Stats 102C, Homework 1

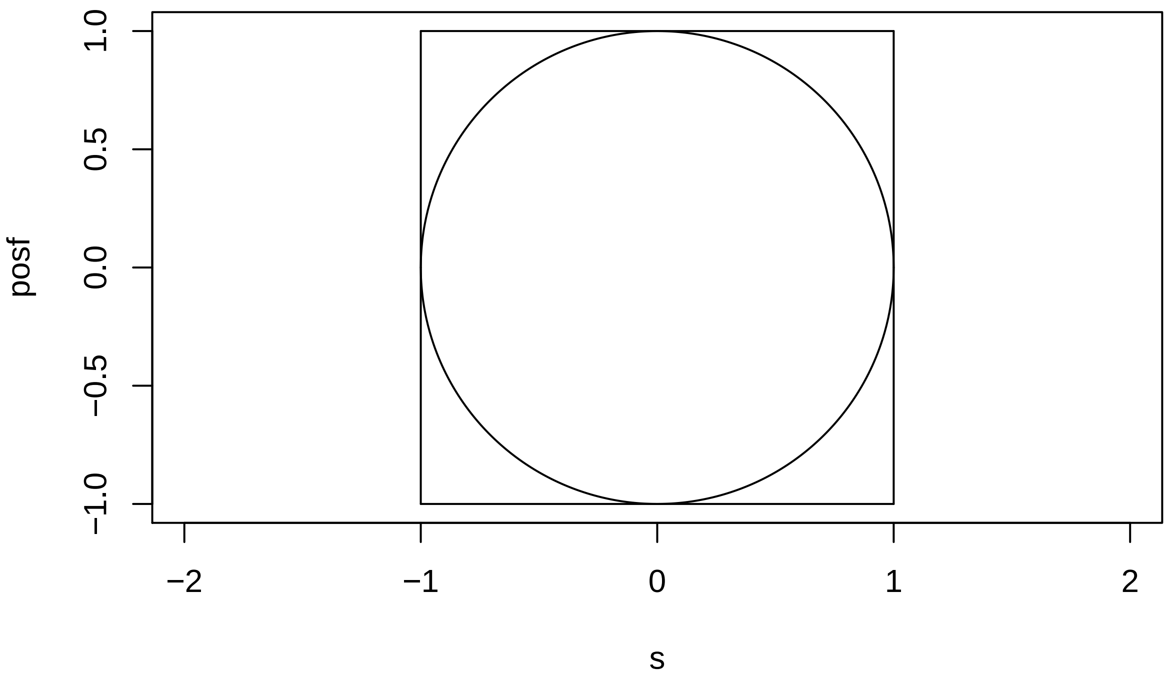
JANG,JUNHYUK

SID : 004 728 134

Problem 1 - Estimate pi (poorly)

Find an estimate of pi by estimating the ratio between the area of a circle and its encompassing square.

```r
s <- seq(-1,1, by = 0.001)
posf <- sqrt(1-s^2)
plot(s, posf, type = "l", asp = 1, ylim = c(-1,1))
lines(s, -1*posf)
segments(-1,-1,-1,1)
segments(-1,-1,1,-1)
segments(1,1,-1,1)
segments(1,1,1,-1)
```



To calculate the area of the circle analytically, we would need to integrate the function drawing the upper semi-circle and then multiply that by 2. This process requires the use of trig substitutions, and while doable, can illustrate a time where the analytic solution is not easy.

$$Area = 2 \times \int_{-1}^{1} \sqrt{1-x^2}dx$$

For the Monte-Carlo approach, we will use `runif(n, min = -1, max=1)` to generate a bunch of random pairs of x and y coordinates. We will see how many of those random uniform points fall within the circle. This is easy - just see if $x^2 + y^2 \leq 1$. The total area of the square is 4. The total area of the circle is pi. Thus, the proportion of coordinates that satisfy the inequality $x^2 + y^2 \leq 1 \approx \pi/4$.

Instructions:

- create a vector x of n random values between -1 and 1. I suggest starting with n = 500
- create a vector y of n random values between -1 and 1. Use the two vectors to make coordinate pairs.
- calculate which of points satisfy the inequality for falling inside the circle.
- Print out your estimate of pi by multiplying the proportion by 4.
- plot each of those (x,y) coordinate pairs. Use pch = 20. Color the points based on whether they fall in the circle or not.

