Caitlyn Jones

Prof. Yangyang Wang

MATH 40A: Introduction to Applied Mathematics

12 September 2024

Homework 1

1. The method `def worldseries(p,N)` function simulates a world series based on `p`, the probability that the best team will win, and `N`, the number of games in the world series; the method will return the final score of the world series as an ordered pair. To start, we must initialize two variables, `win` and `lose`, that tracks the winning team's wins and losses.

```
win = 0 #tracking the number of games the winning team wins
lose = 0 #tracking the number of games the winning team loses
```

   Next, we will actually need to simulate game play by using a while loop that will stop executing when there is a clear winner, or when one team has won more than half of N. In the case of this problem, this loop will stop once either team wins 4 games. During the loop, to simulate the actual games, there will be the drawing of a random number between [0,1]. If the number drawn is less than p, then the winning team wins, and the variable `win` is incremented by 1. If the number is greater than p, then the losing team wins and `lose` is incremented by 1. Lastly, the function calls `save_losses(x,y)`,with `x` being `win` and `y` being `lose` (I will go over this function in the next question). As stated earlier, the method returns `win, lose` as an ordered pair.

```
# function for simulating a world series
def worldseries(p, N):

    """
    Function for simulating a world series
    p is the probability that the best team wins
    N is the number of games in the world series , usually take N=7
    """
    win = 0 #tracking the number of games the winning team wins
    lose = 0 #tracking the number of games the winning team loses
    while (max(win, lose) < math.floor(N/2)+1): #this line states that while the number of games lost or won is more than half of the games in the series, represented by N
       #in this case, this while loop will be running until a team wins or loses four games, which is more than half of 7
        if random.random() < p: #if the random number between [0,1] is less than p
            win = win + 1 #then the winning team wins the game
        else :
            lose = lose +1 #otherwise, the winning team loses

    save_losses(win,lose) #saves the frequency of the number of games the losing team wins
    return win, lose #this returns the number of games the winning team wins and loses
```

2. To run this program 44 times, we must use a loop that will simulate a world series 44 times while saving the score each time. I created the method `run_worldseries_loop_one()` that will call the `worldseries(p,N)` method 44 times and print out the saved values from `saved_losses(x,y)` once the loop has ended.

```
def run_worldseries_loop_one():
    """
    This method holds the loop that will help
    in tally up all 44 games in one place.
    Once the loop is done running, a
    statement is printed.

    """
    i = 0
    while (i<44):
        worldseries(0.65,7)
        i+= 1

    return values
```

In order to count and graph the number of times the losing team wins 0,1,2, and 3 games, we need to save this data in a bar graph. My approach to this problem was to create a list variable, `values = [0,0,0,0]`, with each value initialized to 0. This list will hold the total of 0 games won, 1 game, and so on.

```
values = [0,0,0,0]
```

Next, I updated `values` in `save_losses(x,y)` every time the winning team won the series by saving the frequency of the number of games won by the losing team in `values`.

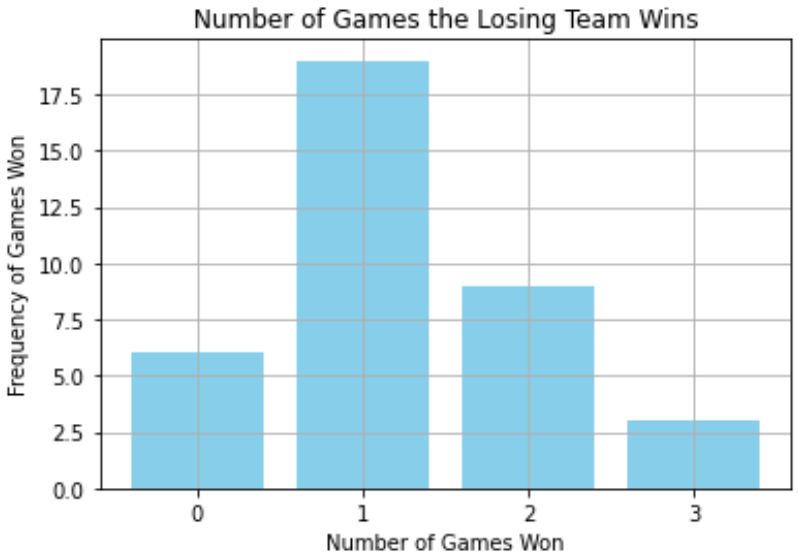
```
def save_losses(x,y): #parameters are x,y. x = wins and y = loss, which are returned by the worldseries(p,N) method
    if(y < 4): #if the losers win less than 4 games (they lose the series), then do the following
        values[y] += 1 #add a tally to the number of the games win by the losing team which corresponds to its position in the list
    else:
        print("The losing team won the series.") #print this statement because you won't need to add a tally if the losing team wins the whole series

    return values
```

