moment_value - mean_value^2
```

Below the sample mean and the theoretical mean are given, respectively:

```r
mean(g_samples)
```

```
## [1] 0.5932709
```

```r
mean_value
```

```
## [1] 0.5922707
```

Below the sample variance and the theoretical variance are given, respectively:

```r
var(g_samples)
```

```
## [1] 0.594877
```

```r
variance_value
```

```
## [1] 0.5949549
```

Problem 3)

Task a)

Consider the probability density function:

$$f(x) = \frac{ce^{\alpha x}}{(1+e^{\alpha x})^2}, \quad -\infty < x < \infty, \alpha > 0$$

We need to ensure that the integral of f over its domain is equal to 1:

$$\int_{-\infty}^{\infty} \frac{ce^{\alpha x}}{(1+e^{\alpha x})^2}\, dx = 1$$

The integral gives a simple expression, $\frac{c}{\alpha} = 1$, hence:

$$c = \alpha$$

Putting it all together:

$$f(x) = \frac{\alpha e^{\alpha x}}{(1+e^{\alpha x})^2}, \quad -\infty < x < \infty, \alpha > 0$$

Task b)

To find the CDF, we need to integrate the pdf f as follows:

$$F(x) = \int_{-\infty}^{x} \frac{\alpha e^{\alpha t}}{(1 + e^{\alpha t})^2} \, dt$$

which yields:

$$F(x) = p = \frac{e^{\alpha x}}{e^{\alpha x} + 1}$$

And the inverse is simply found by solving for x:

$$F^{-1}(p) = \frac{1}{\alpha} \ln \left(\frac{p}{1 - p} \right)$$

Task c)

As in problem 2Task b, we use the inverse transform method to sample from f.

```
generate_samples_from_f <- function(alpha, n) {
  # Check if the alpha parameter is positive
  if (alpha <= 0) {
    stop("Parameter alpha must be positive")
  }

  # Generate n random numbers uniformly distributed between 0 and 1
  p <- runif(n)

  # Transform the uniform random numbers using the inverse CDF
  samples <- (1 / alpha) * log(p / (1 - p))

  # Return the vector of samples generated from the distribution
  return(samples)
}

f_theoretical <- function(x, alpha) {
  alpha * exp(alpha * x) / (1 + exp(alpha * x))^2
}
```

We want to check our code visually, if our generated samples align with the theoretical distribution of f. We set $\alpha = 0.5$ and draw one million samples:

```
# Set the alpha parameter and number of samples
alpha <- 0.5  # Example alpha value
n <- 100000   # Number of samples to generate

# Generate samples from the distribution
f_samples <- generate_samples_from_f(alpha = alpha, n = n)

# Plot
```

```r
ggplot(data.frame(f_samples = f_samples), aes(x = f_samples)) +
  geom_histogram(aes(y = ..density.., fill = "Sample Distribution"), binwidth = 0.5, color = "blue") +
  stat_function(fun = function(x) f_theoretical(x, alpha), aes(color = "Theoretical Distribution"), siz
  scale_fill_manual(name = "Legend", values = c("Sample Distribution" = "blue")) +
  scale_color_manual(name = "Legend", values = c("Theoretical Distribution" = "orange")) +
  labs(title = "Histogram and Density Plot of Samples",
       caption = paste("Figure 3: Generated one million samples from the distribution given by f with a
       x = "Sample Value",
       y = "Density") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5),
        plot.caption = element_text(hjust = 0.5),
        axis.title.x = element_text(hjust = 0.5),
        axis.title.y = element_text(hjust = 0.5),
        legend.title = element_blank())
```

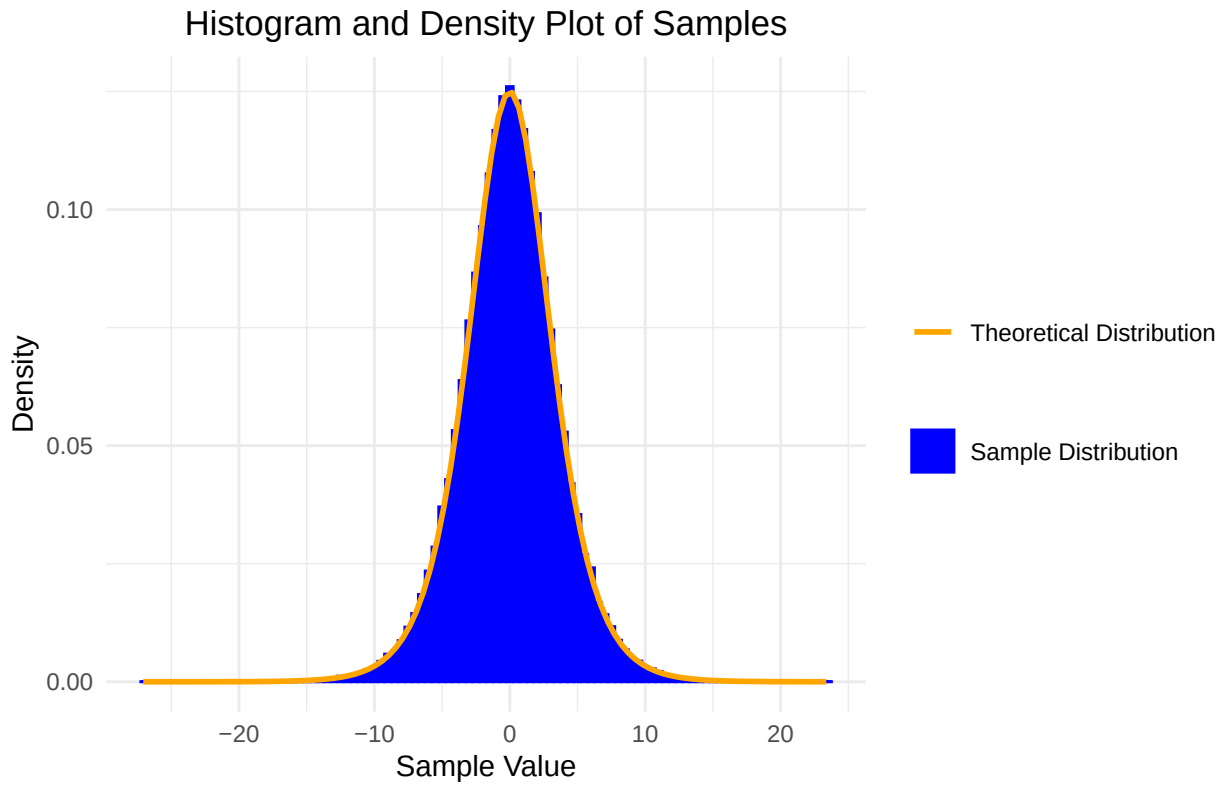

Figure 3: Generated one million samples from the distribution given by f with alpha = 0.5
Plotted with the theoretical pdf

Visually, our code seems to work, but we can further check the workings of the code by comparing the sample mean/variance with the theoretical values for these statistics. The theoretical values are given by solving the integrals:

```r
# Calculate (E[X])
mean_integral <- integrate(function(x) x * f_theoretical(x, alpha), lower = -Inf, upper = Inf)
mean_value <- mean_integral$value
```

Below the sample mean and the theoretical mean are given, respectively:

```r
mean(f_samples)
```

```
## [1] -0.01593158
```

```r
mean_value
```

```
## [1] 2.100488e-16
```

Below the sample variance is given:

```r
var(f_samples)
```

```
## [1] 13.0864
```

When the second-moment integral is calculated using integration software, we approximately get the value 13,16. Since the mean is close to zero, the variance term is also around 13, which checks out good with the calculated sample variance.

Problem 4)

We want to simulate the standard normal distribution using the Box-Muller algorithm. If U_1 and U_2 are uniformly distributed, then $Z_0 = \sqrt{-2\ln(U_1)} * \cos(2\pi \cdot U_2)$ and $Z_1 = \sqrt{-2\ln(U_1)} * \sin(2\pi \cdot U_2)$ are standard normal distributed. This is the Box-Muller algorithm.

