Lab 7

Jack Rellamas

Remember, **follow the instructions below and use R Markdown to create a pdf document with your code and answers to the following questions on Gradescope.** You may find a template file by clicking "Code" in the top right corner of this page.

A. Random sampling in R

1. In your own words, explain the difference between `dnorm()`, `pnorm()`, `qnorm()`, and `rnorm()`.

dnorm() returns the probability density function. It will return the probabilities of the values given in the input vector 'q'.
pnorm() returns the cumulative density function. It will return the cumulative probabilities of the values given in the input vector 'q'.
qnorm() takes in the p-score / the significance level as its parameter. It returns the corresponding z-score of the input.
rnorm() takes in the parameter n, which is the desired length of the return vector. The return vector will contain n normally distributed numbers.
The mean and standard deviation can be change in the input for all of these functions.

2. Suppose we simulate `x <- runif(1)`. What is the distribution of `qnorm(x)`?

qnorm(x) is normally distributed.

3. Suppose we simulate `x <- rnorm(1)`. What is the distribution of `pnorm(x)`?

pnorm(x) will be the probability of the normal randomly generated number x.

B. Gambler's ruin

A and B are playing a coin flipping game. A starts with n_a pennies and B starts with n_b pennies. A coin is flipped repeatedly and if it comes up heads, B gives A a penny. If it comes up tails, A gives B a penny. The game ends when one player has no more pennies.

4. Write a function `run_one_sim(seed, n_a, n_b)` to simulate one game. Repeatedly use your code with different values of `seed` to estimate each player's probability of winning when $n_a = n_b = 10$.

```r
run_one_sim <- function(seed, n_a, n_b) {
  set.seed(seed)
  while(n_a != 0 && n_b != 0) {
    flip <- rbinom(n = 1, size = 1, prob = 0.5)
    # let 0 = heads
    # B gives penny to A
    if (flip == 0) {
      n_b <- n_b - 1
      n_a <- n_a + 1
    }
    # A gives penny to B
    else {
      n_b <- n_b + 1
      n_a <- n_a - 1
    }
  }
  # return winner of game
  if (n_b == 0) {
    return("A")
  }
  else {
    return("B")
  }
}
pennies_0 <- 10
num_games <- 1000
a_wins <- 0
for(i in 1:1000) {
  if (run_one_sim(seed = i, pennies_0, pennies_0) == "A") {
    a_wins <- a_wins + 1
  }

}
print("Ratio of Games that A Wins with Initial Pennies = 10")
```

```
## [1] "Ratio of Games that A Wins with Initial Pennies = 10"
```

```r
a_winrate <- a_wins / 1000
print(a_winrate)
```

```
## [1] 0.524
```

```r
print("Ratio of Games that B Wins with Initial Pennies = 10")
```

```
## [1] "Ratio of Games that B Wins with Initial Pennies = 10"
```

```r
print(1 - a_winrate)
```

```
## [1] 0.476
```

5. Use your function to estimate each player's probability of winning when $n_a = 1, \ldots, 5$ and $n_b = 1, \ldots, 5$, testing every combination. Organize your results in a 5 by 5 matrix and print it out. What do you notice?

```r
a_wins <- 0

# returns num games that A won
run_n_games <- function(n, n_a, n_b) {
  for(i in seq(1, n, by = 1)) {
    if (run_one_sim(seed = i, n_a, n_b) == "A") {
      a_wins <- a_wins + 1
    }
  }
  return(a_wins)
}
num_games <- 100

my_vect <- integer()
for (a in 1:5) {
  for (b in 1:5) {
    a_wins <- 0
    a_wins <- run_n_games(n = num_games, n_a = a, n_b = b)
    a_win_ratio <- a_wins / num_games
    my_vect <- c(my_vect, a_win_ratio)
  }
}
win_matrix <- matrix(data = my_vect,
                     nrow = 5,
                     ncol = 5)
print(win_matrix)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,] 0.53 0.70 0.77 0.83 0.85
## [2,] 0.36 0.51 0.61 0.69 0.74
## [3,] 0.25 0.38 0.49 0.58 0.66
## [4,] 0.21 0.33 0.45 0.55 0.63
## [5,] 0.17 0.27 0.38 0.47 0.57
```

The columns represent the starting number of pennies for player A.
The rows represent the staring number of pennies for player B.
Down the diagonal, the win ratio for player A usually stays near to 0.5.
Additionally, if you take (1 - value of element), that result will be close to the value of its "symmetric" element (ie: A_ij and A_ji).
For elt A[5][1]: 1 - 0.17 = 0.83. Its symmetric elt, A[1][5] = 0.85.

C. One-dimensional random walks

In this part, you will simulate a one-dimensional random walk. Suppose you are at the point x at time t. At time $t+1$, the probability of moving forwards to $x+1$ is p and the chance of moving backwards to $x-1$ is $1-p$. Assume that at time $t=1$, you are at $x_1 = 0$.

