Exercise 2

107.330 - Statistical Simulation and Computerintensive Methods, WS24

11912007 - Yahya Jabary

08.10.2024

Contents

```r
set.seed(11912007)
```

In this exercise, we dig into various aspects of random number generation and distribution sampling. We start off by exploring pseudo-random number generation using the Linear Congruential Random Number Generation algorithm, implementing and analyzing it with different parameters. Next, we focus on the exponential distribution, deriving a method to generate samples from it using uniform random variables. Finally, we tackle the Beta distribution, developing a function that employs an acceptance-rejection approach for sampling. We'll optimize this function for different parameter values and evaluate its effectiveness visually.

We also make sure that all functions implement the number of samples `n` or the parameters of the distribution as arguments where applicable as required by the assignment.

Exploring the LCG Algorithm

In this section we will:

- Summarize the concept of the LCG algorithm, implement and use it (Task 1)
- Try out and compare different values of `m`, `a` to illustrate the behaviour of the method as described on slide 18 (Task 1)

The LCG (Linear congruential Generator) algorithm is a simple and very well known PRNG (pseudo-random number generator). It works by recursively applying a modulo to some large integer m. The algorithm is defined by the recurrence relation $x_{n+1} = (a \cdot x_n + c) \mod m$.

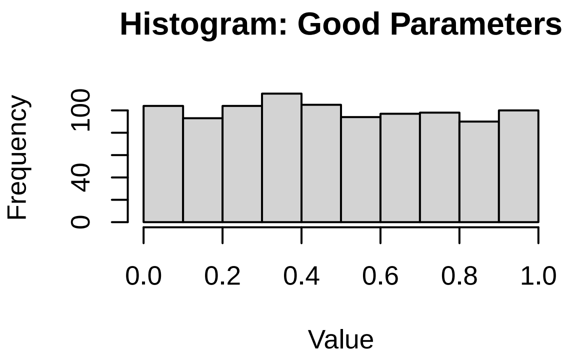
Where:

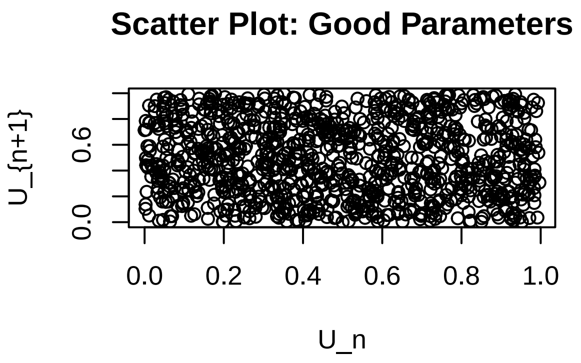
- m is the modulus (a large integer)
- a is the multiplier ($\approx \sqrt{m}$)
- c is the increment ($0 \leq c < m$)
- x_0 is the seed or starting value

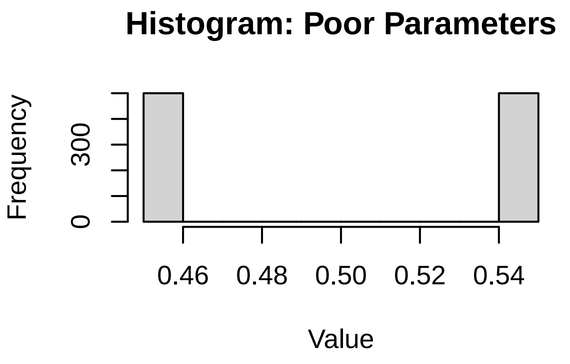
It generates a sequence of pseudo-random numbers u_n by normalizing x_n through $u_{n+1} = x_{n+1}/m$.