```r
generate_standard_normal <- function(n) {
  # We need an even number of samples for Box-Muller, as it generates pairs of normal random variables.
  if (n %% 2 != 0) {
    n <- n + 1  # If n is odd, increment by 1 to make it even
  }

  # Generate n/2 independent uniform random numbers for U1 and U2
  U1 <- runif(n / 2)
  U2 <- runif(n / 2)

  # Apply the Box-Muller transform to convert uniform random numbers into standard normal random number
  Z0 <- sqrt(-2 * log(U1)) * cos(2 * pi * U2)  # First half of transformed samples
  Z1 <- sqrt(-2 * log(U1)) * sin(2 * pi * U2)  # Second half of transformed samples

  # Combine the two vectors of length n/2 each into a single vector of length n
  samples <- c(Z0, Z1)

  return(samples)
}
```

We now, visually, check that the function is working properly by comparing samples generated to the theoretical distribution

```r
n <- 1000000
standard_normal_samples <- generate_standard_normal(n)

# Plot
ggplot(data.frame(Samples = standard_normal_samples), aes(x = Samples)) +
  geom_histogram(aes(y = ..density.., fill = "Sample Distribution"), binwidth = 0.2) +
  geom_density(aes(color = "Theoretical Distribution"), alpha = 0.2) +
  scale_fill_manual(name = "Legend", values = c("Sample Distribution" = "lightblue")) +
  scale_color_manual(name = "Legend", values = c("Theoretical Distribution" = "red")) +
  labs(title = "Histogram and Density Plot of Standard Normal Samples",
       caption = paste("Figure 4: Generated one million samples from a standard normal distribution,\n
       x = "Sample Value",
       y = "Density",) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5),
        plot.caption = element_text(hjust = 0.5),
        axis.title.x = element_text(hjust = 0.5),
        axis.title.y = element_text(hjust = 0.5),
        legend.title = element_blank())
```

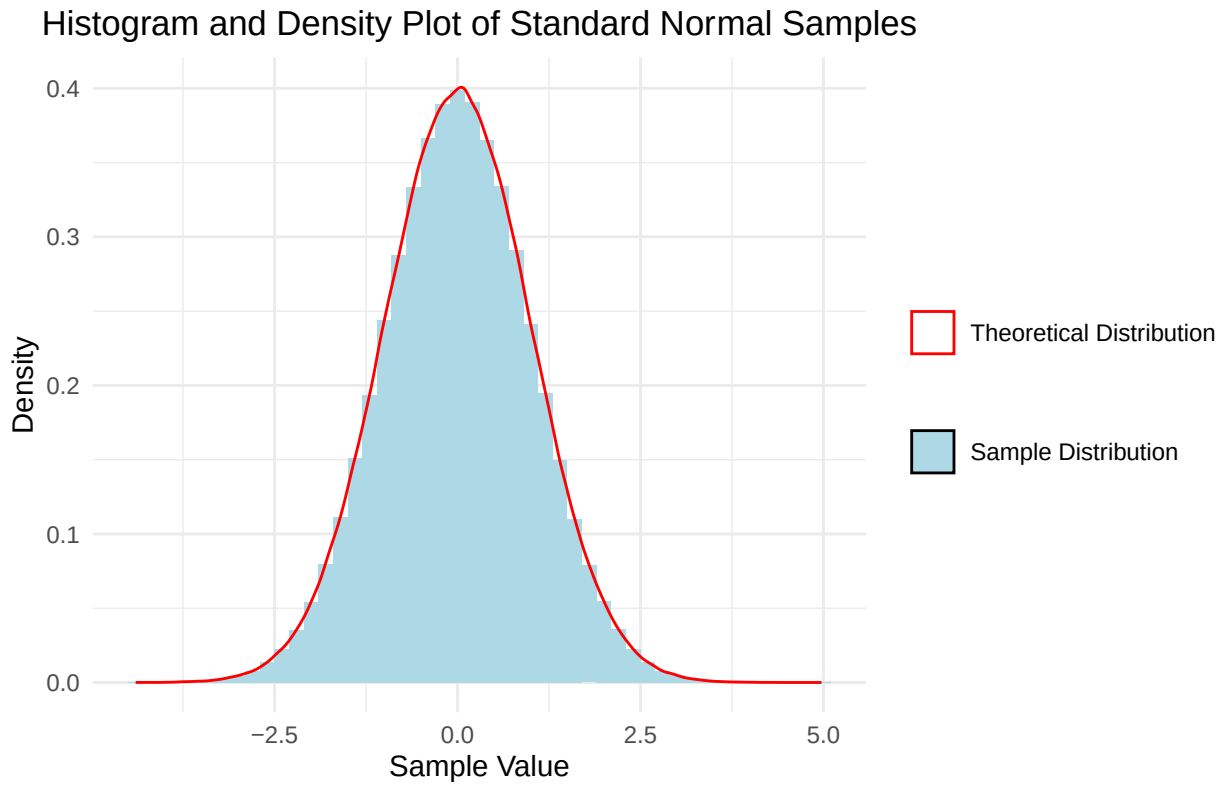

Figure 4: Generated one million samples from a standard normal distribution, overlayed with the theoretical distribution

The visual check checks out. Since it is the standard normal, we can check if the mean is 0 and that the variance is 1:

```r
mean(standard_normal_samples)
```

```
## [1] -0.0004852477
```

```r
var(standard_normal_samples)
```

```
## [1] 1.001139
```

which checks out.

Problem 5)

We want to simulate from a d-variate normal distribution, with mean μ and covariance matrix Σ. Let $Z \sim$ Normal(0,I) where I is the d-variate (dxd) identity matrix. Then:

$$X = \mu + AZ \sim Normal(\mu, AA^T)$$

This representation of the normal function is from the Cholesky decomposition

where:

- A is a lower triangular matrix such that $\Sigma = AA^T$

X then represents a d-variate normal distribution with mean μ and covariance Σ

```r
generate_d_variate_standard_normal <- function(n, mu, sigma) {
  d <- length(mu) # dim of multivariate normal
  A <- t(chol(sigma)) # Compute the Cholesky decomposition
  z <- generate_standard_normal(d*n) # Generate standard normal values
  Z <- matrix(z, nrow = d, ncol = n) # Organize the generated std values into a matrix
  samples <- mu + A %*% Z # Generate multi-variate normal distribution values

  return(samples)
}
```

Plotting the d-variate distribution to see if it checks out is a bit "hard". Instead, let us simulate with known mean and covariance matrices, and see if our samples can lead us back to these. Let:

$$\mu = [1, 2]^T$$

$$\Sigma = \begin{bmatrix} 4 & 1 \\ 1 & 3 \end{bmatrix}$$

Note, for this to work, we have to make sure that Σ is positive-definite.

