HW4 - Intro to MCMC

Charles Liu (304804942)

Do not post your solutions online on a site like github. Violations will be reported to the Dean of Students.

Modify this file with your answers and responses.

Academic Integrity Statement

By including this statement, I, **Charles Liu**, declare that all of the work in this assignment is my own original work. At no time did I look at the code of other students nor did I search for code solutions online. I understand that plagiarism on any single part of this assignment will result in a 0 for the entire assignment and that I will be referred to the dean of students.

I did discuss ideas related to the homework with Josephine Bruin for parts 2 and 3, with John Wooden for part 2, and with Gene Block for part 5. At no point did I show another student my code, nor did I look at another student's code.

Reading:

Reading is important!

Doing Bayesian Data Analysis Textbook is available at: https://www.sciencedirect.com/science/book/9780124058880

- Read chapter 7 of Doing Bayesian Data Analysis
- http://setosa.io/ev/markov-chains/
- http://setosa.io/ev/eigenvectors-and-eigenvalues/ Especially the section on steady states (stationary distributions)

Problem 1 - Transition Matrix and Stationary Distribution (Two state case)

Imagine a two-state Markov chain. With state 1 representing CA and state 2 representing TX.

Let's pretend that each year, 9% of Californians move to TX and that 12% of Texans move to CA.

Create and display a 2x2 transition matrix $\mathbb{P}$ in R to represent the transition probabilities.

Using algebra, find the stationary distribution $\boldsymbol{\pi}$, so that $\boldsymbol{\pi}\mathbb{P} = \boldsymbol{\pi}$.

```
# write your code here
P <- rbind(c(0.91, 0.09),
           c(0.12, 0.88))
```

```r
# Finding the stationary distribution
n <- 10^5
X <- c(1, rep(NA, n-1))
set.seed(1)
for(i in 2:n){
  row <- X[i - 1] + 1
  X[i] <- sample(c(0, 1), size = 1, prob = P[row, ])
}
# Proportion of steps in state 0
state_0 <- sum(X == 0) / n
# Proportion of steps in state 1
state_1 <- sum(X == 1) / n

# Create the stationary distribution and see if it is equal to itself with P
w <- matrix(c(state_0, state_1))
w_P <- t(w) %*% P

# See if the results match
matching <- rbind(c(round(w,3), round(w_P,3))) # rounded up to third place for approximation
colnames(matching) <- c("State 0", "State 1", "State 0 w/ P", "State 1 /w P")
matching # we found the stationary distribution that matches
```

```
##      State 0 State 1 State 0 w/ P State 1 /w P
## [1,]   0.575   0.425        0.575        0.425
```

Find the left eigenvector of $\mathbb{P}$ and normalize it (so it sums to 1). Does it match the stationary distribution you found?

```r
eig_vectors <- eigen(t(P))$vectors
eig_vectors
```

```
##      [,1]       [,2]
## [1,]  0.8 -0.7071068
## [2,]  0.6  0.7071068
```

```r
stationary <- t(eig_vectors[,1])
target_distr <- stationary / (sum(stationary))
target_distr <- round(target_distr, 2)
colnames(target_distr) <- c("State 0", "State 1")
target_distr # they match! (approximately)
```

```
##      State 0 State 1
## [1,]    0.57    0.43
```

Problem 2 - Transition Matrix and Stationary Distribution (island example)

Look at the example with the politician visiting the island chain in chapter 7 of the textbook, Doing Bayesian Data Analysis. Also see Lecture 5-3.

Imagine another chain of 7 islands where the target distribution is equal to the probabilities of a binomial distribution with n = 6 and p = 0.6.

This 'nation' has 7 islands in a chain numbered from 0 to 6. Island 0 has prob = $\binom{6}{0}(.6)^0(.4)^6$ = `dbinom(0, 6, 0.6)`, Island 1 has prob = $\binom{6}{1}(.6)^1(.4)^5$ = `dbinom(1, 6, 0.6)`, Island 6 has prob = `dbinom(6, 6, 0.6)`, etc.

Use the same algorithm as the politician to figure out the transition probabilities. Create and print out the full 7 x 7 transition matrix $\mathbb{P}$. Populate the matrix with actual decimal values, and not symbols (round to 4 decimal places for display purposes).

Start with the initial distribution: $\boldsymbol{\pi}^{(1)}$ = c(0, 0, 0, 1, 0, 0, 0)

Multiply $\boldsymbol{\pi}^{(n)}$ by $\mathbb{P}$ 6 times and print the results after each iteration. (Print the distribution of $\boldsymbol{\pi}^{(2)}$, $\boldsymbol{\pi}^{(3)}$, ... $\boldsymbol{\pi}^{(7)}$)

Find the stationary distribution of the chain by finding the left eigenvector of the transition matrix and normalizing it. Check (using `all.equal()`) to see if it is equal to the target distribution (a binomial distribution with $n = 6$ and $p = 0.6$)

```
# Set up islands' probabilities
island_0 <- dbinom(0, 6, 0.6)
island_1 <- dbinom(1, 6, 0.6)
island_2 <- dbinom(2, 6, 0.6)
island_3 <- dbinom(3, 6, 0.6)
island_4 <- dbinom(4, 6, 0.6)
island_5 <- dbinom(5, 6, 0.6)
island_6 <- dbinom(6, 6, 0.6)
islands <- matrix(c(island_0, island_1, island_2, island_3,
            island_4, island_5, island_6), ncol = 7)
islands <- round(islands, 4)
colnames(islands) <- c("Island 0", "Island 1", "Island 2",
                       "Island 3", "Island 4", "Island 5",
                       "Island 6")
islands # target distribution
```

```
##      Island 0 Island 1 Island 2 Island 3 Island 4 Island 5 Island 6
## [1,]   0.0041   0.0369   0.1382   0.2765    0.311   0.1866   0.0467
```

```
P <- rbind(c(0.5, 0.5, 0, 0, 0, 0, 0), # 0
           c(0.0556, 0.4444, 0.5, 0, 0, 0, 0), # 1
           c(0, 0.1335, 0.3665, 0.5, 0, 0, 0), # 2
           c(0, 0, 0.2499, 0.2501, 0.5, 0, 0), # 3
           c(0, 0, 0, 0.4445, 0.0555, 0.5, 0), # 4
           c(0, 0, 0, 0, 0.5, 0.3749, 0.1251), # 5
           c(0, 0, 0, 0, 0, 0.5, 0.5)) # 6
P
```

```
##        [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]
## [1,] 0.5000 0.5000 0.0000 0.0000 0.0000 0.0000 0.0000
## [2,] 0.0556 0.4444 0.5000 0.0000 0.0000 0.0000 0.0000
## [3,] 0.0000 0.1335 0.3665 0.5000 0.0000 0.0000 0.0000
## [4,] 0.0000 0.0000 0.2499 0.2501 0.5000 0.0000 0.0000
## [5,] 0.0000 0.0000 0.0000 0.4445 0.0555 0.5000 0.0000
## [6,] 0.0000 0.0000 0.0000 0.0000 0.5000 0.3749 0.1251
## [7,] 0.0000 0.0000 0.0000 0.0000 0.0000 0.5000 0.5000
```

```
r_eigen <- eigen(t(P))$vectors[ ,1] # right eigenvector of P^T
stationary <- t(r_eigen) / sum(r_eigen) # normalize the transpose
stationary # the stationary distribution found by finding the left eigenvector
```

```
##              [,1]      [,2]     [,3]      [,4]      [,5]      [,6]       [,7]
## [1,] 0.003550704 0.0319308 0.119591 0.2392777 0.2691538 0.2691538 0.06734227
```

```
stationary %*% P # stationary dist times P remains the same
```

