Exercise 2 - Random Number Generation through CDF and acceptance-rejection sampling

Stefan Merdian

2024-10-21

1) Pseudo-random number generation with Linear Congruential Random Number

Summarise the concept of pseudo-random number generation with Linear Congruential Random Number Generation Algorithm using the available code examples from the course to create a working code example and simulate one pseudo-random sample.

The Linear Congruential Generator (LCG) is a well-known and simple pseudo-random number generator (PRNG). It operates by repeatedly applying a modulo operation on a large integer `m`. The algorithm follows the recurrence relation:

$$x_{n+1} = (a \cdot x_n + c) \mod m$$

Where:

- `m` is the modulus (typically a large integer)
- `a` is the multiplier (usually chosen near $\sqrt{m}$)
- `c` is the increment (where $0 \leq c < m$)
- x_0 is the initial seed (starting value)

The algorithm produces a sequence of pseudo-random numbers, denoted u_n, by normalizing each x_n using the formula:

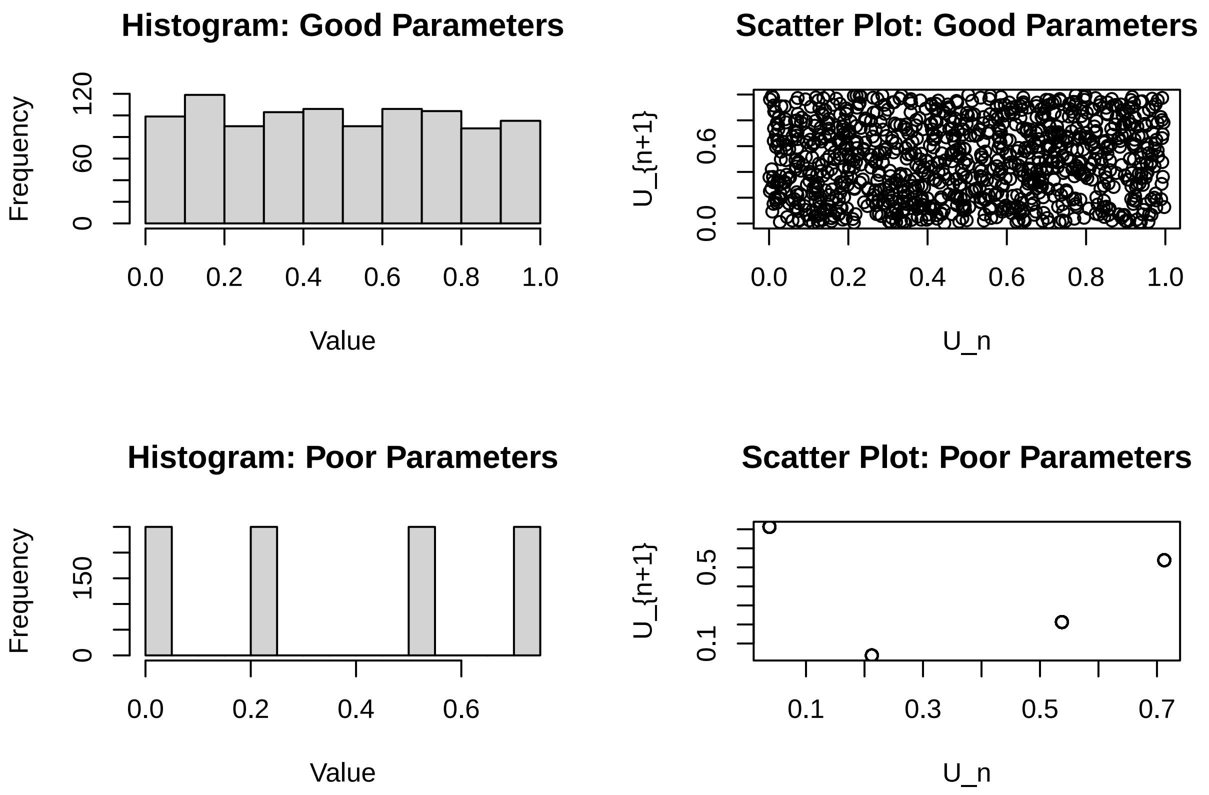
$$u_{n+1} = \frac{x_{n+1}}{m}$$

This process generates random numbers in the interval [0, 1].

```r
lcg <- function(n, m, a, c = 0, x0) {
    us <- numeric(n)
    for (i in 1:n) {
        x0 <- (a * x0 + c) %% m
        us[i] <- x0 / m
    }
    return(us)
}
```

