Random Number Generation through CDF and acceptance-rejection sampling

Mahtab Nahayati

2024-10-23

Summary of LCRNG

The Linear Congruential Generator (LCG) is a type of pseudo-random number generator that generates a sequence of numbers using a linear recurrence relation:

$$X_{n+1} = (aX_n + c) \mod m$$

where: **Xn** is the current state (or seed).

a is the multiplier.

c is the increment.

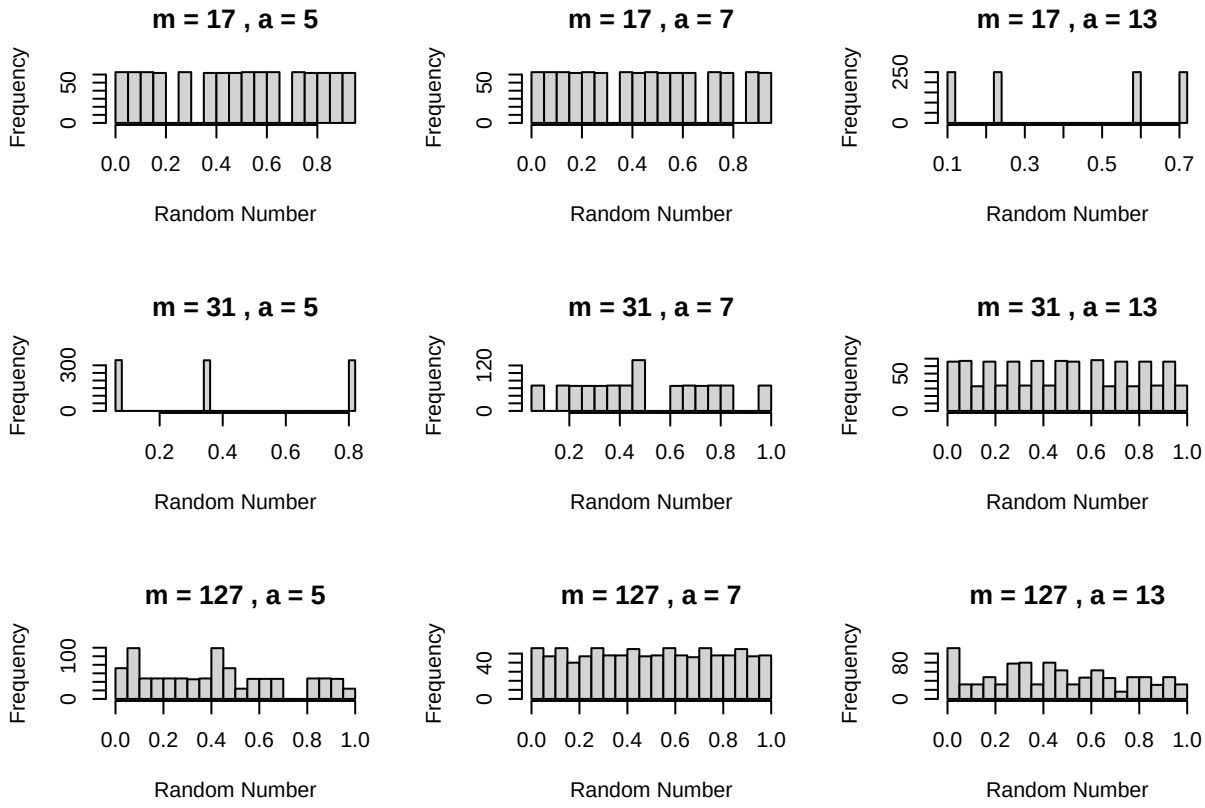
m is the modulus.

The sequence is generated in a cyclic manner, meaning once the sequence returns to its starting point, it will repeat indefinitely. The choice of parameters m,a, and c significantly affects the quality of the generated sequence.