```r
n <- 1000000
mu <- c(1,2)
sigma <- matrix(c(4,1,1,3), nrow = 2)
multi_normal <- generate_d_variate_standard_normal(n,mu,sigma)

rowMeans(multi_normal)
```

```
## [1] 0.9994036 2.0008597
```

```
cov(t(multi_normal))
```

```
##           [,1]      [,2]
## [1,] 3.9940084 0.9990585
## [2,] 0.9990585 2.9968211
```

This checks out!

Problem B: The gamma distribution

Problem 1)

Task a)

The target distribution is given by:

$$f(x) = \begin{cases} \frac{1}{\Gamma(\alpha)} x^{\alpha-1} e^{-x}, & \text{for } 0 < x \\ 0, & \text{otherwise} \end{cases}$$

and the proposal distribution is given by:

$$g(x) = \begin{cases} cx^{\alpha-1}, & \text{for } 0 < x < 1 \\ ce^{-x}, & \text{for } 1 \leq x \\ 0, & \text{otherwise} \end{cases}$$

where:

- c is a normalizing constant
- $\alpha \in (0, 1)$.

As we found out earlier, $c = \frac{\alpha e}{e+\alpha}$
We use the formula:

$$A(x) = \frac{f(x)}{c \cdot g(x)}$$

to find the expression for the acceptance probability in the rejection sampling algorithm.
For $0 < x < 1$:

$$A(x) = \frac{\frac{1}{\Gamma(\alpha)} x^{\alpha-1} e^{-x}}{\frac{\alpha e}{e+\alpha} x^{\alpha-1}} = \frac{1}{\frac{\alpha e}{e+\alpha}\Gamma(\alpha)} e^{-x}$$

For $1 \leq x$:

$$A(x) = \frac{\frac{1}{\Gamma(\alpha)} x^{\alpha-1} e^{-x}}{\frac{\alpha e}{e+\alpha} e^{-x}} = \frac{1}{\frac{\alpha e}{e+\alpha}\Gamma(\alpha)} x^{\alpha-1}$$

Task b)

Given the information above, we can now construct a function that generates a vector of n independent samples from f, using the rejection sampling algorithm.

```r
generate_samples_from_f_rejection_sampling <- function(alpha, n) {
  if (alpha <= 0 || alpha >= 1) {
    stop("Alpha must be in the range (0, 1)")
  }
  c <- (alpha*exp(1))/(exp(1) + alpha) # normalizing constant, as found in previous task

  samples <- numeric(n)  # Vector to store accepted samples
  count <- 1 # count variable

  while(count <= n) {
    u <- runif(1)  # Uniform random number generation
    if (u < c / alpha) {
      x_proposed <- runif(1, 0, 1)
      A_x <- (1 / (c * gamma(alpha))) * exp(-x_proposed) # Acceptance region as defined above, for 0<x<
    } else {
      x_proposed <- rexp(1, 1)
      A_x <- (1 / (c * gamma(alpha))) * x_proposed^(alpha - 1) # Acceptance region as defined above, fo
    }

    # Generate a uniform random number for acceptance check
    u <- runif(1)

    # Calculate the acceptance probability for the proposed x
    if (u < A_x) {
      samples[count] <- x_proposed
      count <- count + 1
    }
  }

  return(samples)
}
```

We can check by inspection if our code is working correctly:

```r
alpha <- 0.5
n <- 1000000

# Generate samples
samples <- generate_samples_from_f_rejection_sampling(alpha, n)

# Theoretical density
theoretical_density <- function(x, alpha) {
  ifelse(x > 0.00001, (1 / gamma(alpha)) * x^(alpha - 1) * exp(-x), 0)
}

# Plot
ggplot() +
  geom_histogram(data = data.frame(x = samples), aes(x = x, y = ..density.., fill = "Sample Distributio
  stat_function(fun = function(x) theoretical_density(x, alpha), aes(color = "Theoretical Distribution"
```

```
scale_fill_manual(name = "Legend", values = c("Sample Distribution" = "blue")) +
scale_color_manual(name = "Legend", values = c("Theoretical Distribution" = "red")) +
labs(title = "Sampled Data and Theoretical Density",
     caption = "Figure 5: Plot of sampled data from the gamma distribution with alpha = 0.5 \n alongs
     x = "x",
     y = "Density") +
theme_minimal() +
theme(plot.title = element_text(hjust = 0.5),
      plot.caption = element_text(hjust = 0.5),
      axis.title.x = element_text(hjust = 0.5),
      axis.title.y = element_text(hjust = 0.5),
      legend.title = element_blank())
```

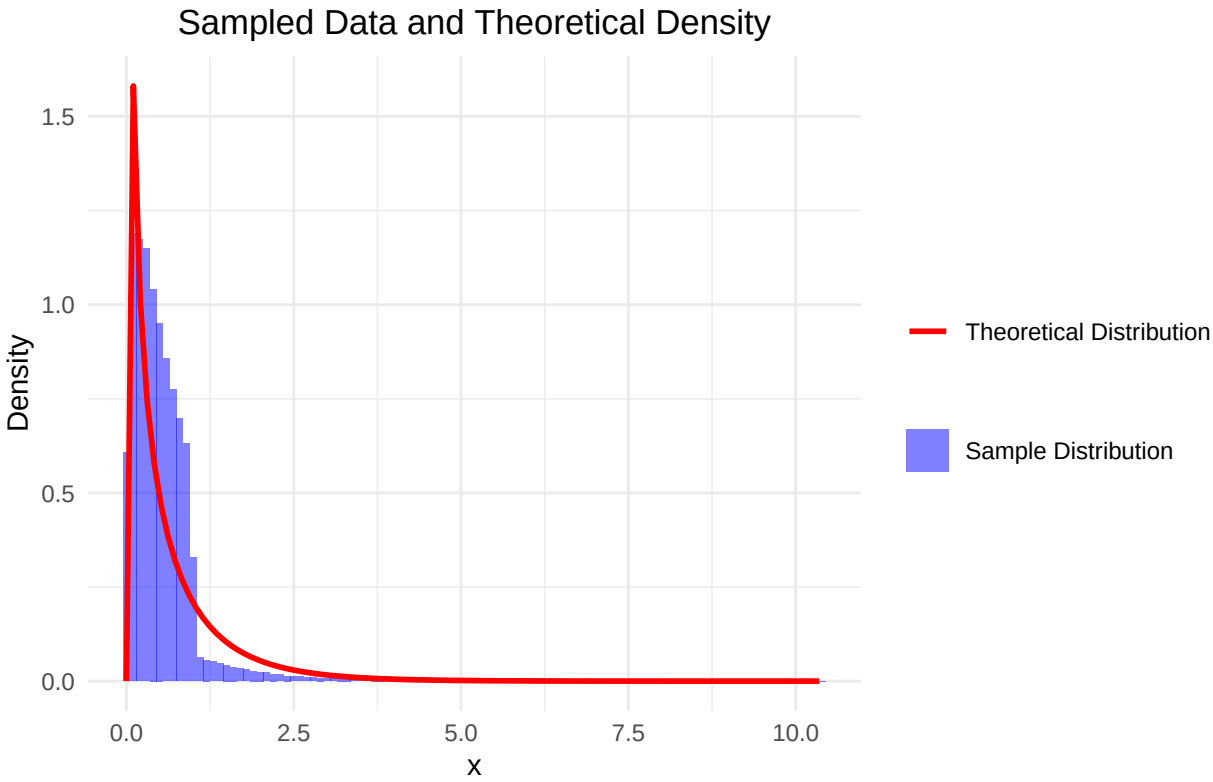

Figure 5: Plot of sampled data from the gamma distribution with alpha = 0.5 alongside the theoretical density function for a given alpha.

This seems about right, with some deviation, but the overall trend is captured, but we can further check the workings of the code by comparing the sample mean/variance with the theoretical values for these statistics. The theoretical values for the gamma function is given by the formulas:

$$E[X] = \alpha\beta = 0.5 * 1 = 0.5$$

$$Var[X] = \alpha\beta^2 = 0.5 * 1^2 = 0.5$$

And compared with the mean and variance calculated using the sample values:

```
mean(samples)
```

```
## [1] 0.5263119
```

```
var(samples)
```

```
## [1] 0.2449895
```

The mean seems to check out, but the variance is a bit off. This may indicate an error in my implementation, also since the visual representation also has some deviation.

Problem 2)

Task a)

Given that $\alpha > 1$ and $\beta = 1$,

$$f^*(x) = \begin{cases} x^{\alpha-1}e^{-x} & \text{for } 0 < x \\ 0 & \text{otherwise} \end{cases}$$

To find the values of a, b^-, and b^+, we use the given formulas:

- $a = \sqrt{\sup_x f^*(x)}$
- $b^+ = \sqrt{\sup_{x \geq 0}(x^2 f^*(x))}$
- $b^- = -\sqrt{\sup_{x \leq 0}(x^2 f^*(x))}$

Finding a is simply done by calculating the maximum of $f^*(x)$, that is:

$$\frac{d}{dx} x^{\alpha-1}e^{-x} = 0$$

$$-x^{\alpha-2} \cdot (x - \alpha + 1)e^{-x} = 0$$

Which gives that $x = \alpha - 1$, and hence

$$a = \sqrt{(\alpha-1)^{\alpha-1}e^{1-\alpha}}$$

Finding b^+ is simply done by calculating the maximum of $x^2 f^*(x)$, that is:

$$\frac{d}{dx} x^2 x^{\alpha-1}e^{-x} = 0$$

$$-x^{\alpha} \cdot (x - \alpha - 1)e^{-x} = 0$$

Which gives that $x = \alpha + 1$, and hence

$$b^+ = \sqrt{(\alpha+1)^{\alpha+1}e^{-(\alpha+1)}}$$

Finding b_- is simply done by calculating the minimum of $x^2 f^*(x)$, that is:

$$\frac{d}{dx} x^2 x^{\alpha-1} e^{-x} = 0$$

$$-x^\alpha \cdot (x - \alpha - 1)e^{-x} = 0$$

Which gives that $x = 0$, and hence

$$b_- = 0$$

Subproblem b)

Implementing the results over in the R-chunk below:

```r
generate_samples_gamma_ratio <- function(alpha, n) {
  if(alpha <= 1) {
    stop("Alpha must be greater than 1")
  }

  # Defining the required constants for the region, as calculated above
  a <- sqrt((alpha - 1)^(alpha - 1) * exp(1 - alpha))
  b_plus <- sqrt((alpha + 1)^(alpha + 1) * exp(-(alpha + 1)))
  b_minus <- 0

  tries <- 0
  samples <- numeric(n)
  count <- 0

  while(count < n) {
    tries <- tries + 1
    # Generate uniform samples within the region [0, a] x [b-, b+]
    x1 <- runif(1, 0, a)
    x2 <- runif(1, b_minus, b_plus)

    # Apply the ratio of uniforms method
    y <- x2 / x1

    # Check if y is positive to have valid numbers for log()
    if(y <= 0) next

    # Check if x1 is positive to have valid numbers for log()
    if(x1 <= 0) next

    # Proceed with the original condition, now that we've ensured y and x1 are valid
    if(log(x1) <= (alpha - 1) * log(y) - y) {
      samples[count + 1] <- y
      count <- count + 1
    }
```

```
  }

  list(samples = samples, tries = tries)
}
```

We now want to check how many tries the algorithm needs to generate 1000 realizations depending on the value of α.

```
# Function to generate plot for different alpha values and track the number of tries
generate_plot <- function(alpha_range, n) {
  results <- data.frame(alpha = numeric(), tries = numeric())

  for(alpha in alpha_range) {
    result <- generate_samples_gamma_ratio(alpha, n)
    # Assuming result is a list that contains a 'tries' element
    results <- rbind(results, data.frame(alpha = alpha, tries = result$tries))
  }

  # Plot the results with figure text and a legend
  ggplot(results, aes(x = alpha, y = tries, color = "Number of Tries")) +
    geom_line() +
    scale_color_manual(values = c("Number of Tries" = "blue"), name = "Legend", labels = c("Number of Tr
    theme_minimal() +
    labs(title = "Number of Tries vs Alpha",
         subtitle = "The relationship between alpha and the computational effort required.",
         x = "Alpha", y = "Number of Tries",
         caption = "Figure 6: The number of tries required to generate samples varies with the alpha par
    theme(plot.title = element_text(hjust = 0.5),
          plot.subtitle = element_text(hjust = 0.5),
          plot.caption = element_text(hjust = 0.5),
          axis.title.x = element_text(hjust = 0.5),
          axis.title.y = element_text(hjust = 0.5),
          legend.title = element_blank())
}

