STAT5400: Homework 5

Dev Narayan Baiju

Due: Oct 4, 2024 9:30 AM

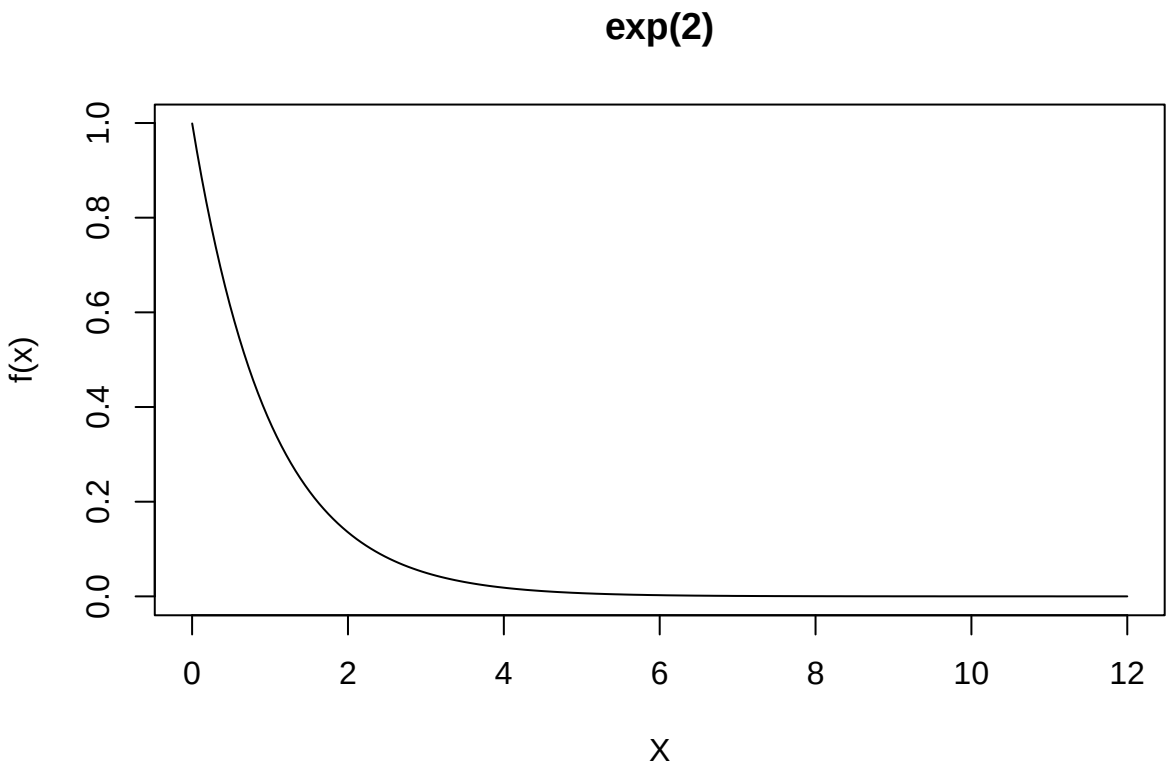
Problems

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

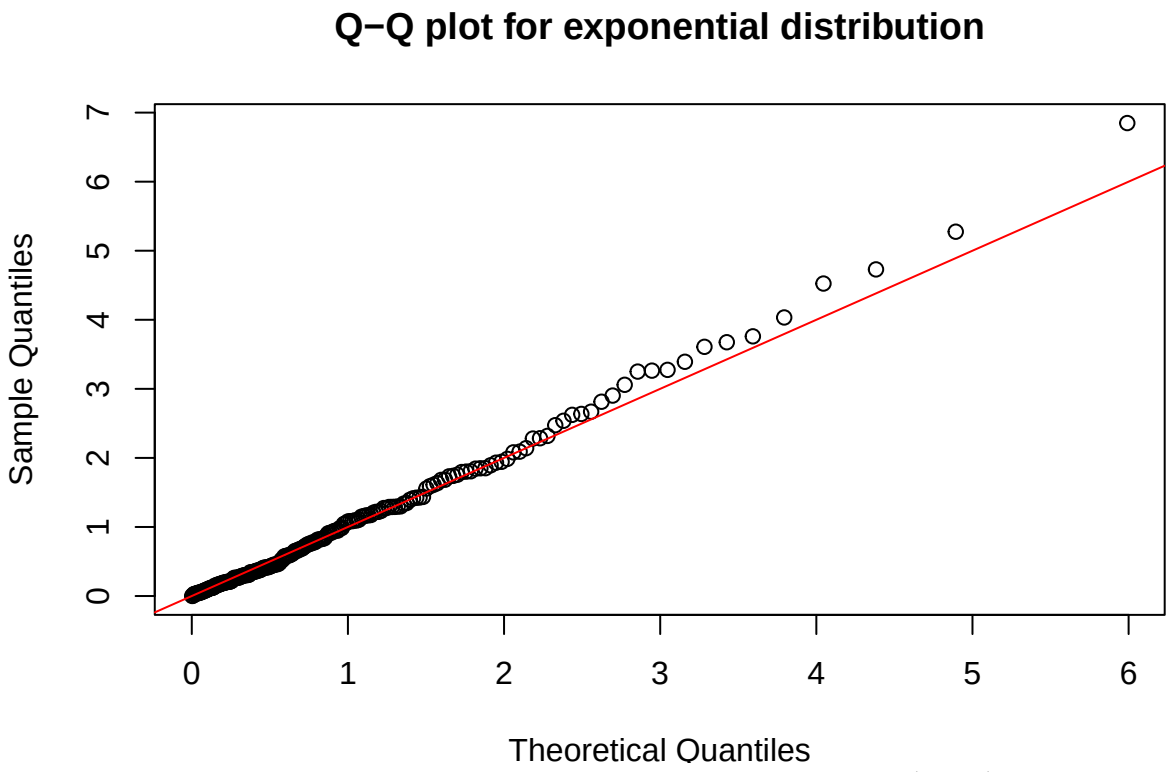
NOTE: Parts of this assignment was done by the help of a classmate ##### 1. Generators of exponential distributions + Fill in the code on Slide 36 of S3P1.pdf. You may either use the pdf of exp(2) manually or call `dexp` in R.

```
expplot_list = seq(0.001, 12, len = 200)
exp_transform = dexp(expplot_list)
plot(expplot_list, exp_transform, type="l",
     xlab="X", main="exp(2)", ylab="f(x)")
```



+ Fill in the code on Slide 37 of S3P1.pdf.

```r
data_points = rexp(200, rate = 1)
quant = ppoints(length(data_points))
theoreticalq = qexp(quant, rate = 1)
sort_data_points = sort(data_points)
plot(theoreticalq, sort_data_points,
     xlab="Theoretical Quantiles", ylab="Sample Quantiles",
     main="Q-Q plot for exponential distribution")
abline(0, 1, col = 'red')
```



(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.