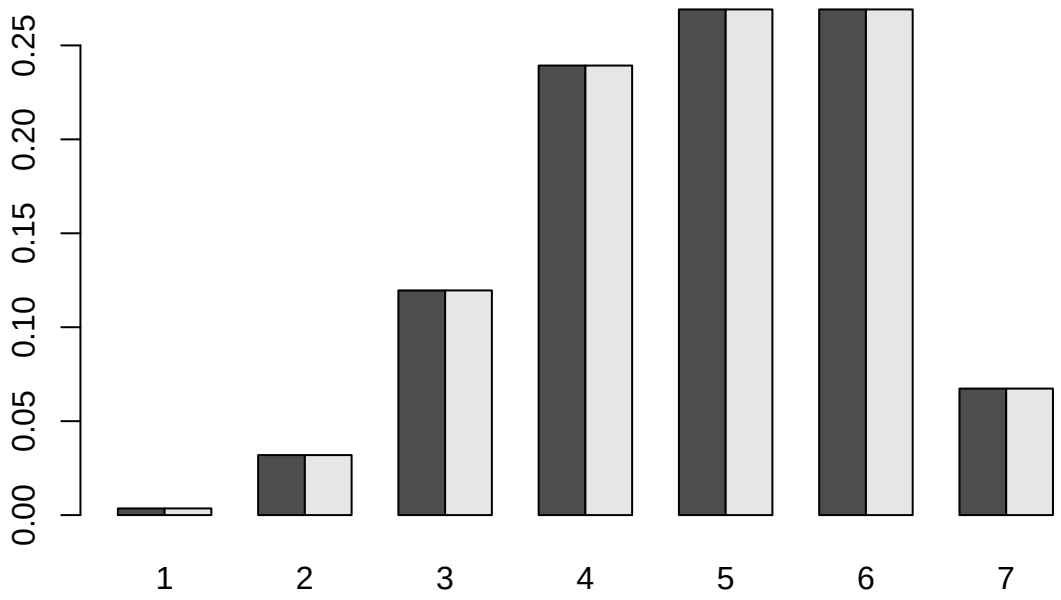
```
##              [,1]      [,2]     [,3]      [,4]      [,5]      [,6]       [,7]
## [1,] 0.003550704 0.0319308 0.119591 0.2392777 0.2691538 0.2691538 0.06734227
```

```
target <- matrix(dbinom(0:6, 6, 0.6), nrow = 1)
all.equal(stationary, target) # NOT EQUAL! but is close
```

```
## [1] "Mean relative difference: 0.2064321"
```

Multiply $\boldsymbol{\pi}^{(1)}$ by $\mathbb{P}$ 500 times to get $\boldsymbol{\pi}^{(501)}$. Show the results after the final iteration. Do NOT show the steps in between. Did the distribution converge to the stationary distribution?

```
pi_n <- matrix(rep(0, 7*500), nrow = 500)
colnames(pi_n) <- 1:7
pi_n[1, ] <- c(0, 0, 0, 1, 0, 0, 0)
pi_n[2, ] <- pi_n[1, ] %*% P
pi_n[3, ] <- pi_n[2, ] %*% P
pi_n[4, ] <- pi_n[3, ] %*% P
pi_n[5, ] <- pi_n[4, ] %*% P
pi_n[6, ] <- pi_n[5, ] %*% P
pi_n[7, ] <- pi_n[6, ] %*% P


for (i in 8:500) {
  pi_n[i, ] <- pi_n[i-1, ] %*% P
}

barplot(pi_n[499:500,], beside = TRUE)
```




```
all.equal(pi_n[499,], pi_n[500,])
```

```
## [1] TRUE
```

Problem 3 - MCMC (Metropolis Algorithm) for the island hopping

Write code to create a Markov chain using the Metropolis Algorithm for the same island nation in problem 2.

Run the Metropolis Algorithm to create two Markov chains, each of length 10^5. For the first chain, start at x = 0 and use `set.seed(1)`. For the second, start at x = 6 and use `set.seed(2)`.

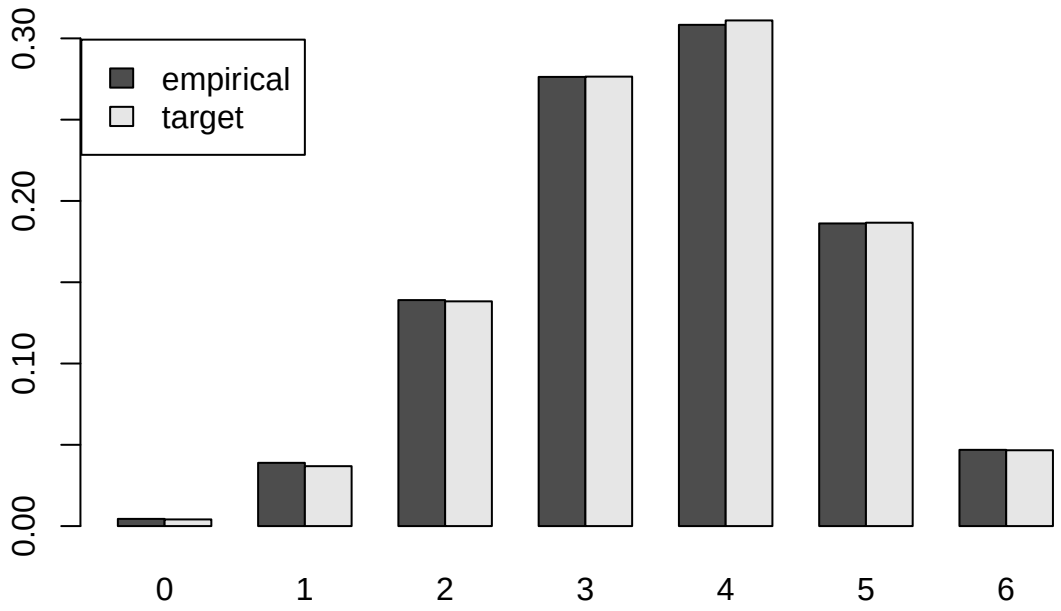
For each completed chain, print out a table of the resulting relative frequencies. Make a side-by-side barchart that shows the empirical PMF of your data and the theoretic PMF according to the binomial distribution. Use a chi-squared goodness-of-fit test to see if the generated values fit the expected probabilities. Be sure to comment on the graph and results of the test.

```
# write a function that returns the probability of the target distribution
target <- function(x){
  ifelse(x %in% 0:6, dbinom(x, 6, 0.6), 0)
}

# write a function that proposes a new value of x given the current value of x
propose <- function(x) {
  x + sample(c(1, -1), size = 1)
```

```
}


## implement the metropolis algorithm
### set.seed(1) & start at 0
results <- rep(NA, 10^5)
n <- 10^5
x <- c(0, rep(NA, n-1)) # start at 0
set.seed(1)
for (t in 1:(n-1)) {
  current <- x[t]
  proposed <- propose(current)
  pmove <- target(proposed) / target(current)
# pmove is min(1, p(propose)/p(current)), but if pmove > 1, random U will be less than it anyway
  u <- runif(1)
  if(u < pmove) {
    x[t + 1] <- proposed
  } else {
    x[t + 1] <- current
  }
}
counts <- table(x)
results <- rbind( counts/n , dbinom(0:6, 6, 0.6) )
rownames(results) <- c("empirical", "target")
round(results, 4)
```

```
##                0      1      2      3      4      5      6
## empirical 0.0044 0.0389 0.1390 0.2763 0.3083 0.1862 0.0469
## target    0.0041 0.0369 0.1382 0.2765 0.3110 0.1866 0.0467
```

```
barplot(results, beside = TRUE, legend.text = row.names(results), args.legend = list(x = 5))
```




```
chisq.test(counts, p = dbinom(0:6, 6, 0.6) )
```