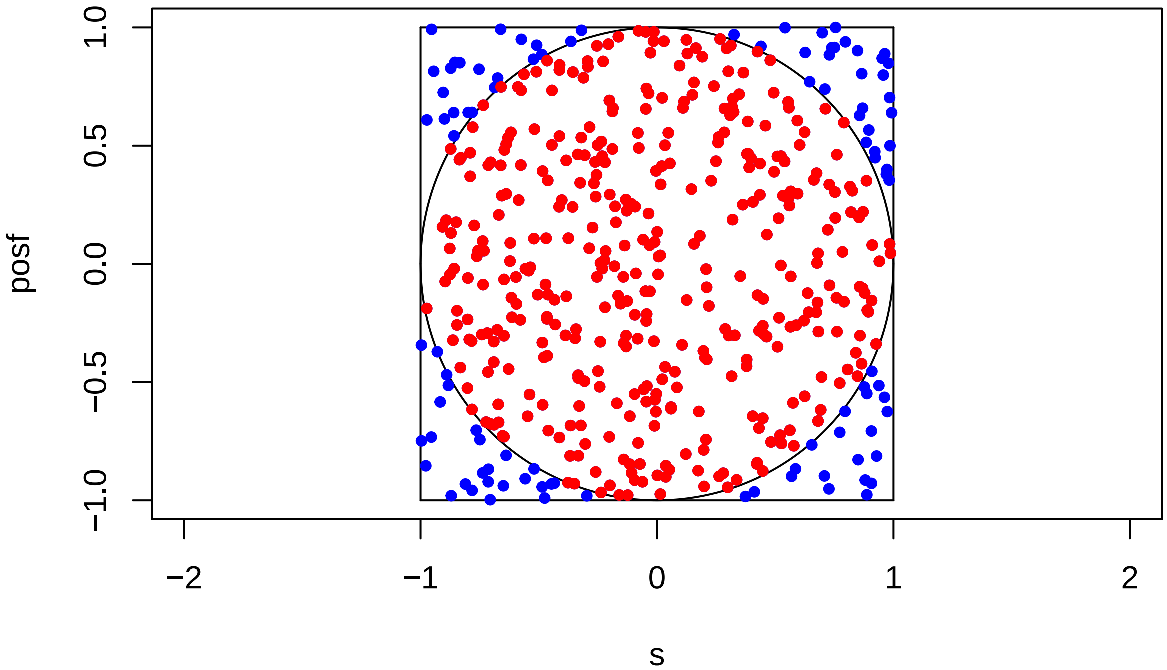
```
set.seed(1)
x <- runif(500,-1,1)
y <- runif(500,-1,1)

x_satis <- c()
y_satis <- c()
satisf <- function(x,y){
        count <- 0
        for(i in 1: length(x)){
                if((x[i])^2 + (y[i])^2 < 1){
                        count <- count + 1
                        x_satis[i] <- x[i]
                        y_satis[i] <- y[i]
                }
        }
        return(list(count = count, x_satis = x_satis, y_satis = y_satis))
}
# The number of total satisfied points among 500
satisf(x,y)$count
```

```
## [1] 401
```

```
# calculate which of points satisfy the inequality for falling inside the circle.
satis_df <- data.frame(satisfied_x = satisf(x,y)$x_satis,satisfied_y = satisf(x,y)$y_satis)
clean_satis <- na.omit(satis_df)

# Print out your estimate of pi by multiplying the proportion by 4.
(pi <- (satisf(x,y)$count / 500) * 4)
```

```
## [1] 3.208
```

```
# plot each of those (x,y) coordinate pairs. Use pch = 20.
# Color the points based on whether they fall in the circle or not.
s <- seq(-1,1, by = 0.001)
posf <- sqrt(1-s^2)
plot(s, posf, type = "l", asp = 1, ylim = c(-1,1))
lines(s, -1*posf)
segments(-1,-1,-1,1)
segments(-1,-1,1,-1)
segments(1,1,-1,1)
segments(1,1,1,-1)

points(x,y, col = "blue",pch = 20)
points(clean_satis$satisfied_x,clean_satis$satisfied_y,col = "red",pch = 20)
```



Problem 2

Generate Random variable from an exponential distribution

Write a function `my_rexp(n, rate)`, that will generate `n` random values drawn from an exponential distribution with lambda = "rate" by using the inverse CDF method. Use `runif()` as your sole source of randomness.

You are not allowed to use any of the functions `dexp()`, `pexp()`, `qexp()`, or `rexp()`.

Use your function to generate 500 random samples from an exponential distribution with lambda = 1.

After generating 500 samples, plot the empirical CDF function of your data (see `ecdf`). Add the theoretic CDF of the exponential distribution to the same plot (in a different color).

Use the Kolmogorov-Smirnov test to compare your generated samples to the theoretic exponential distribution. Be sure to print out the resulting p-value and comment on the sample produced by your function.

```r
my_rexp <- function(n, rate){
  x <- c()
  for(i in 1:n){
          x[i] <- (-1/rate)*log(runif(1))
  }
  return(x)
}
set.seed(1234)
(p <- runif(1))
```

```
## [1] 0.1137034
```

```r
set.seed(1234)
my_rexp(500,1)
```

```
##   [1] 2.174161876 0.474333944 0.495485992 0.472599890 0.149759056
##   [6] 0.445801900 4.656910280 1.458647847 0.406339853 0.665043531
##  [11] 0.365872409 0.607015659 1.263250225 0.079656508 1.229920417
```

