Exercise 2

Random Number Generation 2

Olesia Galynskaia 12321492

2025

Fix the random seed for reproducibility

```
set.seed(12321492)
knitr::opts_chunk$set(echo = TRUE, message = FALSE, warning = FALSE, fig.width = 6, fig.height = 4, dpi
```

Task 3 — Exponential via Inverse Transform

Inverse-transform sampler

This function generates random numbers from an exponential distribution using the inverse transform method.
Formula: X = -log(1 - U) / lambda, where U - Uniform(0, 1).
Using (1 - U) avoids taking log(0) when U is very close to 1.

```
rexpo_inv <- function(n, lambda) {
  stopifnot(is.numeric(n), length(n) == 1, n >= 1, is.finite(n))
  stopifnot(is.numeric(lambda), length(lambda) == 1, lambda > 0, is.finite(lambda))
  u <- runif(n)                          # generate Uniform(0,1) random numbers
  x <- -log1p(-u) / lambda               # numerically stable form of -log(1 - u) / lambda
  return(x)
}
```

QQ-plot helper

This function draws a QQ-plot comparing the generated data with the theoretical exponential distribution.
Points should lie close to the diagonal line if generation is correct.

```
qq_exp <- function(x, lambda, main = NULL, pt_cex = 0.7) {
  n <- length(x)
  probs <- ppoints(n)
  q_theory <- qexp(probs, rate = lambda)
  q_sample <- sort(x)

  old_par <- par(mar = c(4.8, 4.8, 3.2, 1.2))
  on.exit(par(old_par), add = TRUE)
```

```r
  qqplot(q_theory, q_sample,
         xlab = "Theoretical quantiles (Exponential)",
         ylab = "Sample quantiles",
         main = if (is.null(main)) paste0("QQ-plot, lambda = ", lambda) else main,
         pch = 19, cex = pt_cex)
  abline(0, 1, lwd = 2, lty = 2)
}
```

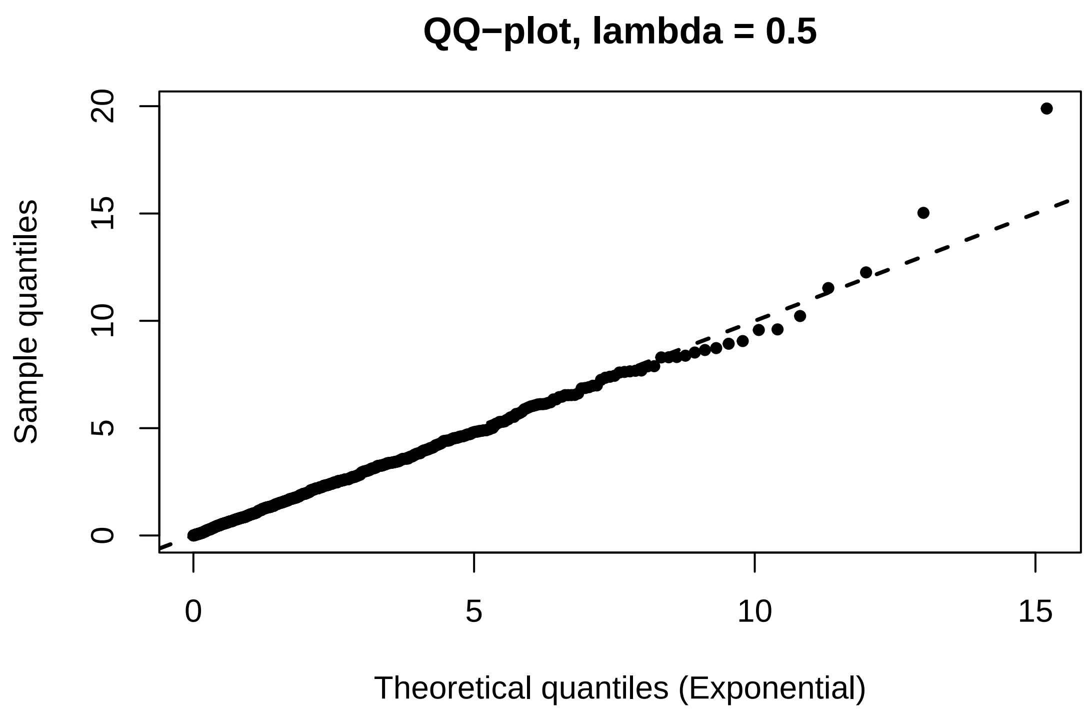
Generate samples for three values of lambda and assess with QQ-plots