Finally, I ran the loop, which now saves the data of all 44 world series' in one list, and plotted the data collected in a bar graph.

```
run_worldseries_loop_one()

"""
The followuing six lines are used to create the bar graph
based on the data saved from running to loop of 44 world series'.
"""
categories = ['0','1', '2', '3']
plt.bar(categories, values, color = "skyblue")
plt.title("Number of Games the Losing Team Wins")
plt.xlabel("Number of Games Won")
plt.ylabel("Frequency of Games Won")
plt.grid(True)
plt.show()
```

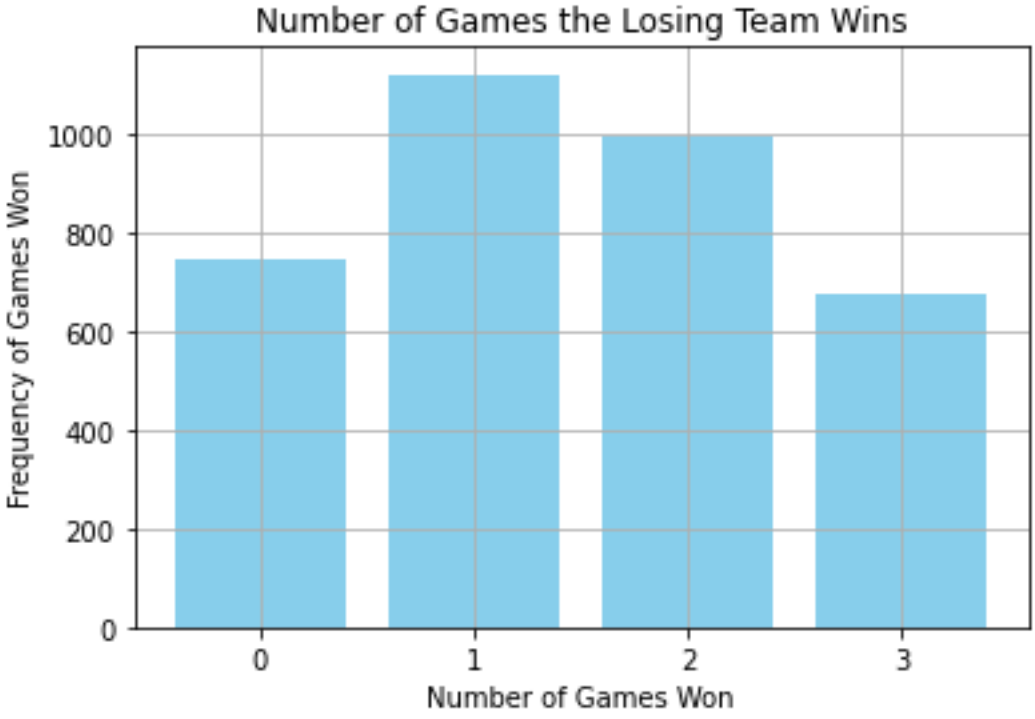


Compared to the paper, these results are similar. Both in the paper and on my graph, the losing team has the highest frequency of lost games at 1. However, in the paper the lowest frequency was at 0, but on my graph, the lowest frequency was at 3.

3. To run the loop 4400, I simply created another method `run_worldseries_loop_two()` that will run `worldseries(p,N)` 4400 times.

```
def run_worldseries_loop_two():
    """
    This method holds the loop that will help
    in tally up all 4400 games in one place.
    Once the loop is done running, a
    statement is printed.

    """
    i = 0
    while (i<4400):
        worldseries(0.65,7)
        i+= 1
    return values
```



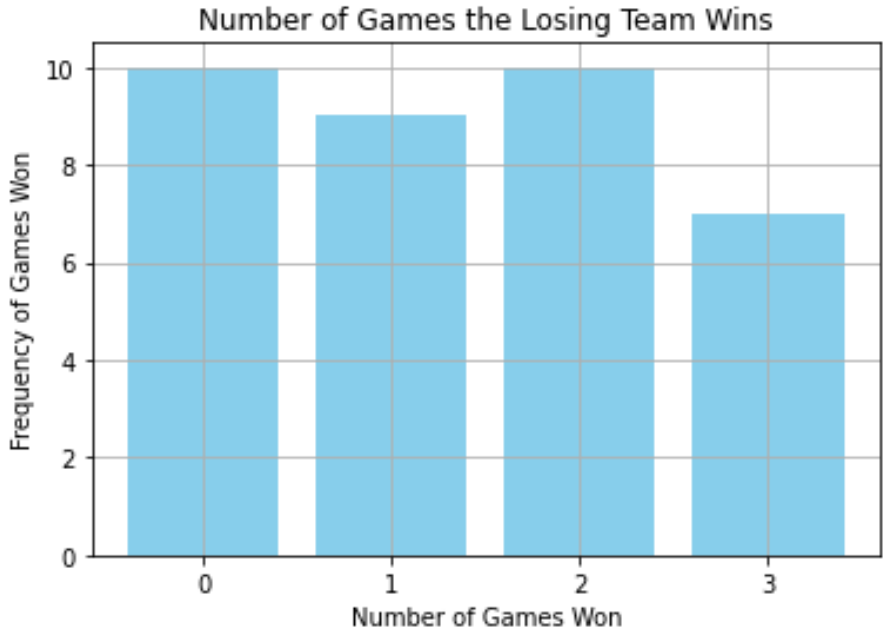
The fraction of games the losing team wins does not change significantly when more games are played. If anything, the values from the first graph are magnified in the second graph. Mosteller's measurements for the first fifty years were accurate to the results of my graph, as the winning team won 31 out of 48 games.