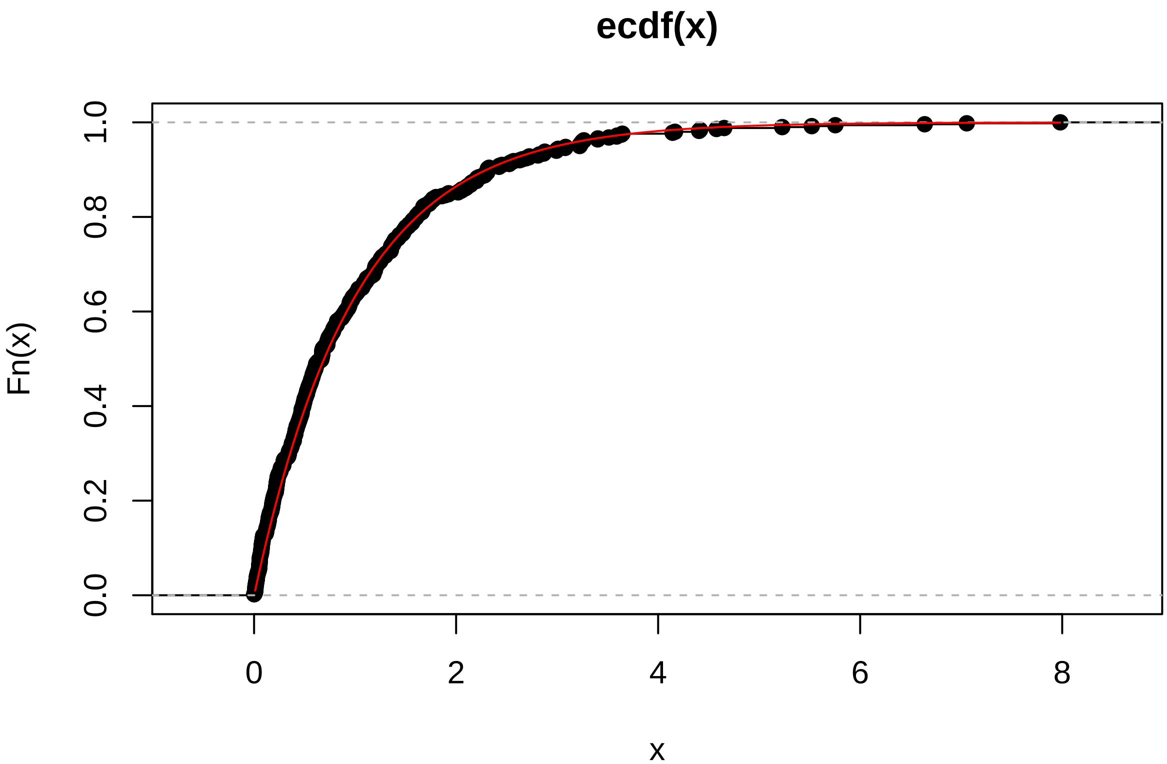
```
##  [16] 0.177578071 1.250983057 1.321178082 1.678130170 1.460044629
##  [21] 1.150076793 1.195034964 1.838561792 3.218977879 1.519599306
##  [26] 0.209982351 0.643029238 0.089204873 0.184710351 3.084120672
##  [31] 0.785061870 1.327321279 1.188518825 0.678639192 1.708726851
##  [36] 0.274870314 1.603217114 1.351661778 0.007880553 0.213995101
##  [41] 0.591794221 0.436327344 1.165315368 0.475105907 1.109359304
##  [46] 0.689160193 0.389944389 0.723624452 1.410878787 0.267278596
##  [51] 2.606669212 1.172194455 0.332300510 0.684096438 1.877324162
##  [56] 0.685310988 0.705298868 0.286083089 1.744972316 0.164412002
##  [61] 0.145217892 3.173489658 1.148279048 4.286720880 1.431184090
##  [66] 0.347439696 1.177347890 0.676196527 2.963330539 0.571691183
##  [71] 2.108004098 0.113351938 4.224868660 0.244467929 2.408375335
##  [76] 0.655485411 0.956418469 2.658510356 1.137422489 0.402725768
##  [81] 0.076448658 0.750967580 1.947604181 0.608310282 1.628749936
##  [86] 0.106938995 0.942891966 1.168377953 1.832402336 0.109607466
##  [91] 1.793398129 0.104888853 2.009332104 2.027880863 2.251060551
##  [96] 0.670244303 1.203309511 3.622459127 1.172320947 0.298244786
## [101] 3.339442283 0.570794845 1.272045470 1.588673412 2.011865897
## [106] 1.121834066 1.863930439 2.040512105 0.831189165 3.253398624
## [111] 0.337850999 2.294924102 0.050972358 2.105229099 1.515689766
## [116] 0.090923273 0.055667985 1.275983715 2.091748257 0.226699291
## [121] 0.295341713 0.087767055 0.005416397 0.059367153 0.721268077
## [126] 1.260685872 1.380130592 0.686657940 0.699233317 1.144302961
## [131] 0.038509226 0.455549605 2.060161419 0.860272010 0.089578039
## [136] 0.759730814 0.096324634 0.514593909 0.459272904 0.140229985
## [141] 0.687662603 0.016500269 1.125821030 0.731108794 1.030055695
## [146] 0.466047172 0.298945008 0.569219829 0.019400471 0.550237614
## [151] 0.823160081 1.475782862 2.499110258 0.162207305 1.449612245
## [156] 0.011903115 0.507668038 0.001259985 0.979232191 0.588559027
## [161] 0.845264017 0.551859827 0.838155849 1.492340605 2.465283595
## [166] 0.450517507 0.841609207 2.621192549 0.220145526 1.123074145
## [171] 0.278010282 0.537389477 0.344126285 0.851028020 1.068356539
## [176] 0.275595431 0.857950581 0.578235367 2.152995303 1.193950537
## [181] 0.736466696 1.064702151 0.509636098 2.575926181 0.045005095
## [186] 3.807355690 0.172318991 0.458166053 1.170879269 0.297638990
## [191] 0.447989624 0.007512158 2.053619512 0.124158793 0.210618089
## [196] 0.196195949 0.180679710 0.310974964 0.017101374 0.447530715
## [201] 0.414372712 0.637978587 1.147296921 0.264153755 0.641867750
## [206] 0.311562440 1.178741343 0.905911653 1.587664457 0.014471115
## [211] 0.568612311 1.271626768 1.687098293 0.276990899 0.567781797
## [216] 0.070236247 0.448330883 0.355606751 0.735590346 0.162152036
## [221] 0.861966681 3.461198275 1.354227451 1.094088333 2.013281821
## [226] 0.694054822 0.220477567 1.087217725 0.675463237 0.704332379
## [231] 0.226834223 0.567468473 2.237763943 0.213628408 0.567198339
## [236] 1.550033197 0.288243224 1.180196546 0.714333140 0.010343416
## [241] 0.857764361 1.408936758 1.527237312 0.372343476 0.019985681
## [246] 0.740169526 0.256799070 0.554580831 0.034653868 0.226996162
## [251] 0.631290213 0.516468580 1.332236293 1.274600194 2.731780896
## [256] 0.574331174 1.338054130 5.719201389 0.528452496 0.653828078
## [261] 0.168851012 3.521440981 0.511210116 1.315203219 2.115202163
## [266] 2.295555245 0.290136896 4.137630256 3.006567854 0.290855387
## [271] 1.029353994 0.275808581 0.978282253 0.223815391 3.661545821
## [276] 0.680510263 0.196953711 0.607416233 1.321689138 1.065262655
## [281] 0.996481912 0.845710923 0.084997726 0.242906540 0.304039757
```