alpha_range <- seq(1.1, 50, length.out = 100)
n <- 1000
generate_plot(alpha_range, n)
```

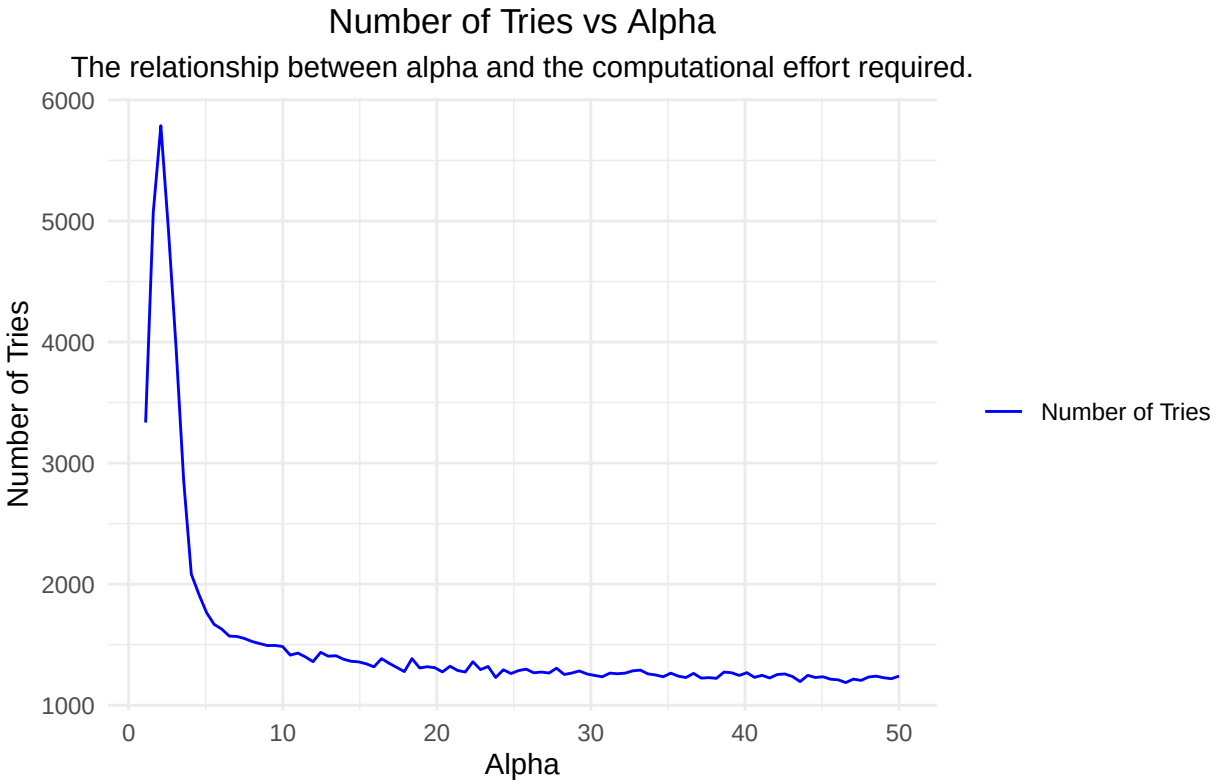

Figure 6: The number of tries required to generate samples varies with the alpha parameter. The maximum seems to be around alpha = 1

There are 3 interesting trends in this plot that is worth mentioning:

1) Initial rapid increase in tries: This increase tells us that for small values of α (seems to be less than 1 (or at least close to 1)), the algorithm might need more tries to generate the descried number of samples. This can be a result of mismatch between how well the proposal region covers the actual area under the target density. For smaller values of α, the "spread" is smaller, and hence, the rejection rate is bigger, and more tries are needed.

2) The peak: This represents the worst-case scenario of the description given above. certainly want to avoid this value of α

3) Decrease and stabilization: After a point, the value of α gets so big that the proposal region and the actual area under the target density overlap "perfectly". Then each generated sample will with high probability be accepted, and the number of tries fall rapidly, and stabilizes around a certain value. As stabilization occurs, there is no need to check for values up to $\alpha = 2000$, for practical reasons.

Just to check that our code works, we do also plot the theoretical and the sampled distributions together. As we can see, they tend to overlap pretty well!

```
library(ggplot2)
library(stats)  # For the dgamma function

alpha <- 2.5
n <- 1000

# Generate samples
```

```r
samples_result <- generate_samples_gamma_ratio(alpha, n)
samples <- samples_result$samples

# Plot
ggplot(data.frame(x = samples), aes(x = x)) +
  geom_histogram(aes(y = ..density.., fill = "Sample Distribution"), binwidth = 0.1, color = "black", al
  stat_function(fun = function(x) dgamma(x, shape = alpha, rate = 1), aes(color = "Theoretical Gamma Dis
  scale_fill_manual(name = "Legend", values = c("Sample Distribution" = "lightblue")) +
  scale_color_manual(name = "Legend", values = c("Theoretical Gamma Distribution" = "red")) +
  labs(title = paste("Samples and Theoretical Gamma Distribution for alpha =", alpha),
       x = "Sample Value",
       y = "Density",
       caption = "Figure 7: Histogram of samples with overlay of the theoretical Gamma distribution. \n
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5),
        plot.caption = element_text(hjust = 0.5),
        axis.title.x = element_text(hjust = 0.5),
        axis.title.y = element_text(hjust = 0.5),
        legend.title = element_blank())
```

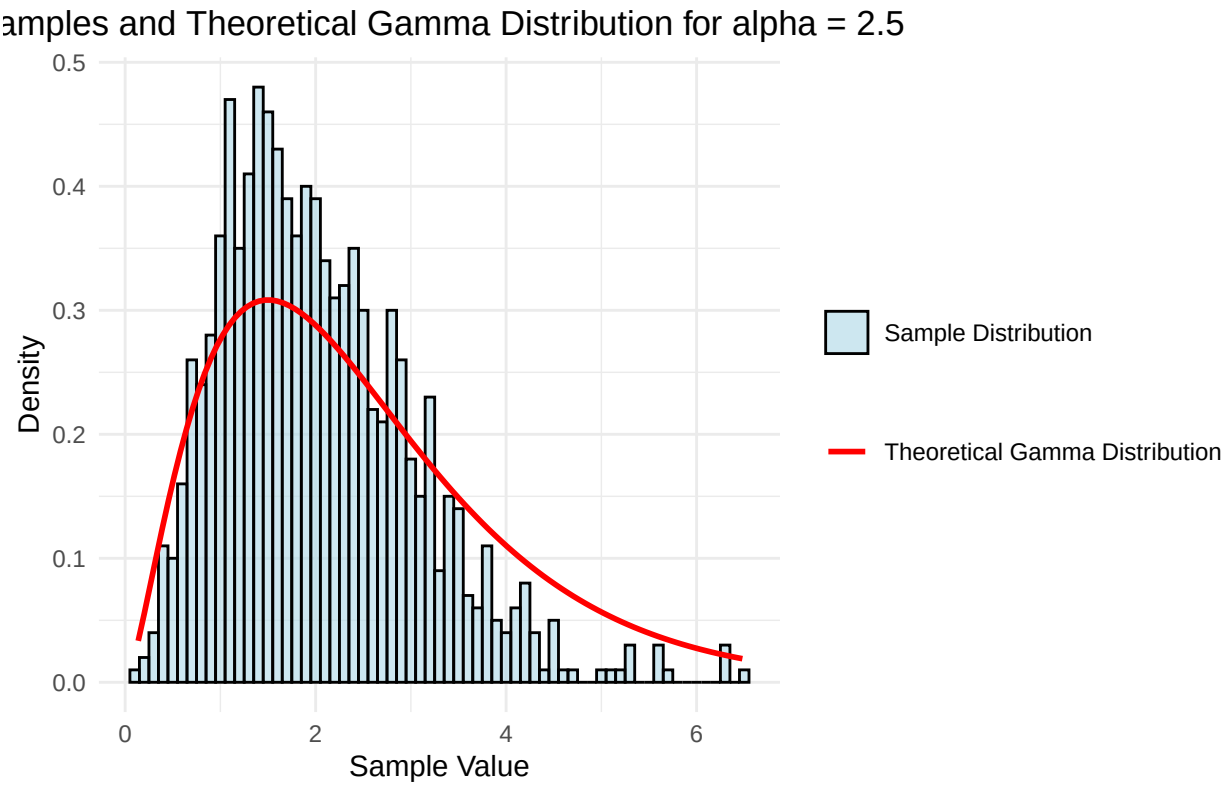

Figure 7: Histogram of samples with overlay of the theoretical Gamma distribution. Alpha is set to 2.5

Problem 3)

Note that β is an inverse scale parameter, which means that we can simply draw from Gamma$(\alpha, 1)$ and divide every sample by β. In other words, the value of β is not of great concern to our simulations.

The value of α can be more troublesome, but we have already defined all the functions we need above to calculate for different α values.

For $\alpha = 1$:

Gamma$(1, \beta) \sim$ Exponential(β). Hence, we can use the generate_samples_from_exp.

For $0 < \alpha < 1$:

We can use the rejection-sampling method

For $\alpha > 1$:

We can use ratio-of-uniforms method.

Hence, we can now sample from Gamma(α, β) for any value of the parameters.

```
generate_general_gamma_samples <- function(alpha, beta, n) {
  if (alpha <= 0 || beta <= 0) {
    stop("Both alpha and beta must be greater than 0")
  }
  if(alpha == 1) {
    return(generate_samples_from_exp(beta, n))
  }
  else if(alpha > 1){
    return(generate_samples_gamma_ratio(alpha, n)/beta)
  }
  else{
    return(generate_samples_from_f_rejection_sampling(alpha, n)/beta)
  }
}
```

Problem 4)

Task a)

To show that $z = \frac{x}{x+y}$ follows a Beta(α, β) distribution when $x \sim$ Gamma$(\alpha, 1)$ and $y \sim$ Gamma$(\beta, 1)$, we can follow these steps (assume x and y independent):

1. Find the joint density of x and y

For x, the density function is given by:

$$f_x(x) = \frac{1}{\Gamma(\alpha)} x^{\alpha-1} e^{-x}$$

For y, the density function is:

$$f_y(y) = \frac{1}{\Gamma(\beta)} y^{\beta-1} e^{-y}$$

The joint density is simply found by taking the product of the given densities. Hence:

$$f_{x,y}(x, y) = f_x(x) f_y(y) = \frac{1}{\Gamma(\alpha)} x^{\alpha-1} e^{-x} \cdot \frac{1}{\Gamma(\beta)} y^{\beta-1} e^{-y}$$

2. Change of variables and express the joint density in terms of z and w

We now make a change of variable, $z = \frac{x}{x+y}$ and $\omega = x + y$. The Jacobian of the transformation is given by:

$$J = \begin{vmatrix} \frac{\partial z}{\partial x} & \frac{\partial z}{\partial y} \\ \frac{\partial w}{\partial x} & \frac{\partial w}{\partial y} \end{vmatrix} = \begin{vmatrix} w & w \\ z & 1-z \end{vmatrix} = w$$

Then:

$$f_{z,w}(z,w) = f_{x,y}(x = \omega z, y = \omega(1-z)) \cdot J = \frac{1}{\Gamma(\alpha)\Gamma(\beta)}(wz)^{\alpha-1}e^{-wz}(w(1-z))^{\beta-1}e^{-w(1-z)} \cdot w = \frac{w^{\alpha+\beta-1}e^{-w}}{\Gamma(\alpha)\Gamma(\beta)}z^{\alpha-1}(1-z)^{\beta-1}$$

3. Integrate out ω to find the marginal density

We integrate over the domain of ω:

$$f_z(z) = \int_0^\infty \frac{w^{\alpha+\beta-1}e^{-w}}{\Gamma(\alpha)\Gamma(\beta)}z^{\alpha-1}(1-z)^{\beta-1}\, dw$$

At closer inspection, the integral over ω is the kernel of the Gamma$(\alpha + \beta, 1)$ density, which integrates to $\Gamma(\alpha + \beta)$. Hence we get:

$$f_z(z) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}z^{\alpha-1}(1-z)^{\beta-1}$$

which is exactly the density we wanted to calculate.

Task b)

The method given above is used in the code chunk below to generate the samples:

```
generate_beta_samples <- function(alpha, beta, n) {
  # Ensure alpha and beta parameters are positive
  if (alpha <= 0 || beta <= 0) {
    stop("Both alpha and beta must be greater than 0")
  }

  # Generate gamma samples for both alpha and beta parameters
  X <- generate_general_gamma_samples(alpha, 1, n)
  Y <- generate_general_gamma_samples(beta, 1, n)

  # Calculate beta samples based on the gamma samples
  beta_samples <- X / (X + Y)

  return(beta_samples)
}
```

Problem C: Monte Carlo integration and variance reduction

Problem 1)

Let $X \sim \mathrm{N}(0,1)$ be a sample, then the Monte Carlo estimator of θ is:

$$\hat{\theta}_{MC} = \frac{1}{n}\sum_{i=1}^{n} h(X_i)$$

where:

- h(x) is the indicator function $\mathrm{I}(x > 4)$

This estimator is unbiased, $E[\hat{\theta}_{MC}] = \theta$, and has a sampling variance

$$Var[\hat{\theta}_{MC}] = \frac{1}{n^2}\sum_{i=1}^{n} Var[h(X_i)] = \frac{1}{n}Var[h(X)] = \frac{1}{n(n-1)}\sum_{i=1}^{n}(h(X_i) - \hat{\theta}_{MC})^2$$

Provided this information, we know that the statistic:

$$T = \frac{\hat{\theta}_{MC} - \theta}{\sqrt{Var[\hat{\theta}_{MC}]}} \sim t_{n-1}$$

This is used in the calculation of the confidence interval.

```r
# Set a seed for reproducibility
set.seed(25935)

# Number of samples
n <- 100000

# Generate samples using the 'generate_standard_normal' function
samples <- generate_standard_normal(n)

h <- function(x) {
  1*(x>4)
}

h_x <- h(samples)

theta_hat <- mean(h_x)

# Sampling variance
var_theta_MC <- var(h_x)

t <- qt(0.025, df = n-1, lower.tail = FALSE)
confidence_MC_upper <- theta_hat + t * sqrt(var_theta_MC/n)
confidence_MC_lower <- theta_hat - t * sqrt(var_theta_MC/n)

# Output the results
cat("Estimate of theta:", theta_hat, "\n")
```

```
## Estimate of theta: 1e-05
```

```
cat("95% Confidence Interval for theta: [", confidence_MC_lower, ",", confidence_MC_upper, "]\n")
```

```
## 95% Confidence Interval for theta: [ -9.599877e-06 , 2.959988e-05 ]
```

Problem 2)

Suppose the proposal distribution:

$$g(x) = \begin{cases} cx \exp\left(-\frac{1}{2}x^2\right), & \text{for } x > 4, \\ 0, & \text{otherwise,} \end{cases}$$

where the normalizing constant $c = \left(\int_4^\infty x e^{-\frac{1}{2}x^2} dx\right)^{-1} = e^8$

Hence:

$$g(x) = \begin{cases} e^8 x \exp\left(-\frac{1}{2}x^2\right), & \text{for } x > 4, \\ 0, & \text{otherwise,} \end{cases}$$

We can sample from the proposal distribution by for example using inversion sampling:

$$G(x) = p = 1 - e^{8-\frac{x^2}{2}} \sim U(0,1)$$

Solving for x gives:

$$G^{-1}(p) = x = \sqrt{16 - 2\ln(1-p)}$$

If X is a sample from the proposal distribution g, then the importance sampling estimator is given by:

$$\hat{\theta}_{IS} = \frac{1}{n}\sum_{i=1}^{n} h(X_i)\omega(X_i)$$

where

- $$\omega = f/g$$

  , which are the weights

This estimator is unbiased, with sampling variance calculated in the same way as in problem C task 1, given by:

$$\frac{1}{n(n-1)}\sum_{i=1}^{n}(h(X_i)\omega(X_i) - \hat{\theta}_{IS})^2$$

The code chunk below gives ut the estimated value by Importance Sampling

```r
# Set a seed for reproducibility
set.seed(25935)

# Number of samples
n <- 100000

sample_proposal <- sqrt(16 - 2*log(1-runif(n)))

weight <- function(x) {
  f <- dnorm(x)
  g <- ifelse(
    test = sample_proposal > 4,
    yes = sample_proposal*exp(-0.5*(sample_proposal^2-16)),
    no = 0
  )
  return(f/g)
}

w <- weight(sample_proposal)

hw = sample_proposal * w

theta_IS <- mean(hw)
var_theta_IS <- var(hw)

t <- qt(0.025, df = n-1, lower.tail = FALSE)

confidence_IS_upper <- theta_IS + t * sqrt(var_theta_IS/n)
confidence_IS_lower <- theta_IS - t * sqrt(var_theta_IS/n)

# Output the results
cat("Estimate of theta:", theta_IS, "\n")
```

```
## Estimate of theta: 0.0001338302
```

```r
cat("95% Confidence Interval for theta: [", confidence_IS_lower, ",", confidence_IS_upper, "]\n")
```

```
## 95% Confidence Interval for theta: [ 0.0001338302 , 0.0001338302 ]
```

The big difference between these methods are the number of samples used to get the precision needed. So the question arises, how many samples of Monte Carlo would we need to achieve the same precision? That can be calculated by looking at the variance terms:

```r
var(samples)/var(hw)
```

```
## [1] 2.591746e+39
```

Problem 3)

Task a)

We want to modify the way we extract our sample so that we return a pair of samples, where one takes $U \sim \text{Uniform}(0,1)$ as an argument and the other takes $1-U \sim \text{Uniform}(0,1)$ as an argument.

Let us draw two pairs of samples from the proposal distribution, $X_n^{(1)}$ and $X_n^{(2)}$, with the importance sampling estimators:

$$\hat{\theta}_{IS}^{(1)} = \frac{1}{n}\sum_{i=1}^{n} h(X_i^{(1)})\omega(X_i^{(1)})$$

$$\hat{\theta}_{IS}^{(2)} = \frac{1}{n}\sum_{i=1}^{n} h(X_i^{(2)})\omega(X_i^{(2)})$$

respectively.

Given these two estimators, we can now express the antihetic estimator as such (per definition):

$$\hat{\theta}_{AS} = \frac{\hat{\theta}_{IS}^{(1)} + \hat{\theta}_{IS}^{(2)}}{2}$$

Which is an unbiased estimator with a pretty "ugly" variance expression, so I do not write it up here. Anyhow, the new sampling function is given below:

```
# Set a seed for reproducibility
set.seed(25935)

sample_proposal_pair <- function(n) {
  u <- runif(n)
  list(
    X1 = sqrt(16 - 2*log(1-u)),
    X2 = sqrt(16 - 2*log(u))
  )
}
```

Task b)

```
# Set a seed for reproducibility
set.seed(25935)

# Number of samples
n <- 50000

sample_pair <- sample_proposal_pair(n)

hw1 <- h(sample_pair$X1) * weight(sample_pair$X1)
hw2 <- h(sample_pair$X2) * weight(sample_pair$X2)

hwAS <- 0.5*(hw1 + hw2)

theta_AS <- mean(hwAS)
theta_var_AS <- var(hwAS)

t <- qt(0.025, df = n-1, lower.tail = FALSE)
confidence_AS_lower <- theta_AS - t * sqrt(theta_var_AS / n)
confidence_AS_upper <- theta_AS + t * sqrt(theta_var_AS / n)
```

```r
# Output the results
cat("Estimate of theta:", theta_AS, "\n")
```

```
## Estimate of theta: 0.0001418561
```

```r
cat("95% Confidence Interval for theta: [", confidence_AS_lower, ",", confidence_AS_upper, "]\n")
```

```
## 95% Confidence Interval for theta: [ 0.0001160627 , 0.0001676495 ]
```

Problem D: Rejection sampling and importance sampling

Problem 1)

Given data:

- $y_1 = 125$
- $y_2 = 18$
- $y_3 = 20$
- $y_4 = 34$

The proposal density, denoted g, is simply given as the Uniform(0,1)

Observed posterior density, not normalized:

$$f^*(\theta|y) \propto (2+\theta)^{y_1}(1-\theta)^{y_2+y_3}\theta^{y_4}$$

To use rejection sampling, we need to normalize the posterior density. That is we need to find a constant C such that $f(\theta|y) = C \cdot f^*(\theta|y)$. To solve for C, we need to evaluate the following integral:

$$\frac{1}{C} = \int_0^1 f^*(\theta|y)d\theta$$

Further, to use rejection sampling we also need that:

$$f(\theta|y) \leq k$$

We set k to be:

$$k = \frac{f(\theta_{max}|y)}{K}$$

Given all this information, the rejection sampling algorithm can simply be implemented. The algorithm goes as follows:

1. Generate θ^* from the Uniform(0, 1) distribution.
2. Calculate the acceptance probability A for θ^* as the ratio between the target density at θ^* and the maximum value of the target density.
3. Generate a random number u from Uniform(0, 1).

4. If $u < A$, accept θ^*, otherwise reject it.
5. Repeat steps 1-4 until enough values are generated.

```
f_theta_given_y_unnormal <- function(theta, y1 = 125, y2 = 18, y3 = 20, y4 = 34) {
  return((2 + theta)^y1 * (1 - theta)^(y2 + y3) * theta^y4)
}

C <- integrate(function(theta)(f_theta_given_y_unnormal(theta)),
               lower = 0,
               upper = 1)$value

f_theta_given_y_normal <- function(theta, y1 = 125, y2 = 18, y3 = 20, y4 = 34) {
  return(f_theta_given_y_unnormal(theta) / C)
}

max_f_theta_given_y_unnormal <- optimize(function(theta)(f_theta_given_y_unnormal(theta)),
                                         interval = c(0,1),
                                         maximum = TRUE)$objective

k <- max_f_theta_given_y_unnormal / C

# The rejection sampling algorithm
rejection_sampling <- function(N, y1 = 125, y2 = 18, y3 = 20, y4 = 34) {
  count <- 0
  sample_vals <- c()

  while(length(sample_vals) < N) {
    u <- runif(1)
    theta <- runif(1)
    A <- f_theta_given_y_normal(theta) / k
    if(u <= A) {
      sample_vals <- rbind(sample_vals, theta)
    }
    count <- count + 1
  }
  return(list("samples" = sample_vals, "count" = count))
}
```

Problem 2)

The Monte Carlo estimate is simply given by:

$$\mu = \frac{1}{M} \sum_{i=1}^{M} \theta_i$$

```
# Estimate the posterior mean using vectorized sampling
N <- 10000
theta <- rejection_sampling(N)

# Calculate the posterior mean
monte_carlo_est <- mean(theta$samples)
```

```
monte_carlo_est
```

```
## [1] 0.6239792
```

We want to verify our result by numerical integration. The expected value of θ given y is given by:

$$E(\theta|y) = \int_0^1 \theta \cdot f(\theta|y) d\theta$$

We can now simply apply the integration using built in functions in R:

```
numeric_est <- integrate(function(theta)(theta * f_theta_given_y_normal(theta)),
                         lower = 0,
                         upper = 1)$value

numeric_est
```

```
## [1] 0.6228061
```

These values are clearly the same, which verifies that our R-code for the rejection sampling and the Monte-Carlo integration is implemented correctly!

We will now draw a histogram of the samples and compare it with the theoretical density distribution. The estimated posterior mean and the numerical calculated mean are also marked. We see that things works out well!

