HW 5 Random Number Generators

STAT 5400

Due: Oct 4, 2024 9:30 AM

Problems

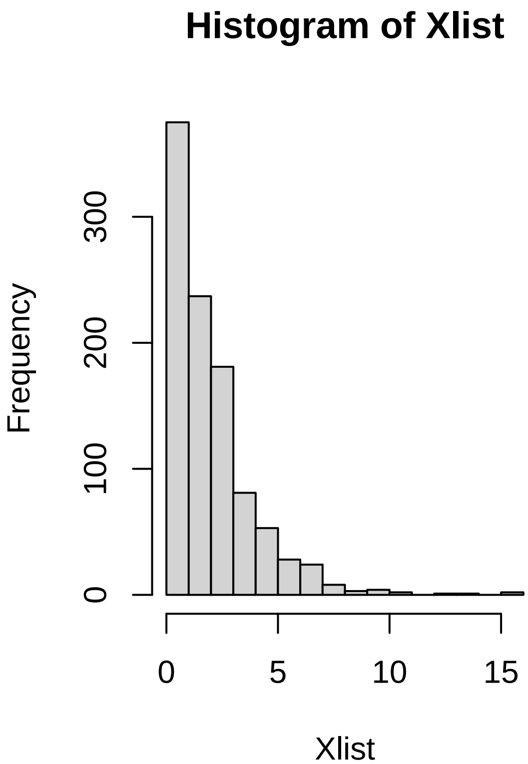
Submit your solutions as an .Rmd file and accompanying .pdf file. Include all the **relevant** R code and output. Always interpret your result whenever it is necessary.

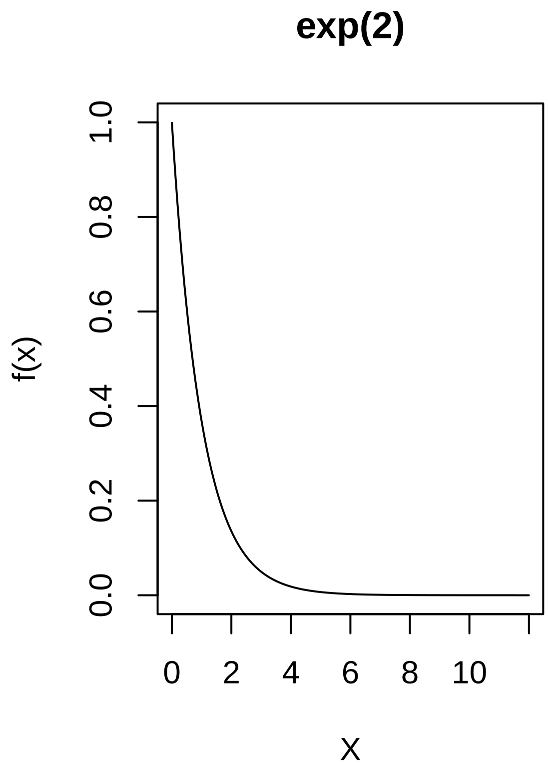
Problems

1. Generators of exponential distributions

- Fill in the code on Slide 36 of S3P1.pdf. You may either use the pdf of exp(2) manually or call `dexp` in R.

