Homework 3

Math and Statistics Foundations for Data Science

2022-24790 Sungwoo PARK
Graduate School of Data Science

May 6, 2022

Problem 1

(a)

Let's consider the joint probability table of one trial of coin toss. The table would look like as below.

	$X=0$	$X=1$
$Y=0$	0	p
$Y=1$	$1-p$	0

Table 1: Joint probability table of X and Y

Now, by the marginal probability, $\Pr(X=1) = \Pr(X=1, Y=0) + \Pr(X=1, Y=1) = p+0 = p$ and $\Pr(Y=1) = \Pr(X=0, Y=1) + \Pr(X=1, Y=1) = 1-p$.

From the table, we can see that the $\Pr(X=1, Y=1) = 0$, since it cannot be the case that the head and tail simultaneously appear in one coin toss. However, $\Pr(X=1)\Pr(Y=1) = p(1-p)$. This is not zero unless $p \in \{0,1\}$. Therefore, we cannot say that $X \perp\!\!\!\perp Y$, thus X and Y are dependent.

(b)

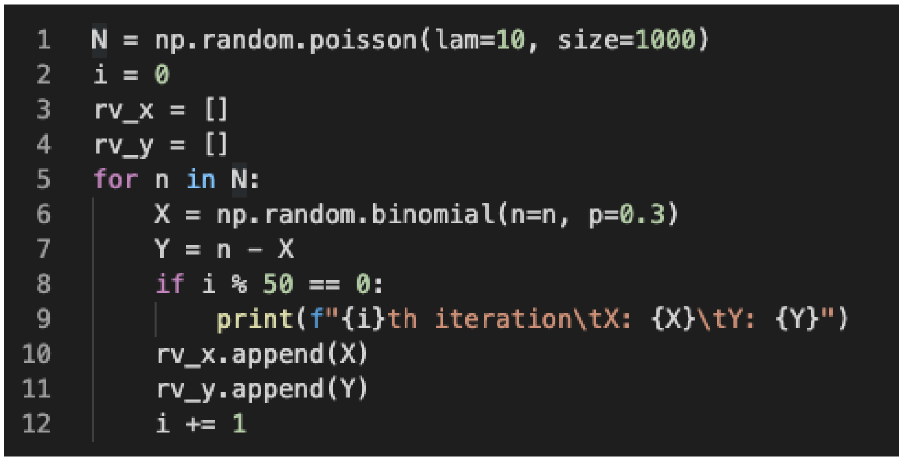
Now N is the number of coin toss and since the X is the number of appearance of head, $N = X+Y$. The appearance of head out of N coin toss trials is equivalent to the number of coins with the head above out of the same N independent coins. Moreover, the event of taking N coins out and the event of X heads turning out are independent to each other, i.e. $\Pr(N = n, X = x) = \Pr(N = n)\Pr(X = x)$. Since $N \sim \text{Poisson}(\lambda)$ and $X \sim \text{Binomial}(n, p)$ the joint pdf would look like as the equation below.

$$\begin{aligned} f_{NX}(n, x) &= f_N(n) f_X(x) \\ &= \lambda^n \frac{e^{-\lambda}}{n!} \cdot \binom{n}{x} p^x (1-p)^{n-x} \\ &= \lambda^{x+y} \frac{e^{-\lambda}}{(x+y)!} \cdot \binom{x+y}{x} p^x (1-p)^y \\ &= \lambda^x \lambda^y \frac{e^{-\lambda}}{(x+y)!} \frac{(x+y)!}{x! \cdot y!} p^x (1-p)^y \\ &= e^{-\lambda} \frac{(\lambda p)^x}{x!} \frac{(\lambda(1-p))^y}{y!} = g(x) h(y) \end{aligned}$$

As shown here, since this joint pdf is factorized into a function of X and Y, $X \perp\!\!\!\perp Y$.

(c)

The code snippet inserted below is the implementation of this experiment.



```
N = np.random.poisson(lam=10, size=1000)
i = 0
rv_x = []
rv_y = []
for n in N:
    X = np.random.binomial(n=n, p=0.3)
    Y = n - X
    if i % 50 == 0:
        print(f"{i}th iteration\tX: {X}\tY: {Y}")
    rv_x.append(X)
    rv_y.append(Y)
    i += 1
```

Figure 1: Python code snippet for experiment

The scatter plot resulted from this experiment is shown in the figure below. In the figure, some dots show stronger color than others. This is because that those dots are generated more than once during the process, so they are overlapped in the plot. The correlation coefficient between X and Y is calculated as 0.068 with the p-value of 0.8294. This implies that we cannot reject the null hypothesis that there exists no correlation between X and Y, thus we can say that they are independent to each other.