```
## [286] 1.270210105 0.783451329 1.246401500 0.361987505 0.197529049
## [291] 0.422885671 0.883086453 0.049369447 1.414243483 0.496454343
## [296] 0.277135921 0.365762739 2.159110085 0.452695618 1.174331907
## [301] 1.041291465 0.019225311 0.618257239 0.811854451 0.051960066
## [306] 0.793004265 1.657442916 0.008426252 0.600649228 0.262903913
## [311] 0.090557651 0.382561409 0.898324544 0.897487870 1.923581761
## [316] 1.626239705 1.649170191 0.895472932 1.054917064 0.180871191
## [321] 1.617470048 0.148725893 0.923352331 1.875660124 1.080919582
## [326] 1.001901891 0.850083878 1.680198930 0.418525119 0.082934555
## [331] 0.309390590 0.125201303 0.047776820 1.635260796 0.749470504
## [336] 0.951786685 0.983056111 3.580719225 0.073264254 0.890308036
## [341] 0.045164518 1.301391458 0.659235741 0.021928847 0.995072917
## [346] 1.169795431 3.375248021 0.404117246 0.082386043 3.101323972
## [351] 1.603790729 0.296366494 2.035946659 0.344063950 0.001168872
## [356] 0.057721229 0.522722952 0.312937474 0.720037146 0.263767654
## [361] 5.761794383 0.583501522 0.775973967 1.109526194 0.179754103
## [366] 0.022550704 0.414735525 1.454252656 0.199426512 0.322018346
## [371] 0.023926824 1.327599021 0.129175132 0.717666923 1.185909318
## [376] 0.928623977 0.275306018 2.294600631 0.864268439 0.418908698
## [381] 1.219885096 1.579944647 6.143818000 2.227081793 1.537735945
## [386] 2.161487749 0.375188944 1.643655565 0.015192937 0.054416085
## [391] 0.368592868 0.301251126 0.311373445 0.426793279 1.789484218
## [396] 0.079930118 0.571629180 0.622696852 3.971574133 1.004030662
## [401] 0.376524544 0.875559806 0.278357355 0.253867139 0.555845619
## [406] 1.667603225 0.276721910 2.378569208 0.474238943 0.877278690
## [411] 0.252944637 0.213437307 0.033250398 1.532924883 0.142706024
## [416] 0.874865651 0.667423459 0.254973130 2.020845761 0.886956124
## [421] 0.412417863 0.096819575 1.065685464 2.251482707 0.070051372
## [426] 1.609771604 2.918432267 0.853691412 1.082357655 1.288800013
## [431] 0.410240680 0.243478797 0.821668592 0.072222509 1.301191717
## [436] 0.416987695 0.966823016 0.066790893 0.307203475 0.527479443
## [441] 0.204993742 0.125099805 0.293778875 0.689570463 0.011565146
## [446] 0.588304307 0.128172255 0.461338458 0.489038803 3.392952474
## [451] 1.367666205 0.253642964 0.001756541 0.040698667 0.286444421
## [456] 0.384921783 1.398826748 0.499946719 0.555177167 3.003867107
## [461] 0.986659161 0.111190758 0.937075890 0.659818270 1.741370148
## [466] 1.647111413 0.604205060 0.933638237 0.469687684 0.558258358
## [471] 1.961700672 1.240389000 7.398523786 0.045663155 0.917766946
## [476] 0.023213822 0.669871769 0.761080923 0.323191131 1.951410376
## [481] 0.647069684 0.579066778 0.414996176 0.988147427 1.993983420
## [486] 2.605300590 0.272449112 0.352690161 0.269324591 1.851151205
## [491] 0.511918543 0.270049397 0.091399549 1.177587804 0.419970999
## [496] 1.085287571 0.503766627 2.477474114 0.501424560 0.517663908
```

```r
-log(0.8010433)
```

```
## [1] 0.2218403
```

```r
x <- my_rexp(500, rate = 1)
plot(ecdf(x))
vals <- seq(0.01, max(x), by = 0.01)
lines(vals, pexp(vals, rate = 1), col = "red")
```




```
# ks test
ks.test(x,pexp(500,1))
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  x and pexp(500, 1)
## D = 0.634, p-value = 0.7345
## alternative hypothesis: two-sided
```

```
# Large p-value indicates that there is an insufficient evididence to reject the
# null hypothesis and coclude that the two data come from the
# same distribution.
```