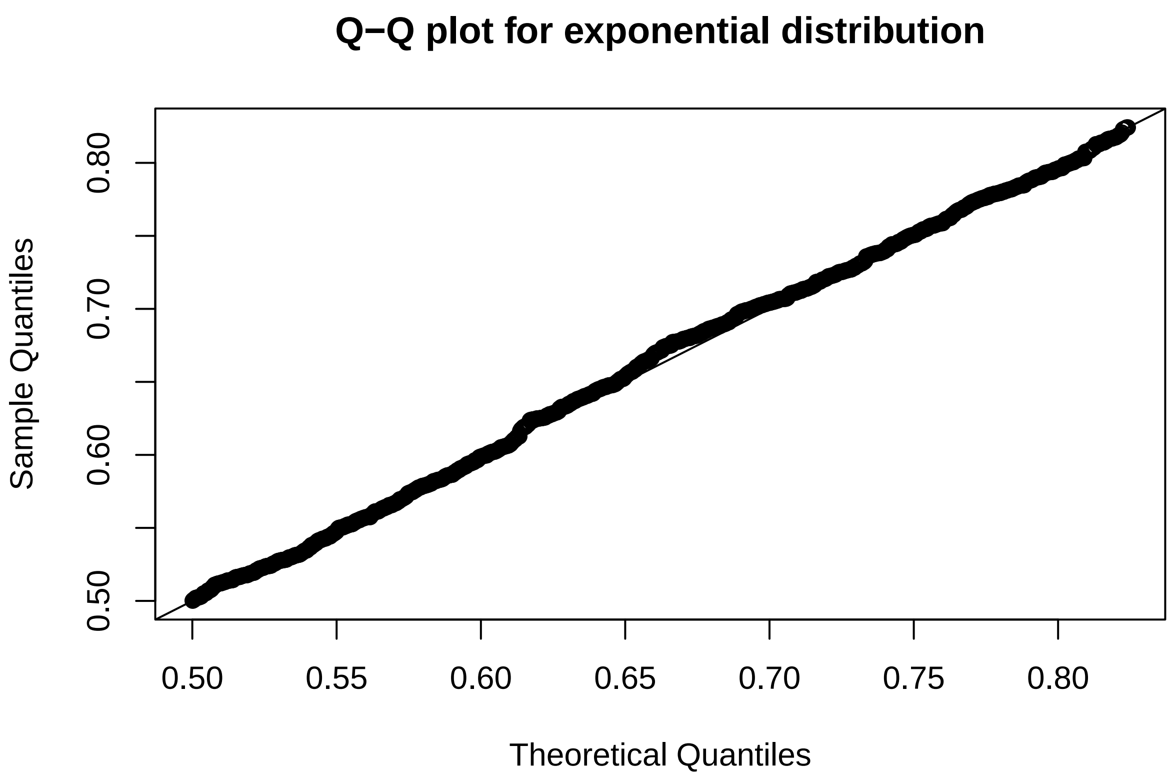
```
set.seed(5400)

Ulist <- runif(1000, 0, 1)
Xlist <- -2 * log(Ulist)


par(mfrow = c(1,2))
hist(Xlist)
plot(seq(0.001, 12, len=200), dexp(seq(0.001, 12, len=200), rate = 1, log = FALSE), type="l", ylim = c(
```





- Fill in the code on Slide 37 of S3P1.pdf.
- (Negative exponential distribution) Suppose there is a distribution with pdf $\frac{1}{2}exp\{\frac{1}{2}(x)\}$, $x < 0$. Generate a random sample with 1000 observations from such distribution. Have a Q-Q plot to check whether the sample comforts with this distribution.