Working Code Example

```
# Linear Congruential Random Number Generator Function
lcg <- function(n, m, a, c, seed) {
  x <- numeric(n)  # Vector to hold generated numbers
  x[1] <- seed

  for (i in 2:n) {
    x[i] <- (a * x[i-1] + c) %% m  # Update state
  }

  return(x / m)  # Normalize to [0, 1]
}

# Parameters
n_samples <- 1000
m_values <- c(17, 31, 127)  # Different moduli
a_values <- c(5, 7, 13)      # Different multipliers
c <- 1                       # Increment
seed <- 2                    # Initial seed

# Generate and plot pseudo-random numbers
par(mfrow = c(length(m_values), length(a_values)))  # Create a plotting area
```

```r
for (m in m_values) {
  for (a in a_values) {
    rand_numbers <- lcg(n_samples, m, a, c, seed)
    hist(rand_numbers, main = paste("m =", m, ", a =", a), xlab = "Random Number", breaks = 30)
  }
}
```



We can observe that the choice of parameters significantly impacts the quality of the generated pseudo-random numbers. The larger the number m produces better results, amximize the cycle length of the sequence.

Random Sample from Exponential Distribution

The cumulative distribution function (CDF) for the exponential distribution is given by:

$$F(x) = 1 - \exp(-x)$$

To generate a random sample from an exponential distribution using uniform random variables, we use the inverse CDF method:

1.Generate a uniform random variable U Uniform(0,1).

2.Apply the inverse CDF : $X = -1/\ln(1-U)$ or $X = -1/\ln(U)$

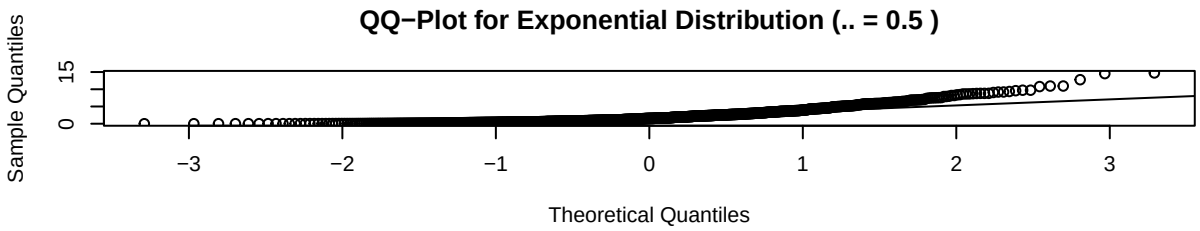
3.Create 1000 random samples for three different values of lambda and use QQ-Plots to evaluate the quality of random number generator.

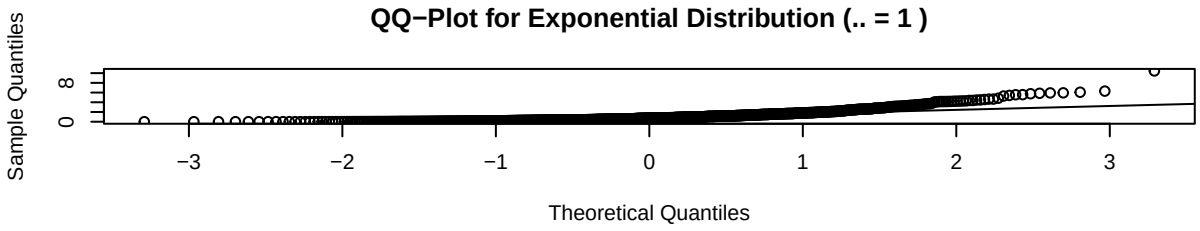
If the points roughly follow the reference line, it means our random sample generator does a good job at mimicking the theoretical exponential distribution.