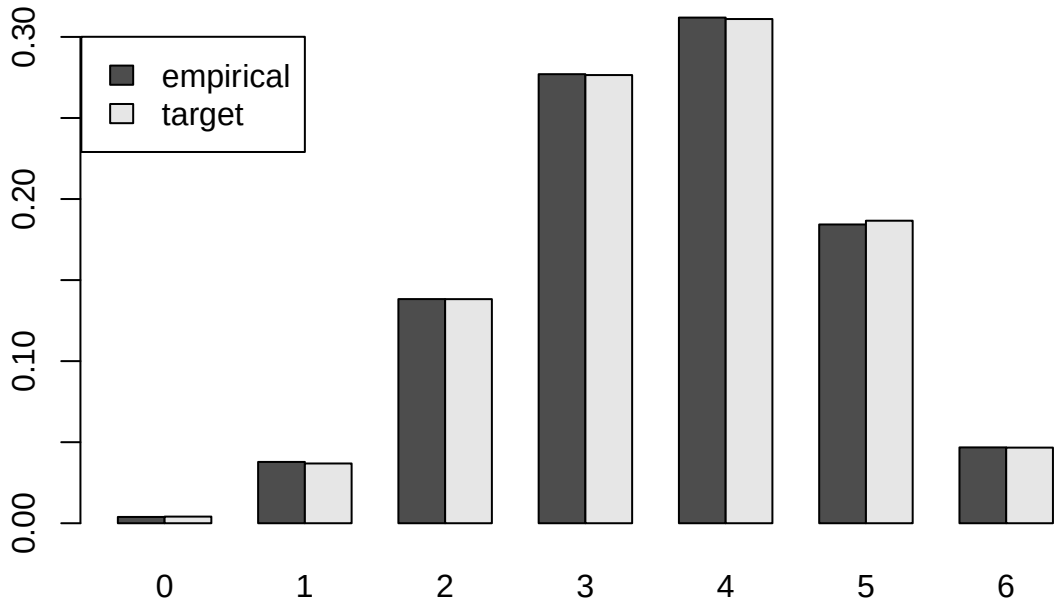
```
##
##  Chi-squared test for given probabilities
##
## data:  counts
## X-squared = 17.037, df = 6, p-value = 0.009149
```

```
### set.seed(2) & start at 6
results <- rep(NA, 10^5)
n <- 10^5
x <- c(6, rep(NA, n-1)) # start at 6
set.seed(2)
for (t in 1:(n-1)) {
  current <- x[t]
  proposed <- propose(current)
  pmove <- target(proposed) / target(current)
# pmove is min(1, p(propose)/p(current)), but if pmove > 1, random U will be less than it anyway
  u <- runif(1)
  if(u < pmove) {
    x[t + 1] <- proposed
  } else {
    x[t + 1] <- current
  }
}
```

```r
counts <- table(x)
results <- rbind( counts/n , dbinom(0:6, 6, 0.6) )
rownames(results) <- c("empirical", "target")
round(results, 4)
```

```
##                0      1      2      3      4      5      6
## empirical 0.0039 0.0378 0.1383 0.2770 0.3119 0.1843 0.0468
## target    0.0041 0.0369 0.1382 0.2765 0.3110 0.1866 0.0467
```

```r
barplot(results, beside = TRUE, legend.text = row.names(results), args.legend = list(x = 5))
```




```r
chisq.test(counts, p = dbinom(0:6, 6, 0.6) )
```

```
##
##  Chi-squared test for given probabilities
##
## data:  counts
## X-squared = 6.9061, df = 6, p-value = 0.3296
```

COMMENTS: We see that when we started with x = 0, the the p-value is less than 0.05. However, we can see that the Empirical and Target are very close to each other. This is because the chi-square is sensitive to large sample sizes (in this case, it is over $n = 10^5$). This is okay and can conclude that we do not have enough evidence to say that the values generated by the Markov Chain do not come from the target distribution.

As for x = 6, the p-value is much greater than 0.05. The Empirical and Target distributions align with each other approximately. This tells us that we do not have enough evidence to say that the values generated by the Markov Chain do not come from the target distribution.

Problem 4 - MCMC (Metropolis Algorithm) for a single continuous random variable

The logisitic distribution is a unimodal and symmetric distribution, where the CDF is a logistic curve. The shape is similar to a normal distribution, but has heavier tails (though not as heavy as a Cauchy distribution).

We will compare Rejection Sampling to the Metropolis Algorithm for producing a sample from a distribution.

The PDF is:

$$f(x;\mu,s) = \frac{1}{s}\frac{e^{-(\frac{x-\mu}{s})}}{\left(1+e^{-(\frac{x-\mu}{s})}\right)^2}$$

Luckily, this is implemented for us in R with `dlogis()`, which you are allowed to use to calculate the probability density of a (proposed) value.