We chose three different rate parameters — 0.5, 1.0, and 2.0 — to illustrate how the exponential distribution changes with its rate. A smaller lambda (0.5) produces a wider distribution with a heavier right tail, meaning large values occur more often. A larger lambda (2.0) makes the distribution steeper and more concentrated near zero. This range covers low, medium, and high rates and clearly shows how scaling affects the generated random samples.

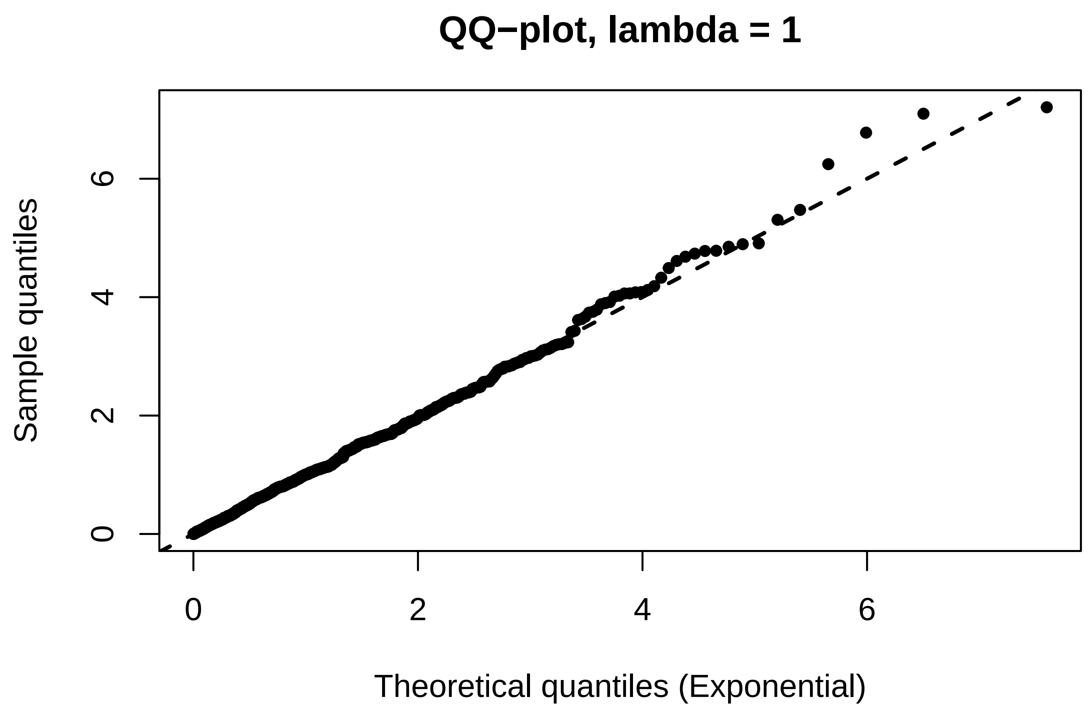
```r
lambdas <- c(0.5, 1.0, 2.0)              # three rate parameters
n_per_lambda <- 1000                     # 1000 samples for each

# Generate the samples
samples <- lapply(lambdas, function(lmb) rexpo_inv(n_per_lambda, lmb))

# Compare sample means to theoretical means
means_df <- data.frame(
  lambda = lambdas,
  theoretical_mean = 1 / lambdas,
  sample_mean = sapply(samples, mean)
)
print(means_df, row.names = FALSE)
```

```
##  lambda theoretical_mean sample_mean
##     0.5              2.0   1.9530065
##     1.0              1.0   1.0023816
##     2.0              0.5   0.5271522
```

