Math 490 Final Exam

Maxwell Levin

May 3, 2018

Problem 1.

Place a king on a 3x3 empty chessboard. The position of the king is denoted by ordered pairs (x,y), where $x, y = 1, 2, 30$. Suppose the king is initially placed at the (2,2) square, which is the center black square. At each step, a fair coin is tossed to decide whether or not the king will make a move. If the coin toss comes up heads, the king will make a random move to a legal position. If the coin toss is a tails, the king will stay in the same square. Let X_j record the kings position at time j = 0,1,...

a. Find the one-step transition matrix P of X_j. For the transition matrix, arrange the states by row in ascending order.

The one-step transition matrix **P** is given by:

$$\mathbf{P} = \begin{bmatrix} \frac{1}{2} & \frac{1}{6} & 0 & \frac{1}{6} & \frac{1}{6} & 0 & 0 & 0 & 0 \\ \frac{1}{10} & \frac{1}{2} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & 0 & 0 & 0 \\ 0 & \frac{1}{6} & \frac{1}{2} & 0 & \frac{1}{6} & \frac{1}{6} & 0 & 0 & 0 \\ \frac{1}{10} & \frac{1}{10} & 0 & \frac{1}{2} & \frac{1}{10} & 0 & \frac{1}{10} & \frac{1}{10} & 0 \\ \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{2} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} \\ 0 & \frac{1}{10} & \frac{1}{10} & 0 & \frac{1}{10} & \frac{1}{2} & 0 & \frac{1}{10} & \frac{1}{10} \\ 0 & 0 & 0 & \frac{1}{6} & \frac{1}{6} & 0 & \frac{1}{2} & \frac{1}{6} & 0 \\ 0 & 0 & 0 & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{2} & \frac{1}{10} \\ 0 & 0 & 0 & 0 & \frac{1}{6} & \frac{1}{6} & 0 & \frac{1}{6} & \frac{1}{2} \end{bmatrix}$$

b. Is X_j irreducible? Explain your answer.

Recall that a Markov Chain is called irreducible if, from any initial state, the chain can reach any other state in some number of steps. Thus X_j is irreducible because the king can reach any square from the center square, and vice-versa: the king can reach the center square directly from any other square.

c. Is X_j aperiodic? Explain your answer.

Recall that a Markov Chain is called aperiodic if the Markov Chain has no periodic states. A state is said to be periodic with period d if the Markov Chain, starting at that state, can only reach that state in steps which are multiples of d. Here we see that the king can navigate from any square back to itself in 2, 3, 4, ... moves. For example the king can move to an adajcent square and move directly back, or the king can move to an adjacent square, then to another adjacent square that borders the first, and then back to the starting square. Additionally, it is possible that the king could stay in the starting square if the right coin toss is made. Thus I claim that X_j is aperiodic.

d. Is X_j regular? Explain your answer. (You may use R).

A Markov Chain X_j is said to be regular if there exists a positive integer n such that P^n has entries that are stictly positive.

Since all we seek is the existence of such a P^n, we can use R to help us:

```
P = matrix( c(1/2, 1/6, 0, 1/6, 1/6, 0, 0, 0, 0,
              1/10, 1/2, 1/10, 1/10, 1/10, 1/10, 0, 0, 0,
              0, 1/6, 1/2, 0, 1/6, 1/6, 0, 0, 0,
              1/10, 1/10, 0, 1/2, 1/10, 0, 1/10, 1/10, 0,
              1/16, 1/16, 1/16, 1/16, 1/2, 1/16, 1/16, 1/16, 1/16,
              0, 1/10, 1/10, 0, 1/10, 1/2, 0, 1/10, 1/10,
              0, 0, 0, 1/6, 1/6, 0, 1/2, 1/6, 0,
              0, 0, 0, 1/10, 1/10, 1/10, 1/10, 1/2, 1/10,
              0, 0, 0, 0, 1/6, 1/6, 0, 1/6, 1/2),
            nrow=9, ncol=9, byrow=TRUE)

# We clip digits after the first 3 places for formatting
round(P %^% 2, digits=3)
```

```
      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]
 [1,] 0.294 0.194 0.027 0.194 0.200 0.027 0.027 0.027 0.010
 [2,] 0.116 0.310 0.116 0.123 0.153 0.123 0.016 0.026 0.016
 [3,] 0.027 0.194 0.294 0.027 0.200 0.194 0.010 0.027 0.027
 [4,] 0.116 0.123 0.016 0.310 0.153 0.026 0.116 0.123 0.016
 [5,] 0.075 0.096 0.075 0.096 0.317 0.096 0.075 0.096 0.075
 [6,] 0.016 0.123 0.116 0.026 0.153 0.310 0.016 0.123 0.116
 [7,] 0.027 0.027 0.010 0.194 0.200 0.027 0.294 0.194 0.027
 [8,] 0.016 0.026 0.016 0.123 0.153 0.123 0.116 0.310 0.116
 [9,] 0.010 0.027 0.027 0.027 0.200 0.194 0.027 0.194 0.294
```

