STOR 535: Python Worksheet 2

Instructor: Dr. Amarjit Budhiraja
Due: October 5, 2021 at 2:00 PM

Submitted by: David Snider

Submission Information

1. **Download the file Python Worksheet 2.ipynb from Sakai.** The file can be found under Resources → Python Worksheets. If the file downloads as a .txt file, open it with Notepad (or a similar text editor), click Save As, under "Save As Type" choose All Files, and manually add the file extension .ipynb. This file can then be opened correctly in Jupyter Notebook. (If you have any issues with this, please e-mail me at iferer@email.unc.edu.) All work for this worksheet should be done in the boxes below.
2. Write your name in the box above.
3. Please make sure to print the output for each of the parts of the problems below. You will be graded based on these outputs.
4. After you complete the worksheet, click File → Print Preview, then right-click on the page, choose Print, and under Destination choose Save as PDF. **Upload the PDF as your submission on Gradescope. Do not submit the .ipynb file.**
5. Your final submission must be a single PDF uploaded on Gradescope.

In [1]:
```
# Importing libraries
# If there are any other libraries you want to use, import them here
import math
import random
import matplotlib.pyplot as plt
import numpy as np
```

Problem 1: Who is Better?

Choose any two sports teams or individuals who have played each other many times over the course of a season or career. These do not have to be sports teams in the traditional sense; you can choose any two individuals or teams who compete against each other in any competitive setting. Some non-sports examples could include: esports teams, chess players, politicians or political parties, or even Jeopardy! contestants ("games played" could be questions answered in this case), among others. (If there are two sides you want to use for this problem but are unsure if they are appropriate for this situation, you can e-mail me at iferer@email.unc.edu.) The following three hypotheses should be reasonable positions for a person to take, where A and B refer to the teams under consideration:

1. H_1: A is slightly better than B
2. H_2: B is slightly better than A
3. H_3: A and B are evenly matched

These hypotheses can be represented mathematically by the following: each time A and B compete against each other, A wins with probability p, and the result of each match between A and B is independent of all other matches between them. Then the hypotheses can be represented as follows:

1. $H_1: p = 0.60$
2. $H_2: p = 0.40$
3. $H_3: p = 0.50$

Before looking at the head-to-head record between A and B, we will assume that each of these three hypotheses is equally likely to be true.

Part (a)

Collect some data about the head-to-head record between the two teams or individuals you choose. At minimum, this data should include the number of matches played between A and B and the number of those matches won by A. Briefly explain where you obtained this data and cite any sources as appropriate. Make sure to identify the number of matches played between A and B, the number of those matches won by A, and the observed win probability of A. (If you cannot think of any two competitors you wish to examine or if you have trouble finding data, consider using the all-time head-to-head record between UNC and Duke men's basketball; this should not be hard to find and has a large sample size.)

** A=Carolina. B=Duke.

Matches played: 254. UNC's wins: 141. Observed win probability = 141/254 = 0.555 Obtained from Wikipedia. **

Part (b)

Given `T`, the number of total matches between A and B, and `W`, the number of matches won by A, the following code will produce a list `results` of length `T` with exactly `W` random entries equal to 1 and all other entries equal to 0, such that `results[i]` is equal to 1 if A wins game $i+1$, and is equal to 0 if A loses game $i+1$ (recall that Python indexing starts at 0, so `results[0]` is the result of game 1). The list `results` represents a random ordering of all `T` matches between A and B such that A wins `W` games.

In [95]:
```
# random ordering
T = 254 # change this to the number of matches played between A and B
W = 141 # change this to the number of matches won by A

# randomly sample W indices from 0 to T - 1 (these represent the game numbers
A_wins = random.sample(range(T), W)
results = [int(i in A_wins) for i in range(T)]
```

Note: If you are able to obtain a historical ordering of the matches between A and B, you can use that instead. Transform the data you have collected and save the transformed data as a .txt file called `match_history.txt` such that each line of the file represents one match and indicates which team won, and such that the matches are in correct historical order. Save this file in the same folder where your Worksheet 2 .ipynb file is located. Uncomment and edit (where indicated) the following block of code, and run it instead of the previous block. (You can also modify this block for any other way you may wish to load your data into Python, as long as you construct something similar to the list `results` as outlined above.)