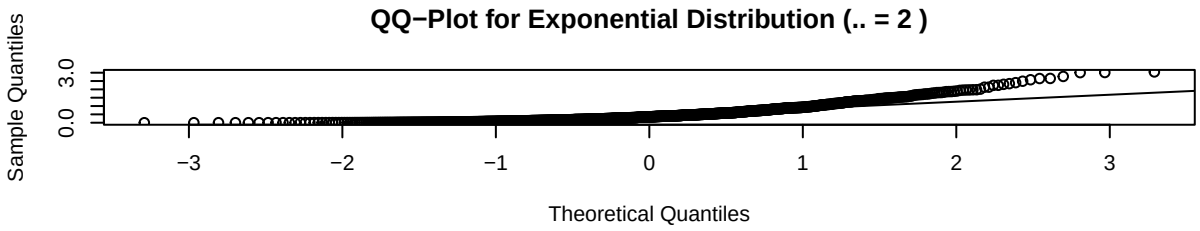
```r
# Function to generate random samples from an exponential distribution
exponential_sample <- function(n, lambda) {
  U <- runif(n)  # Generate uniform random variables
  X <- -log(U) / lambda  # Apply inverse CDF
  return(X)
}

# Parameters for three different values of lambda
lambda_values <- c(0.5, 1, 2)

# Generate samples and evaluate with QQ plots
par(mfrow = c(3, 1))  # Setup plotting area
for (lambda in lambda_values) {
  samples <- exponential_sample(1000, lambda)
  qqnorm(samples, main = paste("QQ-Plot for Exponential Distribution ( =", lambda, ")"))
  qqline(samples)
}
```

Deviation from the line indicate discrepancies.

```
## Warning in title(...): Konvertierungsfehler für 'QQ-Plot for Exponential
## Distribution ( = 0.5 )' in 'mbcsToSbcs': Punkt ersetzt <ce>

## Warning in title(...): Konvertierungsfehler für 'QQ-Plot for Exponential
## Distribution ( = 0.5 )' in 'mbcsToSbcs': Punkt ersetzt <bb>

## Warning in title(...): Konvertierungsfehler für 'QQ-Plot for Exponential
## Distribution ( = 1 )' in 'mbcsToSbcs': Punkt ersetzt <ce>

## Warning in title(...): Konvertierungsfehler für 'QQ-Plot for Exponential
## Distribution ( = 1 )' in 'mbcsToSbcs': Punkt ersetzt <bb>

## Warning in title(...): Konvertierungsfehler für 'QQ-Plot for Exponential
## Distribution ( = 2 )' in 'mbcsToSbcs': Punkt ersetzt <ce>

## Warning in title(...): Konvertierungsfehler für 'QQ-Plot for Exponential
## Distribution ( = 2 )' in 'mbcsToSbcs': Punkt ersetzt <bb>
```