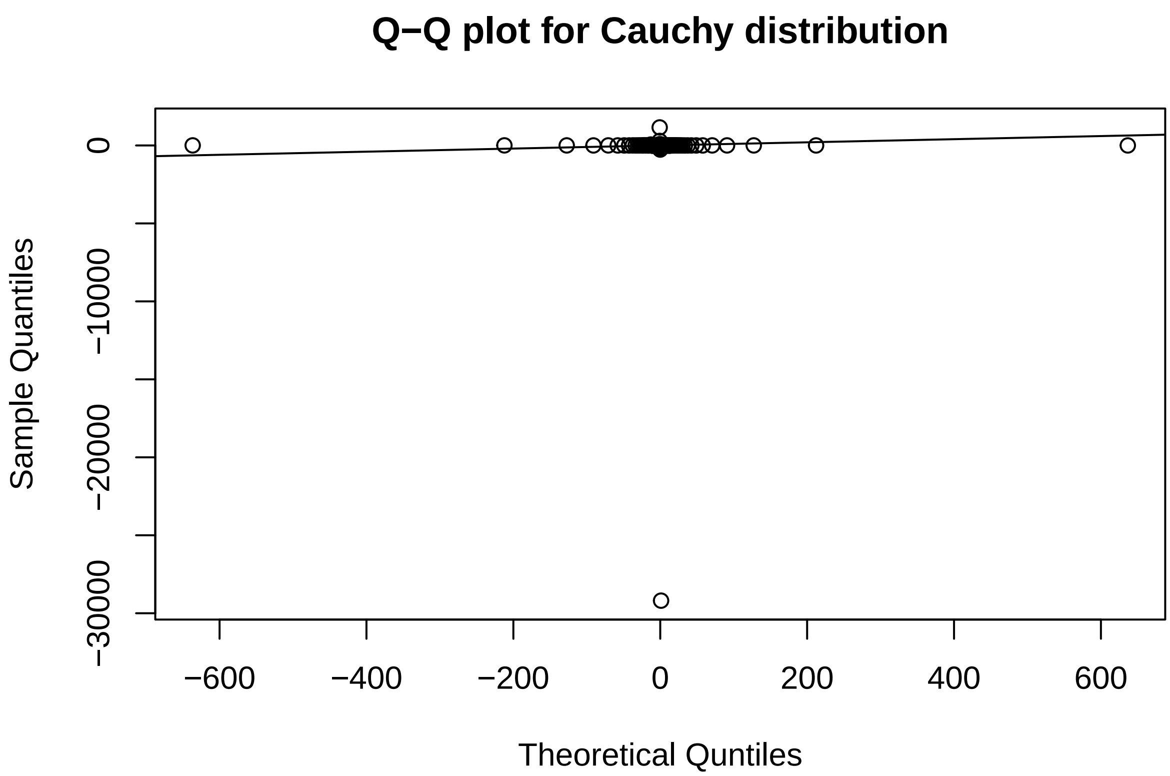
```
ulist <- runif(1000, 0, 1)
xlist <- 0.5 * exp(0.5 * ulist)

theoretical_quantiles <- 0.5 * exp(0.5 * qunif(ppoints(1000)))

plot(theoretical_quantiles, sort(xlist),
xlab="Theoretical Quantiles", ylab="Sample Quantiles",
main="Q-Q plot for exponential distribution")
abline(0, 1)
```



2. Generators of Cauchy distributions.

- Implement an algorithm to generate the standard Cauchy distribution using the inverse CDF.
- Check if the generated numbers conform to the standard Cauchy distribution. Basically, you may check this by filling in the code on Slide 42 of S3P1.pdf.

