HW-05 MANUAL SOLUTIONS

0.0.1 Question 2a)

Let X be a random variable that gives your winnings if you bet on red and the roulette wheel is spun once.

i). What is the probability distribution of X? Give your answer as a table.

ii). Calculate the expected value of your winnings by betting on red.

Write up your full solution in the SAME box below using LaTeX (not code). Show all steps fully justifying your answer.

Solution:

i).

k	P(X=k)
1	18/38
-1	20/38

ii).

There are 18 red numbers and 20 non-red numbers. Thus we have

E**[Winnings w/ Red]=(1 ⋅ 18/38)−(1 ⋅ 20/38) =−2/38 ≈ −0.05263**

0.0.2 Question 2b)

Let's simulate this. In the first code box below, write code to simulate one spin of a roulette wheel. Your output should be a string in the form of the number then the color (i.e. 18R or 00G)

In the 2nd code box below, write code that takes the number of spins and either the color red or black as input, calculates winnings for each spin assuming you bet on that color for all spins, and then outputs the average winnings out of those spins.

Then run the simulation 3 different times for num_spins = 100,000 and compare to your answer from part

A.

To receive credit you must write your code such that all lines are visible in your PDF output.

In [ ]:
```
def spin_roulette():
    numbers = np.array(["0", "00"] + [str(ii) for ii in range(1,36+1)])
    red = [str(ii) for ii in [1,3,5,7,9,12,14,16,18,19,21,23,25,27,30,32,34,36]]
    black = [str(ii) for ii in [2,4,6,8,10,11,13,15,17,20,22,24,26,28,29,31,33,35]]
    green = ["0", "00"]
    number = np.random.choice(numbers)
    color = "R" if number in red else "B" if number in black else "G"
    return number+color
    # Your code above this line

spin_roulette()
```

In [ ]:
```
def color_winnings(color='R', num_spins=100000):
    num_winners = np.sum([color in spin_roulette() for ii in range(num_spins)])
    num_losers = num_spins - num_winners
    return (1*num_winners - 1*num_losers)/num_spins


    # Your code above this line

print("E[Winnings]      =      {:.3f}".format(color_winnings(color="R",      num_spins=int(1e6))))
print("E[Winnings]      =      {:.3f}".format(color_winnings(color="R",      num_spins=int(1e6))))
print("E[Winnings] = {:.3f}".format(color_winnings(color="R", num_spins=int(1e6))))
```

0.0.3 Question 2c)

In Roulette you can bet on one of three “dozens” segments, called 1st 12, 2nd 12, and 3rd 12. They cover 1-12, 13-24, and 25-36, respectively. If you bet $1 on the first dozen (or second dozen, or third dozen) nonzero numbers and win, then you win $2 (i.e. you get your original dollar back, plus another $2.

Let Y be a random variable that gives your winnings if you bet on any one of the three “dozen” nonzero numbers and the roulette wheel is spun once.

i). What is the probability distribution of Y ? Give your answer as a table.
ii). What is E[Y]?
Write up your full solution in the SAME box below using LaTeX (not code). Show all steps fully justifying your answer.

Solution:

i).

k	P(X=k)
2	12/38
-1	26/38

The chance of getting in the correct dozen is 12 in 38. We have:

$$E[\text{Winnings for Single Number}] = (2 \cdot 12/38) - (1 \cdot 26/38) = -2/38 \approx -0.05263$$

0.0.4 Question 2d)

Write code to simulate num_spins spins, record the winnings for each spin if you bet on the first dozen nonzero numbers, and calculate the average winnings out of the total spins.

Then run the simulation 3 different times for num_spins = 100,000 and compare to your answer from part C.

In [ ]:
```
def dozen_winnings(num_spins):
    first_dozen = [str(ii) for ii in range(1,12+1)]
    spins = [spin_roulette() for ii in range(num_spins)]
    num_winners = np.sum([spin[:-1] in first_dozen for spin in spins])
       # spin[:-1] gets only the numeric part of the result
    num_losers = num_spins - num_winners
    return (2*num_winners - 1*num_losers)/num_spins


# Your code above this line

print("E[Winnings]      =      {:.3f}".format(dozen_winnings(num_spins=100000)))
print("E[Winnings]      =      {:.3f}".format(dozen_winnings(num_spins=100000)))
print("E[Winnings] = {:.3f}".format(dozen_winnings(num_spins=100000)))
```

Question 2e)

Recall, we showed in class that the expected winnings if you bet on any number is also −1/19

So you're hopefully onto the pattern by now. The payouts in Roulette are designed so that the expected payout for a winning bet is always −1/19

Since we define these payouts in terms of your winnings after betting $1, we can think of these as payout Odds. For example, since if you bet $1 on the first dozen nonzero numbers and win, then you win $2, we say the odds are 2 to 1 (denoted 2:1).

The odds are 35:1 for landing on any particular number. This means if you bet $1, you'll win $35.

Suppose the casino wanted to develop odds for a new bet in Roulette, where they allow you to bet on any set of 3 different numbers. Let the odds for this new bet be

$$x : 1$$

What should x be so that the expected payout for a winning bet is still −1/19 ?

Show your work using LaTeX below.

Solution:

P(3 numbers) = 3/38
P (3 numbers') = 35/38
$E[Z]$ = (3/38$* x$) −(35/38)
We want $E[Z]$ = −1/19
−2/38 = (3/38$* x$) −(35/38)
33/38 = 3x/38
x=11

Z = x
Z = −1

Question 2f)

Let's generalize this!

Define a function x(n) that describes the odds the casino should give for betting $1 on any set of n numbers if the casino wants to keep the expected payout for a winning bet at −1/19 for any n. (For example, the odds for betting on any 3 different numbers should be set at x(3) to 1. The odds for betting on any 4 different numbers should be set at x(4) to 1).