We see that every entry in P^2 is strictly positive. This confirms that X_j is regular.

e. Find the stationary distribution π of X_j. Explain how you found it.

We can run the following code in R to get the stationary distribution π of X_j (First written for Exam #3):

```
# Function that will compute the stationary distribution
# of any row-oriented transition matrix, if one exists.
getStationary = function(transition) {
  eigen_stuff = eigen(t(transition))
  index = match(1, round(eigen_stuff$values, digits=2))
  eigen_vec1 = eigen_stuff$vectors[ , index[1]]
  abs(eigen_vec1) / sum(abs(eigen_vec1))
}

pi = getStationary(P)
fractions(pi)
```

```
[1] 3/40  1/8 3/40  1/8  1/5  1/8 3/40  1/8 3/40
```

We can confirm that this is a stationary distribution of our transition matrix by taking the matrix product $\pi\mathbf{P}$:

```
fractions(pi %*% P)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
```

```
[1,] 3/40  1/8 3/40  1/8  1/5  1/8 3/40  1/8 3/40
```

Thus we see that this results in π.

f. Is X_j reversible? Explain your answer.

Recall that a Markov Chain X_j is said to be reversible if it satisfies the condition

$$\pi_x P_{x,y} = \pi_y P_{y,x}, \forall x, y \in \{a \in \mathbb{Z} | 1 \leq a \leq 9\}.$$

Here π_x is the xth entry in the stationary distribution π and $P_{x,y}$ is the entry in the xth row and yth column of the probability distribution $\mathbf{P}$. I am not aware of a quick way to inuitively show that a Markov Chain is reversible, so it might just be best to construct a function in R to help us out:

```
# checks the detailed balance condition for one (x,y) pair.
detailedBalance = function(r, c, vec=pi, trans=P) {
  (vec[r] * trans[r, c]) == (vec[c] * trans[c, r])
}
```

This function checks the condition for a single (x,y) pair. Let's try it out for a few values:

```
detailedBalance(1, 1)
```

```
[1] TRUE
```

Great! What about something else, off the main diagonal?

```
detailedBalance(1, 2)
```

```
[1] FALSE
```

Oh no, this is false! This means that our Markov Chain is not π-reversible

g. Does the n-step distribution of the chain X_j stabilize? Explain your answer.

We can see if the long term behavior of our chain stabilizes by raising our transition matrix $\mathbf{P}$ to a high power. We do this in R:

```
fractions(P %^% 2000)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
 [1,] 3/40  1/8 3/40  1/8  1/5  1/8 3/40  1/8 3/40
 [2,] 3/40  1/8 3/40  1/8  1/5  1/8 3/40  1/8 3/40
 [3,] 3/40  1/8 3/40  1/8  1/5  1/8 3/40  1/8 3/40
 [4,] 3/40  1/8 3/40  1/8  1/5  1/8 3/40  1/8 3/40
 [5,] 3/40  1/8 3/40  1/8  1/5  1/8 3/40  1/8 3/40
 [6,] 3/40  1/8 3/40  1/8  1/5  1/8 3/40  1/8 3/40
 [7,] 3/40  1/8 3/40  1/8  1/5  1/8 3/40  1/8 3/40
 [8,] 3/40  1/8 3/40  1/8  1/5  1/8 3/40  1/8 3/40
 [9,] 3/40  1/8 3/40  1/8  1/5  1/8 3/40  1/8 3/40
```

It seems that our markov chain X_j does indeed stabilize, and in fact it appears to have stabilized to a collection of row vectors, each of which is exactly π!

h. Use R to simulate the king's positions after j = 500 steps with 1024 simulations. Make a relative frequency table to describe the likelihoods of the king's positions.