```
u <- runif(1000)

cauchy_samples <- tan(pi * (u - 0.5))

probs = ppoints(1000)
plot(qcauchy(ppoints(1000)), cauchy_samples,
xlab="Theoretical Quntiles",
ylab="Sample Quantiles",
main="Q-Q plot for Cauchy distribution")
abline(0, 1)
```




```
dev.off()
```

```
## null device
##           1
```

3. Generators of Beta distributions and t distributions.

- Generate a Beta(α, α) distribution using

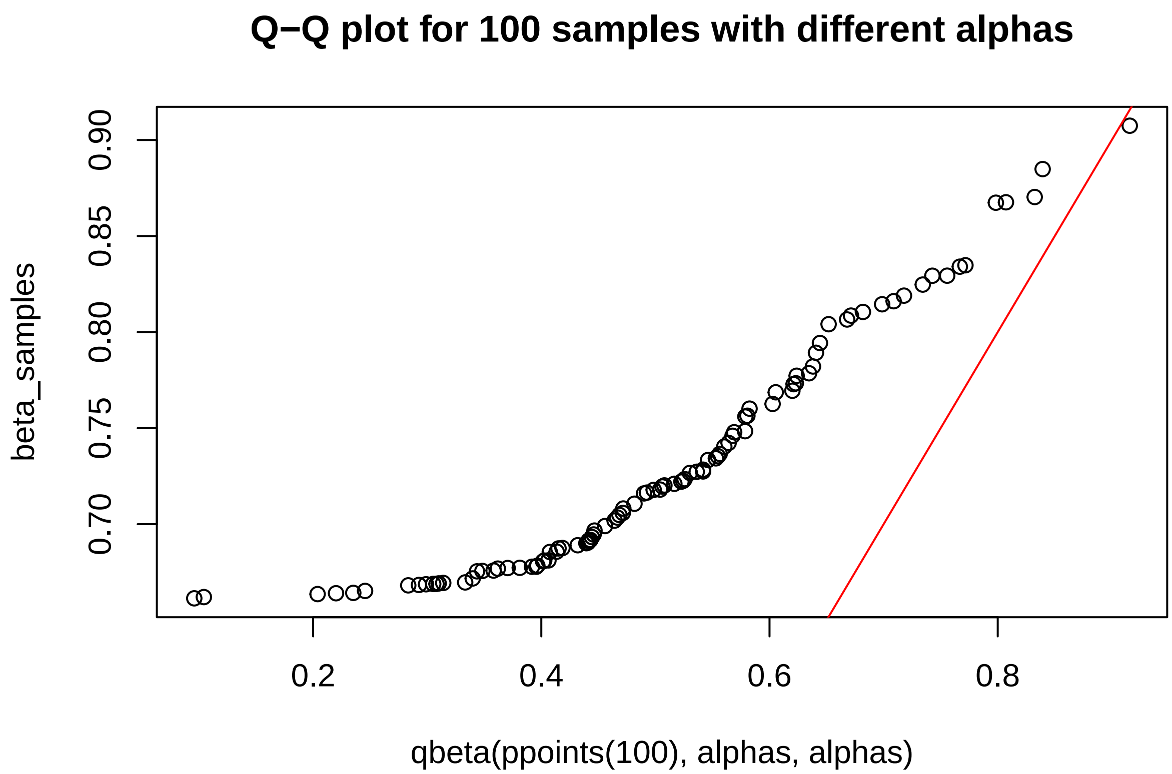
$$X = \frac{1}{2} + \frac{\mathbb{I}_{U_3 \leqslant 1/2}}{2\sqrt{1 + \frac{1}{\left(U_1^{-1/\alpha} - 1\right)\cos^2(2\pi U_2)}}}.$$

