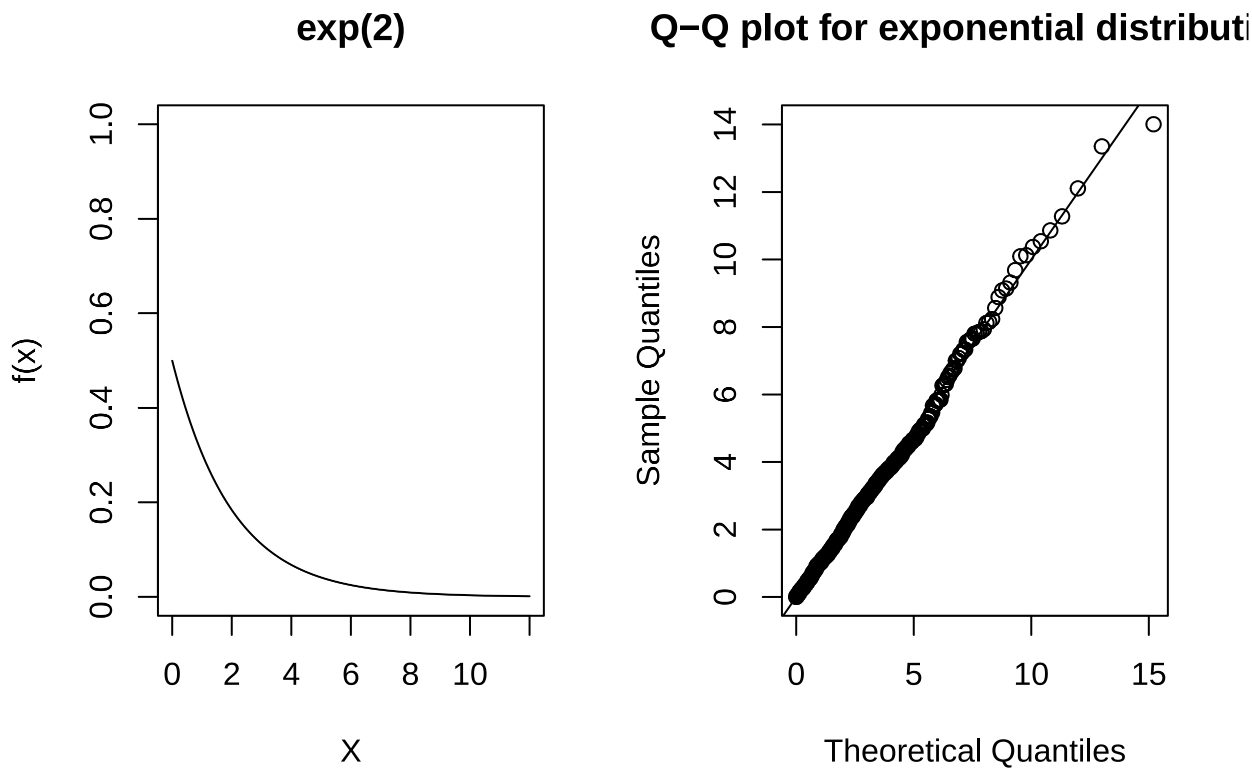
HW 5 CIS

Elias Washor

2024-10-04

Question 1)