We run the following code in R, assuming that the king starts in the center square:

```
mu = c(1, 0, 0, 0, 0, 0, 0, 0, 0)

oneRun = function(step, initial, P) {
  states = numeric(9)
  Xj = sample(1:length(initial), 1, prob=initial)
  for (j in 1:step) {
    rowVec = P[Xj, ]
    Xj = sample(1:length(rowVec), 1, prob=rowVec)
    states[Xj] = states[Xj] + 1
  }
  states / sum(states)
}

manyRun = function(steps, rep, initial, P) {
  states = numeric(9)
  for (j in 1:rep){
    states = states + oneRun(steps, initial, P) / rep
  }
  states
}

kingsMoves = manyRun(500, 1024, mu, P)
round(kingsMoves, digits=4)
```

```
[1] 0.0770 0.1249 0.0742 0.1251 0.2008 0.1238 0.0748 0.1240 0.0754
```

This looks similar to our stationary distribution π.

i. If the king moves to the center square, he will gain 8 points; if the king moves to an edge/corner square, he will lose 1 point. The king's score begins at 8 because he starts in the center square. Let the king move on the chessboard for a large number of steps. What is the king's average net gain? Answer this question using the ergodic theorem.

The average net gain of the king is given by

$$Gain_{net} = 8 * \Pr\{\text{king in center}\} - 1 * \Pr\{\text{king not in center}\}.$$

Here we get Pr{King lands in the center} from our π distribution:

$$\Pr\{\text{king in center}\} = \frac{1}{5}, \Pr\{\text{king not in center}\} = 1 - \frac{1}{5}.$$

Thus we have

$$Gain_{net} = 8 * \frac{1}{5} - 1 * \frac{4}{5} = \frac{4}{5}.$$

j. Answer part (i) by simulation with 500 steps and 1024 repititions. Make a histogram of the 1024 simulated values. Describe the shape, center, and standard deviation of the histogram.

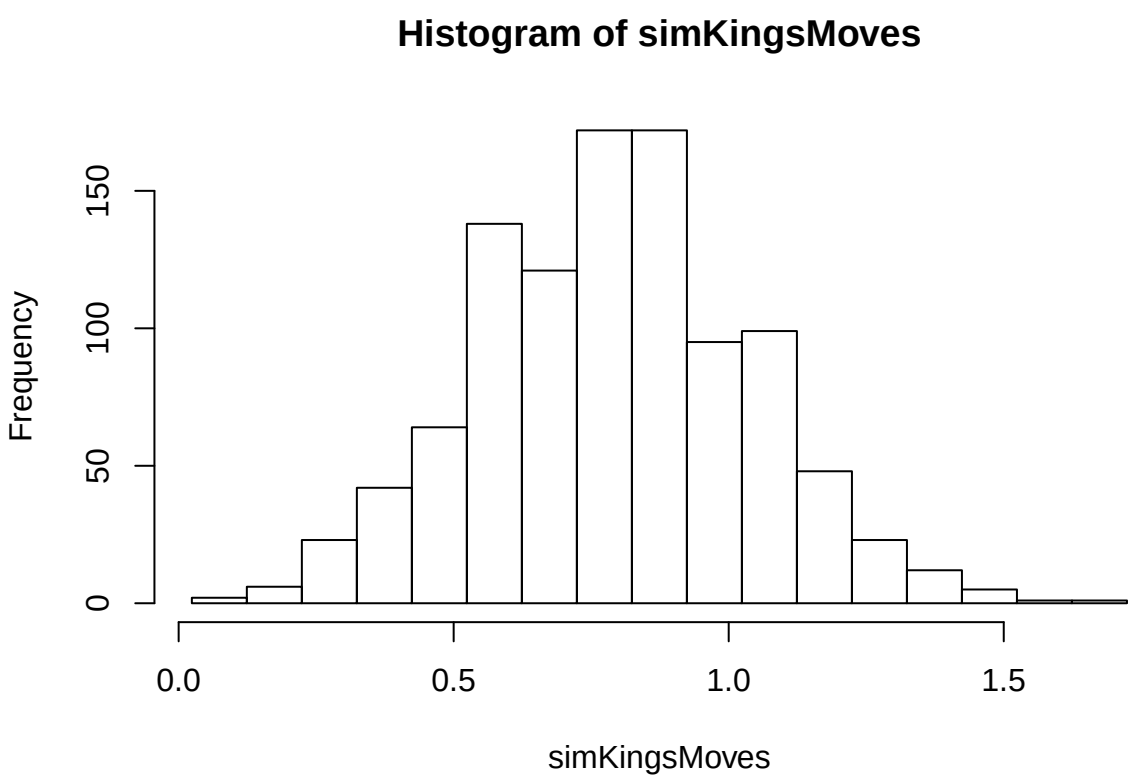
We can run the following code in R:

```r
mu = c(0, 0, 0, 0, 1, 0, 0, 0, 0)

mcKing = function(step, initial, P) {
  Xj = sample(1:length(initial), 1, prob=initial) # initialize King's move
  gain = 0 # initialize net gain to be 0
  if( Xj %% 5 == 1) { # gain 1 if king lands on a dark square
    gain <- gain + 8
  } else { # lose 1 if king lands on light square
    gain <- gain - 1
  }
  for (j in 1:step) {
    rowVec = P[Xj, ] # take the Xj-th row vector of P
    Xj = sample(1:length(rowVec), 1, prob=rowVec) # King's moves
    # update the net gain
    if( Xj %% 5 == 1) { # gain 1 if king lands on a dark square
      gain <- gain + 8
    } else { # lose 1 if king lands on light square
      gain <- gain - 1
    }
  }
  gain/step # return the average net gain
}

mcManyTrials = function(steps, rep, initial, P) {
  scores = numeric(rep)
  for (j in 1:rep){
    scores[j] = mcKing(steps, initial, P)
  }
  scores
}

simKingsMoves = mcManyTrials(500, 1024, mu, P)
hist(simKingsMoves, breaks=seq(min(simKingsMoves), max(simKingsMoves) + 0.1, 0.1))
```