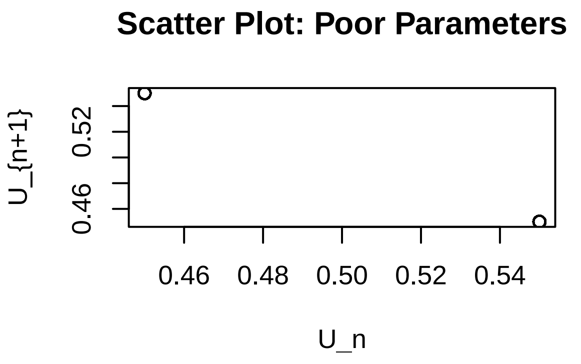
```r
lcg <- function(n, m, a, c = 0, x0) {
    us <- numeric(n)
    for (i in 1:n) {
        x0 <- (a * x0 + c) %% m
        us[i] <- x0 / m
    }
    return(us)
}

sample1 <- lcg(1000, m = 2^31 - 1, a = 16807, c = 0, x0 = 12345) # good params
sample2 <- lcg(1000, m = 100, a = 19, c = 0, x0 = 12345) # poor params
par(mfrow = c(2, 2))
hist(sample1, main = "Histogram: Good Parameters", xlab = "Value")
```

```r
plot(sample1[1:999], sample1[2:1000],
     main = "Scatter Plot: Good Parameters",
     xlab = "U_n", ylab = "U_{n+1}")
hist(sample2, main = "Histogram: Poor Parameters", xlab = "Value")
plot(sample2[1:999], sample2[2:1000],
     main = "Scatter Plot: Poor Parameters",
     xlab = "U_n", ylab = "U_{n+1}")
```









This code generates two samples: one with good parameters and one with poor parameters. The results are then visualized using histograms and scatter plots.

Good Parameters:

- $m = 2^31 - 1$ (a large prime number)
- $a = 16807$ (close to $\sqrt{m}$)
- The histogram shows a relatively uniform distribution.
- The scatter plot shows no obvious patterns, indicating good randomness.

Poor Parameters:

- $m = 100$ (a small, non-prime number)
- $a = 19$ (not close to $\sqrt{m}$)
- The histogram shows a highly non-uniform distribution with gaps.
- The scatter plot reveals clear patterns, indicating poor randomness.

We can observe that the choice of parameters significantly impacts the quality of the generated pseudo-random numbers.

- m: A large prime number for m generally produces better results, maximizing the cycle length of the sequence.
- a: The multiplier a should be carefully chosen, with values close to $\sqrt{m}$ often working well.
- c: With poor parameters, the sequence may repeat quickly, leading to non-random behavior.

Good parameters result in a more uniform distribution of values, while poor parameters can lead to obvious patterns / predictability, non-uniformity and shorter cycle lengths.

Sampling from the Exponential Distribution

Next we will:

- Obtain a random sample from the exponential distribution using uniform random variables (Task 2)
- Write down the mathematical expression and R implementation (Task 2)
- For three different values of λ, generate 1000 random samples and evaluate the quality of the random number generator using QQ-plots comparing against the real exponential distribution with the specified parameter λ (Task 2)

The exponential distribution has the cdf $F(x) = 1 - exp(-\lambda x), \lambda > 0$.

The inverse of its cdf is $F^{-1}(u) = -\frac{1}{\lambda}\ln(1-u)$ where u is a uniform random variable on $[0, 1]$.