2. Generators of Cauchy distributions.

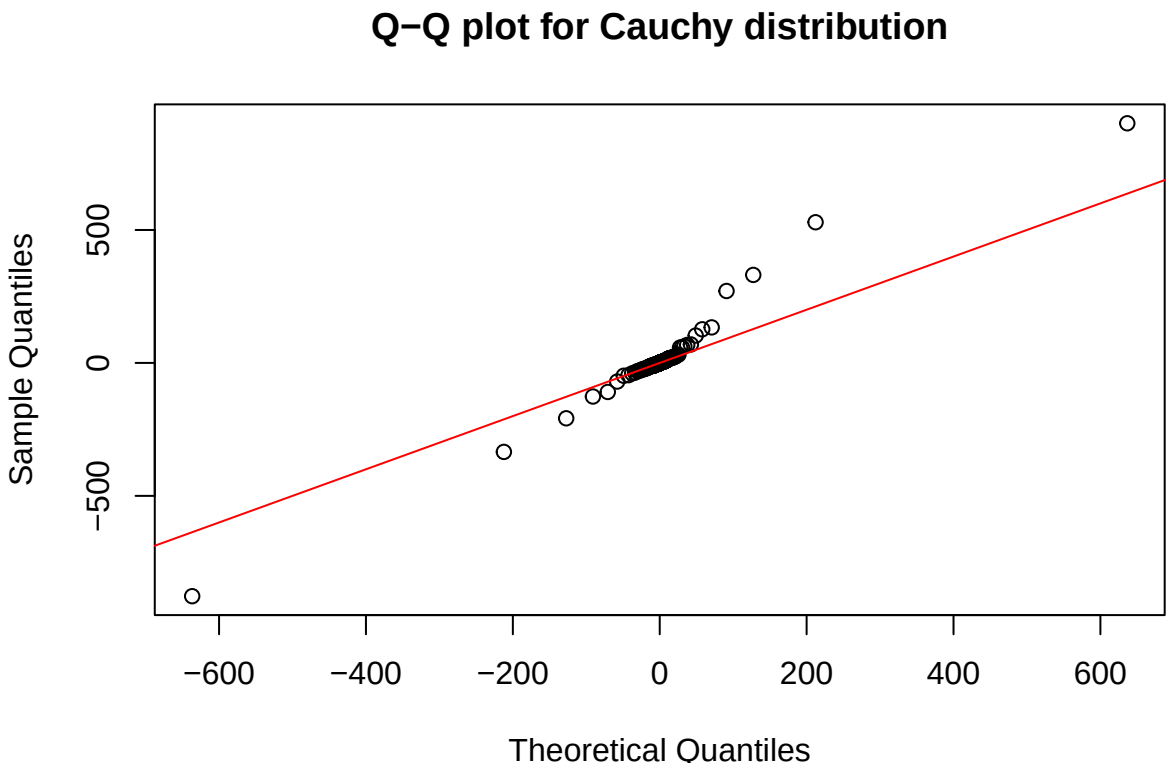
- Implement an algorithm to generate the standard Cauchy distribution using the inverse CDF.
- Check if the generated numbers conform to the standard Cauchy distribution. Basically, you may check this by filling in the code on Slide 42 of S3P1.pdf.

Given below is the code for generating the standard Cauchy distribution and checking if the generated numbers conform to the standard Cauchy distribution using QQplot.

```r
set.seed(2310)
data_p_c <- runif(1000, -10, 10)
cauchy_data <- tan(pi*(data_p_c-0.5))

probs = ppoints(1000)
plot(qcauchy(probs), sort(cauchy_data),
     xlab="Theoretical Quantiles",
```

```
     ylab="Sample Quantiles",
     main="Q-Q plot for Cauchy distribution")
abline(0,1,col = 'red')
```



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.