We will generate two samples drawn from a logistic distribution with mean = 0 and scale = 1.

$$f(x;\mu=0,s=1) = \frac{e^{-x}}{(1+e^{-x})^2} = \texttt{dlogis(x)}$$

Task 4A:

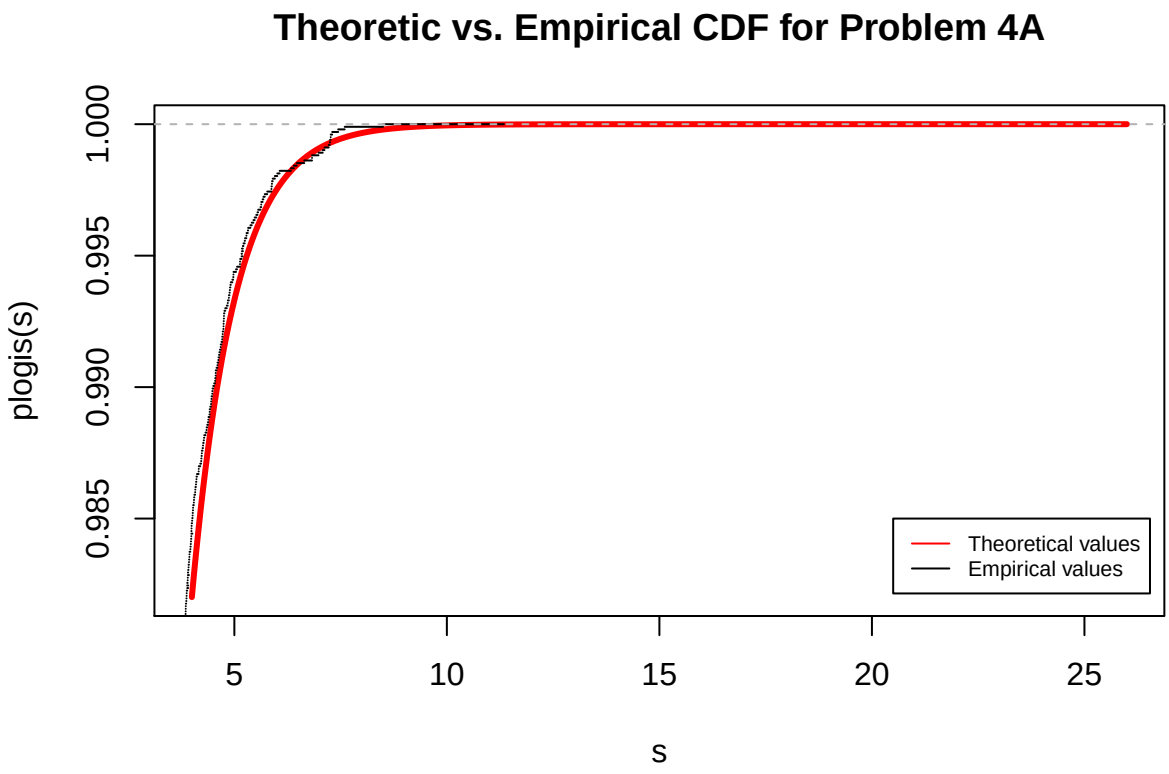
First generate a sample from the logistic distribution using rejection sampling. Propose 10^5 values from a random uniform distribution from -20 to 20. Calculate the necessary constant M, and implement rejection sampling. If you propose 10^5 values, how many values do you end up accepting?

Plot the theoretic CDF of the distribution. Add the empirical CDF of your accepted values to the same plot (in a different color). Use the Kolmogorov-Smirnov test to compare your generated samples to the theoretic distributions.

```
# generate 10^5 proposed x values from the uniform distribution from -20 to +20
n <- 10^5
x <- runif(n, min = -20, max = 20)

# calculate the acceptance ratio f(x)/(M * g(x))
acceptance_ratio <- function(x){
  dlogis(x)/(10 * (1/40)) # M = 10
}

# generate u values to decide to accept or not
U <- runif(n)
r_x <- acceptance_ratio(x)

# create the sample of accepted values
accepted_values <- x[U < r_x]

# how many values did you accept?
length(accepted_values)
```

```
## [1] 10146
```

```r
# plot the empirical cdf vs the theoretic cdf. you can use plogis
s <- seq(4, 26, by = .01)
plot(s, plogis(s), col = "red", lwd = 3, type = "l",
     main = "Theoretic vs. Empirical CDF for Problem 4A")
plot(ecdf(accepted_values), add = TRUE)
legend(20.5, 0.985, legend = c("Theoretical values", "Empirical values"),
       col = c("red", "black"),
       lty = c(1, 1), cex = 0.65)
```




```r
# K-S test to compare generated samples to theoretic distribution
ks.test(accepted_values, plogis)
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  accepted_values
## D = 0.006999, p-value = 0.7029
## alternative hypothesis: two-sided
```

ANSWER: We accepted 10,195 values out of the 100,000 of x. From the K-S Test, the p-value is much greater than 0.05. Therefore, we can conclude that we have no reason to doubt that these values came from an logistic distribution.

Task 4B:

Use the Metropolis algorithm to generate values from the logisitic distribution.

For your proposal distribution, use a random uniform distribution that draws a random value between $X_{current} - 1$ and $X_{current} + 1$.

As a reminder, the steps of the algorithm are as follows:

- Propose a single value from the proposal distribution.
- Calculate the probability of moving = min(1, P(proposed)/P(current))
- Draw a random value to decide if you will move or not. If you move, update the current position. If you do not move, keep the current position for another iteration.
- Repeat.

Start at the terrible location $x^{(1)} = -19$.

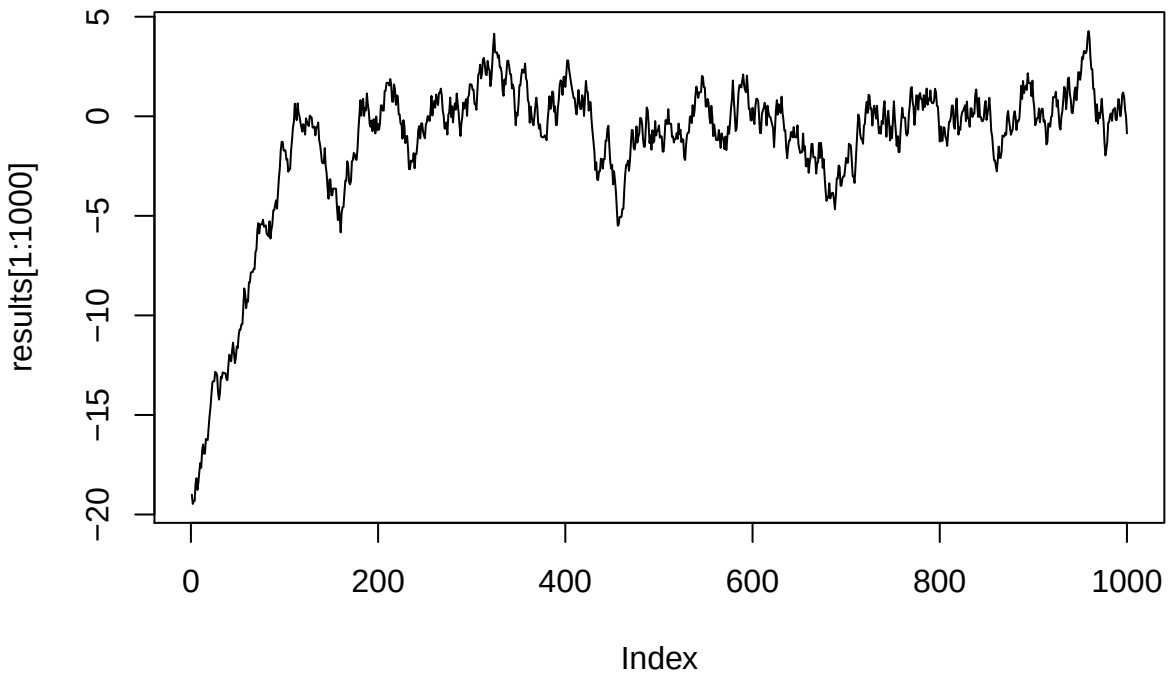
Run the Markov Chain for 10,000 iterations. Plot the first 1000 values of the chain and eyeball where you think the chain starts has finished 'burning-in' and is now drawing values from the target distribution. Throw away those initial values.