We see that this is fairly normally distributed with a center around 0.8.

The actual mean of our simulation is

```
[1] 0.7908457
```

The standard deviation is

```
[1] 0.2501882
```

Problem 2.

Let π be the probability distribution of the random variable recording the maximum value obtained from rolling two fair dice. We would like to use the Metropolis-Hastings algorithm to sample from π.

a. Find the probability distribution π. Also compute the mean and the standard deviation of this distribution.

Here I can recycle code I wrote in python for homework #21:

```
unique_sum = {}
for d1 in range(1, 7):  # In python the range(1, 6) excludes 6
  for d2 in range(1, 7):
    new_sum = max(d1, d2)
    if new_sum in unique_sum:
      unique_sum[new_sum] += 1
    else:
      unique_sum[new_sum] = 1
```

```
for key in unique_sum:
  print(key, ": ", (unique_sum[key] / 36))
```

```
1 :  0.027777777777777776
2 :  0.08333333333333333
3 :  0.1388888888888889
4 :  0.19444444444444445
5 :  0.25
6 :  0.3055555555555556
```

Thus the probability distribution π is given by:

$$\pi = \left[\frac{1}{36}, \frac{1}{12}, \frac{5}{36}, \frac{7}{36}, \frac{1}{4}, \frac{11}{36}\right]$$

b. Alice uses the transition matrix:

$$\mathbf{Q}_a = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 \\ 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 \\ 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 \\ 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

for proposing states in carrying out the Metropolis Hastings algorithm. Use R to help Alice run the algorithm with n = 1200 steps and 1024 repititions. Make a relative frequency table to describe the likilihoods of the 1024 simulated values. Also report the mean and the standard deviaiton of the 1024 simulated values.

We run the following code in R:

```
Qa = matrix(c(1/2, 1/2, 0, 0, 0, 0,
              1/4, 1/2, 1/4, 0, 0, 0,
              0, 1/4, 1/2, 1/4, 0, 0,
              0, 0, 1/4, 1/2, 1.4, 0,
              0, 0, 0, 1/4, 1/2, 1/4,
              0, 0, 0, 0, 1/2, 1/2),
            nrow=6, ncol=6, byrow=TRUE)

pi = c(1/36, 1/12, 5/36, 7/36, 1/4, 11/36)

initial = c(1, 0, 0, 0, 0, 0)

metro = function(step, initial, Q) {
  x = sample(length(initial), 1, prob=initial)
  # chain starts with the initial state
  for (j in 1:step) {
    q.x = Q[x, ]
    y = sample(length(q.x), 1, prob=q.x)
    u = runif(1)
    if ( u <= pi[y]*Q[y,x]/(pi[x]*Q[x,y]) ) {
      x = y
```

```
    }
  }
  x
}

aliceSimul = replicate(1024, metro(1200, initial, Qa))
table(aliceSimul) / length(aliceSimul)
```

```
aliceSimul
         1          2          3          4          5          6
0.02441406 0.06250000 0.09375000 0.36425781 0.21093750 0.24414062
```

The mean of our simulation is

```
[1] 4.407227
```

The standard deviation of our simulation is

```
[1] 1.25964
```

c. Use the R code in part (b) to estimate the proportion of steps in which a proposed state is accepted.