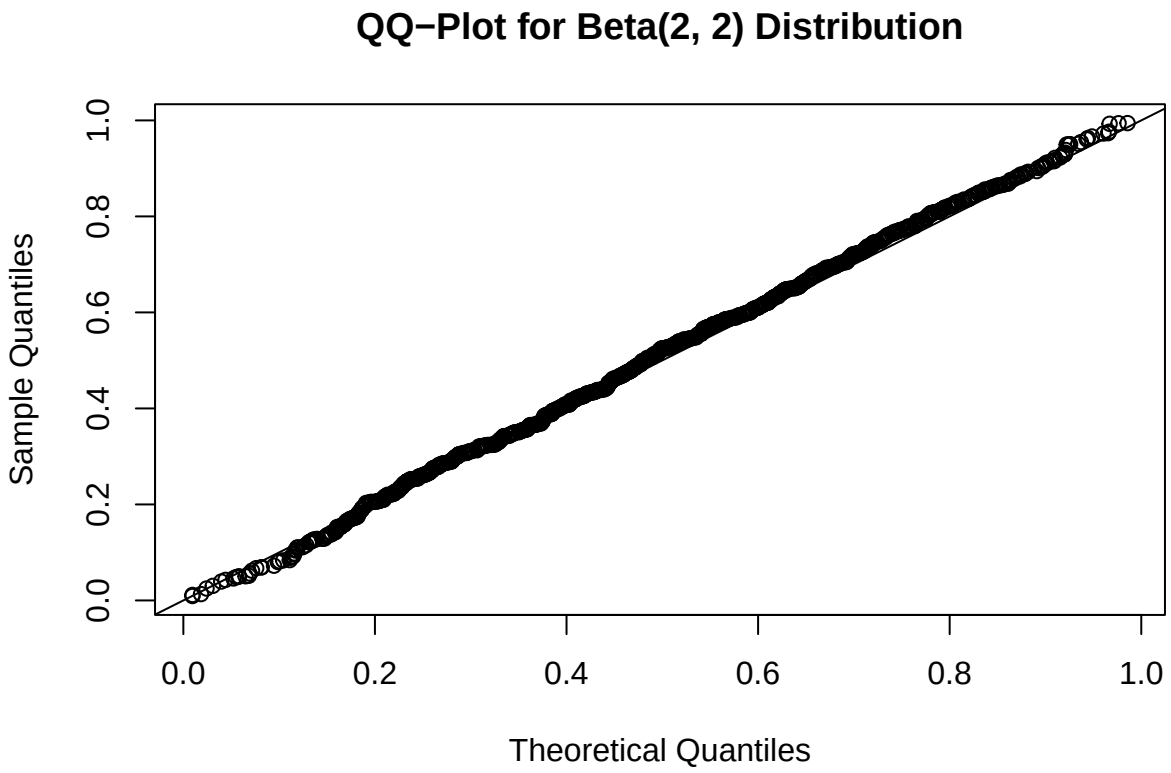
Acceptance-Rejection Sampling for Beta Distribution

The probability density function (PDF) of the Beta distribution is given by:f(x; ,)=Γ(+)Γ( )Γ( )x −1(1−x) −

```r
# Beta distribution PDF (unnormalized version for rejection sampling)
beta_pdf <- function(x, alpha, beta) {
  return(x^(alpha - 1) * (1 - x)^(beta - 1))
}

# Function to sample from Beta distribution using acceptance-rejection method
beta_sample_ar <- function(n, alpha, beta) {
  samples <- numeric(n)  # Vector to hold accepted samples
  accepted <- 0          # Counter for accepted samples

  # Constant 'c' should be the maximum of the Beta(alpha, beta) PDF on [0, 1].
  # For simplicity, the constant can be approximated by evaluating at the mode:
  # Mode of Beta(alpha, beta) = (alpha - 1) / (alpha + beta - 2) for alpha, beta > 1
  mode <- (alpha - 1) / (alpha + beta - 2)
  c <- beta_pdf(mode, alpha, beta)  # Compute maximum of the unnormalized PDF

  while (accepted < n) {
    y <- runif(1)  # Sample from proposal distribution (Uniform)
    u <- runif(1)  # Sample from Uniform(0, 1) for acceptance/rejection

    # Accept/Reject condition based on Beta PDF
```

```
    if (u <= beta_pdf(y, alpha, beta) / c) {
      accepted <- accepted + 1
      samples[accepted] <- y
    }
  }

  return(samples)
}

# Generate samples from Beta(2, 2) using acceptance-rejection
beta_samples_2_2 <- beta_sample_ar(1000, 2, 2)

# QQ-plot to compare generated samples with the true Beta(2, 2) distribution
true_samples_2_2 <- rbeta(1000, 2, 2)
qqplot(true_samples_2_2, beta_samples_2_2, main = "QQ-Plot for Beta(2, 2) Distribution",
       xlab = "Theoretical Quantiles", ylab = "Sample Quantiles")
abline(0, 1)
```