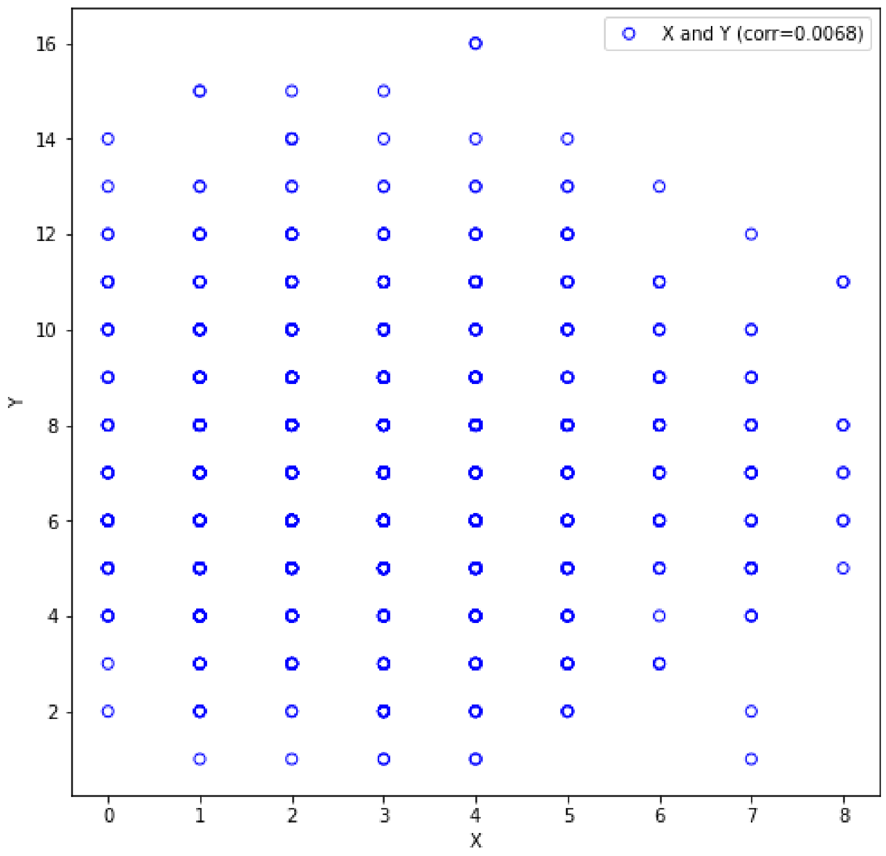
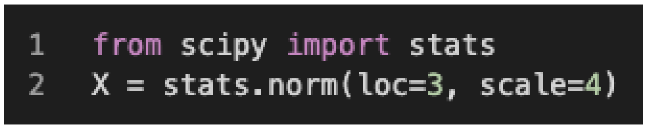


Figure 2: Scatter plot between X and Y

Problem 2

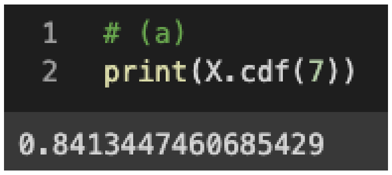
The `Python` code for the calculation is shown in the figures below. First, we need to set up the random variable $X \sim N(3, 4^2)$. This can be implemented using the `scipy` package provided by `Python`.



```python
from scipy import stats
X = stats.norm(loc=3, scale=4)
```

Figure 3: Random variable $X \sim N(3, 4^2)$

(a)



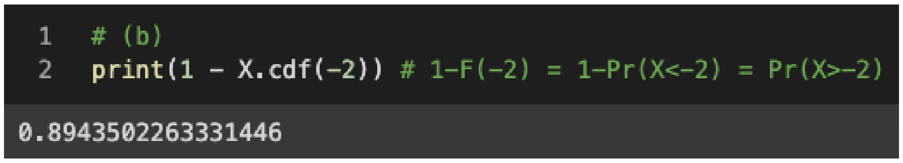
```
# (a)
print(X.cdf(7))
```

```
0.8413447460685429
```

Figure 4: Answer for (a)

Because $\Pr(X < 7) = F_X(7)$. Thus, $\Pr(X < 7) \approx 0.8413$.

(b)



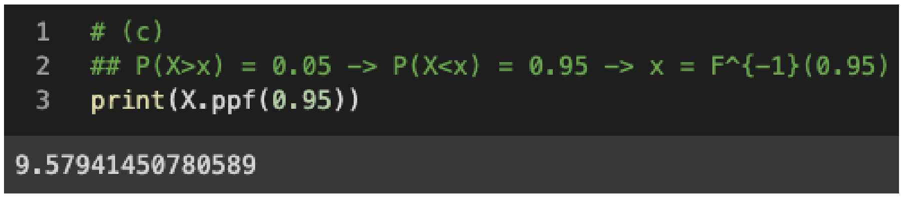
```
# (b)
print(1 - X.cdf(-2)) # 1-F(-2) = 1-Pr(X<-2) = Pr(X>-2)
```

```
0.8943502263331446
```

Figure 5: Answer for (b)

Because $\Pr(X > -2) = 1 - \Pr(X < -2) = 1 - F_X(-2)$. Thus, $\Pr(X > -2) \approx 0.8944$.