Plot a density histogram of the remaining values and add the density of the logistic distribution to the histogram.

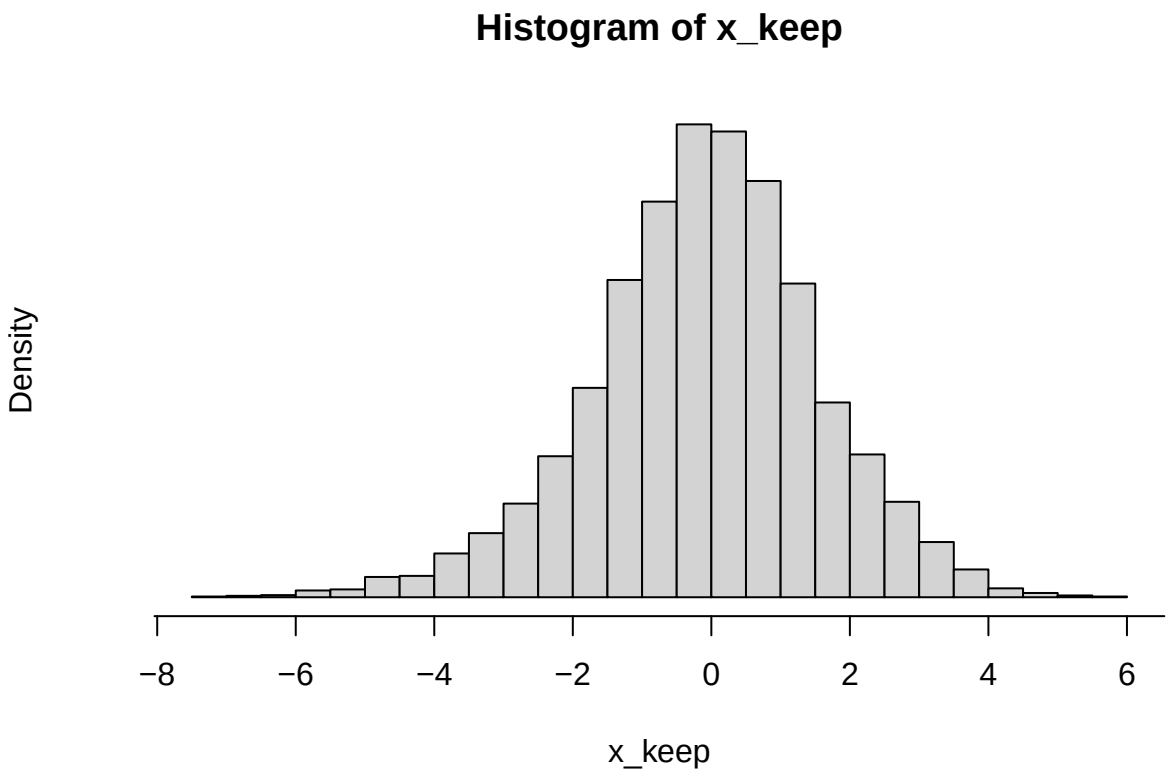
Plot the theoretic CDF of the distribution. Add the empirical CDF of your values (after removing burn-in) to the same plot. Use the Kolmogorov-Smirnov test to compare your generated samples to the theoretic distributions.

```
target <- function(x) {
  dlogis(x)
}

# write a function that will return a proposed value given the current value
# the proposed value comes from the uniform distribution with
# low = current - 1, to high = current + 1
propose <- function(x) {
  runif(1, min = (x - 1), max = (x + 1))
}

results <- rep(NA, 10^4)
results[1] <- -19  # we start at a terrible location

# write a loop to perform MCMC
set.seed(1)
for(i in 1:10^4){
  current <- results[i]
  # propose a new value based on the current value
  proposed <- propose(current)

  # calculate the probability of moving
  p_move <- target(proposed) / target(current)

  # decide to accept or reject the proposed value
  U <- runif(1)

  # store the appropriate value in results[i]
```