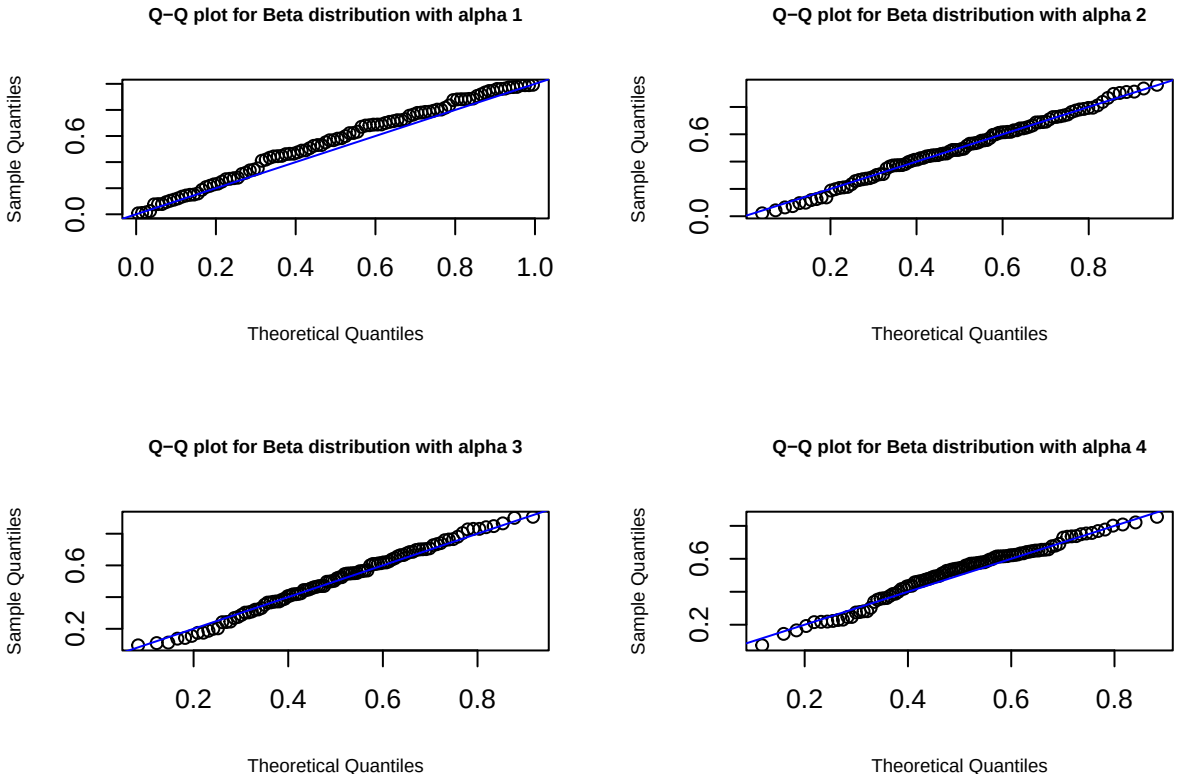
```
beta <- function(alpha){
  U1 = runif(100)
  U2 = runif(100)
  U3 = runif(100)
  IU3 = 1*(U3<=0.5)
  IU3 = replace(IU3, IU3 == 0, -1)
  denom = (U1^(-1/alpha)-1)*(cos(2*pi*U2))^2
  Xdis = 0.5 + IU3/(2*sqrt(1+(1/denom)))
  return(Xdis)
}
beta(2)
```

```
##    [1] 0.48879677 0.28961738 0.15675183 0.33530775 0.86618643 0.71283773
```

```
##   [7] 0.25325262 0.57656339 0.04160128 0.12429741 0.24354102 0.18688751
##  [13] 0.20811655 0.03171293 0.08337544 0.51176458 0.70095175 0.43944593
##  [19] 0.23758414 0.58194981 0.68400635 0.30840673 0.22161936 0.06291231
##  [25] 0.39983189 0.35151415 0.44124433 0.55009236 0.30124952 0.72620491
##  [31] 0.08962149 0.55051436 0.58995218 0.64003342 0.73004035 0.33091123
##  [37] 0.61303790 0.61734953 0.86176313 0.49906362 0.47422487 0.55323648
##  [43] 0.40378528 0.75471969 0.63823759 0.25137722 0.46662120 0.67229041
##  [49] 0.49259294 0.41847870 0.45308588 0.72662741 0.51728009 0.42982715
##  [55] 0.53553146 0.83039422 0.65529874 0.59589293 0.20827214 0.17010523
##  [61] 0.82581984 0.82962063 0.46445355 0.73513314 0.56802346 0.69676246
##  [67] 0.21807076 0.12130665 0.17622502 0.57193462 0.32079234 0.66090267
##  [73] 0.36326151 0.68789470 0.84472523 0.42608485 0.65240637 0.37545666
##  [79] 0.38272779 0.77834857 0.74699436 0.43105236 0.38056695 0.85988213
##  [85] 0.28080379 0.24337370 0.37282423 0.54010613 0.40513092 0.60377514
##  [91] 0.57090430 0.57106190 0.22193642 0.57056338 0.88794022 0.45375806
##  [97] 0.08322288 0.50706846 0.77193668 0.91764569
```