(c)



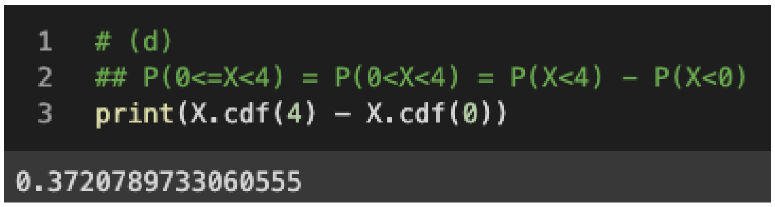
```
# (c)
## P(X>x) = 0.05 -> P(X<x) = 0.95 -> x = F^{-1}(0.95)
print(X.ppf(0.95))
```

```
9.57941450780589
```

Figure 6: Answer for (c)

Because $\Pr(X > x) = 1 - F_X(x) = 0.05$, thus $F_X(x) = 0.95$, and $x = F_X^{-1}(0.95)$. Thus, $x \approx 9.5794$.

(d)



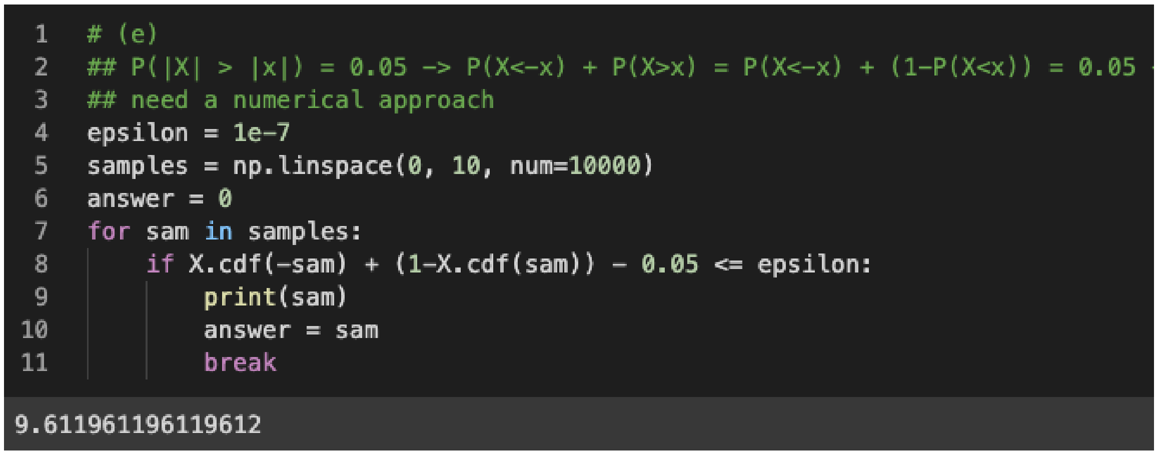
```python
# (d)
## P(0<=X<4) = P(0<X<4) = P(X<4) - P(X<0)
print(X.cdf(4) - X.cdf(0))
```

```
0.3720789733060555
```

Figure 7: Answer for (d)

Since X is continuous, $\Pr(0 \leq X < 4) = \Pr(0 < X < 4)$. By the properties of probability, $\Pr(0 < X < 4) = \Pr(X < 4) - \Pr(X < 0)$. Thus, $\Pr(0 \leq X < 4) \approx 0.3721$.

(e)

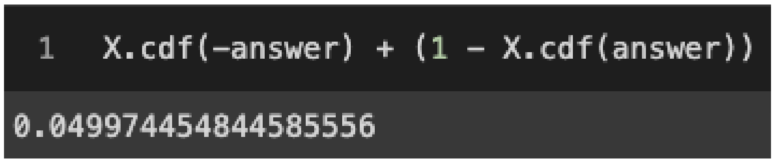


```python
# (e)
## P(|X| > |x|) = 0.05 -> P(X<-x) + P(X>x) = P(X<-x) + (1-P(X<x)) = 0.05
## need a numerical approach
epsilon = 1e-7
samples = np.linspace(0, 10, num=10000)
answer = 0
for sam in samples:
    if X.cdf(-sam) + (1-X.cdf(sam)) - 0.05 <= epsilon:
        print(sam)
        answer = sam
        break
```

```
9.611961196119612
```

Figure 8: Answer for (e)

First of all, $\Pr(|X| > |x|) = \Pr(X < -x) + \Pr(X > x) = F_X(-x) + (1 - F_X(x))$. I took a numerical approach to find x such that the difference between $F_X(-x) + (1 - F_X(x))$ and 0.05 is less than the pre-defined $\epsilon, 10^{-7}$. The result was calculated as $x \approx 9.612$, and I calculated again the $F_X(-x) + (1 - F_X(x))$ using the obtained answer. The verification result is shown in the figure below. We can see that the value is very close to 0.05.



```python
X.cdf(-answer) + (1 - X.cdf(answer))
```

```
0.049974454844585556
```