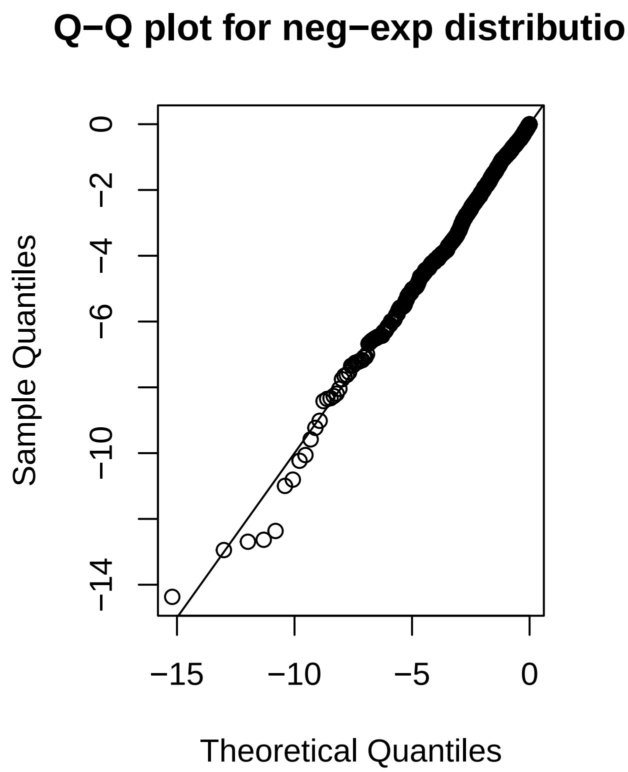
```r
Ulist <- runif(1000, 0, 1)
Xlist <- -2 * log(Ulist)


par(mfrow=c(1,2))
# 1-a)
plot(seq(0.001, 12, len=200), dexp(seq(0.001, 12, len=200), rate = 1/2) , type="l",
    ylim = c(0, 1), xlab="X", main="exp(2)", ylab="f(x)")

## 1-b)
plot(qexp(ppoints(1000), rate= 1/2), sort(Xlist),
    xlab="Theoretical Quantiles", ylab="Sample Quantiles",
    main="Q-Q plot for exponential distribution")
abline(0, 1)
```




```r
Ulist <- runif(1000, 0, 1)
Xlist <- 2 * log(Ulist)
Xlist[1:3]
```

```
## [1] -0.4254094 -0.6823601 -0.8018428
```

```r
par(mfrow=c(1,2))

## 1c
plot(- 1 * qexp(1 - ppoints(1000), rate= 1/2), sort(Xlist),
    xlab="Theoretical Quantiles", ylab="Sample Quantiles",
    main="Q-Q plot for neg-exp distribution")
abline(0, 1)
```