- 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
par(mfrow = c(2,2))
for(i in 1:4){
  beta_dis = beta(i)
  qqplot(qbeta(ppoints(100), shape1 = i, shape2 = i),
         sort(beta_dis),
         xlab="Theoretical Quantiles",
         ylab="Sample Quantiles",
         main=paste("Q-Q plot for Beta distribution with alpha", i),
         cex.main = 0.7,
         cex.lab = 0.7)
  abline(0,1, col = 'blue')
}
```



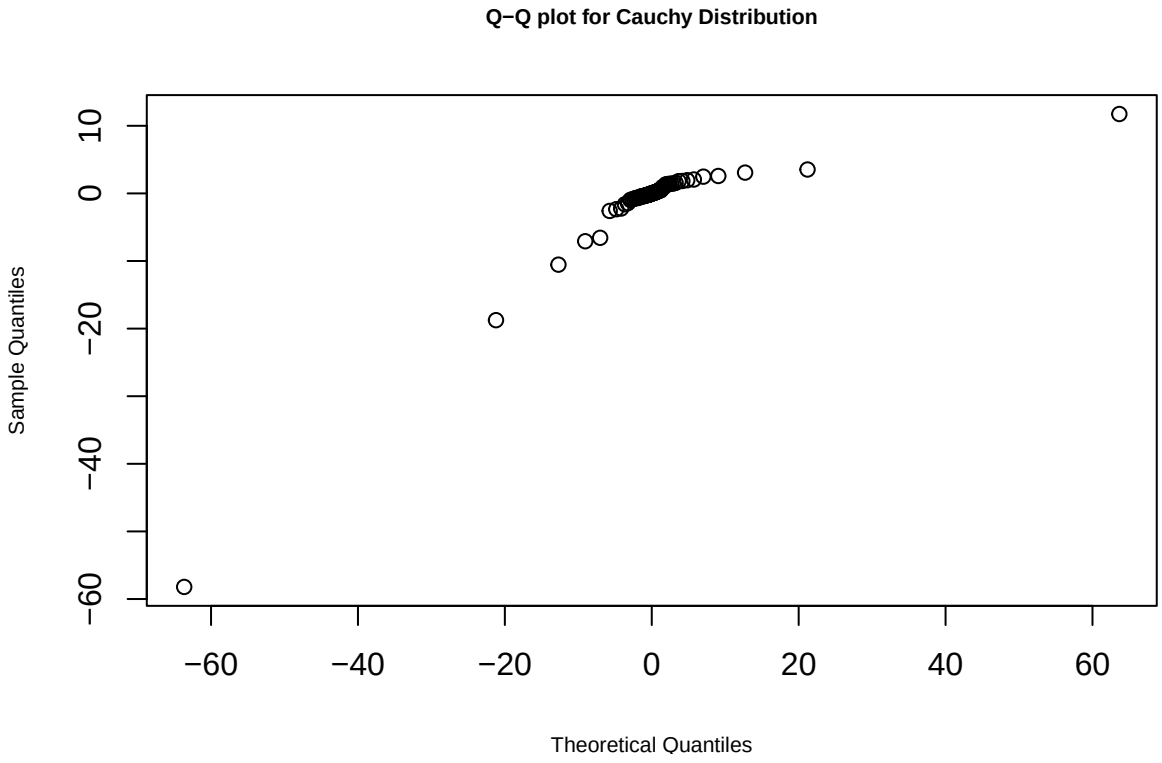
+ Using the above method to generate $Y \sim \text{Beta}(n, n)$. Generate a t-distribution with degree of freedom $2n$ using

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

```
#Function to generate t-distribution
tdist <- function(dof){
  Y <- beta(dof/2)
  Z <- sqrt(dof/2)*(Y-0.5)/(2*sqrt(Y*(1-Y)))
}
```

- 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.

```
qqplot(qcauchy(ppoints(100)),
         sort(tdist(1)),
         xlab="Theoretical Quantiles",
         ylab="Sample Quantiles",
         main=paste("Q-Q plot for Cauchy Distribution"),
         cex.main = 0.7,
         cex.lab = 0.7)
```



The QQplot of the previous cauchy plot and this cauchy (t(1)) is not exactly the same but there are similarities.

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.

NOTE: This answer was done completely by internet assistance

```
trunc_norm <- function(n, mu, sigma, a = -Inf, b = Inf){
  norm_dist <- qnorm(runif(n, pnorm(a, mu, sigma), pnorm(b, mu, sigma)),
                     mu, sigma)
  return(norm_dist)
}
print(trunc_norm(20, 10, 5, 7.5, 13))
```