4. The function worldseries2 does the same actions as worldseries1 except a small change in `worldseries(p,N)`. Instead of comparing a random number to p like we did in worldseries1, worldseries2 takes a different approach to make the simulation more real. They define the variable `p_real,` which holds p with a small perturbation. This comes from the result of multiplying the random number by 0.05 and adding it to p. This will result in p having small but noticeable fluctuations, which will simulate a more real-world example of random games.

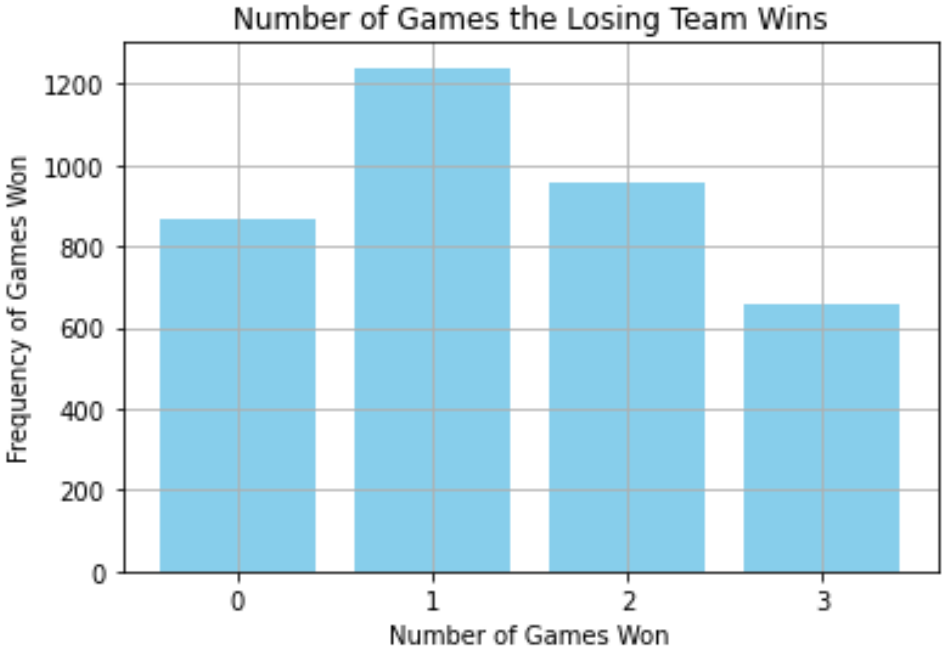
```
def worldseries(p, N):
    '''
    Function for simulating a world series
    p is the probability that the best team wins
    N is the number of games in the world series, usually take N=7
    '''
    win = 0 #tracking the number of games the winning team wins
    lose = 0 #tracking the number of games the winning team loses
    while (max(win,lose) < math.floor(N/2)+1):#this line states that while the number of games lost or won is more than half of the games in the series, represented by N
       #in this case, this while loop will be running until a team wins or loses four games, which is more than half of 7
      '''
      Random.random() retunrs a random decimal between 0.0 and 1.0.
      Multiplying the number by 0.05 scales the number down to be between 0.0 and 0.05.
      Lastly, adding this number to p will make p be slightly larger than it is in
      worldseries1. P_real represents random pertubations to simulate a more
      real world randomness.
      '''
      p_real = p + 0.05*random.random()
      if random.random() < p_real:
            win = win + 1 #then the winning team wins
      else:
            lose = lose + 1 #otherwise, the winning team loses
    save_losses(win,lose)  #saves the frequency of the number of games won by the losing team
    return win, lose #returns the result of the world series as an ordered pair
```

5. In order to graph worldseries2, I follow the steps from part 2 and 3 to receive the following graphs:

Loop ran 44 times:



Loop ran 4400 times:



The results of worldseries2 are incredibly similar to the results in worldseries1. However, when running worldseries2 44 times, the frequency of the losing team one game was not the highest; in fact, the highest frequency of games won was a tie between 0 and 2.

6. I am unhappy with Mosteller's model because there is truly no way of predicting the random events that happen during baseball games, such as injuries, bad referees, and motivation of the team. There really isn't a way to calculate these things, but it is something to think about when computing the probability of a "better" team winning.