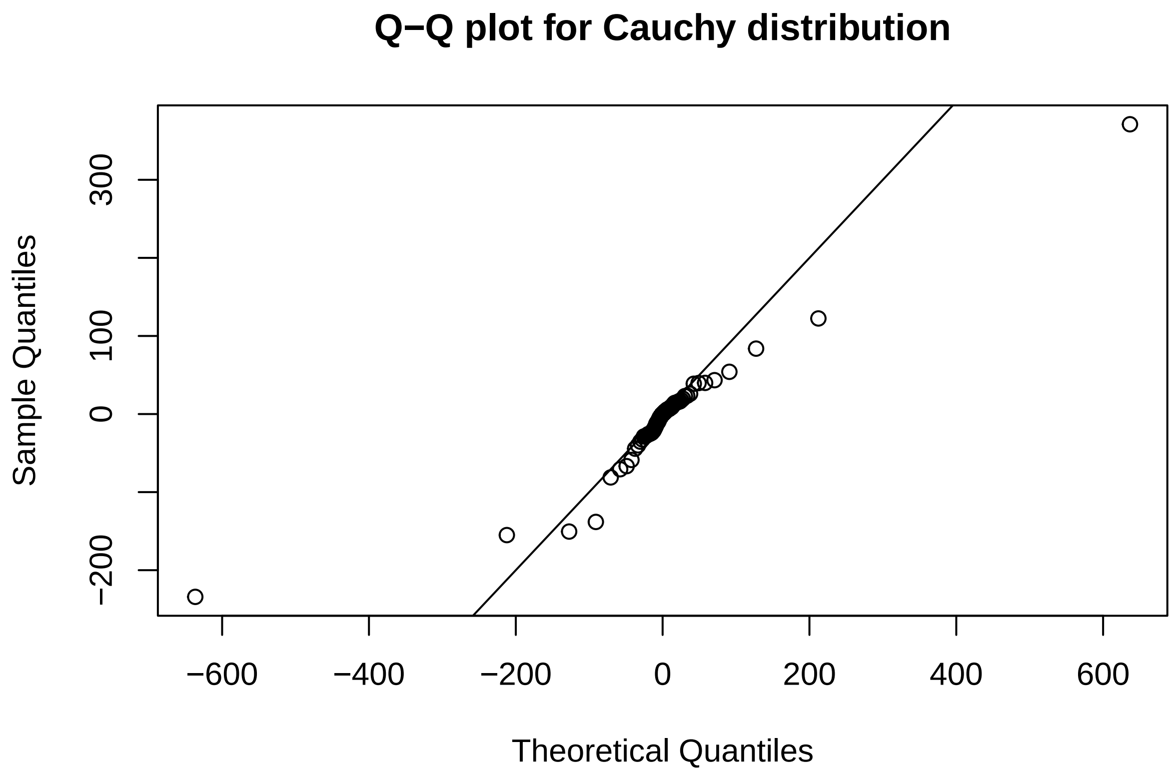


1b) The first Q-Q plot shows that the sample conforms to exponential distribution with a close fit between sample and theoretical quantiles.

Also, based on the Q-Q plot for Part C, we can see the points fit the reference line fairly well and thus the sample conforms to the negative exponential distribution.

2)

```
## gen 1000 numbers then transform with inverse CDF -- part a
U2 <- runif(1000, 0, 1)
cauchy_array <- tan(pi * (U2 - 0.5))

#min(cauchy_array)

probs = ppoints(1000)

## plot of standard cauchy -- part b
plot(qcauchy(probs) , sort(cauchy_array),
    xlab="Theoretical Quantiles",
    ylab="Sample Quantiles",
    #ylim = c(-1000,1000),
    main="Q-Q plot for Cauchy distribution")
abline(0, 1)
```