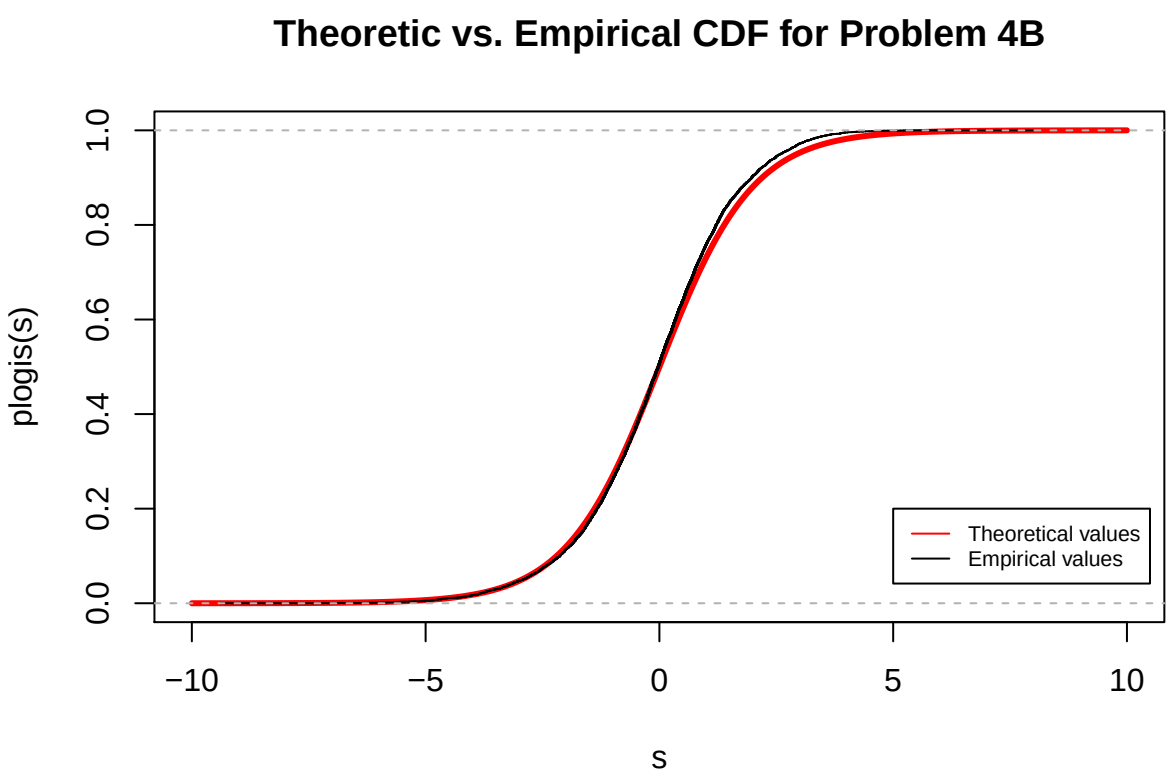
```
  if(U < p_move){
    x_new <- proposed
  } else {
    x_new <- current
  }
  results[i + 1] <- x_new
}

# create a plot of the first 1000 values in the chain.
plot(results[1:1000], type = "l")
```




```
# Eyeball guess where the values seem to have reached a 'good' area.
# The 'bad' area values are the burn-in values

# only keep the values after the 'burn in' period.
x_keep <- results[-(1:100)]

# Plot a histogram of the remaining values, and add the density
hist(x_keep, breaks = 30, freq = FALSE, axes = FALSE)
axis(1, at = seq(-10, 10, by = 2))
```




```
# plot the empirical CDF (of the kept values) versus the theoretic CDF
s <- seq(-10, 10, by = 0.01)
plot(s, plogis(s), col = "red", lwd = 3, type = "l",
     main = "Theoretic vs. Empirical CDF for Problem 4B")
plot(ecdf(x_keep), add = TRUE)
legend(5, 0.20, legend = c("Theoretical values", "Empirical values"),
       col = c("red", "black"),
       lty = c(1, 1), cex = 0.65)
```




```
# K-S Test
ks.test(x_keep, plogis)
```

```
## Warning in ks.test(x_keep, plogis): ties should not be present for the
## Kolmogorov-Smirnov test

##
##  One-sample Kolmogorov-Smirnov test
##
## data:  x_keep
## D = 0.033664, p-value = 3.588e-10
## alternative hypothesis: two-sided
```

COMMENTS: For the K-S Test, we see that the p-value is less than 0.05, however from the plots and numerical comparisons, we can clearly see that the Empirical and Theoretical values match approximately. The K-S Test's p-value is lower because we only considered the first 1000 values and burned away other values. We burned values 1 through 100 and kept the rest. Thus, we had 900 values to work from.

Problem 5 - MCMC - the effect of sigma in the proposal distribution

Write code to perform 50,000 iterations of the Metropolis Algorithm for a single continuous random variable.

Let the PDF of the target distribution be:

$$f(x) = c \cdot (sin(x) + 2)$$

for $0 \leq x \leq 3 * \pi$, where c is some constant so that $\int_0^{3\pi} f(x)dx = 1$.

For your proposal distribution, use a normal distribution, centered at the current value, with a standard deviation of σ, which we will adjust in this problem.

Begin your Markov Chain at the location x = 2.

Keep in mind that the probability of a value greater than 3π or less than 0 is 0.

Gather 50,000 samples using MCMC **three** different times.

The first time, use a sigma of 0.1 for the proposal distribution.

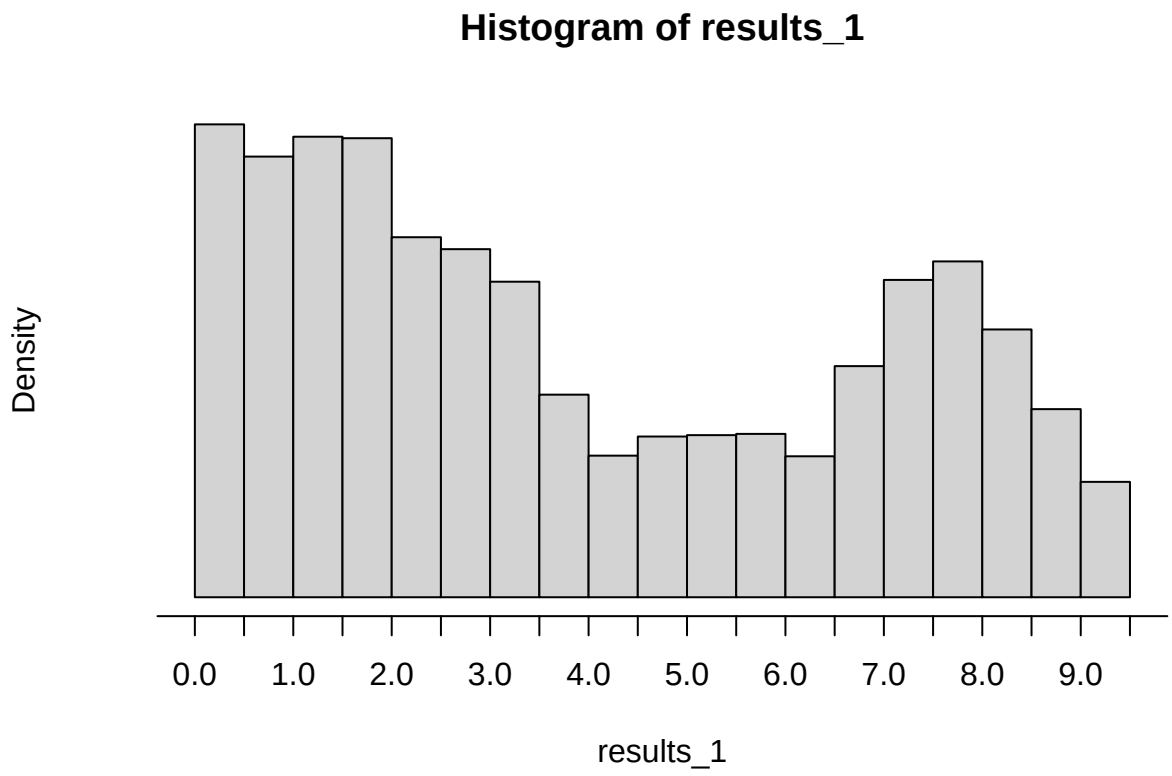
The second time, use a sigma of 2.5 for the proposal distribution.

The third time, use a sigma = 20.

Keep track of whether your proposed values are accepted or rejected, and print out the acceptance ratio.