In [3]:
```
# historical ordering
#f = open('match_history.txt')
#results = []
#for line in f:
    #if line.strip('\n').strip() == '(string you used to indicate A won)': #
        #results.append(1)
    #else:
        #results.append(0)
```

Write code to update the subjective probabilities that each hypothesis H_1, H_2, H_3 is true after the result of each match is determined. Keep three lists, each storing the updated probabilities that each hypothesis is true, such that the ith entry of each list represents the updated probability that the corresponding hypothesis is true after the ith match is played ($i = 0, \ldots, T$). Some code is provided to help you get started.

In [176…

```
# Write any additional functions you want to use for part (b) here

#defined for 1,...,len(results)
def add(k):
    pLatestAndA = 0
    pLatestAndAB = 0
    pLatestAndB = 0
    if (results[k]==1):
        pLatestAndA = P_current[2]*H[2]
        pLatestAndAB = P_current[1]*H[1]
        pLatestAndB = P_current[0]*H[0]
    else:
        pLatestAndA = P_current[2]*(1-H[2])
        pLatestAndAB = P_current[1]*(1-H[1])
        pLatestAndB = P_current[0]*(1-H[0])

    pLatest = pLatestAndA + pLatestAndAB + pLatestAndB

    P_current[2] = pLatestAndA/pLatest
    P_current[1] = pLatestAndAB/pLatest
    P_current[0] = pLatestAndB/pLatest
    A_better.append(P_current[2])
    AB_equal.append(P_current[1])
    B_better.append(P_current[0])

# list of each possible value of p, according to possible hypotheses
H = [0.40, 0.50, 0.60]

# list of current subjective probabilities: P_current[i] = Pr(p = H[i])
# HINT: after each match, you should update each entry of this list
P_current = [1/3, 1/3, 1/3]

# A_better = list of subjective probabilities over time that H1 is true: Pr(p
# B_better = list of subjective probabilities over time that H2 is true: Pr(p
# AB_equal = list of subjective probabilities over time that H3 is true: Pr(p
# HINT: after each match, you should append something to each of these lists
A_better = [1/3]
B_better = [1/3]
AB_equal = [1/3]

# Write the rest of your code for part (b) here
for i in range(0,len(results)):
    add(i)
```

In [177…

```
# Print statements for part (b)
# Uncomment and run when you have finished the previous block
for i in range(len(H)):
    print('Pr(p = {0:.2f}) = {1:.5f}'.format(H[i], P_current[i]))
```

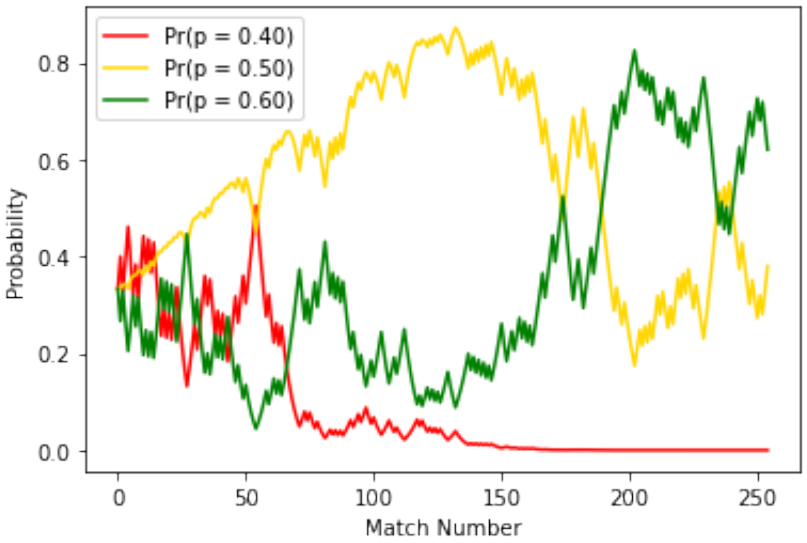
```
Pr(p = 0.40) = 0.00001
Pr(p = 0.50) = 0.37939
Pr(p = 0.60) = 0.62060
```