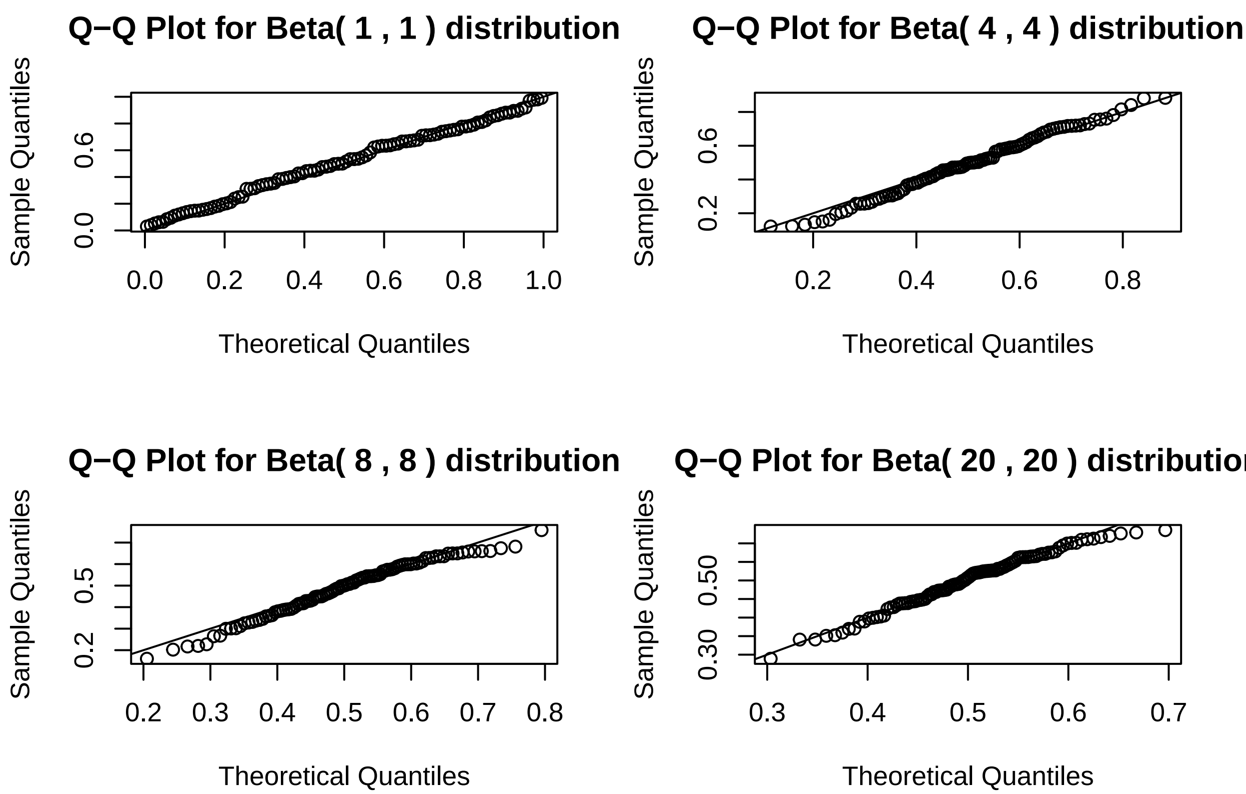


Most of the points on the Q-Q plot are close to the Cauchy theoretical quantiles, so this sample conforms to the standard Cauchy distribution.

3-a) Beta dist.

```
## gen beta dist 3a
beta_dist <- function(alpha, n) {
  Ulist1 <- runif(n)
  Ulist2 <- runif(n)
  Ulist3 <- runif(n)
  Indicator_List <- ifelse(Ulist3 <= 0.5, 1, -1)

  denom <- (Ulist1 ^ (-1/ alpha) - 1) * (cos(2 * pi * Ulist2))^2
  root <- sqrt(1 + (1 / denom ))
  Xlist <- (1/2) + ((Indicator_List)/(2 * root))
  return (Xlist)
}

## alphas
alpha_list <- c(1,4,8,20)

## gen 100 observations from beta(a,a)
## PART 3b
par(mfrow = c(2,2))
```

```r
for (a in alpha_list) {
  Xlist <- beta_dist(a, 100)

  title <- paste("Q-Q Plot for Beta(",a,",",a, ") distribution")
  ##qq plot for each sample, with specific alpha
  plot(qbeta(ppoints(length(Xlist)), a, a), sort(Xlist),
       xlab= "Theoretical Quantiles",
       ylab= "Sample Quantiles",
       main= title)
  abline(0,1)
}
```