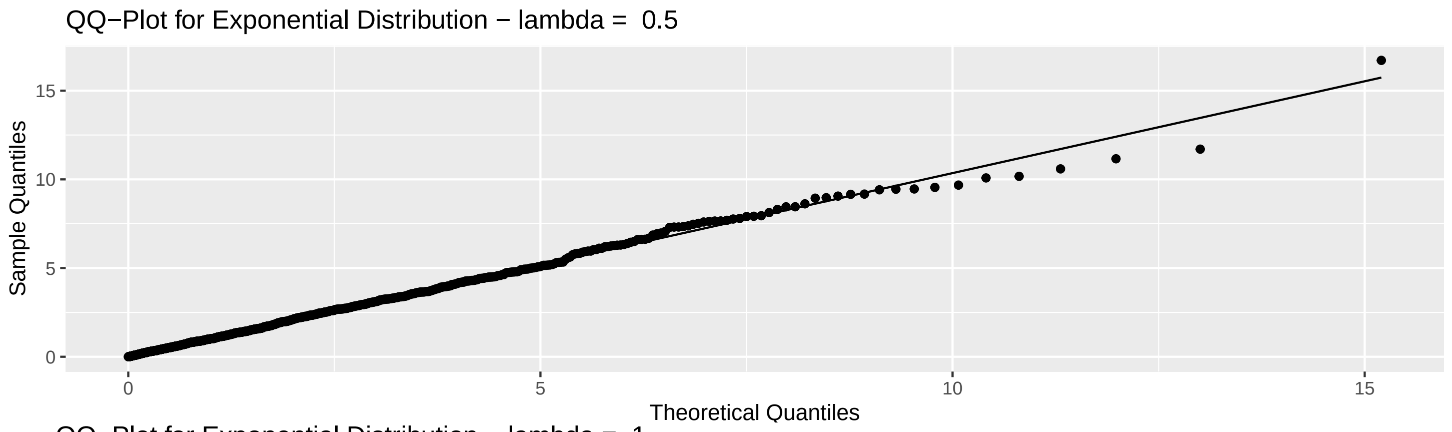
This function is preimplemted in R as `rexp(n, rate)`[1] where `rate` is the rate parameter λ and `n` is the number of samples.

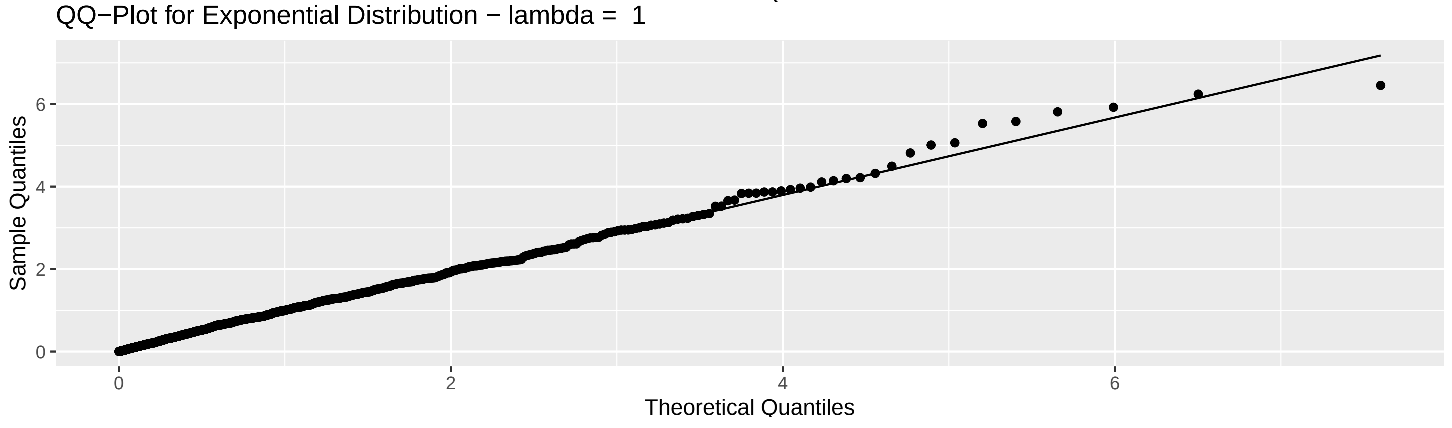
The following is our custom implementation of `rexp`. We generate three QQ-plots, one for each value of λ (0.5, 1, and 2). The plots compare the quantiles of our custom-generated exponential samples against the theoretical exponential distribution.

If the points in the QQ-plots closely follow the diagonal line, it indicates that our custom random number generator is producing samples that closely match the theoretical exponential distribution. Any significant deviations from the line would suggest discrepancies between our generated samples and the expected distribution.