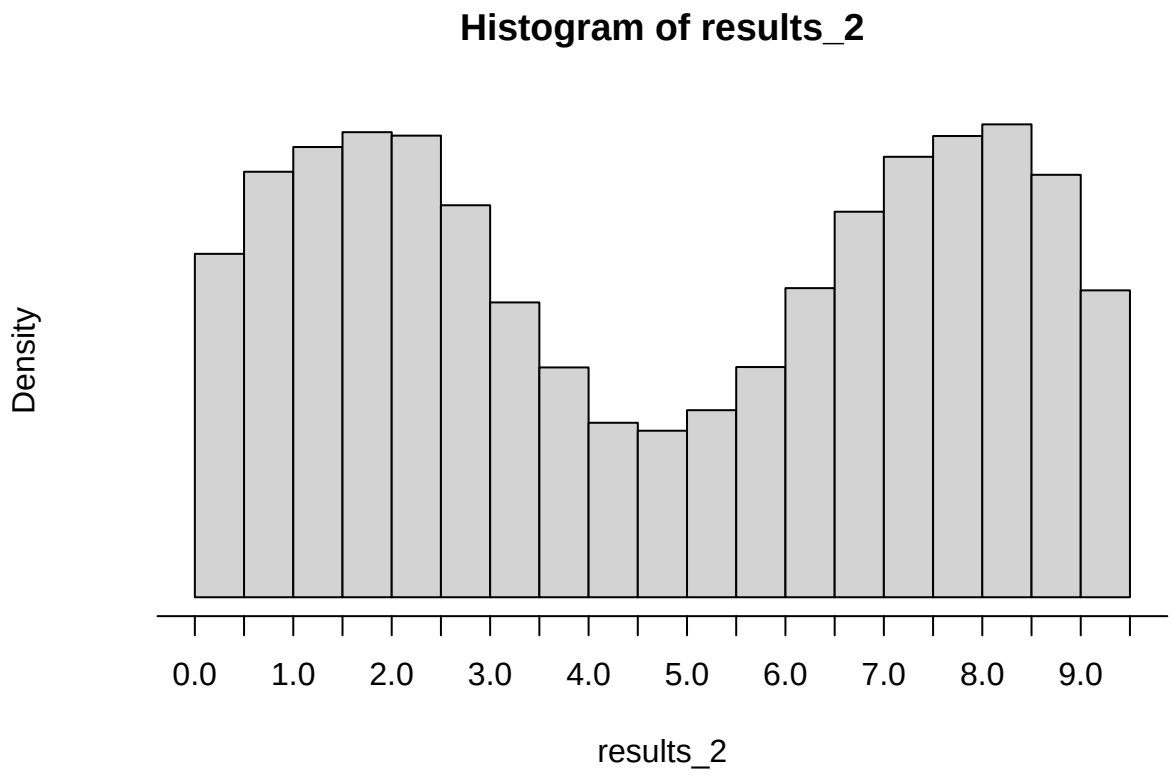
For each MCMC run, print out the acceptance ratio, create a histogram of the sampled values, and plot the first 500 values of the chain `plot(x[1:500], type = "l")`.

```
# Write a function for the PDF. Try to make it a vectorized function by using
# the ifelse() conditional statement rather than if()
f <- function(x) {
  ifelse(x >= 0 & x <= (3*pi), (sin(x) + 2), 0)
}

propose <- function(x, Sigma) {
  rnorm(1, mean = x, sd = Sigma)
}

n <- 5*10^4
# Sigma = 0.1
results <- rep(NA, n)
results[1] <- 2  # we start at x = 2
Sigma <- 0.1
# write a loop to perform MCMC
set.seed(1)
for(i in 1:(n - 1)){
  current <- results[i]
  proposed <- propose(current, Sigma = Sigma)
  p_move <- f(proposed) / f(current)
  U <- runif(1)
  if(U < p_move){
    x_new <- proposed
  } else {
    x_new <- current
  }
  results[i + 1] <- x_new
}
results_1 <- results
hist(results_1, breaks = 30, freq = FALSE, axes = FALSE)
axis(1, at = seq(-10, 10, by = 0.5))
```

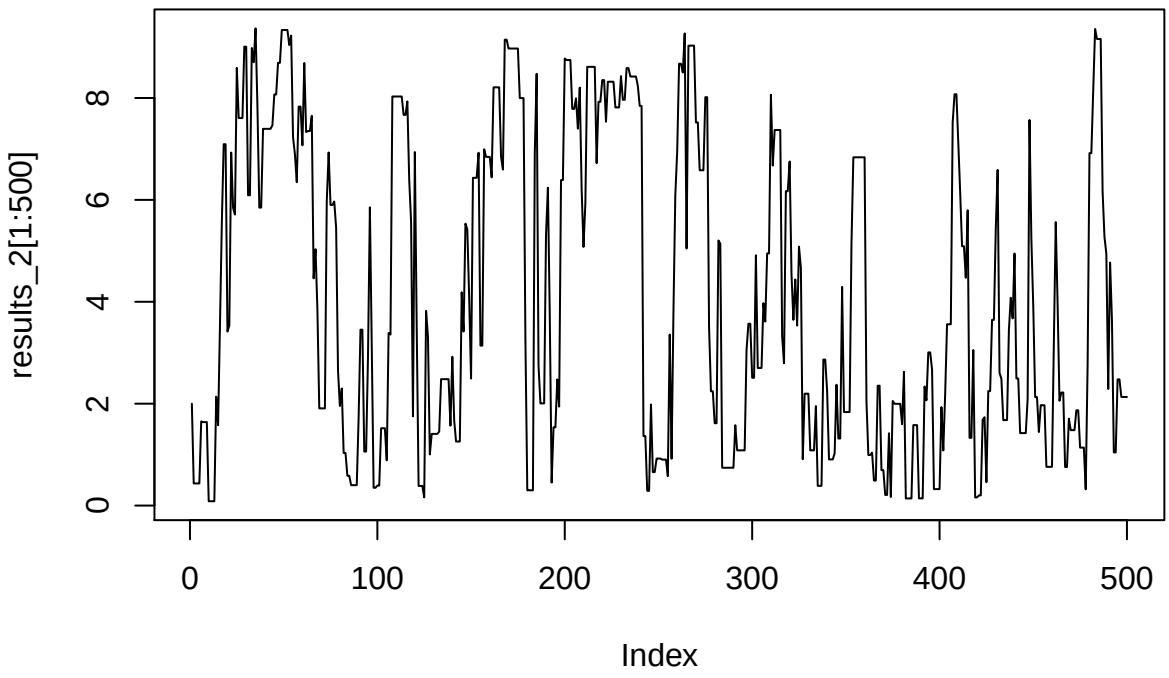



```
plot(results_1[1:500], type = "l")
```