Problem 3

Generate Random variable from an binomial distribution

Write a function `my_rbinom(n, size, prob)`, that will generate `n` random values drawn from a binomial distribution with size = `size` and probability of success = `prob` by using the inverse CDF method. Use `runif()` as your sole source of randomness.

Do not use any of R's binom functions. Do not use `dbinom`, `pbinom`, `qbinom()`, or `rbinom()`

Use your function `my_rbinom()` to generate 200 values from a binomial distribution with n = 6, and p = 0.4.

After generating 200 samples, make a side-by-side barchart that shows the empirical PMF of your data and the theoretic PMF according to the binomial distribution.

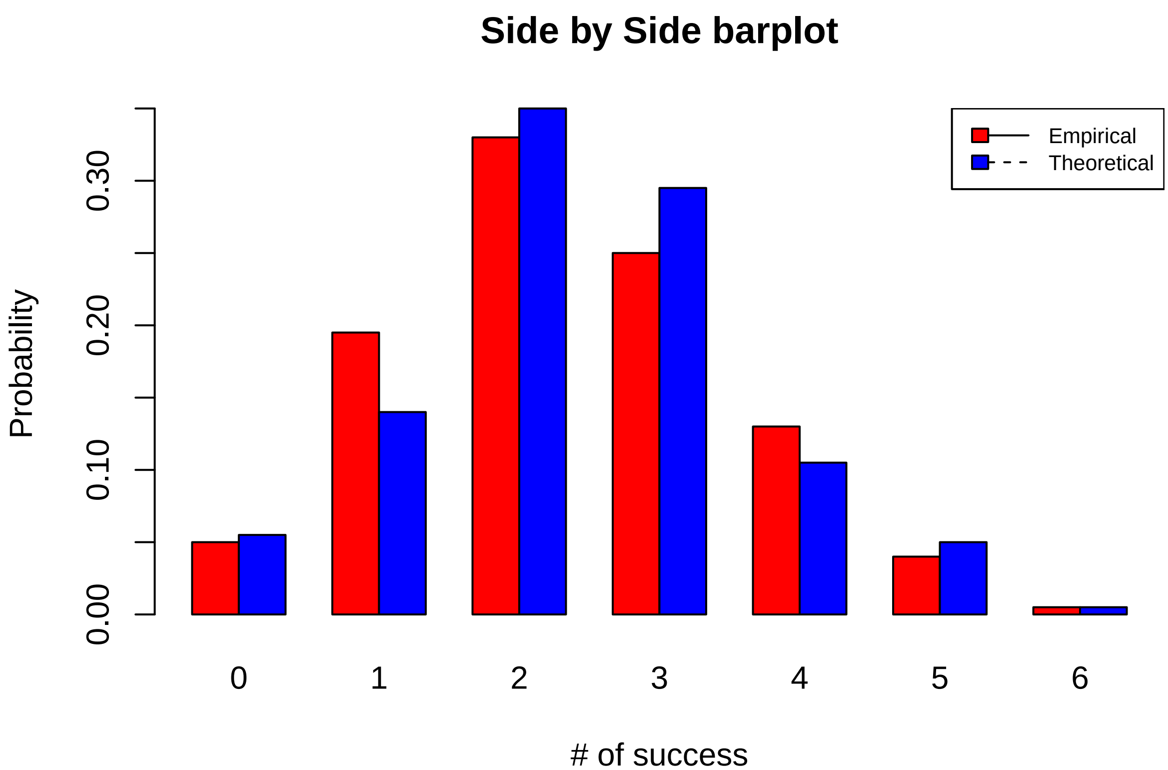
Use a chi-squared goodness-of-fit test to see if the generated values fit the expected probabilities. Be sure to comment on the graph and results of the test.

```r
# write your code here
# PDF
# p(x=0) = choose(6,0)*(0.4)^0*(0.6)^6 = 0.046656
# p(x=1) = choose(6,1)*(0.4)^1*(0.6)^5 = 0.186624
# p(x=2) = choose(6,2)*(0.4)^2*(0.6)^4 = 0.31104
# p(x=3) = choose(6,3)*(0.4)^3*(0.6)^3 = 0.276480
# p(x=4) = choose(6,4)*(0.4)^4*(0.6)^2 = 0.138240
# p(x=5) = choose(6,5)*(0.4)^5*(0.6)^1 = 0.036864
# p(x=6) = choose(6,6)*(0.4)^6*(0.6)^0 = 0.004096

p <- c(0.046656,0.186624,0.31104,0.276480,0.138240,0.036864,0.004096)
# CDF
(cdf <- cumsum(p))
```

```
## [1] 0.046656 0.233280 0.544320 0.820800 0.959040 0.995904 1.000000
```

```r
my_binom <- function(n,size,prob){
        c0 <- choose(6,0)*(prob)^0*(1-prob)^6
        c1 <- choose(6,1)*(prob)^1*(1-prob)^5
        c2 <- choose(6,2)*(prob)^2*(1-prob)^4
        c3 <- choose(6,3)*(prob)^3*(1-prob)^3
        c4 <- choose(6,4)*(prob)^4*(1-prob)^2
        c5 <- choose(6,5)*(prob)^5*(1-prob)^1
        c6 <- choose(6,6)*(prob)^6*(1-prob)^0
        cd <- c(c0,c1,c2,c3,c4,c5,c6)
        cumcd <- cumsum(cd)
        q <- c()
        ran <- runif(n, min = 0, max = 1)
        for(i in 1:n){
                if(cumcd[1] < ran[i] && ran[i] <= cumcd[2]){
                        q[i] = 1
                }else if (cumcd[2] < ran[i]  && ran[i] <= cumcd[3]){
                        q[i] = 2
                }else if (cumcd[3] < ran[i] && ran[i] <= cumcd[4]){