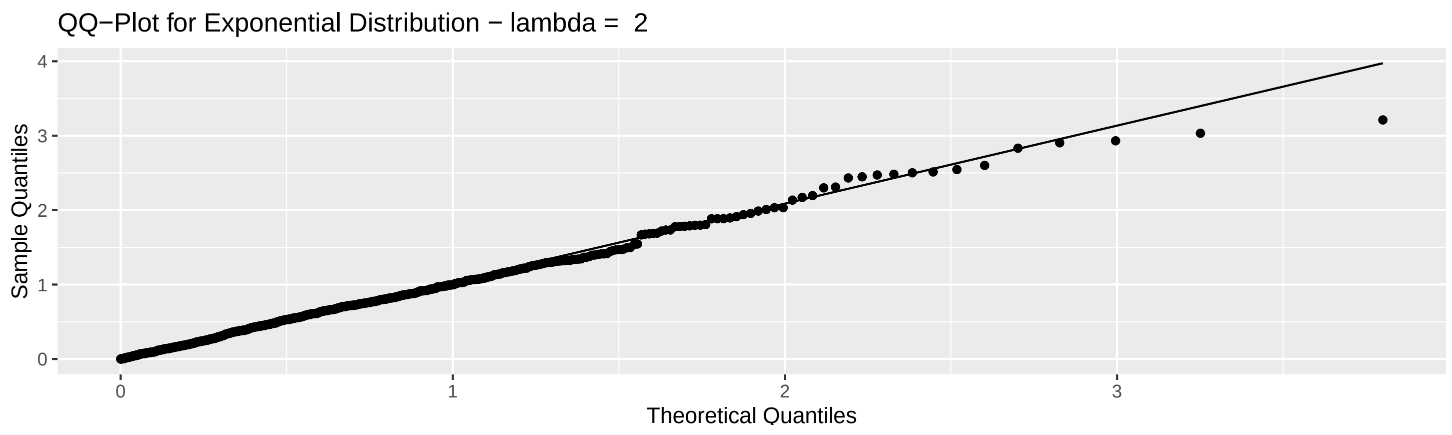
By examining these QQ-plots, we can visually assess the quality of our random number generator for different parameter values of the exponential distribution.

```r
rexp_custom <- function(n, lambda) {
    u <- runif(n) # uniform random variables
    x <- -1/lambda * log(1 - u)
    return(x)
}

generate_qqplot <- function(lambda) {
    df <- data.frame(custom = rexp_custom(1000, lambda), theoretical = rexp(1000, rate = lambda))
    ggplot(df, aes(sample = custom)) +
        stat_qq(distribution = qexp, dparams = list(rate = lambda)) +
        stat_qq_line(distribution = qexp, dparams = list(rate = lambda)) +
        ggtitle(paste("QQ-Plot for Exponential Distribution - lambda = ", lambda)) +
        xlab("Theoretical Quantiles") +
        ylab("Sample Quantiles")
}

lambda_values <- c(0.5, 1, 2)
plots <- lapply(lambda_values, generate_qqplot)
for (plot in plots) {
    print(plot)
}
```





[1]See: https://stat.ethz.ch/R-manual/R-devel/library/stats/html/Exponential.html



It is evident from the QQ-plots that our custom random number generator is producing samples that closely match the theoretical exponential distribution for all three values of λ we tested.

Acceptance-rejection sampling for the Beta Distribution

In this section we will:

- Write a function which uses an acceptance-rejection approach to sample from a beta distribution. (Task 3)
- Argue what a natural candidate for a proposal distribution could be. (Task 3)
- Implement and find the optimal parameters for the acceptance-rejection method for different values of α and β. (Task 3)

The Beta distribution has the pdf $f(x;\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}x^{\alpha-1}(1-x)^{\beta-1}$.

To implement an acceptance-rejection method for sampling from a Beta distribution, let's break this down into more digestible steps by starting off with the special case where $\alpha = \beta = 2$. In this case the Beta distribution has the probability density function $f(x;2,2) = \frac{\Gamma(2+2)}{\Gamma(2)\Gamma(2)}x^{2-1}(1-x)^{2-1} = 6x(1-x)$.

A natural candidate for the proposal distribution is the `Uniform(0,1)` distribution, as the Beta distribution is defined on the interval $[0,1]$. The uniform distribution has a constant density of 1 over this interval.