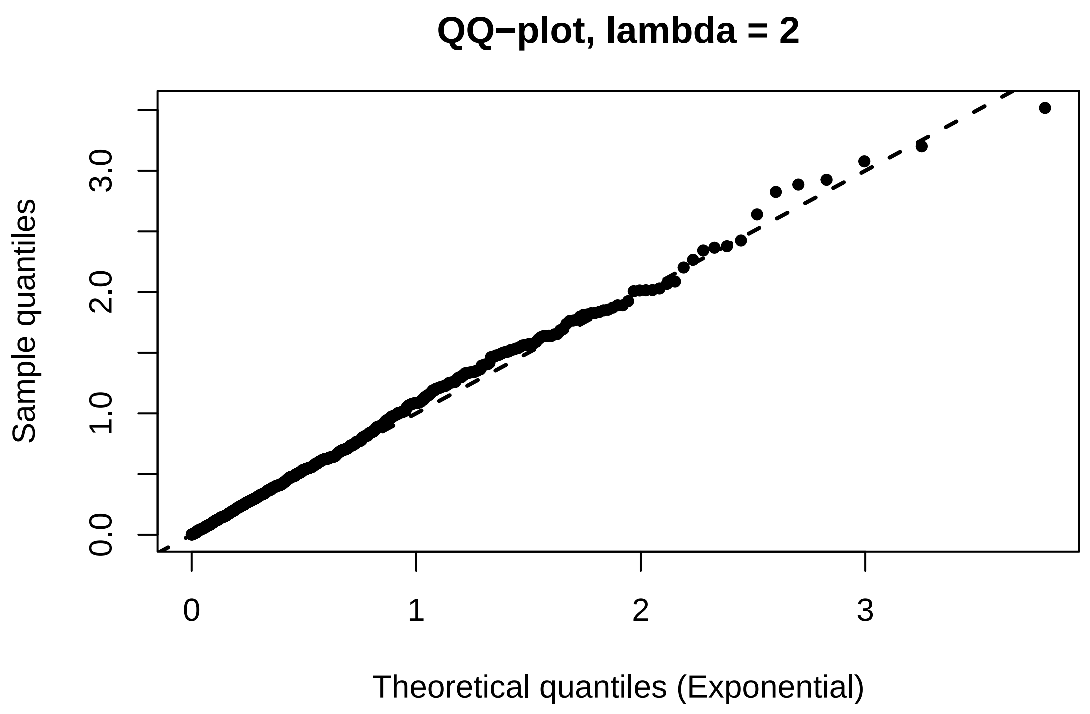
```r
qq_exp(samples[[1]], lambdas[1])
```



```r
qq_exp(samples[[2]], lambdas[2])
```



```
qq_exp(samples[[3]], lambdas[3])
```



Comment

The QQ-plots for lambda = 0.5, 1.0, and 2.0 follow the diagonal line closely, confirming that the inverse-transform sampler correctly generates exponential random variables. At the extreme upper tail, the points rise slightly above the line, which indicates that a few large observations occurred — a normal random fluctuation for exponential data with heavy tails. The deviation is more noticeable for lambda = 0.5 because this distribution is the most spread out. Overall, the results confirm that the sampling procedure works properly, and the observed differences are within the range expected for 1000 random samples.

Task 4 — Beta distribution

Beta distribution: pdf, cdf, support, parameters

PDF: f(x; a, b) = x^(a - 1) * (1 - x)^(b - 1) / B(a, b), for 0 < x < 1, a > 0, b > 0.
CDF: F(x; a, b) = integral_0^x f(t; a, b) dt. In practice we evaluate it via pbeta in R.
Support: values lie strictly between 0 and 1 (with mass approaching the ends if a < 1 or b < 1).
Parameters: a > 0 and b > 0 (shape parameters). Larger a shifts mass toward 1; larger b shifts mass toward 0.

We explore two ways to generate Beta samples — using the Gamma relationship and using the acceptance–rejection method — and then compare their performance. ## 1) Sampling from Beta via Gamma relationship

If X ~ Gamma(shape = a, rate = 1) and Y ~ Gamma(shape = b, rate = 1), then Z = X / (X + Y) ~ Beta(a, b).