                        q[i] = 3
                }else if (cumcd[4] < ran[i] && ran[i] <= cumcd[5]){
                        q[i] = 4
                }else if (cumcd[5] < ran[i] && ran[i] <= cumcd[6]){
                        q[i] = 5
                }else if (cumcd[6] < ran[i] && ran[i] <= cumcd[7]){
                        q[i] = 6
                }else {
                        q[i] = 0
                }
        }
        return(q)
}
# random generationa and collapse 6 and 7
set.seed(1234)
my_samp <- my_binom(200,6,0.4)
th_prob <- rbinom(200, 6, 0.4)

# Side by side bar plot
my_data <- cbind(my_samp,"emp")
```

```
theo_data <- cbind(th_prob,"theo")
df123 <- rbind(my_data,theo_data)
df123 <- as.data.frame(df123)

data_tb <- table(df123$my_samp, df123$V2)
pro_data_tb <- data_tb/200
pro_data_tb <- matrix(unlist(t(pro_data_tb)), byrow=F, 2, 7)
colnames(pro_data_tb) <- 0:6
rownames(pro_data_tb) <- c("empirical","theoretical")

barplot(pro_data_tb,beside = T,
        xlab = "# of success",ylab = "Probability",col = c("red","blue"),main = "Side by Side barplot")
legend("topright", lty=1:2, cex=0.7,
       legend = c("Empirical", "Theoretical"), fill = c("red", "blue"))
```




```
# Chisq.test
(bar <- table(my_samp))
```

```
## my_samp
##  0  1  2  3  4  5  6
## 10 39 66 50 26  8  1
```

```
inti <- as.integer(bar)
inti[6] <- inti[6] + inti[7]
inti <- inti[-7]
inti <- as.table(inti)
names(inti) <- c("0","1","2","3","4","5 or 6")
(inti)
```

```
##      0      1      2      3      4 5 or 6
```

```
##     10     39     66     50     26      9
```

```r
prop_inti <- prop.table(inti)

tb <- table(th_prob)
inti2 <- as.integer(tb)
inti2[6] <- inti2[6] + inti2[7]
inti2 <- inti2[-7]
names(inti2) <- c("0","1","2","3","4","5 or 6")
(inti2)
```

```
##      0      1      2      3      4 5 or 6
##     11     28     70     59     21     11
```

```r
chisq.test(inti,p=prop.table(inti2))
```

```
##
##  Chi-squared test for given probabilities
##
## data:  inti
## X-squared = 7.5679, df = 5, p-value = 0.1817
```

```r
# p > 0.05
# p-value is large indicates that there is an insufficient evidence to reject the
# null hypothesis. We conclude that our sample data are consistent with a specified
# distribution.
```