Part (c)

The following code visualizes the updated probabilities that each hypothesis is true after each match is played.

In [178…

```
# Uncomment and run when you have finished part (b)
x = list(range(len(A_better)))
plt.plot(x, B_better, label = 'Pr(p = 0.40)', color = 'red')
plt.plot(x, AB_equal, label = 'Pr(p = 0.50)', color = 'gold')
plt.plot(x, A_better, label = 'Pr(p = 0.60)', color = 'green')
plt.xlabel('Match Number')
plt.ylabel('Probability')
plt.legend()
plt.show()
```



Comment briefly on your output from part **(b)** and on the plot above. What conclusions can we draw about the probability of each hypothesis being true?

We can conclude that it is more likely that UNC is the better team than that they are equal, and both these outcomes are more likely than Duke being the better team.

Problem 2: House Edge

Craps is a casino game where a player rolls two six-sided dice and wins or loses depending on the sum of the dice. The basic rules for one round of play are as follows:

- On the player's first roll of the round, if 7 or 11 comes up, the player instantly wins.
- If 2, 3, or 12 comes up, the player instantly loses.
- If any other number $x \in \{4, 5, 6, 8, 9, 10\}$ comes up, the player repeatedly rolls the dice until their sum is either x (their first roll) or 7. If x comes up before 7, the player wins. If 7 comes up before x, the player loses.

We want to compute the probability of winning a single round of craps to determine whether it is worth playing in the long run.

Part (a)

Write a function called `win_prob(k)` that returns the probability of winning a round of craps, given that your first roll is `k`. Your function should return the **theoretical** probability of winning a round given that your first roll is `k`; you should **not** estimate the probability through simulation. Print the values of `win_prob(3)`, `win_prob(7)`, and `win_prob(10)`. Some code has been provided to help you get started.

In [179…

```
# D = list of tuples representing all 36 possible dice rolls
# S = list of sums of each dice roll in D
# P = probability of rolling each number 2, ..., 12 on one roll
D = [(i + 1, j + 1) for i in range(6) for j in range(6)]
S = [sum(d) for d in D]
P = [S.count(x) / len(S) for x in sorted(list(set(S)))]

# Write your code for part (a) here
def win_prob(k):
    # k = first roll of the round
    if (k==2 or k==3 or k==12):
        output=0
    elif (k==7 or k==11):
        output = 1
    else:
        output = (P[k-2])/(P[k-2]+P[5])

    # code for any computations

    return output
test = [3,7,10]
for i in test:
    print(win_prob(i))
```

```
0
1
0.3333333333333333
```

Part (b)

Using your function `win_prob(k)`, compute the theoretical probability of winning a round of craps and print the result.

In [180…

```
# Write your code for part (b) here
prob = 0
for i in range(2,12+1):
    prob = prob + P[i-2]*win_prob(i)

print(prob)
```

```
0.4929292929292929
```

Problem 3: Six Degrees of Separation

"Six degrees of separation" is the idea that any two people can be connected through (on average) 6 other people. It is famously applied in the game Six Degrees of Kevin Bacon, where players try to connect any actor to Kevin Bacon in as few steps as possible (two actors are connected if they appear in a movie or television show together). In mathematics, a similar idea is the collaboration distance between two people, where two people are connected if they co-authored a paper together.