We just pick for **m** a high prime number like `1000023301` and for **a** a number which is close to $\sqrt{m}$ in our case, we pick `31620`

```r
lcg_good <- lcg(1000, m = 1000023301, a = 31620, c = 0, x0 = 77777)
lcg_bad <- lcg(1000, m = 80, a = 19, c = 0, x0 = 77777)

par(mfrow = c(2, 2))
hist(lcg_good, main = "Histogram: Good Parameters", xlab = "Value")
plot(lcg_good[1:999], lcg_good[2:1000],
     main = "Scatter Plot: Good Parameters",
     xlab = "U_n", ylab = "U_{n+1}")
hist(lcg_bad, main = "Histogram: Poor Parameters", xlab = "Value")
plot(lcg_bad[1:999], lcg_bad[2:1000],
     main = "Scatter Plot: Poor Parameters",
     xlab = "U_n", ylab = "U_{n+1}")
```



The scatter plot for good parameters shows a well-spread distribution of random numbers, indicating minimal correlation between consecutive values and proper randomness. The poor parameters result in clear patterns and clustering, as seen in the histogram and scatter plot, where the values cycle through only a small subset of possible numbers, leading to poor randomness.

This highlights the importance of carefully selecting parameters, particularly the modulus mm, in PRNGs. Since the method is cyclic, once a specific value $x_k = x_0$ is reached, the sequence will repeat. A poor choice of parameters results in short cycles and high correlation between consecutive numbers, severely impacting randomness.

2) The exponential distribution

Next we will :

- Write down the mathematical expression for this.
- Write an R function which takes as input the number of samples required and the parameter lamda. We will use runif function to create the uniform random variables.
- Try three different values of lambda and create 1000 random samples. We will than evaluate the quality of your random number generator using qq-plots which compare against a real exponential distribution with the specified parameter lambda.

Mathematical expression:**

1. **The exponential distribution has the following cdf:**

$$F(x) = 1 - e^{-\lambda x}$$

2. **Set** $F(x) = U$: Since U is uniformly distributed between 0 and 1, we can equate the CDF $F(x)$ to U:

$$U = 1 - e^{-\lambda x}$$

3. **Solve for** x

$$U = 1 - e^{-\lambda x}$$

Rearranging:

$$e^{-\lambda x} = 1 - U$$

Taking the natural logarithm of both sides:

$$-\lambda x = \ln(1 - U)$$

Finally, solve for x:

$$x = -\frac{1}{\lambda}\ln(1 - U)$$

Since U is a uniform random variable, $1 - U$ is also uniformly distributed in $[0, 1]$, so we can simplify this to:

$$x = -\frac{1}{\lambda}\ln(U)$$

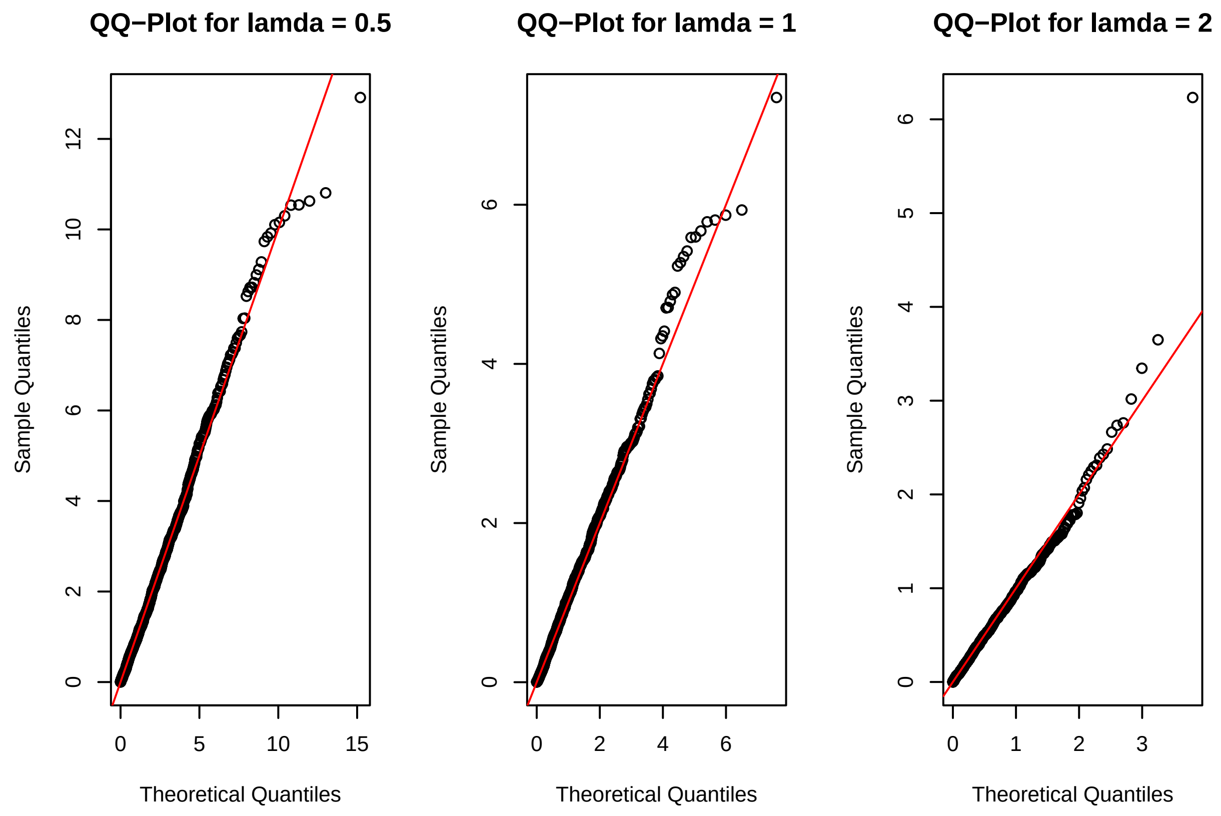
Function for $x = -\frac{1}{\lambda}\ln(U)$

```r
custom_expo <- function(n, lambda) {
  U <- runif(n, min = 0, max = 1)
  X <- -log(U) / lambda
  return(X)
}
```

Evaluation

```r
lambda_values <- c(0.5, 1, 2)

par(mfrow = c(1, 3))
```

```r
for (lambda in lambda_values) {
  samples <- custom_expo(1000, lambda)

  qqplot(qexp(ppoints(1000), rate = lambda), samples,
         main = paste("QQ-Plot for lamda =", lambda),
         xlab = "Theoretical Quantiles", ylab = "Sample Quantiles")


  abline(0, 1, col = "red")
}
```



```r
par(mfrow = c(1, 1))
```

The random samples generated by your function closely match the exponential distribution for all three values of lambda, particularly in the central range of the data. In all three plots, you can see some deviations at the upper tail.These outliers in the upper tail often deviate from the theoretical line because they represent rare events that occur less predictably.

3) Beta distribution

Beta Distribution Overview:

The Beta distribution has the pdf:

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

Beta distribution = 2 ($\alpha = 2$, $\beta = 2$):

For $\alpha = \beta = 2$, the PDF simplifies to:

$$f(x; 2, 2) = 6x(1 - x)$$

A natural proposal distribution is **Uniform(0, 1)**, as it shares the same support, and has constant density $g(x) = 1$. To determine a suitable constant c, we need to make sure that $c \cdot g(x) \geq f(x)$ for all $x \in [0, 1]$, where $g(x)$ is the uniform density. The maximum value of $f(x)$ is at $x = 0.5$, where $f(0.5) = 1.5$. Hence, we can set $c = 1.5$.