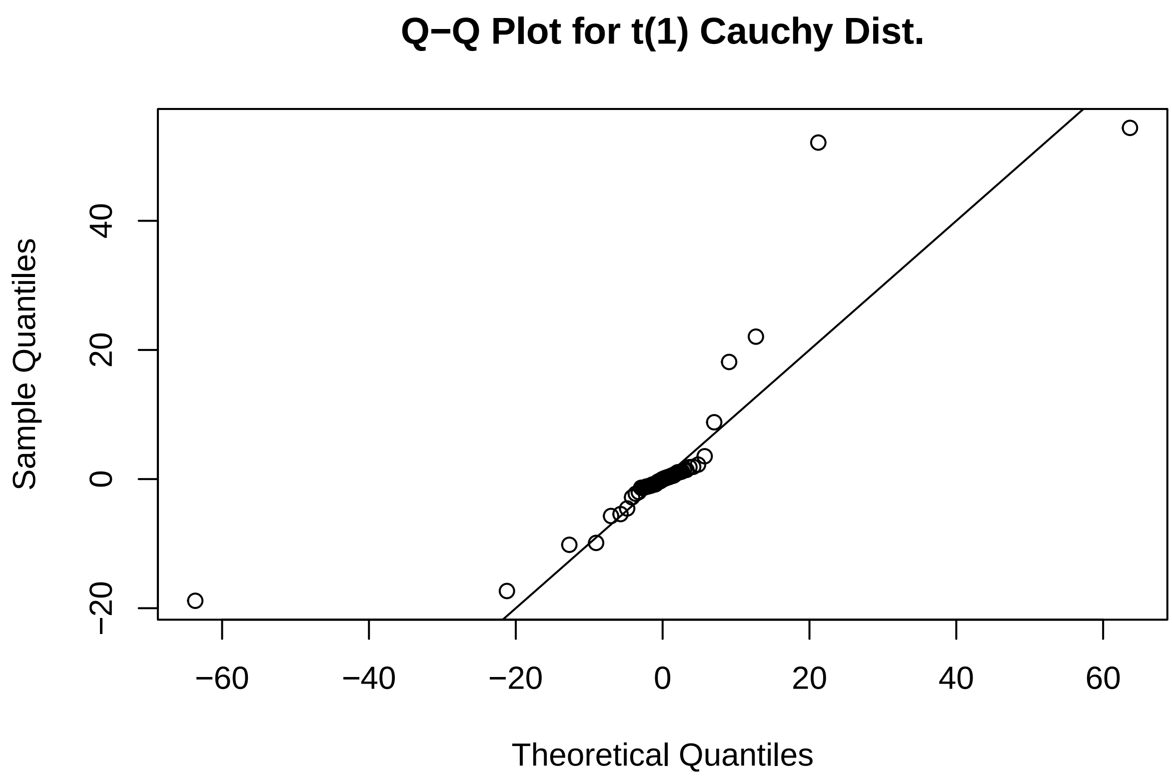


For all of the alphas ranging from 1 to 20, the Q-Q plots show that the sample conforms to each Beta(alpha, alpha) distribution because the points strongly follow the theoretical reference line.

3c and 3d) Cauchy dist.

```r
## alpha needs to be 0.5 to get t(1) dist. -- 100 obs.
Y <- beta_dist(0.5,100)

Z <- (sqrt(2*1/2)*(Y - 0.5)) / (2 * sqrt(Y * (1-Y)))

## 3-d QQ plot for Cauchy vs t(1) dist.
plot(qt(ppoints(100), 1), sort(Z),
```

```
      xlab = "Theoretical Quantiles",
      ylab = "Sample Quantiles",
      main = "Q-Q Plot for t(1) Cauchy Dist.")
abline(0,1)
```



This sample conforms to the t(1) distribution as shown by the points closely on the theoretical reference line. Also, the Q-Q plot from plotting the sample against the t(1) distribution closely resembles the Q-Q plot of Q2. The heavy-tails are evident in both Q-Q plots but the majority of points fall on the reference line, so they both conform to their respective distributions.

4 Accept-Reject Sampling)

```
accept_reject <- function(n, mu, sigma, a, b) {
  ## generate from normal(mu, sigma^2)
  temp <- rnorm(2*n, mu, sigma) ##
  ## discard
  Y_i <- temp[(temp > a) & (temp < b)]

  ## most likely won't be needed but in case the vector is short
  ## this will ensure a vector of length n
  while (length(Y_i) < n) {
    extras <- rnorm(n, mu, sigma)

    ## discard outside interval
```

```
    accepted <- extras[(extras > a) & (extras < b)]
    Y_i <- c(Y_i, accepted)
  }
  ## return vector of n truncated normal variables
  return (Y_i[1:n])
}

## test function for vector of 40 obs'ns mean 10, sd 3, (a,b = 4,16)
accept_reject(40, 10, 3, 4, 16)
```

```
##  [1] 12.084816  8.709704  8.299264 11.819960 12.626644  6.441079  8.860979
##  [8] 13.346882  6.425985  7.436086  9.795494  7.124832 11.055849  7.810412
## [15]  7.626114 14.277172 11.835682 10.494749  9.420581 13.395791 15.887113
## [22]  6.784088 13.209059  9.981577 15.978823  6.770383  9.535947  9.619941
## [29]  9.357034 15.941828 14.446427 13.171294  5.799630  8.267119  6.557553
## [36] 10.945691  7.037005  9.226958  6.792396  5.611895
```