This uses the well-known connection between Beta and Gamma distributions.

```r
rbeta_from_gamma <- function(n, a, b) {
  stopifnot(n >= 1, a > 0, b > 0)
  x <- rgamma(n, shape = a, rate = 1)
  y <- rgamma(n, shape = b, rate = 1)
  z <- x / (x + y)
  return(z)
}
```

2) Acceptance–rejection samplers using a Uniform proposal on [0, 1]

Natural proposal: Uniform(0, 1). It matches the Beta support and is easy to sample.
We need a constant M such that f(x) <= M * g(x) for all x, where g is Uniform(0, 1) with density 1.
For a, b > 1 the Beta pdf is maximized at the mode m = (a - 1) / (a + b - 2), so M = f(m) = dbeta(m, a, b). This gives a tight bound and a decent acceptance rate (about 1 / M).

2.1) Special case a = b = 2

```r
rbeta_ar_22 <- function(n) {
  a <- 2; b <- 2
  M <- dbeta(0.5, a, b)   # = 1.5 exactly for Beta(2, 2)
  out <- numeric(0)
  while (length(out) < n) {
    y <- runif(n)          # proposal draws
    u <- runif(n)
    keep <- u <= dbeta(y, a, b) / M
    out <- c(out, y[keep])
  }
  out[seq_len(n)]
}
```

2.2) General case a, b > 1 with data-dependent M

```r
rbeta_ar_ab <- function(n, a, b) {
  stopifnot(n >= 1, a > 1, b > 1)
  m <- (a - 1) / (a + b - 2)      # mode
  M <- dbeta(m, a, b)             # supremum of the pdf on (0,1)
  out <- numeric(0)
  while (length(out) < n) {
    y <- runif(n)                 # proposal from Uniform(0,1)
    u <- runif(n)
    keep <- u <= dbeta(y, a, b) / M
    out <- c(out, y[keep])
  }
  out[seq_len(n)]
}
```

Plot helpers

```r
plot_beta_hist <- function(x, a, b, main = NULL) {
  old <- par(mar = c(4.6, 4.6, 3.0, 1.0)); on.exit(par(old), add = TRUE)
  hist(x, breaks = 40, freq = FALSE,
       xlab = "x", ylab = "Density",
       main = if (is.null(main)) paste0("Histogram with Beta(", a, ",", b, ") density") else main)
  curve(dbeta(x, a, b), add = TRUE, lwd = 2, lty = 2)
}

qq_beta <- function(x, a, b, main = NULL, pt_cex = 0.65) {
  n <- length(x)
  probs <- ppoints(n)
  q_theory <- qbeta(probs, shape1 = a, shape2 = b)
  q_sample <- sort(x)
  old <- par(mar = c(4.8, 4.8, 3.0, 1.0)); on.exit(par(old), add = TRUE)
  qqplot(q_theory, q_sample,
         xlab = "Theoretical quantiles (Beta)",
         ylab = "Sample quantiles",
         main = if (is.null(main)) paste0("QQ-plot, Beta(", a, ",", b, ")") else main,
         pch = 19, cex = pt_cex)
  abline(0, 1, lwd = 2, lty = 2)
}
```