Problem 4

Acceptance-rejection sampling

Let $f(x)$ and $g(x)$ be the target and candidate (proposal) distributions, respectively, in acceptance-rejection sampling. Find the optimal constant M that maximizes the acceptance rates for the following designs.

$f(x) = \frac{1}{2}\sin(x)$ for $0 \leq x \leq \pi$

$g(x) = \text{Unif}(0, \pi)$

Answer: M is pi/2

M * 1/pi = 1/2

M = pi/2

Implement the rejection sampling design, using `runif(n, 0, pi)` as your source of randomness. Generate 500 samples.

```r
M <- pi/2
set.seed(1234)
a <- runif(500,0,pi)
b <- runif(500,0,1)
# target
tar <- function(x){
        p <- c()
        for(i in 1:length(x)){
                p[i] <- 0.5 * sin(x[i])
        }
        return(p)
```

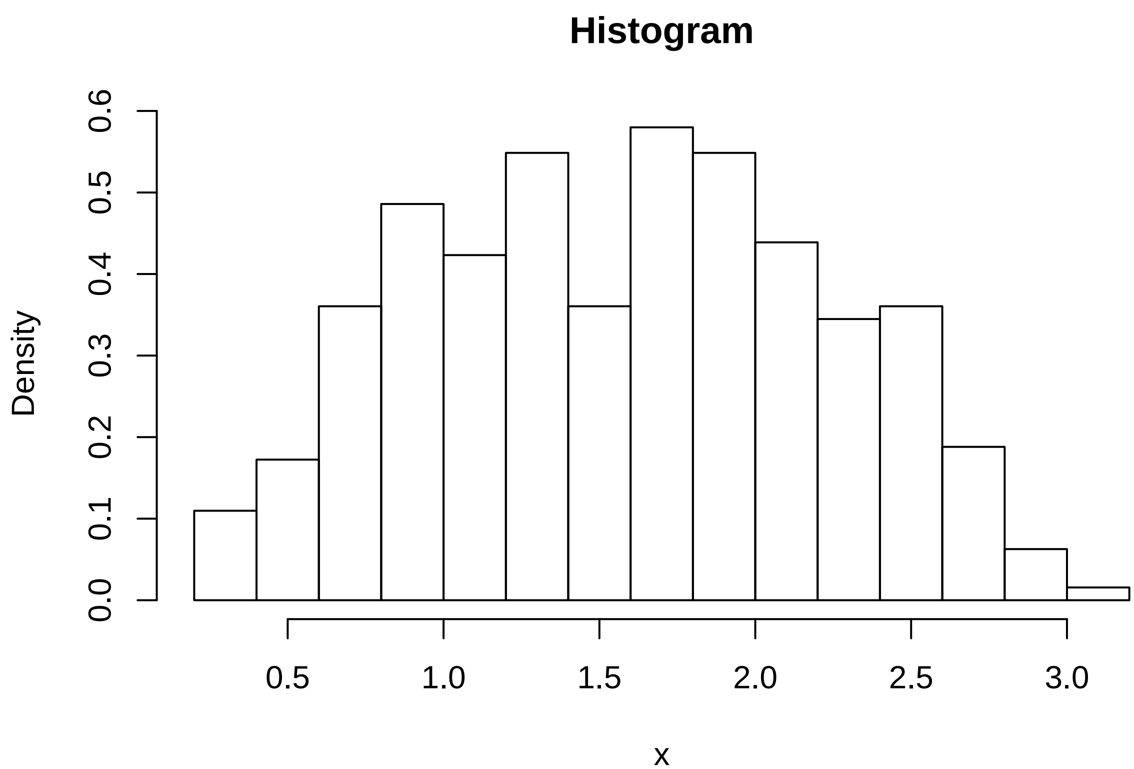
```r
}
# propose
pro <- rep(1/pi,500)

para <- function(x){
        return(tar(x)/(M*pro))
}

acc <- function(x){
        rati <- para(a)
        return(x <= rati)
}
c <- acc(b)
accepted <-  b[c]
length(accepted)
```

```
## [1] 319
```

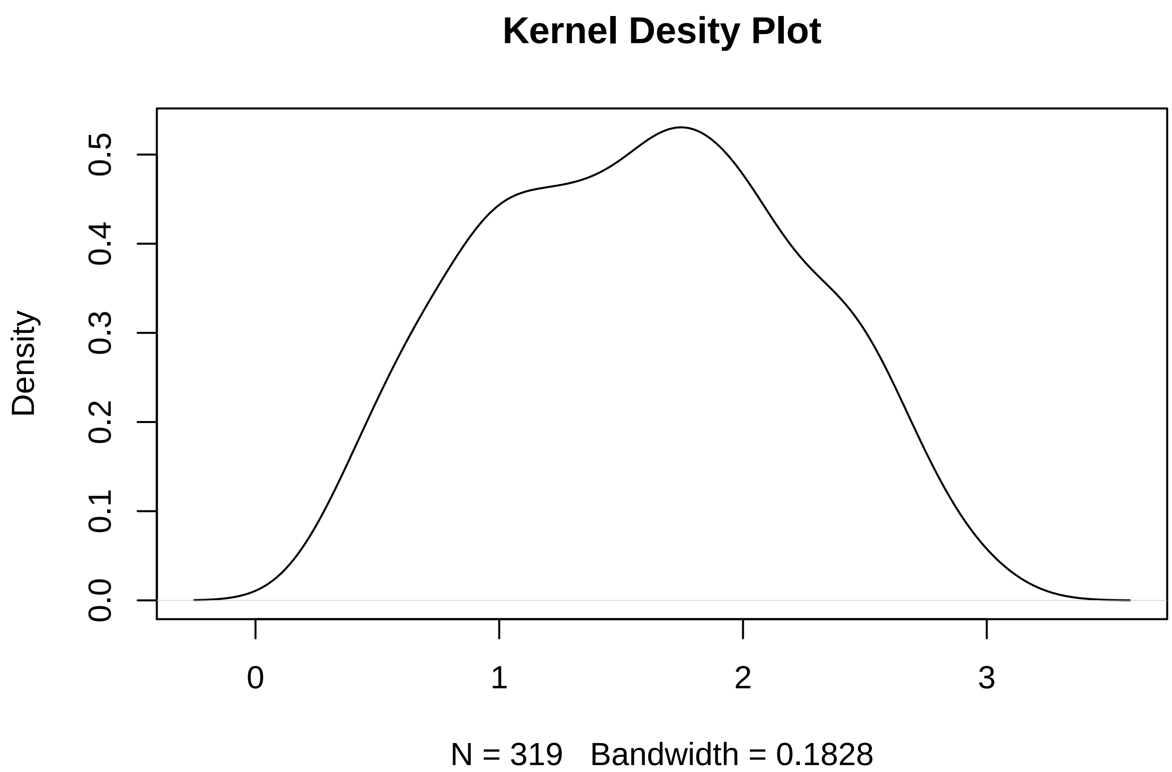
```r
(acceptance_rate <- 100*(length(accepted)/500))
```

```
## [1] 63.8
```

```r
hist(a[c],main = "Histogram",xlab = "x",freq = F)
```




```r
plot(density(a[c]),main = "Kernel Desity Plot")
```



What is your acceptance rate?

Create a histogram of your generated (accepted) sample.

Plot a kernel density of the resulting (accepted) sample.