Consider the following simplification of this situation: given a positive integer N, we will randomly choose a positive integer $n_1 \in \{1, \ldots, N-1\}$ where each number has equal probability of being chosen. If $n_1 = 1$, we stop the process, but otherwise, we then randomly choose a positive integer $n_2 \in \{1, \ldots, n_1 - 1\}$. We continue this process for k iterations until $n_k = 1$. Let X be the number of steps k we take via this process to get from N to 1. For a given N, we want to estimate the PMF of X, as well as values for $\mathbb{E}(X)$ and $\mathrm{Var}(X)$.

Part (a)

Write a function called `degree_sim(N)` that simulates the process described above for a positive integer `N` and returns the number of steps `X` taken to get to 1. To verify that your function works correctly, think about what values `degree_sim(1)` and `degree_sim(2)` should always return and what values `degree_sim(3)` could return, and then test your function to make sure it works correctly (you do **not** need to include the printed results of these tests in your final submission). **Hint:** a recursive function will work well to simulate this process, but it can also be easily simulated using a while loop.

In [181…

```
# Write your code for part (a) here
def degree_sim(N):
    # N = starting integer N

    # code for any computations
    choice = random.choice(list(range(1,N)))
    k=1
    while (choice != 1):
        choice = random.choice(list(range(1,choice)))
        k = k + 1
    return k
```

Part (b)

For all future parts, we let $N = 50$. The following code generates 10,000 samples and computes an approximation to the PMF of X:

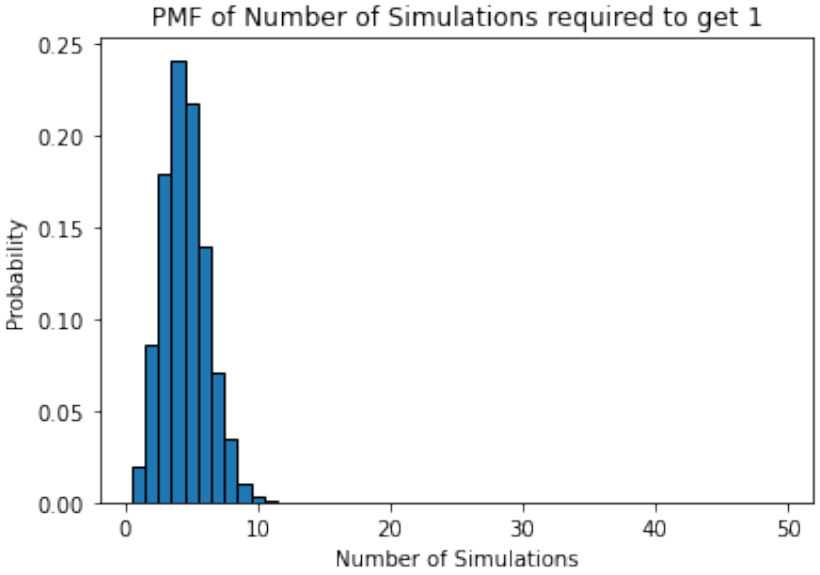
In [182…

```
# Uncomment and run when you have finished part (a)
N = 50
S = 10000
X = [degree_sim(N) for i in range(S)]
PMF_X = [X.count(k)/S for k in range(1, N)]
```

Plot the approximate distribution `PMF_X` generated above. You may use given code from class or from Python Worksheet 1 or 2 to help you plot this distribution.

In [183…

```
# Write your code for part (b) here
plt.bar(list(range(1,N)), PMF_X, width = 1, edgecolor = 'black')
plt.xlabel('Number of Simulations')
plt.ylabel('Probability')
plt.title('PMF of Number of Simulations required to get 1')
plt.show()
```



Part (c)

Using the distribution `PMF_X`, compute estimates for the expected value and variance of X and print the results.