To find a good constant c, we need to ensure that $\forall x \in [0,1] : c \cdot g(x) \geq f(x)$, where $g(x)$ is the uniform density. The maximum of $f(x)$ occurs at $x = 0.5$, where $f(0.5) = 1.5$. Therefore, we can use $c = 1.5$.

So far so good. Now let's implement the acceptance-rejection method for the special case of $Beta(\alpha = 2, \beta = 2)$‘ and then generalize it for arbitrary Beta distributions with $\alpha > 1$ and $\beta > 1$.

Here's a function to sample using acceptance-rejection:

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

    while (accepted < n) {
        y <- runif(1)  # proposal from Uniform(0,1)
        u <- runif(1)

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

    return(samples)
}
```

For an arbitrary Beta distribution with $\alpha > 1$ and $\beta > 1$, we can use a similar approach, but we need to adapt the constant c based on the parameters. The maximum of the Beta density occurs at $x = (\alpha - 1)/(\alpha + \beta - 2)$, and we can calculate the value of c accordingly.

Here's a function to sample using acceptance-rejection from an arbitrary $Beta(\alpha, \beta)$ distribution:

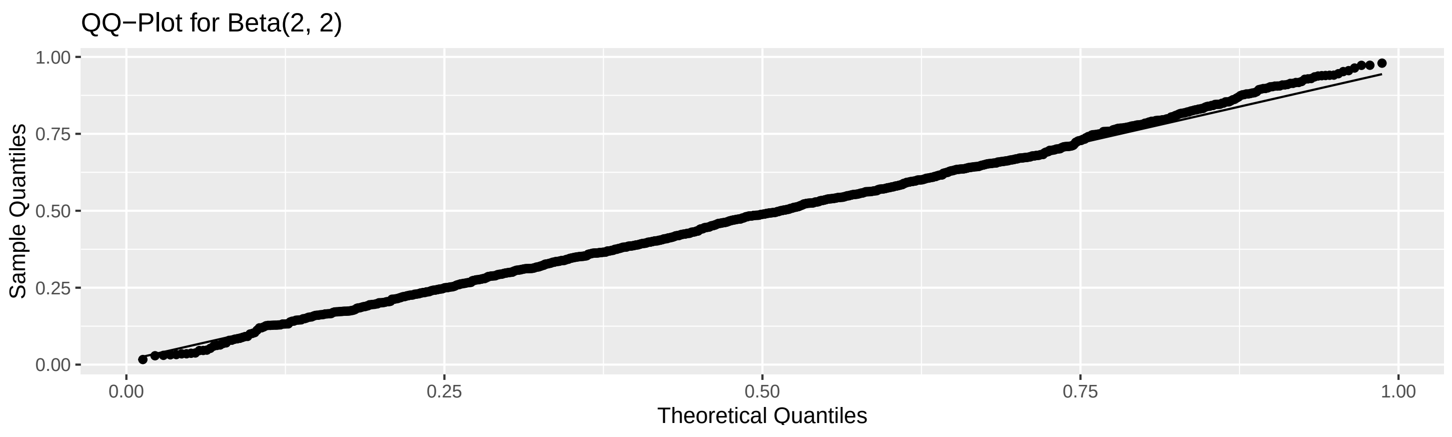
```r
sample_beta <- function(n, alpha, beta) {
    if (alpha <= 1 || beta <= 1) {
        stop("invalid params")
    }

    samples <- numeric(n)
    accepted <- 0

    mode <- (alpha - 1) / (alpha + beta - 2)
    c <- dbeta(mode, alpha, beta)

    while (accepted < n) {
        y <- runif(1)  # proposal from Uniform(0,1)
        u <- runif(1)