  where U_1, U_2, and U_3 are independently generated random vaiables on $(0, 1)$, and the indicator function $\mathbb{I}_{U_3 \leqslant 1/2}$ takes the value of 1 if the condition is true or the value of -1 otherwise.

```
beta <- function(a,b){
  ind_u3 = ifelse(runif(1)> 0.5,1,-1)

  return (0.5 + ind_u3/(2* sqrt(1+1/(((runif(1)^(-1/a))-1)* cos(2 * pi * runif(1))^2) )))
}
```

- With several values of α, generate a sample of 100 observations from Beta(α, α). Have a Q-Q plot to check whether the sample comforts with the beta distribution. You may use the built-in function in R to give the inverse CDF function.

```r
set.seed(123)
alphas <- runif(100, 1, 10)  # Randomly generate 100 different alpha values between 1 and 10
beta_samples <- beta(alphas, alphas)

qqplot(qbeta(ppoints(100), alphas, alphas), beta_samples, main="Q-Q plot for 100 samples with different
abline(0, 1, col="red")
```



+ Using the above method to generate $Y \sim \text{Beta}(n, n)$. Generate a t-distribution with degree of freedom $2n$ using

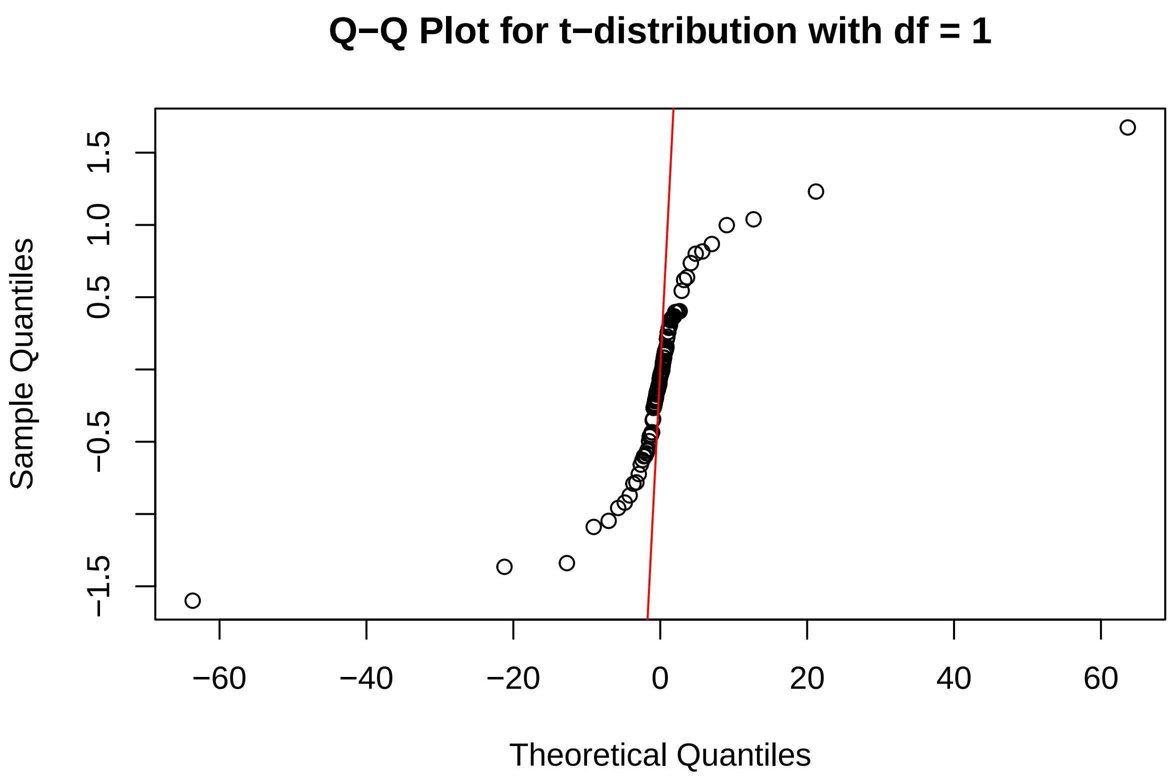
$$Z = \frac{\sqrt{n}(Y - 1/2)}{2\sqrt{Y(1-Y)}}.$$

```r
generate_t_distribution <- function(n, sample_size = 100) {
  beta_samples <- replicate(sample_size, beta(n))  # Generate Beta(n, n) samples

  # Apply the given formula to generate t-distributed samples
  t_samples <- sqrt(n) * (beta_samples - 0.5) / (2 * sqrt(beta_samples * (1 - beta_samples)))

  return(t_samples)
}
```

- Generate a sample of 100 observations from $t(1)$, namely Cauchy distribution. Have a Q-Q plot to check whether the sample comforts with the $t(1)$ distribution. Compare the Q-Q plot with the one you plotted in the previous question.

```r
t_samples <- generate_t_distribution(1)

qqplot(qt(ppoints(100), df=1), t_samples,
```

```r
       main=paste("Q-Q Plot for t-distribution with df =", 1),
       xlab="Theoretical Quantiles", ylab="Sample Quantiles")
abline(0, 1, col="red")  # Add the y=x reference line
```



4. More on accept-reject sampling.

- Generate n values from the truncated normal distribution:

$$Y_i \sim (X|a < X < b), \text{ where } X \sim N(\mu, \sigma^2).$$

  Hint: generate observations from normal distribution using 'rnorm"' and discard it if it falls outside the interval.
  This function should have five arguments:

  - `n`: number of observations,
  - `mu`: the value of mean μ,
  - `sigma`: the value of standard deviation σ,
  - `a`: the left endpoint of the interval of `-Inf`,
  - `b`: the right endpoint of the interval of `Inf`, This function should return a vector of n truncated normal variables.