```
##  [1] 11.020978 10.325432 11.242880 11.765091 10.884902  8.310742  8.696314
##  [8] 12.499440 12.013123  7.741628  9.683755 10.860696 11.967916  8.674728
## [15]  8.699652  8.124087  8.927692 10.231410  8.067695  7.731649
```

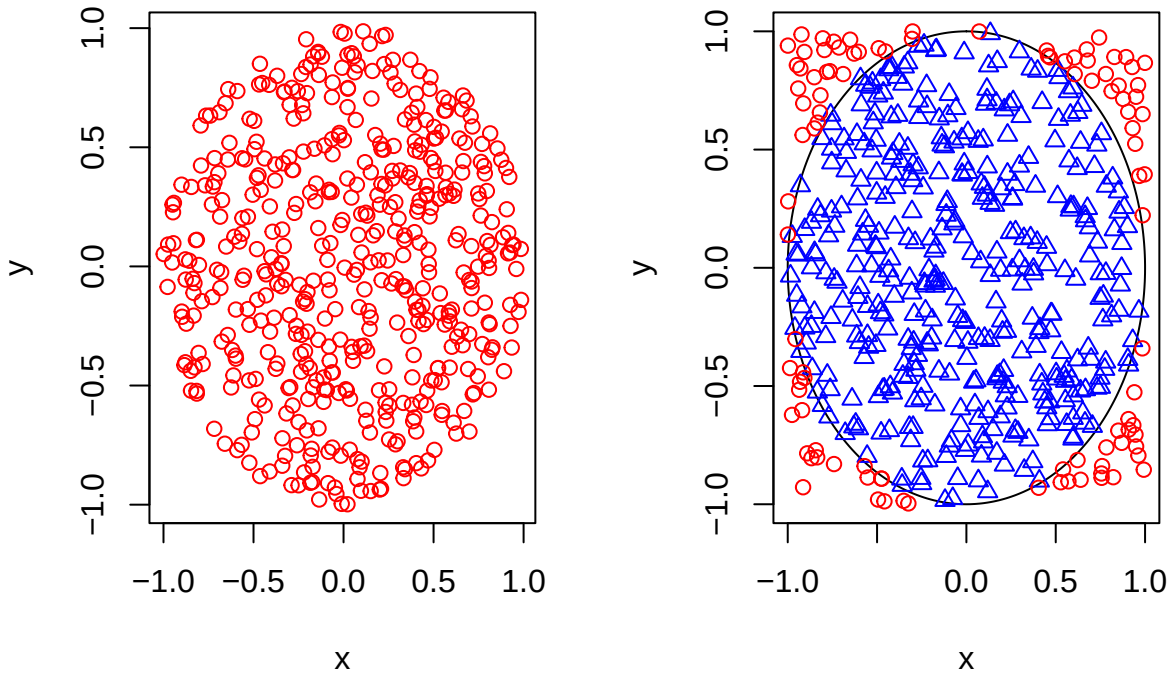
5. Uniformly generating points within a circle.

- 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).

```
par(mfrow = c(1,2))
algo <- function(n){
  x <- rep(NA, n)
  y <- rep(NA, n)
  for(i in 1:n){
    theta <- runif(1, 0, 2*pi)
    r <- sqrt(runif(1))
    x[i] <- r*cos(theta)
    y[i] <- r*sin(theta)
  }
  plot(x, y, col = 'red')
}
accept_reject <- function(n){
  set.seed(5400)
  # Each row is a pair of V1 and V2. n rows in total.
  dat <- matrix(runif(n*2, -1, 1), n, 2)
  # Accept if V_1^2 + V_2^2 <= 1.
  accept <- (dat[,1]^2 + dat[,2]^2) <= 1
  mean(accept) # which is about pi/4
  # Accepted pairs of V1 and V2
  V <- dat[accept,]
  # Rejected pairs of V1 and V2
  V_out <- dat[!accept,]
  # draw a circle
  theta <- seq(0, 2*pi, len=1000)
  plot(cbind(sin(theta)*2/2, cos(theta)*2/2),
  type='l', xlab='x', ylab='y')
  points(V, pch=2, col="blue") # Accepted pairs
  points(V_out, pch=1, col="red") # Rejected pairs
}
print(system.time(algo(500)))
```

```
##    user  system elapsed
##   0.012   0.000   0.012
```

```
print(system.time(accept_reject(500)))
```




```
##    user  system elapsed
##   0.016   0.000   0.016
```

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.