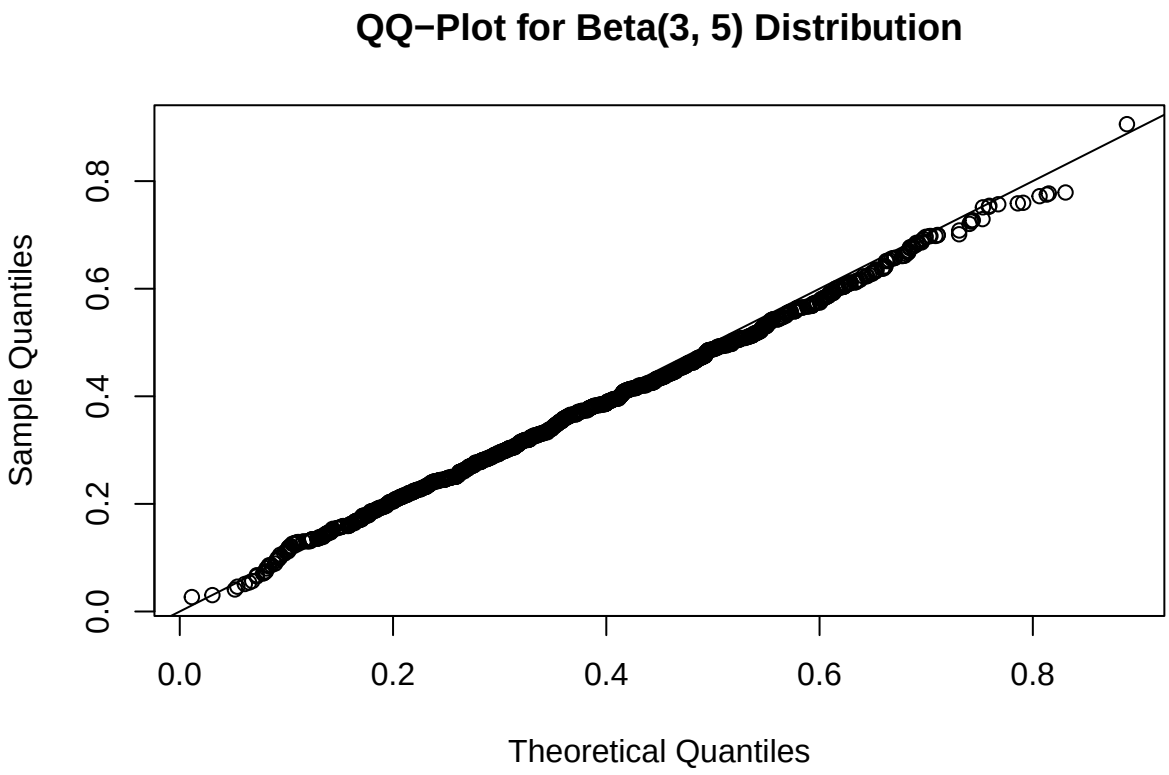
A natural candidate for a proposal distribution when sampling from the Beta distribution is the uniform distribution g(x)=1 on the interval [0,1] since it covers the support of the Beta distribution.



If the points in the QQ-plot closely follow the reference line, so the random number generator is accurately capturing the distribution of the Beta samples. From the plot, we can see that generated samples align quite well with the theoretical quantiles, suggesting that the acceptence-rejection method is working effectively.

```
# Generate samples from Beta(3, 5) using acceptance-rejection
beta_samples_3_5 <- beta_sample_ar(1000, 3, 5)
```

```r
# QQ-plot to compare generated samples with the true Beta(3, 5) distribution
true_samples_3_5 <- rbeta(1000, 3, 5)
qqplot(true_samples_3_5, beta_samples_3_5, main = "QQ-Plot for Beta(3, 5) Distribution",
       xlab = "Theoretical Quantiles", ylab = "Sample Quantiles")
abline(0, 1)
```



QQ-Plot Analysis

When comparing Beta-distributed samples to theoretical quantiles:

Close Fit to the Line: Points closely following the diagonal reference line indicate good alignment with the theoretical Beta distribution. This implies the generator works accurately.

Deviation: If the points straying far from the line:

-Tails: Deviations at the extremes might suggest underestimation or overestimation of the distribution tails.

-Center: Deviations in the middle suggest discrepancies in central values.

The QQ-plot shows a satisfactory fit for both Beta(2, 2) and Beta(3, 5) distributions.