We can modify the code in part (b) by adding a counter variable that gets incremented whenever we enter our conditional statement:

```
metroProbAccept = function(step, initial, Q) {
  numAccept = 0
  x = sample(length(initial), 1, prob=initial)
  # chain starts with the initial state
  for (j in 1:step) {
    q.x = Q[x, ]
    y = sample(length(q.x), 1, prob=q.x)
    u = runif(1)
    if ( u <= pi[y]*Q[y,x]/(pi[x]*Q[x,y]) ) {
      x = y
      numAccept = numAccept + 1
    }
  }
  numAccept/step
}

prAcceptSimul = replicate(1024, metroProbAccept(1200, initial, Qa))
mean(prAcceptSimul)
```

```
[1] 0.7273088
```

Thus we see that the probability of acceptance is about 72.8%.

d. Find the transition matrix $P_a = [p_{a(x,y)}]$ of the Markov Chain constructed using $\mathbf{Q}_a$ for proposing states in the Metropolis Hastings algorithm. Display the entries of $\mathbf{P}_a$ in fractions.

We can run the following code in R to find the transition matrix for Alice.

```
getTransition = function(stationary, Q) {
  P <- matrix(c(rep(0, 36)), nrow = 6, ncol=6, byrow = T);
```

```
  for (x in 1:6) {
    for (y in (1:6)[-x]) {
      if (Q[x, y] > 0) {
        P[x, y] = Q[x, y] * min(1, (stationary[y] * Q[y,x]) / (stationary[x] * Q[x,y]));
      }
    }
  }
  for (x in 1:6) {
    P[x, x] = 1 - sum(P[x,]);
  }
  P
}

Pa = fractions(getTransition(pi, Qa))
Pa
```

```
     [,1]  [,2]  [,3]  [,4]  [,5]  [,6]
[1,]  1/2   1/2     0     0     0     0
[2,]  1/6  7/12   1/4     0     0     0
[3,]    0  3/20   3/5   1/4     0     0
[4,]    0     0  5/28   1/2  9/28     0
[5,]    0     0     0   1/4   1/2   1/4
[6,]    0     0     0     0  9/44 35/44
```

e. Bob uses a different transition matrix:

$$\mathbf{Q}_b = \begin{bmatrix} \frac{1}{3} & \frac{2}{3} & 0 & 0 & 0 & 0 \\ \frac{1}{3} & 0 & \frac{2}{3} & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & \frac{2}{3} & 0 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & \frac{2}{3} & 0 \\ 0 & 0 & 0 & \frac{1}{3} & 0 & \frac{2}{3} \\ 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{2}{3} \end{bmatrix}$$

for proposing states in carrying out the Metropolis Hastings algorithm to sample from π. Use R to help Bob run the algorithm with n = 1200 steps and 1024 repititions. Make a relative frequency table to describe the likilihoods of the 1024 simulated values. Also report the mean and the standard deviation of the 1024 simulated values.

We run the following code in R to carry out the Metropolis-Hastings Algorithm for Bob:

```
Qb = matrix(c(1/3, 2/3, 0, 0, 0, 0,
              1/3, 0, 2/3, 0, 0, 0,
              0, 1/3, 0, 2/3, 0, 0,
              0, 0, 1/3, 0, 2/3, 0,
              0, 0, 0, 1/3, 0, 2/3,
              0, 0, 0, 0, 1/3, 2/3),
            nrow=6, ncol=6, byrow=TRUE)

bobSimul = replicate(1024, metro(1200, initial, Qb))
table(bobSimul) / length(bobSimul)
```

```
bobSimul
```

```
         1          2          3          4          5          6
0.03613281 0.09472656 0.14257812 0.18847656 0.21972656 0.31835938
```

The mean of our simulation for Bob is

```
[1] 4.416016
```

The standard deviation of our simulation is

```
[1] 1.472716
```

f. Use the R code in part (e) to estimate the proportion of steps in which a proposed state is accepted. Compare the estimate with the answer you obtain in part (c).

We can run the following code in R to determine this ratio:

```
prAcceptSimul = replicate(1024, metroProbAccept(1200, initial, Qb))
mean(prAcceptSimul)
```

```
[1] 0.8424683
```

In part (c) we found that we had a probability of acceptance of about 72.8%. Here we see that our probability of acceptance is about 84.2%, which is a little higher.

g. Find the transition matrix $P_b = [p_{b(x,y)}]$ of the Markov Chain constructed using $\mathbf{Q}_b$ for proposing states in the Metropolis Hastings algorithm. Display the entries of $\mathbf{P}_b$ in fractions.

We can run the following code in R to find the transition matrix for Bob:

```
Pb = fractions(getTransition(pi, Qb))
Pb
```

```
     [,1]  [,2]  [,3]  [,4]  [,5]  [,6]
[1,]  1/3   2/3     0     0     0     0
[2,]  2/9   2/9   5/9     0     0     0
[3,]    0   1/3   1/5  7/15     0     0
[4,]    0     0   1/3  5/21   3/7     0
[5,]    0     0     0   1/3  7/27 11/27
[6,]    0     0     0     0   1/3   2/3
```

h. Whose transition matrix has a faster rate of convergence for the Metropolis Hastings algorithm? Explain your answer.

To do this we can compare the second largest eigenvalues of the two transition matrices P_a and P_b. We can access their eigenvalues by running the following code in R:

```
eigen(t(Pa))$values
```

```
[1] 1.0000000 0.9116568 0.7289646 0.5015109 0.2243452 0.1123103
```

```
eigen(t(Pb))$values
```

```
[1]  1.0000000  0.8486438  0.5391204 -0.4781379  0.1844571 -0.1745067
```

Thus we see that P_b has the smaller leading eigenvalue, which means that P_b also has the higher rate of convergence for the Metropolis Hastings Algorithm.

We can confirm this more experimentally by analyzing the total variation difference statistic. To do so, we construct the following method in R:

```
# Compute the total variation distance
tvd = function(initial, stationary, transition, power) {
  u_power = initial %*% (transition %^% power)
  sum( abs( u_power - stationary ) ) / 2
}
```

We now use this to calculate the total variation difference for Alice and Bob:

```
# TVD for Alice
tvd(initial, pi, Pa, 20)
```

```
[1] 0.1581165
```

```
# TVD for Bob
tvd(initial, pi, Pb, 20)
```

```
[1] 0.03879923
```

Our results here confirm that Bob's transition matrix yields a faster rate of convergence.

Problem 3.

Let $\pi = [\pi_1, \pi_2, \ldots, \pi_r]$ be a probability distribution on support $S = \{s_1, s_2, \ldots, s_r\}$. Consider the following algorithm for constructing a Markov Chain to sample from π:
1. Choose a transition matrix $Q = [q_{x,y}]$ on the state space S with $q_{x,y} > 0$ for all $x, y \in S$.
2. Construct a Markov Chain X_j with the transition matrix $P = [p_{x,y}]$ such that

$$p_{x,y} = \pi_y q_{y,x} q_{x,y}, \text{for} x \neq y; p_{x,x} = 1 - \sum_{y:y \neq x} p_{x,y},$$

and run the Markov Chain X_j to sample from π.

a. Show that this algorithm works for sampling from π

b. Use R to implement this algorithm to answer Problem 2(e) with n = 1200 steps and 1024 repititions.

We first propose our transition matrix $Q = [q_{x,y}]$ to be the same as the transition matrix that Bob proposed in 2(e). Now we build the transition matrix $P = [p_{x,y}]$ based on the algorithm above:

```
P = matrix(numeric(36), nrow=6, ncol=6, byrow=TRUE)
for (x in 1:6) {
  for (y in 1:6) {
    if (x != y) {
      P[x, y] = pi[y] * Qb[y, x] * Qb[x, y]
    }
  }
}
for (xy in 1:6) {
  P[xy, xy] = 1 - sum(P[xy, ])
}

P = fractions(P)
P
```

```
     [,1]   [,2]   [,3]   [,4]   [,5]   [,6]
[1,]  53/54   1/54      0      0      0      0
[2,]  1/162  26/27  5/162      0      0      0
[3,]      0   1/54  76/81  7/162      0      0
[4,]      0      0  5/162  74/81   1/18      0
[5,]      0      0      0  7/162    8/9 11/162
[6,]      0      0      0      0   1/18  17/18
```

Now that we've constructed P, which models the Markov Chain X_j, we want to use P to sample from π. To do this we run the following code in R:

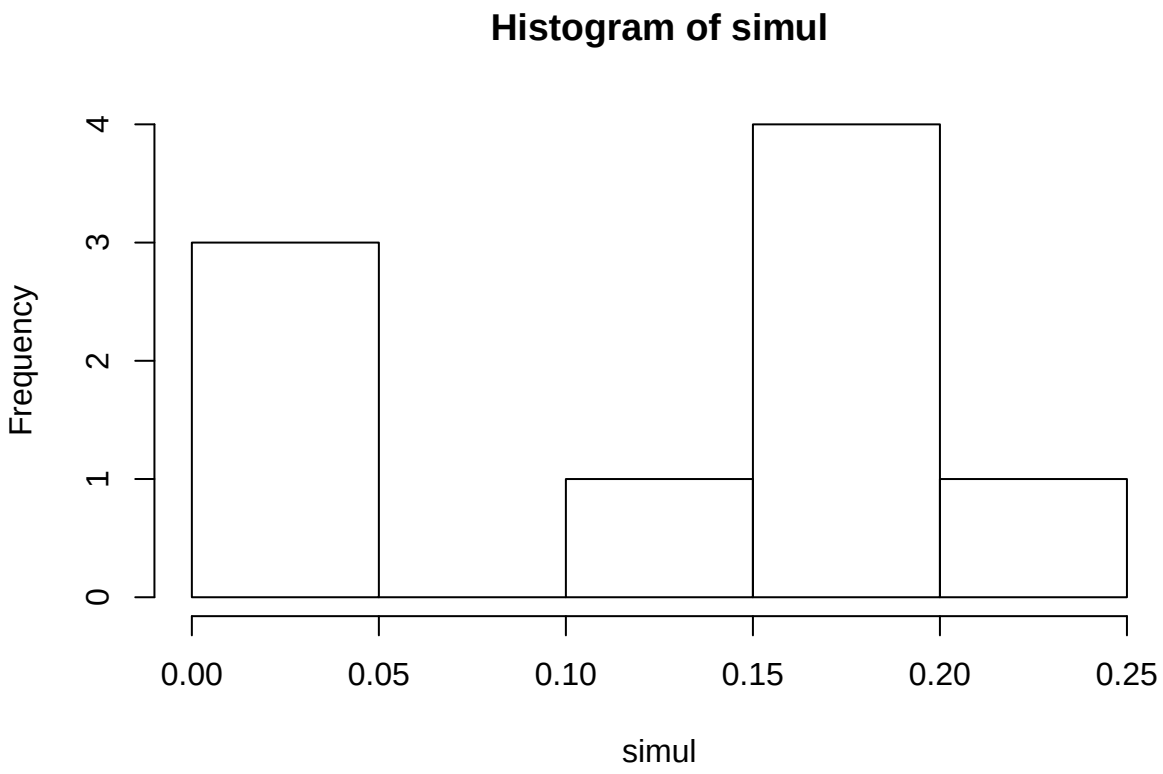
```
once = function(step, initial, P) {
  states = numeric(6)
  Xj = sample(1:length(initial), 1, prob=initial)
  for (j in 1:step) {
    rowVec = P[Xj, ]
    Xj = sample(1:length(rowVec), 1, prob=rowVec)
    states[Xj] = states[Xj] + 1
  }
  states / sum(states)
}

many = function(steps, rep, initial, P) {
  states = numeric(6)
  for (j in 1:rep){
    states = states + oneRun(steps, initial, P) / rep
  }
  states
}

simul = many(500, 1024, initial, P)
```

```
Warning in states + oneRun(steps, initial, P)/rep: longer object length is
not a multiple of shorter object length
```

```
hist(simul)
```



The mean of this table is

```
[1] 0.1111111
```

The standard deviation of this table is

```
[1] 0.08541213
```

c. Compare the convergence rate of the Markov Chain of this algorithm and the convergence rate of P_b in problem 2(g).

To do this we use the total variation distance method by running the following code in R:

```
# TVD for Bob:
tvd(initial, pi, Pb, 20)
```

```
[1] 0.03879923
```

```
#TVD for the new method:
tvd(initial, pi, P, 20)
```

```
[1] 0.8169751
```

Thus we see that the Metropolis Hastings algorithm yields a much faster rate of convergence. Sorry Yung-Pin, it appears that your algorithm has been beaten.

Problem 4.