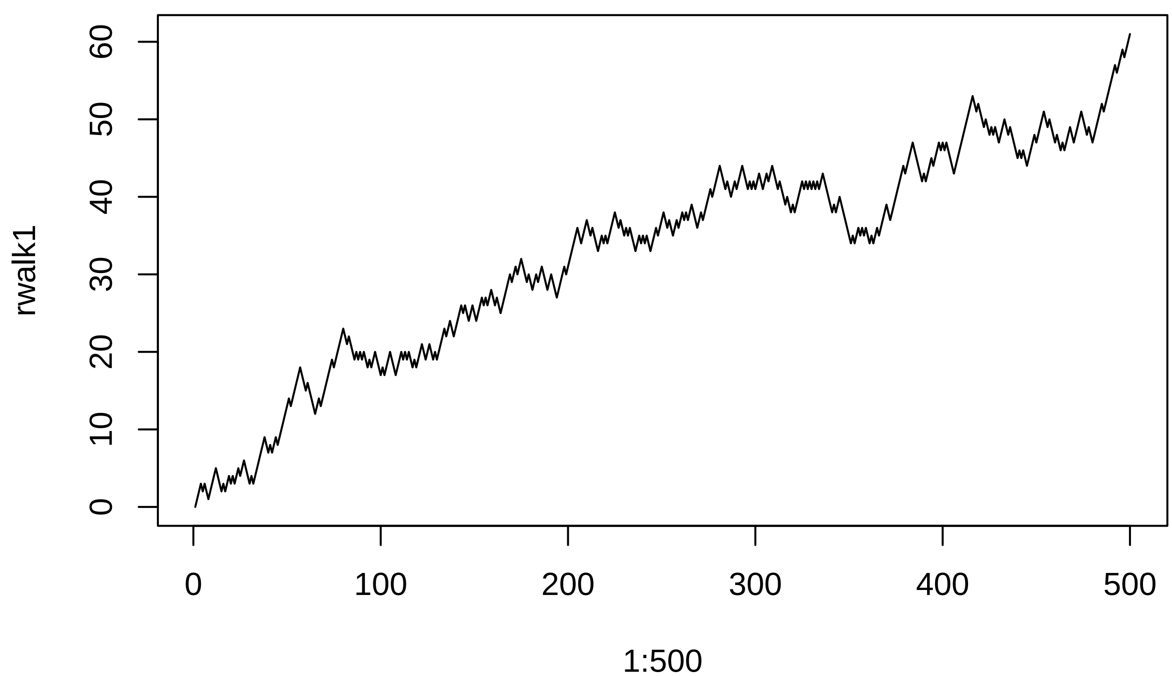
6. Write a function `random_walk()` that takes as input a numeric `n_steps` and a numeric p and simulates `n_steps` steps of the one-dimensional random walk with forward probability p. You may have other

input arguments if desired. The output should be a length vector of length `n_steps` starting with 0 where the ith entry represents the location of the random walker at time $t = i$. For example, `random_walk(5, .5)` may return the vector $(0, 1, 2, 1, 2)$.

```
random_walk <- function(n_steps, p) {
  x <- c(0)
  walk <- rbinom(n_steps - 1, 1, p)
  for (i in walk) {
    if (i == 1) {
      x <- c(x, tail(x, 1) + 1)
    }
    else {
      x <- c(x, tail(x, 1) - 1)
    }
  }
  return(x)
}
```

7. Use your function to generate a random walk of 500 steps with probability .55 and generate a line graph with $t = 1, \ldots, 500$ on the x-axis and $x_1, \ldots, x_{500}$ on the y-axis.

```
set.seed(5000)
rwalk1 <- random_walk(500, .55)
plot(x = 1:500, y = rwalk1, type = "l")
```



8. Use your function to generate two more random walks of 500 steps with probability p, where $p \sim \text{Unif}(0,1)$ and create a line graph with all three of your random walks, using different colors for each walk.

```
library(ggplot2)
set.seed(10000)
rwalk2 <- random_walk(n_steps = 500, p = runif(1))
set.seed(15000)
rwalk3 <- random_walk(n_steps = 500, p = runif(1))

rwalk_table <- data.frame(cbind(rwalk1, rwalk2, rwalk3))
ggplot(data = rwalk_table, aes(x = 1:500)) +
  geom_line(aes(y = rwalk1, color = "red")) +
  geom_line(aes(y = rwalk2, color = "green")) +
  geom_line(aes(y = rwalk3, color = "blue")) +
  xlab("Time") +
  ylab("Position") +
  ggtitle("Position vs. Time Graph of Random Walks") +
  theme(legend.position = "none")
```