Problem 5

Use rejection sampling to generate samples from the normal distribution, by using the folded-normal distribution method discussed in class.

The standard normal distribution has the pdf:

$$f(z) = \frac{1}{\sqrt{2\pi}} \exp\left(-z^2/2\right), \text{ for } z \in (-\infty, \infty)$$

The target distribution f(x) will be the positive half of the standard normal distribution, which will have PDF:

$$f(x) = 2 \times \frac{1}{\sqrt{2\pi}} \exp\left(-x^2/2\right), \text{ for } x \geq 0$$

Use an exponential distribution with lambda = 1 as your trial (proposal) distribution.

$$g(x) = e^{-x}, \text{ for } x \geq 0$$

Find the optimal constant M that maximizes the acceptance rates for the rejection sampling design.

Implement the rejection sampling design as discussed in class.

- Use `runif` and inverse CDF to get a proposal value X from the exponential distribution.

- Calculate the ratio: $\frac{f(X)}{M \times g(X)}$
- Use `runif` to generate U to decide whether to accept or reject the proposed X.
- keep the accepted X
- Use `runif` to generate S to decide whether the accepted X will be positive or negative with probably 0.5.

Use the above algorithm to generate a vector of 200 random values from the normal distribution.

Create a histogram of your generated sample.

Create a QQ-norm plot.

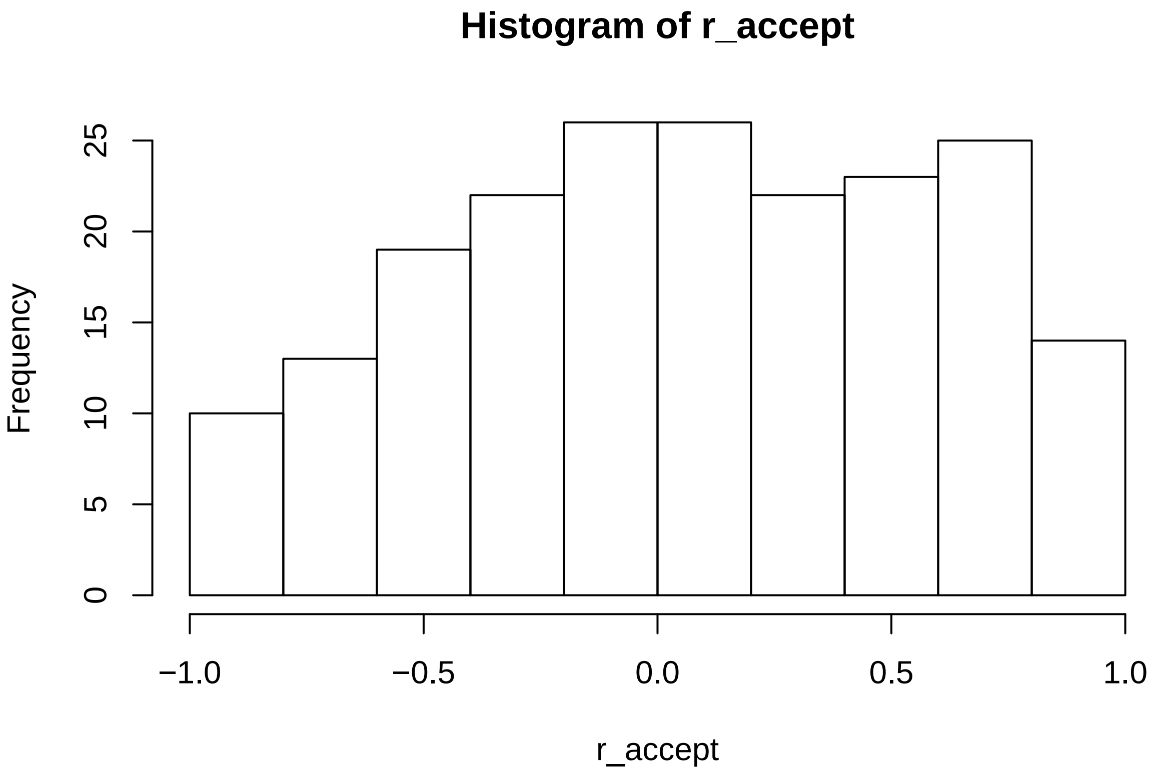
```
# Find the optimal constant M
set.seed(1234)
s <- seq(-0.05,3, by  = 0.001)
density <- function(x) ifelse(x < 0,0,2*(1/sqrt(2*pi))*exp(-((x^2)/2)))
propose <- function(x) ifelse(x < 0,0,exp(-x))

(M <- max(density(s)/propose(s),na.rm = TRUE))
```

```
## [1] 1.301802
```

```
rejec_s <- function(x){
        accept <- c()
        for(i in 1:x){
        u1 <- runif(1)
        j <- -log(u1)
        if(u1 <= (2*(1/sqrt(2*pi))*exp(-((j^2)/2)))/(M*j)){
                accept[i] <- j
        }else{
               accept[i] <- NA
        }
        }
        return(accept)
}
accepted <- rejec_s(500)
r_accept <- na.omit(accepted)
r_accept <- r_accept[1:200]
length(r_accept)
```

```
## [1] 200
```

```
s <- runif(length(r_accept),0,1)
s_minus <- s < 0.5
r_accept[s_minus] = - r_accept[s_minus]

hist(r_accept)
```




```r
qqnorm(r_accept)
```