In [184…

```
# Write your code for part (c) here
ex = 0
for i in range(1,N):
    ex = ex + PMF_X[i-1]*i
print("Expected value is: " + str(ex))

var = 0
for i in range(1,N):
    var = var + ((i-ex)**2)*PMF_X[i-1]
print("Variance is: " + str(var))
```

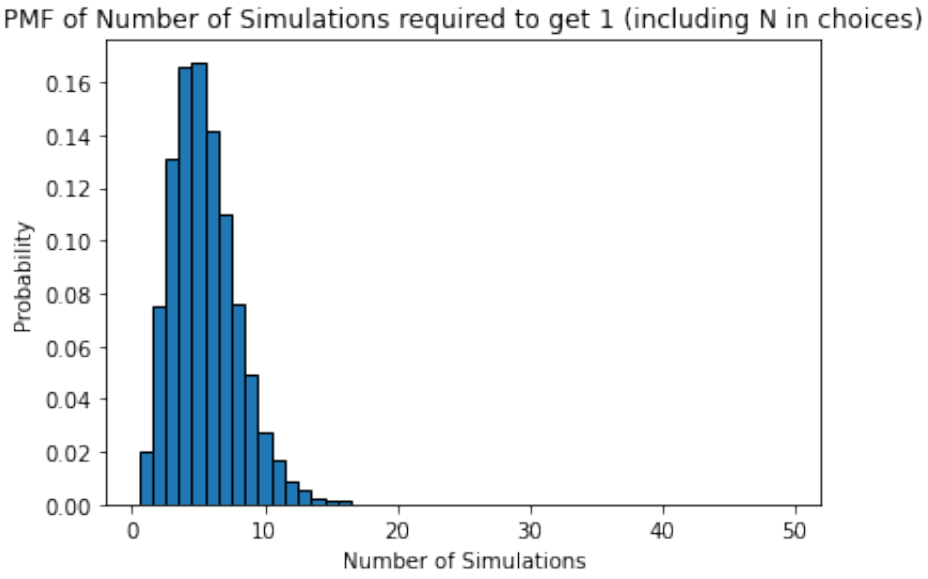
```
Expected value is: 4.5139
Variance is: 2.8088067899999998
```

Part (d)

What if at each step k, instead of choosing a positive integer $n_k \in \{1, \ldots, n_{k-1} - 1\}$, we allow for the possibility of choosing $n_k = n_{k-1}$ as well (such that each number $1, \ldots, n_{k-1}$ has equal probability of being chosen)? Replicate the code for parts **(a)** through **(c)** to plot the distribution of X and compute estimates for $\mathbb{E}(X)$ and $\mathrm{Var}(X)$ under this new formulation. What is the **theoretical** range (support) of X under this new formulation? **Hint:** the theoretical range of X is not necessarily the same as the range of values we observe when simulating X many times.

In [185…

```
# Write your code for part (d) here
def degree_sim_prime(N):
    # N = starting integer N

    # code for any computations
    choice = random.choice(list(range(1,N+1)))
    k=1
    while (choice != 1):
        choice = random.choice(list(range(1,choice+1)))
        k = k + 1
    return k

Xprime = [degree_sim_prime(N) for i in range(S)]
PMF_Xprime = [Xprime.count(k)/S for k in range(1, N)]

plt.bar(list(range(1,N)), PMF_Xprime, width = 1, edgecolor = 'black')
plt.xlabel('Number of Simulations')
plt.ylabel('Probability')
plt.title('PMF of Number of Simulations required to get 1 (including N in cho
plt.show()

exprime = 0
for i in range(1,N):
    exprime = exprime + PMF_Xprime[i-1]*i
print("Expected value is: " + str(exprime))

varprime = 0
for i in range(1,N):
    varprime = varprime + ((i-exprime)**2)*PMF_Xprime[i-1]
print("Variance is: " + str(varprime))
```




```
Expected value is: 5.4584
Variance is: 6.262269439999999
```

$$R(X) = \{\textbf{all positive integers}\}$$

In [ ]: