TDT4171 - Assignment 1
Uncertainty

Arthur Testard - id: 105022

1 Exercise 1

Consider the set of all possible five-card poker hands dealt fairly from a standard deck of fifty-two cards.

a *How many atomic events are there in the joint probability distribution (i.e., how many five-card hands are there)?*

We want to know the number of five-card hands existing, it means we want a 5-subset of the 52 cards set. Thus we have the number of five-card hand:

$$\binom{52}{5} = \frac{52!}{5!(52-5)!} = 2598960$$

.

b *What is the probability of each atomic event?*

Each atomic event has the same probability than the others, and each five-card hands are unique, thus, the probability of each atomic event is one over the number of five-card hands existing, i.e,

$$\binom{52}{5}^{-1} = \frac{1}{2598960} \approx 3,85 \times 10^{-7}$$

.

c *What is the probability of being dealt a royal straight flush? Four of a kind?*

There are only four hand of royal straight flush, and the four of them are mutually incomptible, thus the probability of being dealt a royal straight flush is the sum of each probability of being dealt a royal straight flush is:

$$4 \times \binom{52}{5}^{-1} = \frac{4}{2598960} = \frac{1}{649740} \approx 1,54 \times 10^{-6}$$

Concerning the probability of being dealt four of a kind; there are 13 kinds existing in each color. The 5th card can be any card of the $52 - 4 = 48$ cards left. So we get the number of hands being dealt for of a kind, which is 13×48. Thus, we have the probability, which is the number of hands being dealt four of a kind over the number of hands existing, which is

$$13 \times 48 \times \binom{52}{5}^{-1} = 1/4165 \approx 2,40 \times 10^{-4}$$

2 Exercise 2

Deciding to put probability theory to good use, we encounter a slot machine with three independent wheels, each producing one of the four symbols BAR, BELL, LEMON, or CHERRY with equal probability. The slot machine has the following payout scheme for a bet of 1 coin (where "?" denotes that we don't care what comes up for that wheel):

BAR/BAR/BAR *pays 20 coins*
BELL/BELL/BELL *pays 15 coins*
LEMON/LEMON/LEMON *pays 5 coins*
CHERRY/CHERRY/CHERRY *pays 3 coins*
CHERRY/CHERRY/? *pays 2 coins*
CHERRY/?/? *pays 1 coin*

The payouts do not stack. For instance, if you get 3 cherries, you get 3 coins. You do not get 6 coins, because of 3 coins for 3 cherries, 2 coins for 2 cherries and 1 coin for 1 cherry. The wheel order matter. Cherries need to be on specific wheels as outlined in the list. For example, to get 1 coin: wheel 1 = cherry, wheel 2 = ?(non-cherry) and wheel 3 = ?(non-cherry), and not some other combination with a single cherry.

a *Compute the expected "payback" percentage of the machine. In other words, for each coin played, what is the expected coin return?*

Let's write the random variable X which gives the number of coins x_i. We write $p(x/y/z)$ the probability of having the outcome (in the right order) $x/y/z$. We can notice that,

$$p(x/x/x) = 1/4^3, \text{ for all } x \in A = \{BAR, BELL, LEMON, CHERRY\}$$

Thus, we can express $P(X = 20) = p(BAR/BAR/BAR) = 1/4^3$, $P(X = 20) = p(BELL/BELL/BELL) = 1/4^3$, etc.

If we want to show it a more formal way, because all the outcomes are independent, $p(x/y/z) = p(x)p_x(y)p_{x/y}(z) = p(x)p(y)p(z)$. Thus, $P(CHERRY/CHERRY/?) = 1/4^2 \times 3/4$ and $P(CHERRY/?/?) = 1/4 \times (3/4) \times (4/4)$.

We can thus calculate the expected value,

$$\begin{aligned} E(X) &= \sum x_i P(X = x_i) \\ &= 20 \times 1/4^3 + 15 \times 1/4^3 + 5 \times 1/4^3 + 3 \times 1/4^3 + 2 \times 3/4^3 + 1 \times 3/4^2 \\ &= \frac{1}{4^3}(20 + 15 + 5 + 3 + 6 + 12) \\ &= 61/64 \end{aligned}$$

b *Compute the probability that playing the slot machine once will result in a win.*

We want to compute the probability that playing the slot machine one will result in a win, i.e, we want to compute $P(X \geq 1)$.

We notice that it event is the union of all of the event told by the statement. So we have,

$$\begin{aligned} P(X \geq 1) &= P((\bigcup_{x \in A} x/x/x) \cup (CHERRY/CHERRY/?) \cup (CHERRY/?/?)) \\ &= \sum_{x \in A} P(x/x/x) + P(CHERRY/CHERRY/?) + P(CHERRY/?/?) \\ &= 4 \times \frac{1}{4^3} + \frac{3}{4^3} + \frac{3}{4^2} \\ &= \frac{19}{64} \end{aligned}$$

c *Estimate the mean and median number of plays you can expect to make until you go broke, if you start with 10 coins. Run a simulation in Python to estimate this. Add your results to your PDF report.*

Let c_n be the number of coins at the step n (after n parties). We can define this sequence the following way:

$$\begin{aligned} &c_0 = 10 \\ &c_n = c_{n-1} - 1 + E(X) \qquad\qquad \forall n \in \mathbf{N}^\star \end{aligned}$$

We have the first line because we start with 10 coins, and the second because, each turn, we start from the number of coins we had on last turn, on which we suppress one coin for the party (price of the game), and we had the mean value of the payout $E(X)$.

Thus, we recognize an arithmetic sequence, which can be written the following way:

$$\forall n \in \mathbf{N},\ c_n = c_0 - n(1 - E(X))$$

We directly notice the strict decrease of the sequence c_n ($1 - E(X) > 0$, and then deduce that there is only one value of $n^\star$ such as $c_{n^\star} = 0$. Thus, we want $n^\star$. Using the definition of c_n, for $n \in \mathbf{N}$, we have:

$$n^\star = \frac{c_0}{1 - E(X)} = \frac{10}{1 - \frac{61}{64}} = \frac{640}{3} \approx 213$$

The median m is defined such as $P(C \geq m) = \frac{1}{2}$, where C is the random variable which counts the number of tries to get zero coins. Thus we get the same formula for m_n but instead of $E(X)$, we have $0,5$. Making,

$$m^\star = 20$$

.

We want to get similar results using python. We create the following functions:

```python
from random import randint
from statistics import median

outputs = ['BAR', 'BELL', 'LEMON', 'CHERRY']

def run_wheel():
    return outputs[randint(0, 3)]

def play_one_game(coins = 1):
    if coins < 1:
        print("Do not have enough coin")
        return coins
    coins -= 1
    output = [run_wheel(), run_wheel(), run_wheel()]

    payout = 0

    if output[0] == 'CHERRY':
        payout = 1
    if output[0] == output[1]:
        if output[0] == 'CHERRY':
            payout = 2
        if output[0] == output[2]:
            payout = 20 if output[0] == 'BAR' else
                15 if output[0] == 'BELL' else
                5 if output[0] == 'LEMON' else
                3

    return coins + payout

def play_parties_until_broke(coins = 10):
    parties_output = []
    while coins > 0:
        coins = play_one_game(coins)
        parties_output.append(coins)
    return parties_output

def compute_mean_and_median_of_parties_to_get_broke(number_of_parties = 10000):
    parties = []
    for _ in range(number_of_parties):
        parties.append(len(play_parties_until_broke()))
    return sum(parties) / len(parties), median(parties)
```

Calling `compute_mean_and_median_of_parties_to_get_broke()` gives us a mean of 211.8492 and a median of 21 based on 10 000 tries. These results are similar to the one we found mathematically.

3 Exercise 3

3.1 Part 1

Peter is interested in knowing the possibility that at least two people from a group of N people have a birthday on the same day. Your task is to find out what N has to be for this event to occur with at least 50% chance. We will disregard the existence of leap years and assume there are 365 days in a year that are equally likely to be the birthday of a randomly selected person.

a *Create a function that takes N and computes the probability of the event via simulation.*

Firstly, we have to determine how we are going to computes the function asked. Let's write p_n the probability of having at least two people who have their birthday on the same day in a group of n people. This is the value we want to compute. We have thus, the contrary event, which is, everyone in the group of n people have their birthday on a different date. The probability of this event is $\bar{p}_n = 1 - p_n$.

We can easily compute $\bar{p}_n$ if we notice that it is the number of combination of n people having a different birthday over the number of birthday's combinations. The first one is a n-permutations of the dates in a year, which is equal to $\frac{365!}{(365-n)!}$ and the second is the number of combination, which is 365^n

Thus, we can easily compute our desired probability:

$$p_n = 1 - \bar{p}_n$$
$$p_n = 1 - \frac{365!}{365^n \times (365-n)!}$$

We provide the code of the function computing this probability:

```
from scipy.special import perm

def compute_probability_birthday_same_date(N):
    P = perm(365, N) / (365 ** N)
    return 1 - P
```

For example, if we call this function for $n = 22$, we get $p(22) \approx 0.475695$

b *Use the function created in the previous task to compute the probability of the event given N in the interval [10, 50]. In this interval, what is the proportion of N where the event happens with the least 50% chance? What is the smallest N where the probability of the event occurring is at least 50%?*

For the task asked I created the following code, using the function defined in the previous question:

```
def get_proportion_of_N_and_smallest_N(range_of_N = range(10, 51)):
    Ns = []
    for N in range_of_N:
        P = compute_probability_birthday_same_date(N)
        if P > .5:
            Ns.append(N)
    return min(Ns), len(Ns)/len(range_of_N)
```

When we call this function, we get in this interval, the proportion of N for which the event happens with the least 50% chance. The result is 68%. We also get the smallest N for which the probability of the event occurring is at least 50%, which is 23.