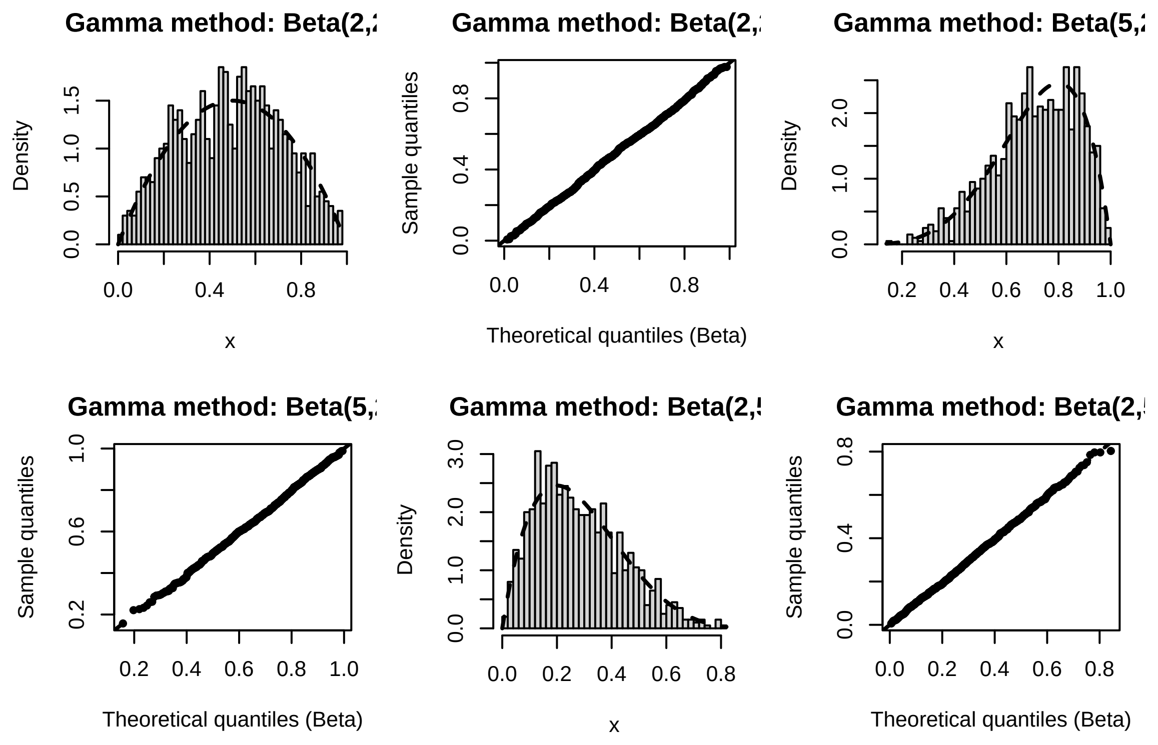
Generate samples and evaluate visually

```r
set.seed(12321492)

n <- 1000
param_sets <- list(
  c(2, 2),    # symmetric, bell-shaped on (0,1)
  c(5, 2),    # skewed toward 1
  c(2, 5)     # skewed toward 0
)

# Draw with both methods
samp_gamma <- lapply(param_sets, function(p) rbeta_from_gamma(n, p[1], p[2]))
samp_ar    <- list(
  rbeta_ar_22(n),
  rbeta_ar_ab(n, 5, 2),
  rbeta_ar_ab(n, 2, 5)
)

# Quick numeric check: sample means vs theory a/(a+b)
check_df <- data.frame(
  a = sapply(param_sets, `[[`, 1),
  b = sapply(param_sets, `[[`, 2),
  theoretical_mean = sapply(param_sets, function(p) p[1] / (p[1] + p[2])),
  mean_gamma = sapply(samp_gamma, mean),
  mean_ar    = sapply(samp_ar, mean)
)
print(check_df, row.names = FALSE)
```

```
##  a b theoretical_mean mean_gamma   mean_ar
##   2 2        0.5000000  0.4935022 0.4848278
##   5 2        0.7142857  0.7091388 0.7133984
##   2 5        0.2857143  0.2813499 0.2809194
```