The expected value

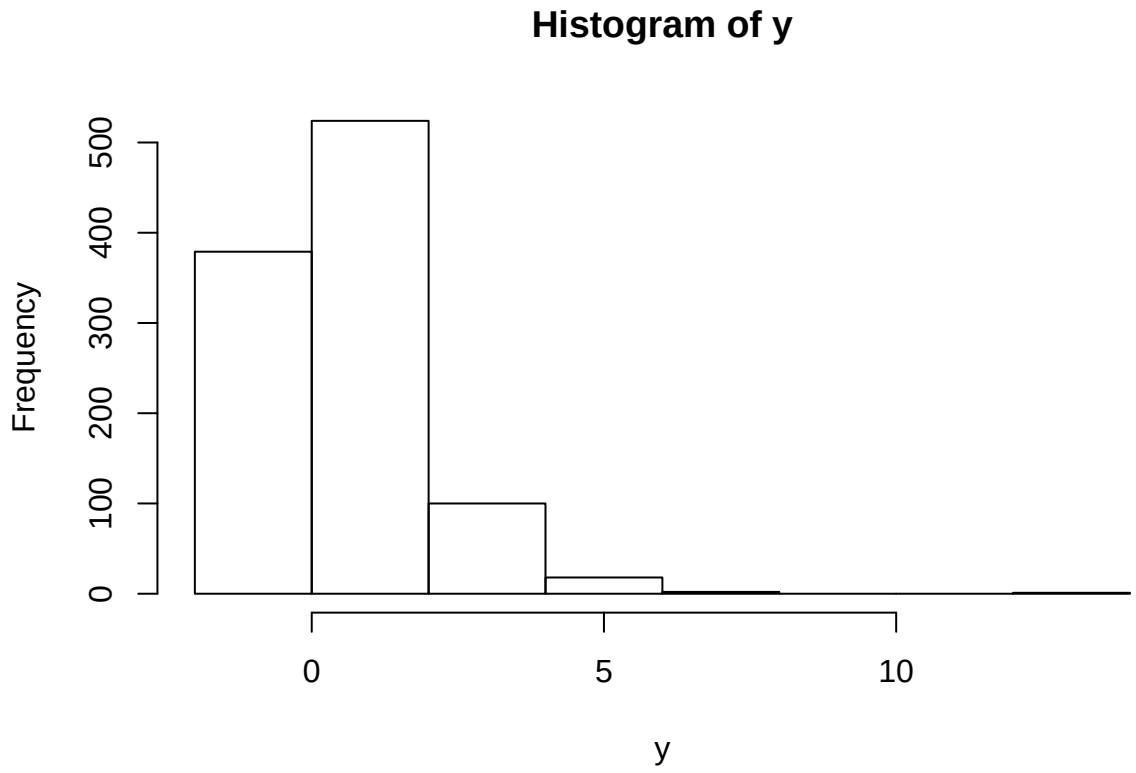
$$E(Y) = E\left[ln\left(\frac{1}{X}\right)\right] = \int_0^\infty ln\left(\frac{1}{x}\right)e^{-x}dx = -\int_0^\infty ln(x)e^{-x}dx = \gamma \approx 0.57721566490153\ldots,$$

gives the Euler constant, where X is an exponential random variable with frequency parameter $\lambda = 1$. It is a well known open problem to determine whether γ is rational or irrational.

a. Use R to sample 1024 values from an exponential distribution with $\lambda = 1$, then transform x to y. Make a histogram of y and report its shape, mean, and standard deviation.

we can run the following code in R:

```
x = rexp(1024, 1)
y = log(1/x)
hist(y)
```



This plot looks skewed to the right, almost like a chi-square distribution.

The mean of the distribution is

```
[1] 0.5583681
```

The standard deviation is

```
[1] 1.326054
```

If we run the Monte Carlo method in part (a) 900 times, what is the proportion of runs in which the Monte Carlo method would yield an estimate of E[Y] within ± 0.01? Answer this question using the following approaches:

b. Use R to give an estimate.

We use R to help us answer this question:

```
gammaEstimator = function(steps, rep) {
  gamma = 0.5772
  accept = 0
  for (j in 1:rep) {
    x = rexp(steps, 1)
    y = log(1/x)
    if (mean(y) <= gamma + 0.01 & mean(y) >= gamma -0.01) {
      accept = accept + 1
    }
  }
  accept/rep
}

gammaEstimator(1024, 900)
```

```
[1] 0.2066667
```

We see that about 18% of these runs result in an estimate of γ within ± 0.01 of its actual estimated value.

c. Use the central limit theorem to find the answer. Explain your work.