3.2 Part 2

Peter wants to form a group where every day of the year is a birthday (i.e., for every day of the year, there must be at least one person from the group who has a birthday). He starts with an empty group, and then proceeds with the following loop:

1. *Add a random person to the group.*
2. *Check whether all days of the year are covered.*
3. *Go back to step 1 if not all days of the year have at least one birthday person from the group.*

How large a group should Peter expect to form? Make the same assumption about leap years as in Part 1.
For this task we define many functions detailed bellow:

```
def check_group_full(group):
    """Check if every values in group are not equal to zero"""
    for g in group:
        if g == 0:
            return False
    return True

def make_a_group():
    """Create a group according to the instructions in the statement"""
    group = [ 0 for k in range(365)]
```

```
    while not check_group_full(group):
        index = randint(1, 365) - 1
        group[index] += 1
    return sum(group)
```

To get some precision, we can call `make_a_group()` a thousand time to get a better precision on the mean value of how large a group should Pierre expect to form:

```
def compute_group_size_mean(nb_of_gp = 1000):
    groups = []
    for g in range(nb_of_gp):
        groups.append(make_a_group())
    return sum(groups)/len(groups), groups
```

If we run 50000 times, we get a mean size for the group of 2365.9146.

Running the following code, we can get the distribution of groups size, which we can find on Figure 1.

```
import matplotlib.pyplot as plt
plt.hist(groups, bins=100)
```

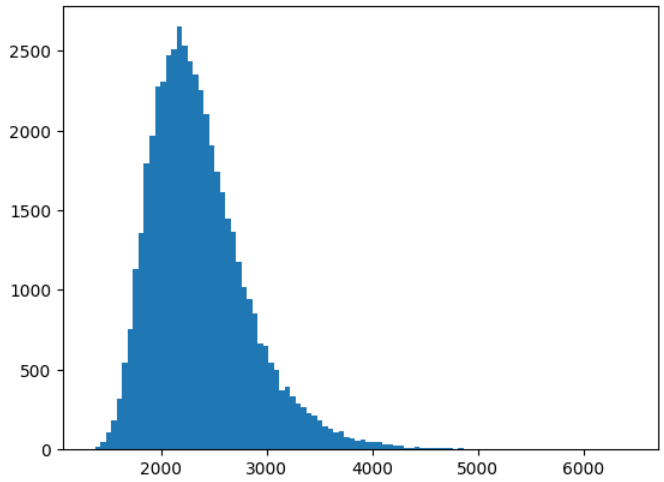

Figure 1: Distribution of the groups size in which for every day of the year, there is at least one person from the group who has a birthday

Moreover, I found it interesting to make the demonstration of the solution. Let consider the sequence of random variables X_i, counting the number of people needed to have someone having his birthday on the i-th date. We consider here, the dates ordered in the order in which the dates have been attributed to someone, and secondly, the natural order.

We can show that $\forall i \in [\![1, 365]\!]$,

$$X_i \sim Geometric(p_i)$$

where p_i is the probability to have someone who's birthday is on the i-th date. Remembering that it is the i-th date that we pick. Thus we only have $365 - (i-1)$ choice left over all the choices (365). We thus have,

$$p_i = \frac{365 - (i-1)}{365} = \frac{365 - i + 1}{365}$$

We then notice that, the probability to get at the k-th person, the i-th date attributed:

$$P(X_i = k) = (1 - p_i)^{k-1} p_i$$

as the geometric law specifies it, which can be interpreted by $k-1$ first tries to attribute the person to the i-th date which fail $(1 - p_i)$ and success on the k-th person (p_i).

We then consider the random variable

$$X = \sum_{i=1}^{365} X_i$$

.

Thus, we have:

$$
\begin{aligned}
E(X) = E(\sum_{i=1}^{365} X_i) = \sum_{i=1}^{365} E(X_i) \quad & \text{By the linearity of the expected value of independent random variables} \\
= \sum_{i=1}^{365} \frac{1}{p_i} \quad & \text{Because } X_i \sim Geometric(p_i) \\
= \sum_{i=1}^{365} \frac{365}{365 - i + 1} & \\
= 365 \times (\sum_{i=1}^{365} \frac{1}{365 - i + 1}) & \\
E(X) = 365 \times (\sum_{k=1}^{365} \frac{1}{k}) \quad & \text{Change index } k = 365 - i + 1
\end{aligned}
$$

Then, recognising that $E(X) = N$, the desired value, we can compute with the following program:

```
def harmonic_number(n):
    h = 0
    for k in range(1, n + 1):
        h += 1 / k
    return h

def get_number_of_people_needed(n = 365):
    return n * harmonic_number(n)
```

We obtain then, $E(X) \approx 2364.646$, meaning that the mean-size of Peter's group is 2365, which confirms our primary result.