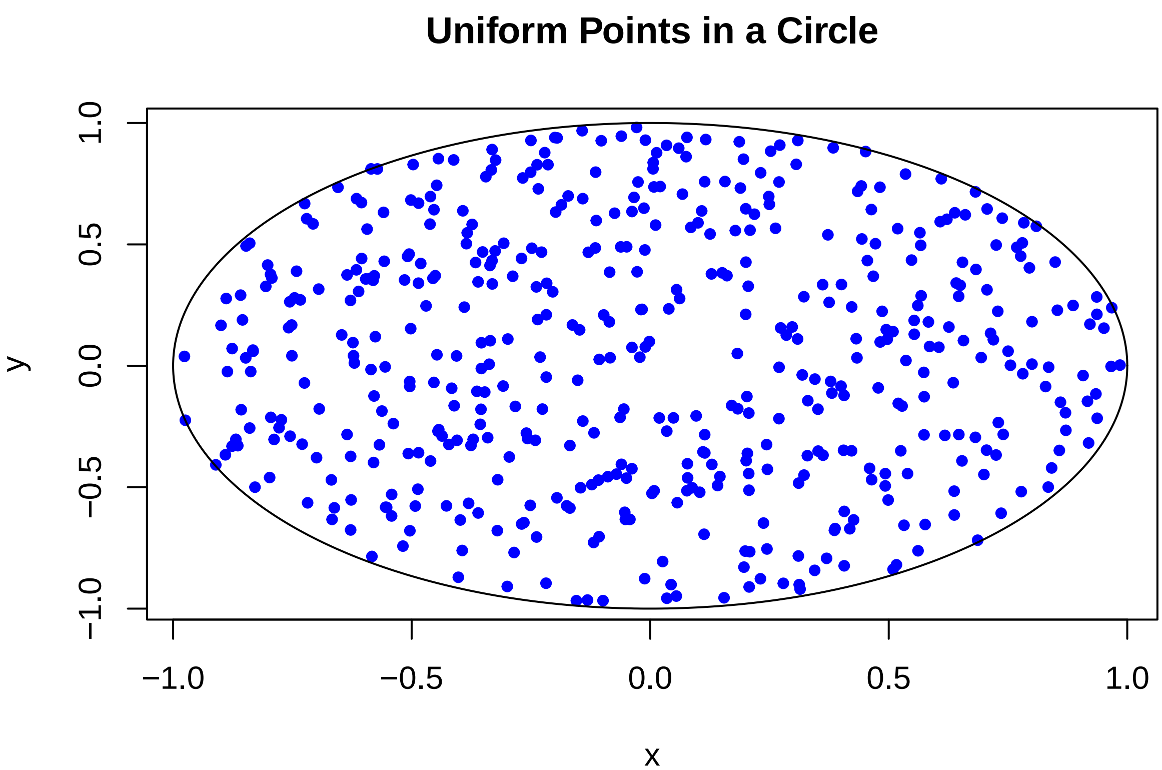
```r
Y <- function(n, mu, sigma, a, b) {
  i = 1
  while(i <=n) {
    val <- rnorm(1, mu, sqrt(sigma))
    vec <- c()
    if(val < a || val > b) {
      continue;
```

```
    } else {
      i = i + 1
      vec.append(val)
    }
  }
  return(vec)
}
```

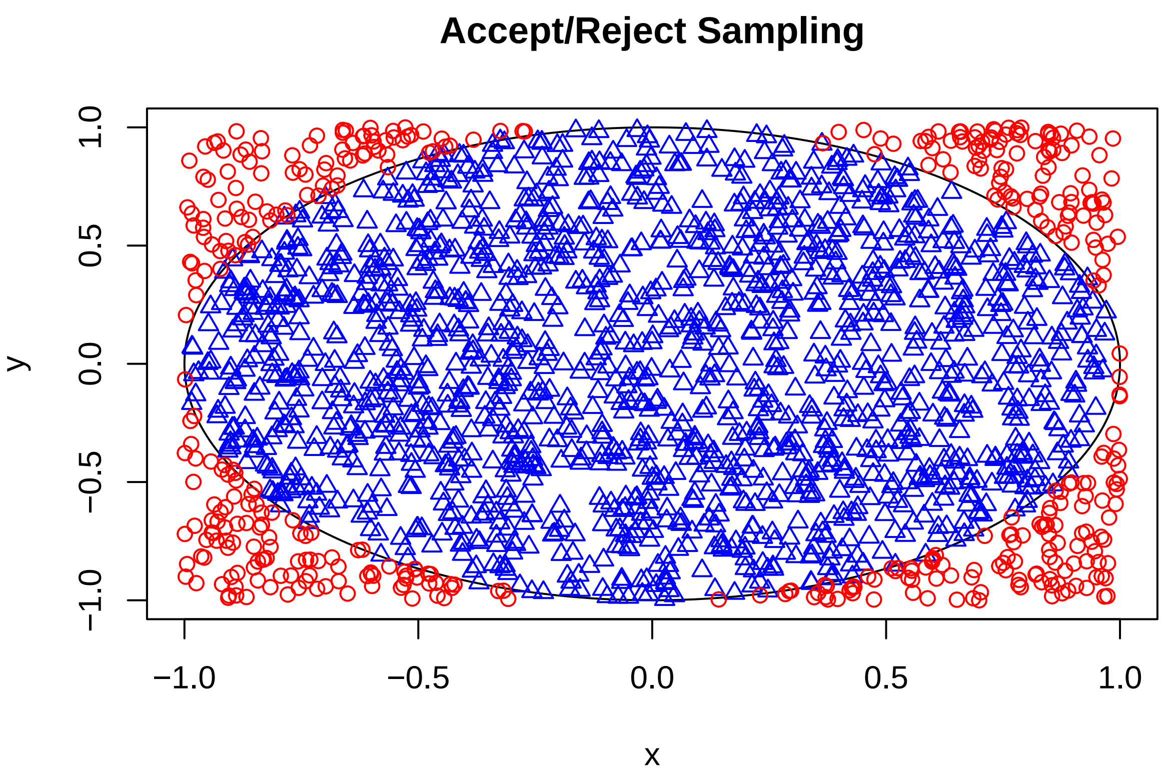
5. Uniformly generating points within a circle.