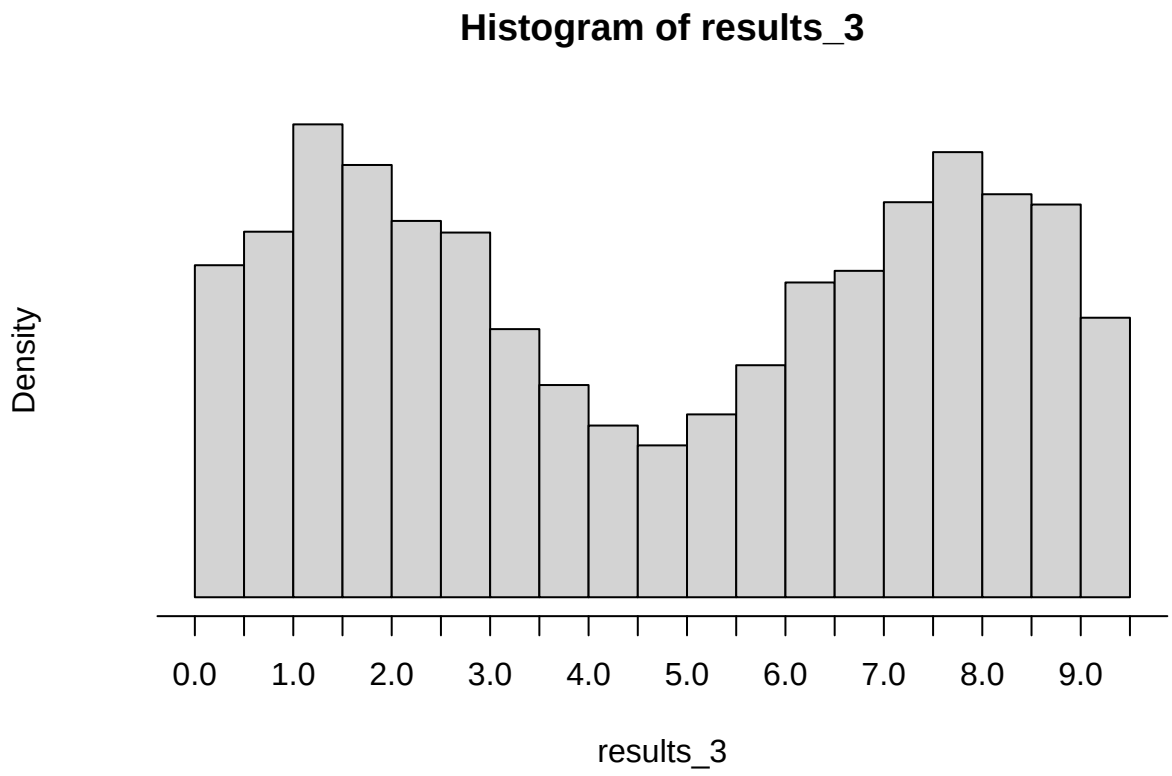

```r
# Sigma = 2.5
results <- rep(NA, n)
results[1] <- 2  # we start at x = 2
Sigma <- 2.5
# write a loop to perform MCMC
set.seed(1)
for(i in 1:(n - 1)){
  current <- results[i]
  proposed <- propose(current, Sigma = Sigma)
  p_move <- f(proposed) / f(current)
  U <- runif(1)
  if(U < p_move){
    x_new <- proposed
  } else {
    x_new <- current
  }
  results[i + 1] <- x_new
}
results_2 <- results
hist(results_2, breaks = 30, freq = FALSE, axes = FALSE)
axis(1, at = seq(-10, 10, by = 0.5))
```



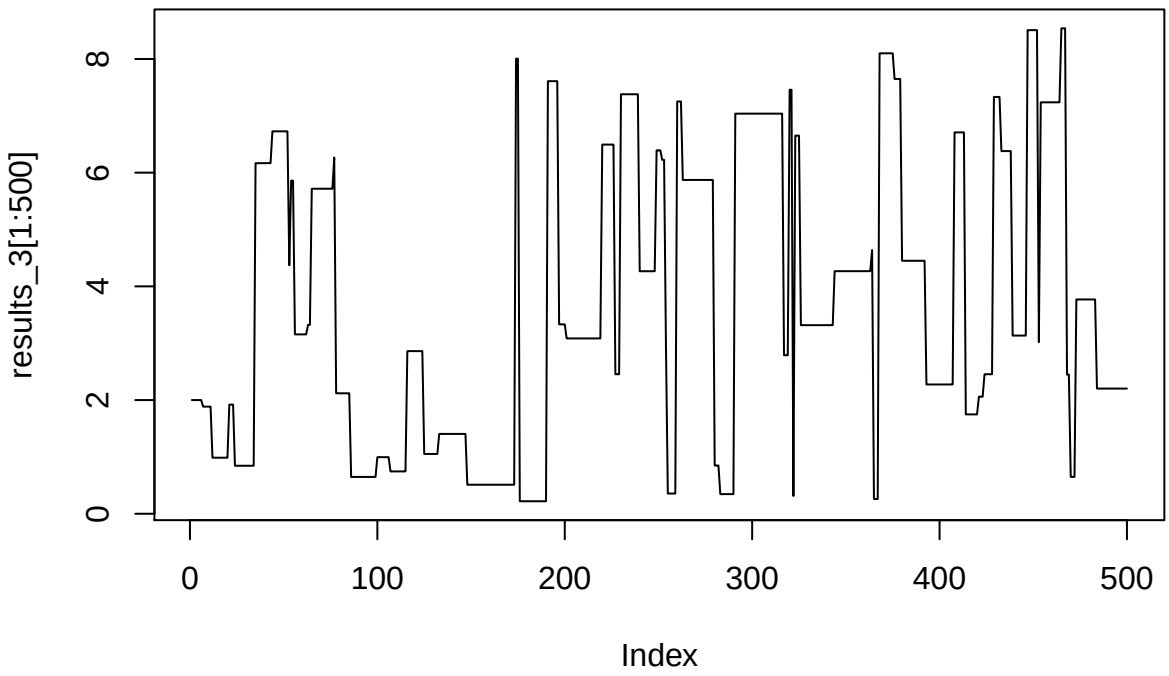

```
plot(results_2[1:500], type = "l")
```




```r
# Sigma = 20
results <- rep(NA, n)
results[1] <- 2  # we start at x = 2
Sigma <- 20
# write a loop to perform MCMC
set.seed(1)
for(i in 1:(n - 1)){
  current <- results[i]
  proposed <- propose(current, Sigma = Sigma)
  p_move <- f(proposed) / f(current)
  U <- runif(1)
  if(U < p_move){
    x_new <- proposed
  } else {
    x_new <- current
  }
  results[i + 1] <- x_new
}
results_3 <- results
hist(results_3, breaks = 30, freq = FALSE, axes = FALSE)
axis(1, at = seq(-10, 10, by = 0.5))
```




```
plot(results_3[1:500], type = "l")
```


results_3[1:500]
0
2
4
6
8
0
100
200
300
400
500
Index