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

    return(samples)
}
```

This function adapts the constant c to the user-specified parameters α and β of the Beta distribution. It calculates the mode of the Beta distribution and uses the density at that point as the value for c.

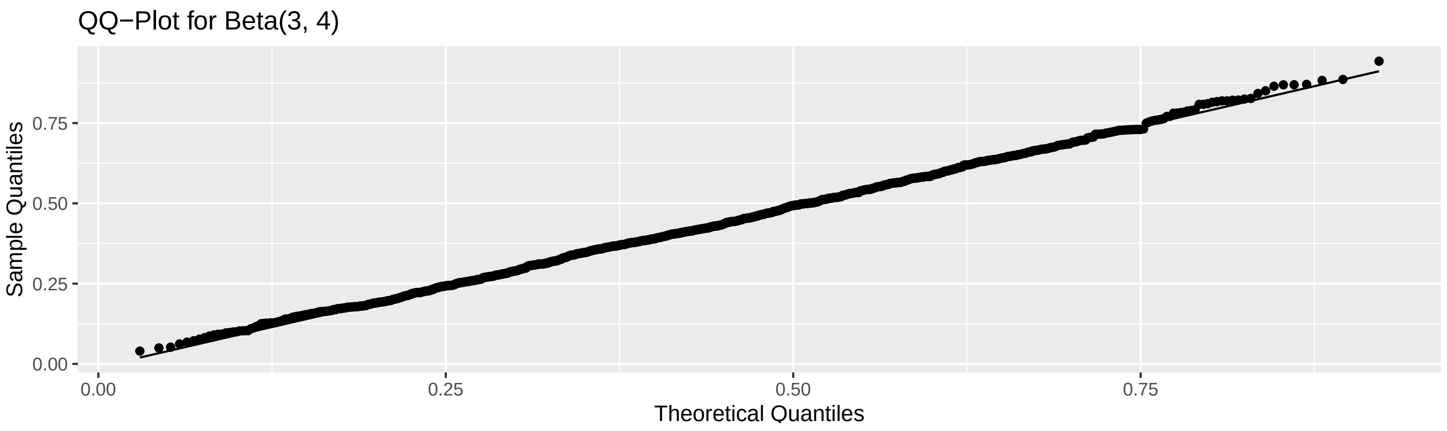
The $Uniform(0, 1)$ distribution remains a natural candidate for the proposal distribution for any $Beta(\alpha, \beta)$ with $\alpha > 1$ and $\beta > 1$, as the Beta distribution is always defined on $[0, 1]$.

To use these functions:

```
samples_2_2 <-  sample_beta_2_2(1000)
samples_3_4 <- sample_beta(1000, 3, 4)

beta_2_2 <- rbeta(1000, 2, 2)
beta_3_4 <- rbeta(1000, 3, 4)

ggplot(data.frame(samples = samples_2_2, theoretical = beta_2_2), aes(sample = samples)) +
    stat_qq(distribution = qbeta, dparams = list(shape1 = 2, shape2 = 2)) +
    stat_qq_line(distribution = qbeta, dparams = list(shape1 = 2, shape2 = 2)) +
    ggtitle("QQ-Plot for Beta(2, 2)") +
    xlab("Theoretical Quantiles") +
    ylab("Sample Quantiles")
```



```
ggplot(data.frame(samples = samples_3_4, theoretical = beta_3_4), aes(sample = samples)) +
    stat_qq(distribution = qbeta, dparams = list(shape1 = 3, shape2 = 4)) +
    stat_qq_line(distribution = qbeta, dparams = list(shape1 = 3, shape2 = 4)) +
    ggtitle("QQ-Plot for Beta(3, 4)") +
    xlab("Theoretical Quantiles") +
    ylab("Sample Quantiles")
```



These functions provide a flexible way to sample from Beta distributions using the acceptance-rejection method. The efficiency of the method depends on how close the proposal distribution is to the target distribution. For Beta distributions with parameters far from 1, the rejection rate may become high, and other methods (like inverse transform sampling) might be more efficient.

This solution was inspired by the following resources:

- https://wiki.math.uwaterloo.ca/statwiki/index.php?title=acceptance-Rejection_Sampling
- https://stackoverflow.com/questions/47268956/acceptance-rejection-for-beta-distribution-r-code
- http://www.columbia.edu/~ks20/4703-Sigman/4703-07-Notes-ARM.pdf
- https://www.jstor.org/stable/2287667