- Implement an algorithm to perform the following steps to uniformly generating points within a circle.
  - Generate a random angle θ uniformly distributed in the range $[0, 2\pi)$.
  - Genreate a random radius r uniformly distributed in the range [0, 1) with a square root scale, i.e., `r` <- sqrt(runif(1))‘.
  - Give polar coordinates (r, θ), get the Cartesian coordinates $x = r\cos\theta$ and $y = r\sin\theta$.
- Generate 500 points and use the code on Slide 48 of S3P1.pdf to plot the points.
- Compare the run time between this algorithm and the one based on accept/reject sampling (Slide 45 of S3P1.pdf).

```
# Function to generate uniformly distributed points within a circle
generate_circle_points <- function(n) {
  theta <- runif(n, 0, 2 * pi)      # Generate random angles between 0 and 2pi
  r <- sqrt(runif(n))               # Generate random radii with square root scaling
  x <- r * cos(theta)               # Convert to Cartesian coordinates (x, y)
  y <- r * sin(theta)

  return(cbind(x, y))               # Return as matrix of x and y values
}

# Set seed for reproducibility
set.seed(5400)

# Generate 500 points
points_circle <- generate_circle_points(500)

# Plot the points within the circle
plot(points_circle, pch=20, col="blue", xlab="x", ylab="y", main="Uniform Points in a Circle")
theta <- seq(0, 2 * pi, length.out = 1000)
lines(cos(theta), sin(theta))      # Draw the unit circle boundary
```




```r
# Accept/Reject Sampling to generate points in a circle
set.seed(5400)
n <- 2000                               # Total number of samples to generate
dat <- matrix(runif(n * 2, -1, 1), n, 2)  # Generate n pairs of (V1, V2) uniformly in [-1, 1]

# Accept points where V1^2 + V2^2 <= 1 (i.e., inside the unit circle)
accept <- (dat[, 1]^2 + dat[, 2]^2) <= 1
accepted_points <- dat[accept, ]     # Accepted pairs of (V1, V2)
rejected_points <- dat[!accept, ]    # Rejected pairs of (V1, V2)

# Plot the accepted and rejected points
theta <- seq(0, 2 * pi, length.out = 1000)
plot(cos(theta), sin(theta), type = 'l', xlab = 'x', ylab = 'y', main = 'Accept/Reject Sampling')
points(accepted_points, pch = 2, col = "blue")    # Accepted points
points(rejected_points, pch = 1, col = "red")     # Rejected points
```




```r
# Time for polar coordinates method
system.time({
  points_circle <- generate_circle_points(2000)
})
```

```
##    user  system elapsed
##       0       0       0
```

```r
# Time for accept/reject method
system.time({
  dat <- matrix(runif(n * 2, -1, 1), n, 2)
  accept <- (dat[, 1]^2 + dat[, 2]^2) <= 1
  accepted_points <- dat[accept, ]
})
```

```
##    user  system elapsed
##       0       0       0
```

6. (Optional) Advanced readings on accept-reject sampling. Read more on accept-reject sampling on Section 4.7 and 4.8 of http://statweb.stanford.edu/~owen/mc/Ch-nonunifrng.pdf. Implement Algorithm 4.8, which generates a gamma distribution using the accept-reject sampling.