Solution:
We want the expected payout for a winning bet to always be −2/38. In other words, we want

($x(n)\cdot n$/38)−1$\cdot$(38−n)/38 = −2/38

Solving for $x(n)$ we obtain

$x(n)$= (36/n)−1

Answer all of the parts below in the SAME cell below using LaTeX. Show all of your steps.

3a). Determine the value of a such that this defines a valid probability distribution. Use that value for the rest of the problem.

3b). Calculate P (X ≤ 3).

3c). What is E[X]? (Show steps calculating this).

3d). What is the standard deviation of X? (Show all steps calculating this).

Answer all of the parts above in the SINGLE cell provided below using LaTeX.

Solution:

a).

We know $\sum x\ P\ (X = x)$ = 1 , therefore, $p(2)+p(3)+p(4)=1$
.

$\Rightarrow$(8a−4a)+(18a−6a)+(32a−8a)=1

$\Rightarrow$4a+12a+24a=1

$\Rightarrow$40a=1
implies a = 1/40
.

b). $P(X\leq 3) = 1-P(X>3) = 1-P(X=4) = 1-1/20(16-4) =$ ⅗

c). $E[X] = 2P(X=2) + 3P(X=3) + 4P(X=4)$
$=(2)1/20(2)+(3)1/20(6)+(4)1/20(12)$
$=3.5$

d). $Var[X]=E[X2]-(E[X])2$

$E[X2]=22P(X=2)+32P(X=3)+42P(X=4)$

$=(4)120(2)+(9)120(6)+(16)120(12)=12.7$

$\Rightarrow Var[X]=12.7-(3.5)2 = 9/20$

$\Rightarrow Std[X]=Var[X]\_\_\_\_\_\sqrt{9/20}$

0.0.5 Question 3e

Plot a histogram of the discrete probability distribution for X.
Use the same plotting guidelines as shown in Problem 1 so we can interpret area in the histogram as representing probability: - Set the bin widths to be equal to 1 - Add white lines between each bar
Be sure to include a title on your plot.

In [ ]: ...

```
X = np.array([2,3,4])
p = 1/20*(X**2-X)

fig, ax = plt.subplots()

ax.bar(X, p, width=1, ec='white');
ax.set_axisbelow(True)
ax.grid(alpha=0.25)
plt.xlim(1,4.5)
plt.title("Distribution of X");
```

```
# Your code for the histogram above this line
```

4d). What is the probability that the self check-out tends exactly 6 customers in the next 10 minutes?
Show all steps in the cell below using LaTeX.

$P(6)=P(6|working)P(working)+P(6|frozen)P(frozen) = 56e-56!\cdot 0.9+16e-16!\cdot 0.1$

≈ 0.13165

0.0.6 Question 4e)

S'pose John is working a 5-hour shift from 4-9 PM after school. He gets no breaks, because the year is 1870 and worker's rights is not yet a thing.
Plot a histogram of the probability mass function (pmf) of the number of customers he serves in his 5 hour Shift. For the domain of the histogram, include x values between 80 and 160 in your plot.
Be sure to include labels for your x and y axes and a title for your plot.

Hint: Python has a built-in function to calculate the Poisson distribution for different values of λ. See the documentation for poisson.pmf in scipy.stats (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.poisson.html)

Hint: Since we are changing the time interval over which we are counting customers, you will need to update the parameter λ in the Poisson distribution to be the average number of customers John can serve in a 5-hour shift. You can assume that his rate of 4 customers per 10 minutes scales up consistently during his 5 hour shift.

In [ ]:
```
from scipy.stats import poisson
```

In [ ]:
```
...
        x = np.arange(80, 160)
        fig, ax = plt.subplots(1, 1)
        mu = 4*5*60/10
        p=poisson.pmf(x, mu)
        #mu=4

        ax.bar(x, p, width=1, ec='white');

        plt.title("Distribution of Pois(120)");



    # your code above this line
```

0.0.7 Question 4g)

Time to simulate!

In Questions 4e and 4f above it was possible to calculate the probabilities exactly. However, in some situations it may be too challenging to calculate an exact theoretical probability in which case we can use numerical simulation to estimate the probability.

In this part we're going to re-do Questions 4e and 4f via simulation to demonstrate that we're able to get very close to the theoretical probabilities via simulation.

That is, we're going to simulate the distribution of the number of customers John serves in his 5 hour shift.
i). Use the np.random.poisson function to randomly sample 100,000 times (with replacement) from a Poisson distribution with the same parameter λ you used in Question 4e. Save these simulated values in a numpy array called random_sample

ii). Then plot a density histogram of your simulated values. Your histogram should look very similar to the theoretical histogram you plotted in Question 4e.
Set the bin parameter to be bins = np.arange(80,160). Be sure to include a title and labels for the units of your x and y axes.

iii). Then use your simulated values to calculate the (simulated) probability that John serves 100 or more customers in a 5-hour block. Your answer should be very close to the theoretical answer you calculated in part 4f.

In [ ]:
```
random_sample = ...

    #Peek at the first 10 values
    random_sample[:10]
```

In [ ]:
```
...
        fig, ax = plt.subplots()

        ax.hist(random_sample, density=True, bins = np.arange(80,160))
        ax.grid(alpha=0.25)
        #plt.xlim(1,60)
        plt.title("Poisson Distribution, lambda =120 ");



    # Your code for part 4g(ii) above this line
```

In [ ]: ...

```
p100_john = len([cnt for cnt in random_sample if cnt>=100])/len(random_sample)   # SOLUTION

# Your code for part iii above this line

print('Simulated Probability that John serves 100 or more customers in a 5-hour block = {:.3f}'# Output should

match your theoretical answer to Part 4f
```