Generalizing the Approach:

We'll first implement the method for $\alpha = 2$, $\beta = 2$ and then extend it to arbitrary $\alpha > 1$, $\beta > 1$.

Here's the function for the acceptance-rejection method:

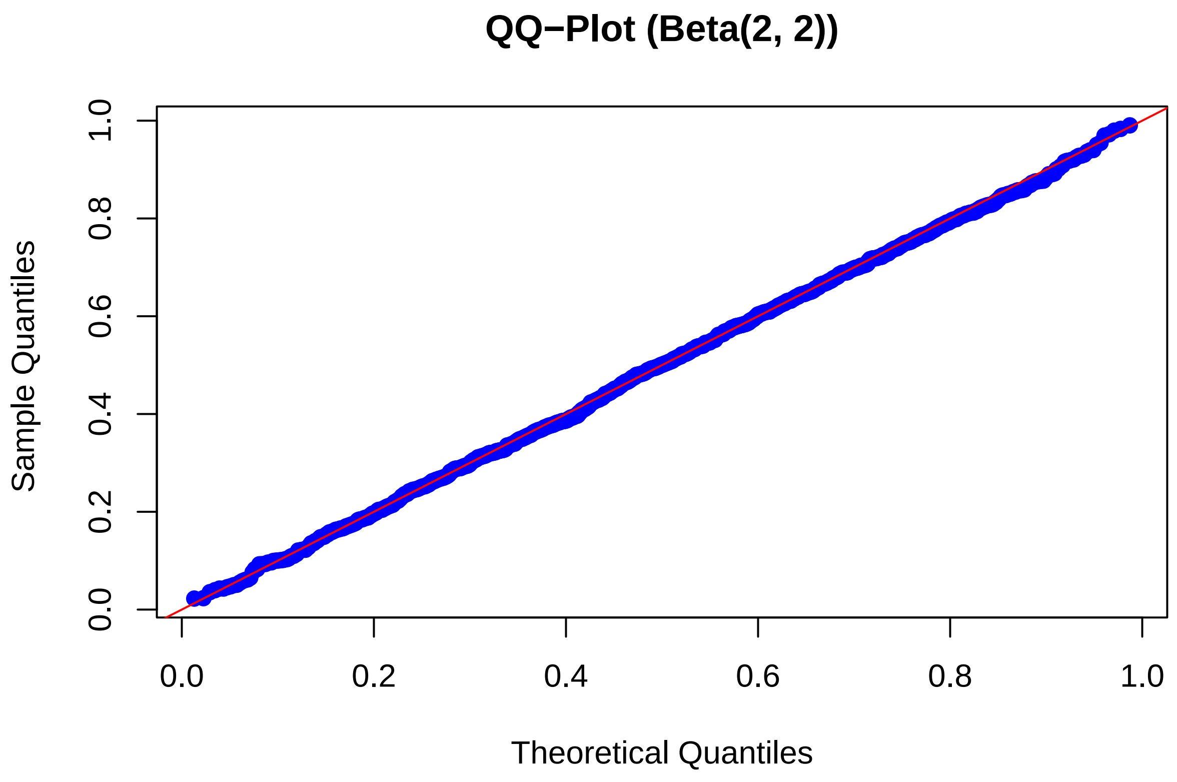
```r
sample_beta_2_2 <- function(n) {
    samples <- numeric(n)
    accepted <- 0
    c <- 1.5

    while (accepted < n) {
        y <- runif(1)
        u <- runif(1)

        if (u <= (6 * y * (1 - y)) / c) {
            accepted <- accepted + 1
            samples[accepted] <- y
        }
    }

    return(samples)
}
```

For a general Beta distribution with $\alpha > 1$ and $\beta > 1$, the method remains similar, but the constant c must be adjusted according to the specific parameters. The maximum value of the Beta density is at $x = \frac{\alpha-1}{\alpha+\beta-2}$, allowing us to calculate the appropriate c.

Below is a function that implements acceptance-rejection sampling for any Beta(α, β) distribution:

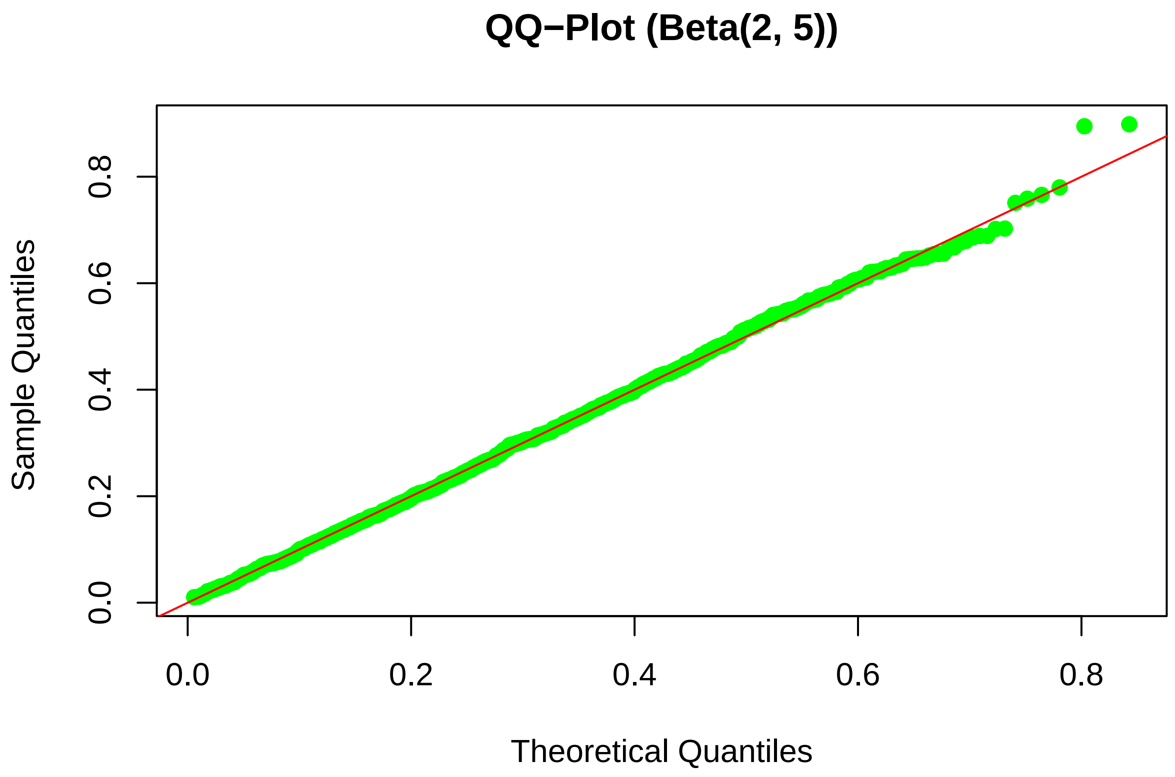
```r
sample_beta <- function(n, alpha, beta) {
    if (alpha <= 1 || beta <= 1) {
        stop("alpha and beta must be greater than 1")
    }

    mode_x <- (alpha - 1) / (alpha + beta - 2)
    max_beta_val <- dbeta(mode_x, alpha, beta)

    samples <- numeric(n)
    count <- 0

    while (count < n) {
        y <- runif(1)
        u <- runif(1)
        if (u <= dbeta(y, alpha, beta) / max_beta_val) {
            count <- count + 1
            samples[count] <- y
        }
    }

    return(samples)
}
```

```
library(ggplot2)


set.seed(12433732)
samples_beta_22 <- sample_beta_2_2(1000)
samples_beta <- sample_beta(1000, 2, 5)

theoretical_quantiles_2_2 <- qbeta(ppoints(1000), 2, 2)
theoretical_quantiles <- qbeta(ppoints(1000), 2, 5)

qqplot(theoretical_quantiles_2_2, samples_beta_22,
       main = "QQ-Plot (Beta(2, 2))",
       xlab = "Theoretical Quantiles", ylab = "Sample Quantiles ",
       pch = 19, col = "blue")
abline(0, 1, col = "red")
```

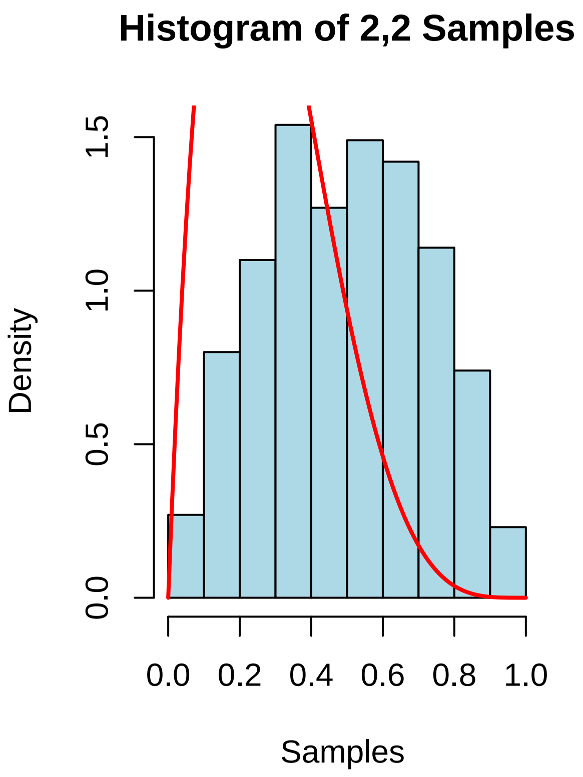


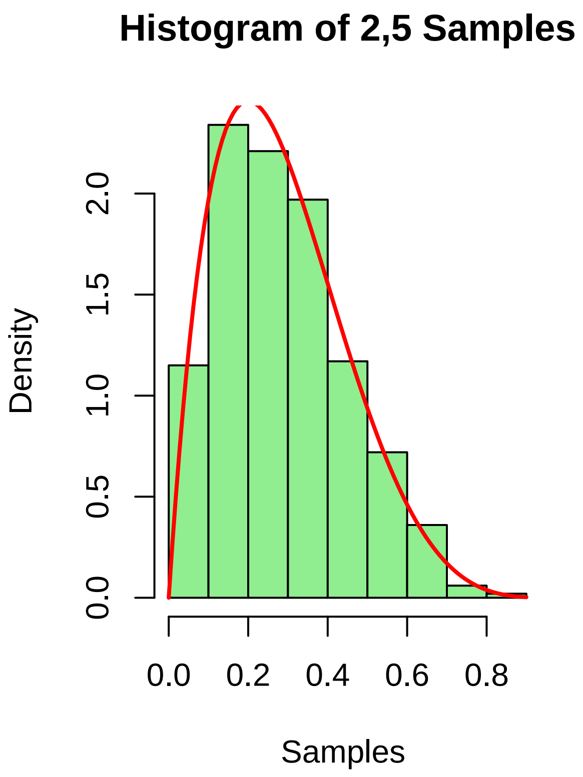
```
qqplot(theoretical_quantiles, samples_beta,
       main = "QQ-Plot (Beta(2, 5))",
       xlab = "Theoretical Quantiles", ylab = "Sample Quantiles ",
```

```r
       pch = 19, col = "green")
abline(0, 1, col = "red")
```



```r
par(mfrow = c(1, 2))
hist(samples_beta_22, probability = TRUE, main = "Histogram of 2,2 Samples",
     xlab = "Samples ", col = "lightblue")
curve(dbeta(x, 2, 5), col = "red", add = TRUE, lwd = 2)

hist(samples_beta, probability = TRUE, main = "Histogram of 2,5 Samples",
     xlab = "Samples ", col = "lightgreen")
curve(dbeta(x, 2, 5), col = "red", add = TRUE, lwd = 2)
```





```r
par(mfrow = c(1, 1))
```

The shape and skewness of the Beta distribution are determined by the relationship between alpha and beta. When they are equal, the distribution is symmetric; when they differ, the distribution skews toward the lower or higher end of the interval.