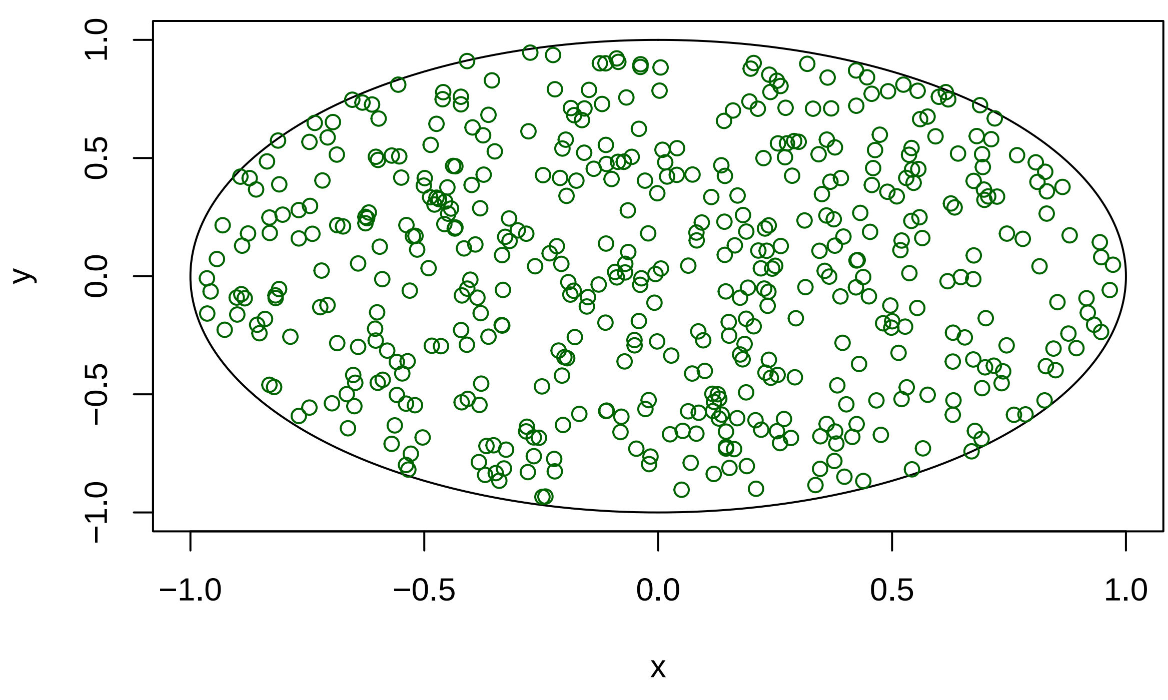
5 Uniform points within a circle)

```
## 5a Algorithm
circle_pts <- function(n) {
  M <- matrix(NA, n, 2)
  for (i in 1:n) {
    theta <- runif(1, max = 2*pi)
    r <- sqrt(runif(1, max = 1))
    x <- r * cos(theta)
    y <- r * sin(theta)
    M[i,] <- c(x,y)
  }
  return (M)
}

#draw circle
seq_1 <- seq(0, 2*pi, len=1000)
plot(cbind(sin(seq_1)*2/2, cos(seq_1)*2/2),
     type='l', xlab='x', ylab='y')

## 5b
## plot points on circle
points(circle_pts(500), pch=1, col='darkgreen')

## accept and reject algorithm from slide 45
accept_reject_circle_pts <- function(n) {
  M <- matrix(NA, n, 2)
  row <- 0
  while (row < n ) {
    V1 <- runif(1, -1, 1)
    V2 <- runif(1, -1, 1)
    if ((V1^2 + V2^2) <= 1) {
      row <- row + 1
      M[row,] <- c(V1, V2)
```

```r
    }
  }
  return (M)
}

#accept_reject_circle_pts(500)

## 5c comparing runtimes of both algorithms
library(microbenchmark)
```

```
## Warning: package 'microbenchmark' was built under R version 4.2.3
```

```r
microbenchmark(
  ## uniformly generates pts within circle
  circle_pts(500),
  ## accept reject approach
  accept_reject_circle_pts(500)
)
```




```
## Unit: milliseconds
##                           expr      min       lq     mean   median       uq
##                circle_pts(500) 1.599501 2.612201 4.137181 3.592100 4.456450
##  accept_reject_circle_pts(500) 1.930402 3.209950 5.205426 4.160751 5.488051
##       max neval cld
```

```
##  16.2796   100  a
##  25.1221   100   b
```

Using microbenchmark, we can see that the algorithm that generates theta and a radius within certain ranges has an edge on the accept-reject sampling approach (based on generation of 500 points within a circle). It's mean runtime is apx. 3.3 ms, while the accept-reject algorithm has a mean runtime of apx. 4.5 ms.