```
# Plot the histogram of the sampled values
hist(theta$samples, breaks = 50, probability = TRUE, main = "Posterior Distribution of Theta", xlab = "

# Overlay the true distribution curve
curve(f_theta_given_y_normal(x, y1 = 125, y2 = 18, y3 = 20, y4 = 34), add = TRUE, col = "blue", lwd = 2

# Mark the estimated posterior mean from Monte Carlo simulation
abline(v = monte_carlo_est, col = "red", lwd = 2, lty = 2)

# Mark the numerical estimated mean
abline(v = numeric_est, col = "green", lwd = 2, lty = 2)

# Add a legend
legend("topright",
       legend = c("Theoretical density", "Monte Carlo mean", "Numerical mean"),
       col = c("blue", "red", "green"),
       lwd = 2,
       lty = c(1, 2, 2),
       merge = TRUE)

# Add figure text
mtext("Figure 8: Posterior distribution of Theta along with the Monte Carlo and numerical means, overlay
      side = 1,
      line = 4,
      cex = 0.8,
      adj = 0)
```

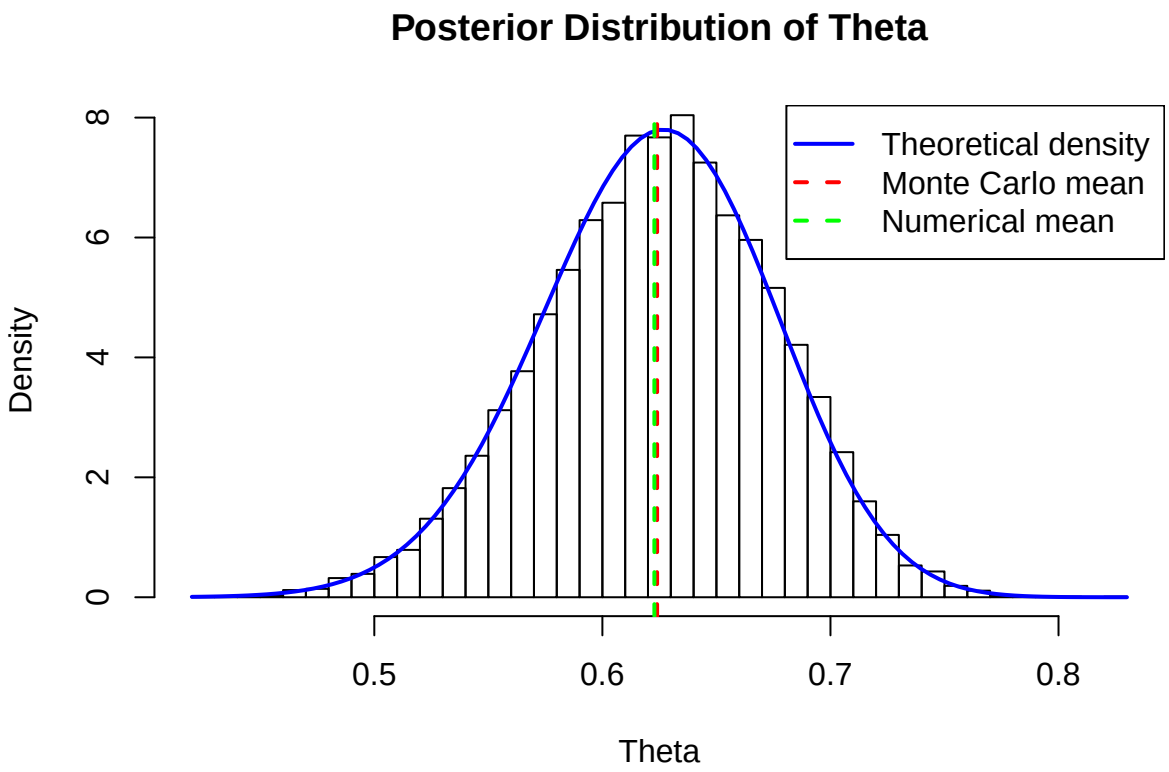

Figure 8: Posterior distribution of Theta along with the Monte Carlo and numerical means, ov

Problem 3)

k can be seen as a measure of the expected number of trials up the first sample from $f(\theta|y)$. As all the draws are independent, this number represents the average number of tries needed on average to obtain one sample from f. Hence we get:

```
k
```

```
## [1] 7.799308
```

We can also use the sampler to estimate this number. We can look at the number of times the while loop runs divided by the amount of samples obtained of $f(\theta|y)$.

```
theta$count / N
```

```
## [1] 7.8432
```

We see that both these numbers are approximately equal. We need to approximately generate 7.8 random numbers in order to obtain one sample from f.

Problem 4)

To calculate the posterior mean under the new prior Beta(1,5), we can use the importance weights based on the samples calculated in 2. This saves us alot of computation. What we do need to calculate is the importance sampling weights, which is the ratio of the new prior density f_{new} to the old prior density f_{old}.

$$\omega(theta_i) = \frac{f_{new}}{f_{old}} \propto (1-\theta_i)^4$$

We want to estimate μ using importance sampling. The posterior mean under the new prior is evaluated as the weighted average of the samples:

$$\mu = \frac{\sum_{i=1}^{n} \theta_i \omega(\theta_i)}{\sum_{i=1}^{n} \omega(\theta_i)}$$

Then we get:

```
weights <- (1 - theta$samples)^4

# Weighted average for posterior mean under Beta(1, 5) prior
weighted_posterior_mean <- sum(theta$samples * weights) / sum(weights)

# Display the estimated posterior mean under the new prior
weighted_posterior_mean
```

```
## [1] 0.5966464
```

We want to verify this result numerically, using the integrate function in R.

```
f_theta_given_y_unnormal_new_prior <- function(theta, y1 = 125, y2 = 18, y3 = 20, y4 = 34) {
  return((2 + theta)^y1 * (1 - theta)^(y2 + y3 + 4) * theta^y4)
}

C <- integrate(function(theta)(f_theta_given_y_unnormal_new_prior(theta)),
               lower = 0,
               upper = 1)$value

true_mu <- integrate(function(theta)(theta * f_theta_given_y_unnormal_new_prior(theta) / C),
lower = 0,
upper = 1)$value

true_mu
```

```
## [1] 0.5959316
```

Which we see, the true value and the numerically estimated value are similar, which indicates our code is working correct. Another additional observation of interest is that the change of prior had a big affect on the mean. The Beta(1,1) posterior mean has a bigger value than the Beta(1,5) posterior mean. This can clearly be seen from the plot of the both priors - where one sees that the Beta(1,5) prior prefers smaller values for θ, while the Beta(1,1) prior does not prefer any values for θ.

```
theta_values <- seq(0, 1, length.out = 100)
df <- data.frame(theta = theta_values)

# Calculate Beta(1,1) and Beta(1,5) densities
df$beta_1_1 <- dbeta(df$theta, 1, 1)
df$beta_1_5 <- dbeta(df$theta, 1, 5)
```

```r
# Plot
ggplot(df, aes(x = theta)) +
  geom_line(aes(y = beta_1_1, color = "Beta(1,1)"), size = 1) +
  geom_line(aes(y = beta_1_5, color = "Beta(1,5)"), size = 1) +
  labs(title = "Beta(1,1) vs Beta(1,5) Distributions",
       x = expression(theta),
       y = "Density") +
  scale_color_manual(values = c("Beta(1,1)" = "blue", "Beta(1,5)" = "red")) +
  theme_minimal() +
  theme(legend.title = element_blank())
```