Visuals for the Gamma-based sampler

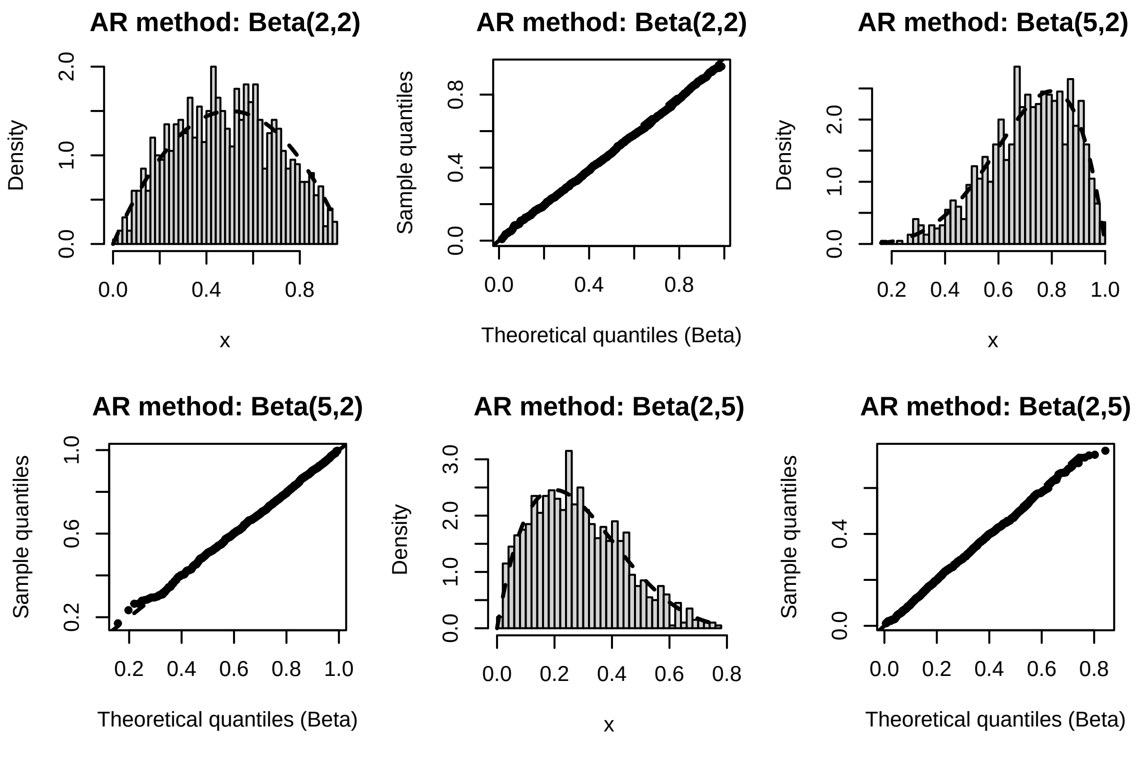
```r
op <- par(mfrow = c(2, 3))
for (i in seq_along(param_sets)) {
a <- param_sets[[i]][1]; b <- param_sets[[i]][2]
plot_beta_hist(samp_gamma[[i]], a, b,
main = paste0("Gamma method: Beta(", a, ",", b, ")"))
qq_beta(samp_gamma[[i]], a, b,
main = paste0("Gamma method: Beta(", a, ",", b, ")"))
}
```



```r
par(op)
```

Visuals for the acceptance–rejection sampler

```r
op <- par(mfrow = c(2, 3))
for (i in seq_along(param_sets)) {
a <- param_sets[[i]][1]; b <- param_sets[[i]][2]
plot_beta_hist(samp_ar[[i]], a, b,
main = paste0("AR method: Beta(", a, ",", b, ")"))
qq_beta(samp_ar[[i]], a, b,
main = paste0("AR method: Beta(", a, ",", b, ")"))
}
```



```r
par(op)
```

Comment

1. Sampling from Gamma distributions

We choose this method because Gamma sampling is easy in R and numerically stable.
The histogram and QQ-plots confirm that this method reproduces the theoretical Beta shape accurately.
The sample means for (a, b) = (2, 2), (5, 2), (2, 5) are very close to their theoretical means 0.5, 0.714, 0.286.

2. Acceptance–rejection approach

1) Case a = b = 2

We use the Uniform(0, 1) distribution as the proposal because its support matches the Beta distribution.
The constant M must satisfy f(x) <= M · g(x) for all x, where g(x)=1 for Uniform(0,1).
For Beta(2, 2), the mode is x = 0.5 and M = f(0.5) = 1.5, which gives an acceptance rate of about 1 / 1.5 = 0.67.
The histograms and QQ-plots show that accepted samples match the target distribution very well.

2) General case a, b > 1

We adapt M to any (a, b) > 1 by computing it at the mode m = (a - 1)/(a + b - 2). This keeps the rejection rate reasonable for different shapes.
Again, the graphs show that the accepted samples follow the theoretical curves closely.
The sample means are almost the same as in the Gamma method.

For both methods, the histograms line up with the theoretical Beta densities, and the QQ-plots fall nearly on the diagonal line.

Minor deviations near 0 or 1 are expected because the Beta density rises sharply at the boundaries, and finite samples add random variation.
Both algorithms thus produce good Beta random numbers.

Both implementations successfully generate Beta-distributed samples.
The Gamma-based method is simpler and faster since it relies on built-in rgamma.
The acceptance–rejection method is more general, illustrating how to construct a sampler from a known proposal and an adjustable constant M.
In our results, both methods achieve accurate means and almost identical shapes.
Hence, for practical use, the Gamma approach is preferred for efficiency,
while the AR version is a good conceptual example of the acceptance–rejection principle.