Figure 9: Verification

Problem 3

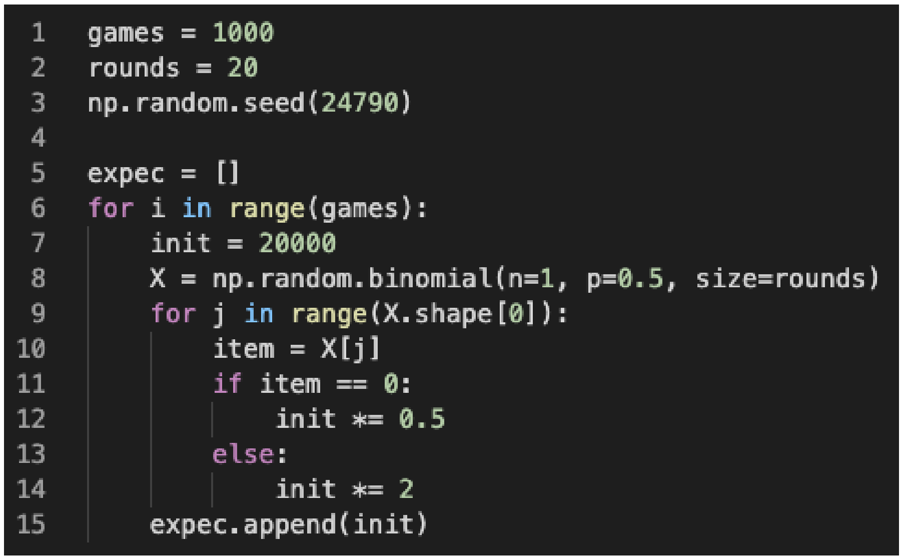
(a)

Let X_n be the amount of dollars that the gamer receives. Then, the conditional expectation of X_{n+1}, $\mathbb{E}\left[X_{n+1} \mid X_n\right] = \frac{1}{2}\left(2X_n + \frac{1}{2}X_n\right) = \frac{5}{4}X_n$. Now, by the law of iterated expectation, $\mathbb{E}\left[\mathbb{E}\left[X_{n+1} \mid X_n\right]\right] = \mathbb{E}X_{n+1} = \frac{5}{4}\mathbb{E}X_n$.

Let the initial amount of money, $X_0 = \mathbb{E}X_0 = M$, then $\mathbb{E}X_n = \left(\frac{5}{4}\right)^n M$.

(b)

When the initial amount of money is \$2000, the expectation computed from the formula equals to \$1734723.48. The `Python` code for the simulation is shown in the figure below.

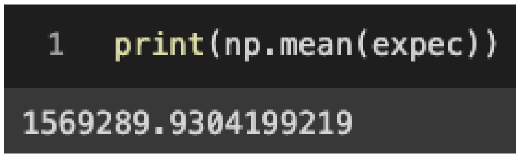


```python
games = 1000
rounds = 20
np.random.seed(24790)

expec = []
for i in range(games):
    init = 20000
    X = np.random.binomial(n=1, p=0.5, size=rounds)
    for j in range(X.shape[0]):
        item = X[j]
        if item == 0:
            init *= 0.5
        else:
            init *= 2
    expec.append(init)
```

Figure 10: Code for simulation

And the result is computed as in the figure below. There is some difference, but is close to the theoretical value.



```python
print(np.mean(expec))
```

```
1569289.9304199219
```

Figure 11: Computation result

Problem 4

Let C be a random variable where $C = 1$ if the head appears for one coin toss, $C = 0$ otherwise. Then the variable X can be denoted as the equation below.

$$X \sim \begin{cases} \text{Uniform}\,(0,1) & \text{if } C = 1 \\ \text{Uniform}\,(3,4) & \text{if } C = 0 \end{cases}$$

(a)

Then, X can be consolidated as $X = C \cdot \text{Uniform}\,(0,1) + (1-C) \cdot \text{Uniform}\,(3,4)$. Furthermore, we can see that the C and the Uniform distribution are independent to each other. Thus, using the linearity of the expectation and the independence,

$$\begin{aligned} \mathbb{E}X &= \mathbb{E}\left[C \cdot U\,(0,1)\right] + \mathbb{E}\left[(1-C) \cdot U\,(3,4)\right] \\ &= \mathbb{E}C \cdot \mathbb{E}U\,(0,1) + \mathbb{E}\left[1-C\right] \cdot \mathbb{E}U\,(3,4) \\ &= \frac{1}{2}\left(\frac{1}{2} + \frac{7}{2}\right) = 2 \end{aligned}$$

(b)

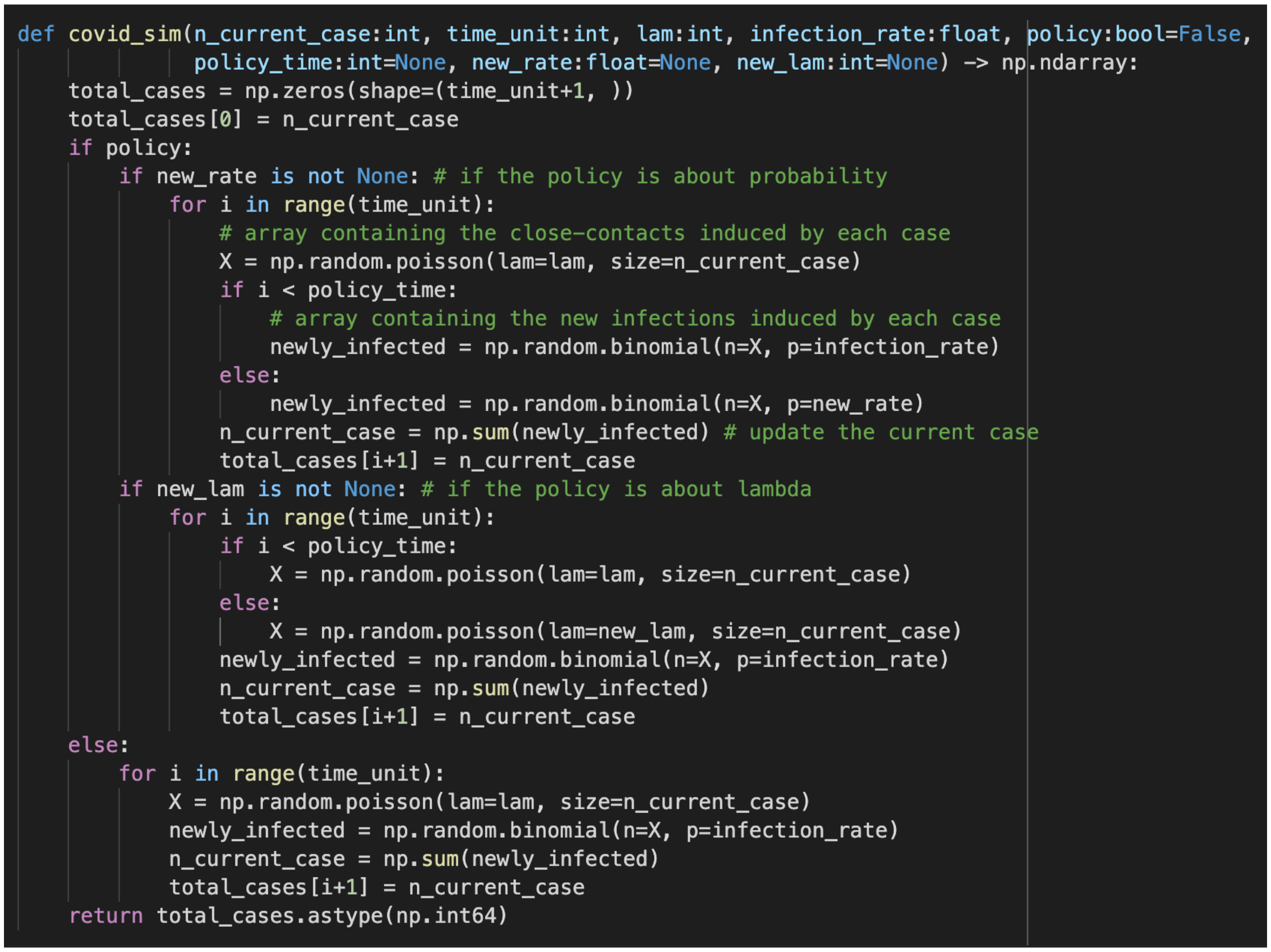
To obtain the standard deviation, we need to obtain the variance first. To obtain the variance, it is necessary to obtain the $\mathbb{E}X^2$. Furthermore, since $VarC = p\,(1-p) = \frac{1}{4}$ and $\mathbb{E}C = \frac{1}{2}$, $\mathbb{E}C^2 = VarC + (\mathbb{E}C)^2 = \frac{1}{2}$.

$$\begin{aligned} \mathbb{E}X^2 &= \mathbb{E}\left[C \cdot U\,(0,1) + (1-C) \cdot U\,(3,4)\right]^2 \\ &= \mathbb{E}\left[C \cdot U\,(0,1)\right]^2 + \mathbb{E}\left[(1-C) \cdot U\,(3,4)\right]^2 + 2\mathbb{E}\left[C \cdot U\,(0,1)\,(1-C)\,U\,(3,4)\right] \\ &= \mathbb{E}C^2 \mathbb{E}U\,(0,1)^2 + \mathbb{E}\,(1-C)^2\,\mathbb{E}U\,(3,4)^2 \quad (\because C\,(1-C) = 0) \\ &= \frac{1}{2}\left(\int_0^1 x^2 dx + \int_3^4 x^2 dx\right) = \frac{1}{2}\left(\frac{1}{3} + \frac{37}{3}\right) = \frac{19}{3} \end{aligned}$$

Therefore, the variance equals to $\mathbb{E}X^2 - (\mathbb{E}X)^2 = \frac{19}{3} - 2^2 = \frac{7}{3}$, thus the standard deviation equals to $\sqrt{\frac{7}{3}} \approx 1.528$.

Problem 5

First of all, let X be a random variable that represents the number of close-contact induced by one COVID case, then $X \sim \text{Poisson}\,(\lambda)$, where $\lambda = 10$. Next, let p denote the probability to be infected by each close-contact, then $p = 0.2$. Also, let N represent the number of infections induced by the total close-contacts X, then the conditional distribution of $N \mid X \sim \text{Binomial}(X, p)$. The function for this simulation is implemented as shown in the figure below.

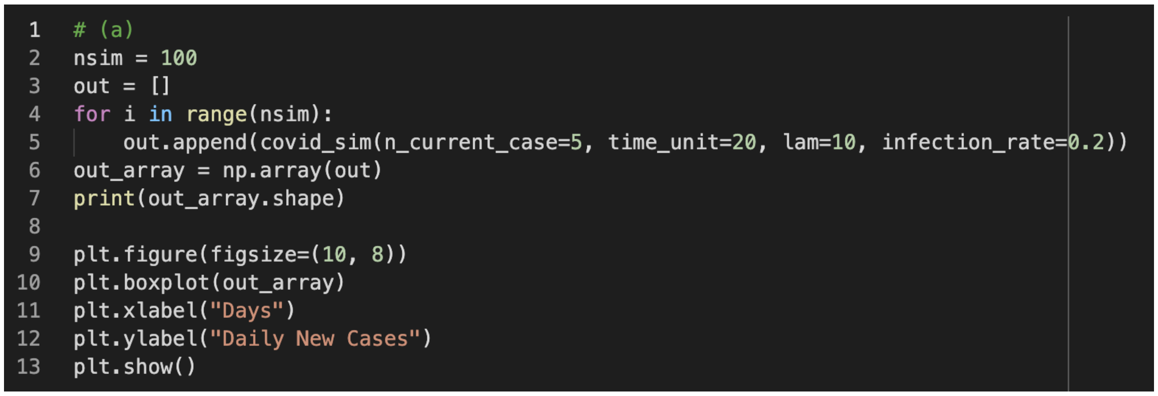


```python
def covid_sim(n_current_case:int, time_unit:int, lam:int, infection_rate:float, policy:bool=False,
              policy_time:int=None, new_rate:float=None, new_lam:int=None) -> np.ndarray:
    total_cases = np.zeros(shape=(time_unit+1, ))
    total_cases[0] = n_current_case
    if policy:
        if new_rate is not None: # if the policy is about probability
            for i in range(time_unit):
                # array containing the close-contacts induced by each case
                X = np.random.poisson(lam=lam, size=n_current_case)
                if i < policy_time:
                    # array containing the new infections induced by each case
                    newly_infected = np.random.binomial(n=X, p=infection_rate)
                else:
                    newly_infected = np.random.binomial(n=X, p=new_rate)
                n_current_case = np.sum(newly_infected) # update the current case
                total_cases[i+1] = n_current_case
        if new_lam is not None: # if the policy is about lambda
            for i in range(time_unit):
                if i < policy_time:
                    X = np.random.poisson(lam=lam, size=n_current_case)
                else:
                    X = np.random.poisson(lam=new_lam, size=n_current_case)
                newly_infected = np.random.binomial(n=X, p=infection_rate)
                n_current_case = np.sum(newly_infected)
                total_cases[i+1] = n_current_case
    else:
        for i in range(time_unit):
            X = np.random.poisson(lam=lam, size=n_current_case)
            newly_infected = np.random.binomial(n=X, p=infection_rate)
            n_current_case = np.sum(newly_infected)
            total_cases[i+1] = n_current_case
    return total_cases.astype(np.int64)
```

Figure 12: Function for simulation

(a)

The code snippet for simulation and the box plot resulted from the simulation are shown in the figures below.



```
# (a)
nsim = 100
out = []
for i in range(nsim):
    out.append(covid_sim(n_current_case=5, time_unit=20, lam=10, infection_rate=0.2))
out_array = np.array(out)
print(out_array.shape)

plt.figure(figsize=(10, 8))
plt.boxplot(out_array)
plt.xlabel("Days")
plt.ylabel("Daily New Cases")
plt.show()
```

Figure 13: Simulation code for (a)

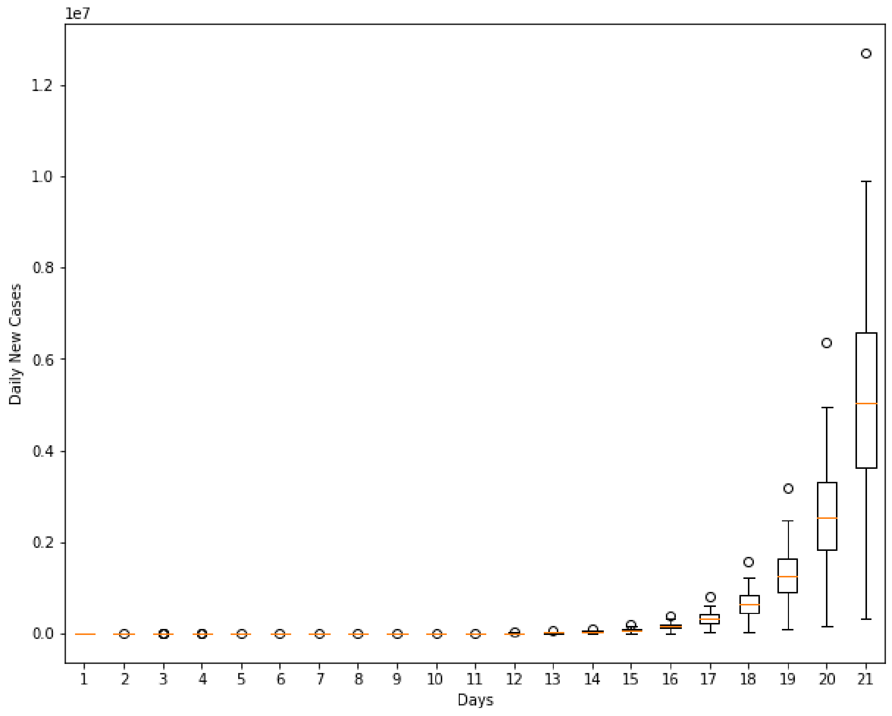
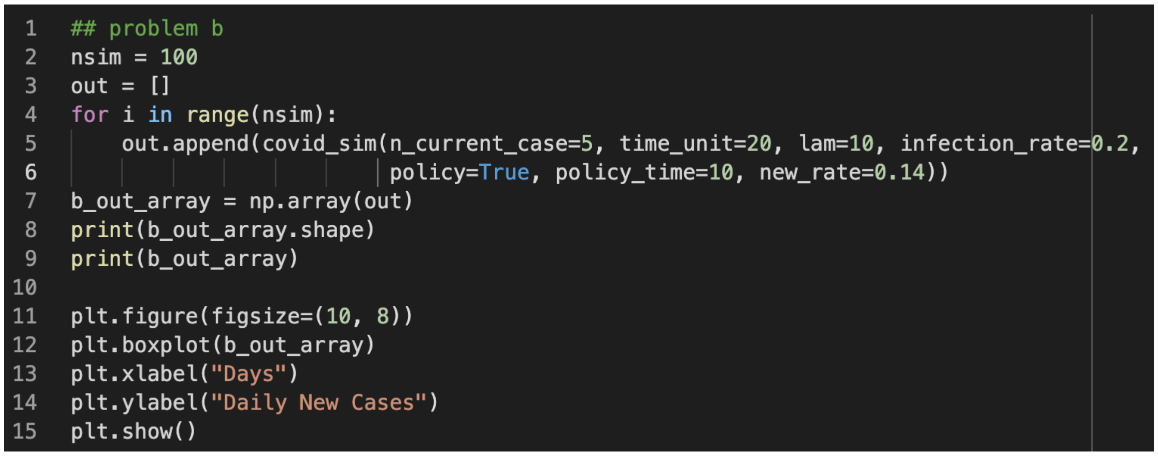


Figure 14: Box plot obtained from the simulation

We can see that the new cases increases by twice per day, being the peak at the 21th unit, which corresponds to 100 days after the first time unit.

(b)

The code snippet for simulation and the box plot resulted from the simulation are shown in the figures below.



```
## problem b
nsim = 100
out = []
for i in range(nsim):
    out.append(covid_sim(n_current_case=5, time_unit=20, lam=10, infection_rate=0.2,
                         policy=True, policy_time=10, new_rate=0.14))
b_out_array = np.array(out)
print(b_out_array.shape)
print(b_out_array)

plt.figure(figsize=(10, 8))
plt.boxplot(b_out_array)
plt.xlabel("Days")
plt.ylabel("Daily New Cases")
plt.show()
```

Figure 15: Simulation code for (b)

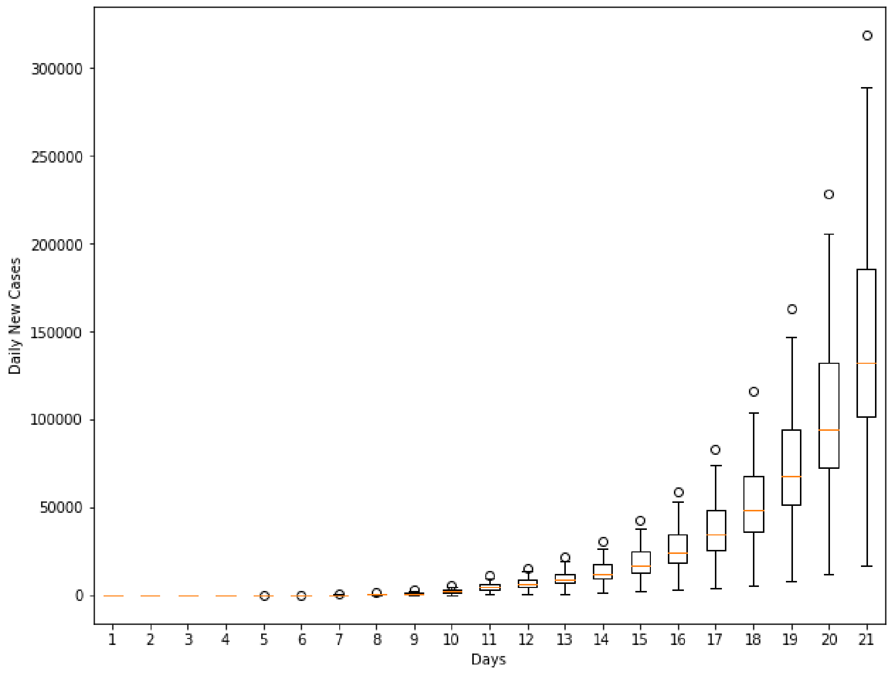
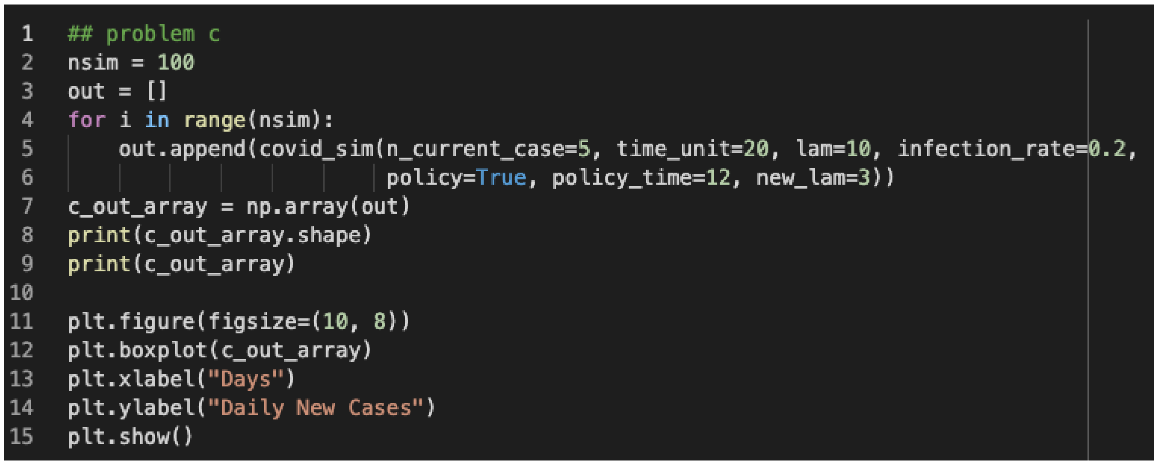


Figure 16: Box plot obtained from the simulation

Like the result in (a), we can see that the new cases increases by twice per day, being the peak at the 21th unit, which corresponds to 100 days after the first time unit. However, the hand-wash campaign helps flatten the curve, because the maximum new case count has decreased from 12,709,232 to 319,156(reduction rate 97.5%). More specifically, from the figure 14 we can see that the new cases skyrockets in the last few days, while the figure 16 shows that the new cases increases in much more gradual manner after introduction of the campaign.

(c)

The code snippet for simulation and the box plot resulted from the simulation are shown in the figures below.



```python
## problem c
nsim = 100
out = []
for i in range(nsim):
    out.append(covid_sim(n_current_case=5, time_unit=20, lam=10, infection_rate=0.2,
                         policy=True, policy_time=12, new_lam=3))
c_out_array = np.array(out)
print(c_out_array.shape)
print(c_out_array)

plt.figure(figsize=(10, 8))
plt.boxplot(c_out_array)
plt.xlabel("Days")
plt.ylabel("Daily New Cases")
plt.show()
```

Figure 17: Simulation code for (c)

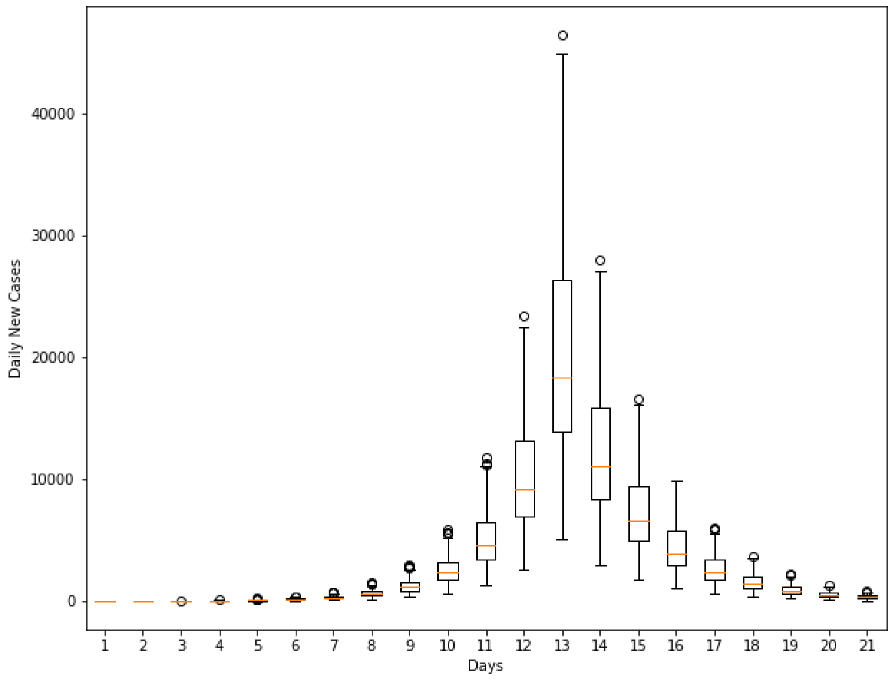


Figure 18: Box plot obtained from the simulation

Interestingly, we can see that peak is the thirteenth day, the day when the policy has introduced, and new cases steadily decreases after the policy. Furthermore, this policy has flattened the curve much more than the campaign has done in (b), since the maximum cases has decreased from 324,451 to 46,446(reduction rate 85.4%).

(d)

To make this model more realistic, I want to relieve the constraint that the newly infected individuals can spread the infection only in the next time unit. This can be implemented by updating the variable `n_current_case` cumulatively (i.e. updating the current cases by the summation of the total cases by far rather than the one-period previous cases.)