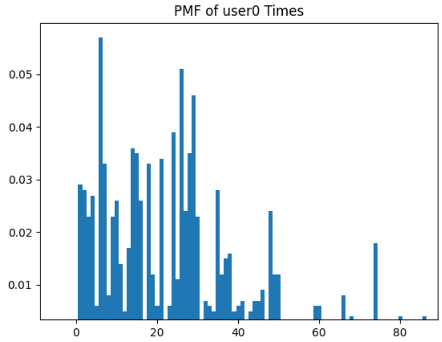
Q1(a)



Q1(b)

Mapping the values for X_0 to all the values for user0 we get 582 values where $X_0 = 1$. Since $Prob(X_0 = 1) = E[X_0]$. We can use the formula for empirical mean:

$$\frac{1}{N}\sum_{k=1}^{N} X_k = \frac{582}{1000}$$

$Prob(X_0 = 1) = 0.582$

Q1(d)

Code in appendix

Q1(c)

Chebyshev:

- Gives full distribution of X_0
- Only requires mean and variance to full describe distribution
- Con: Approximation when N is finite, hard to determine accuracy

$$\mu = 0.582, \sigma = \sqrt{\mu(1-\mu)} = 0.493, N = 1000$$
$$\mu - \frac{\sigma}{\sqrt{0.05N}} \leq X_0 \leq \mu + \frac{\sigma}{\sqrt{0.05N}} = 0.582 - \frac{0.493}{\sqrt{0.05(1000)}} \leq X_0 \leq 0.582 + \frac{0.493}{\sqrt{0.05(1000)}}$$
$$0.512 \leq X_0 \leq 0.651$$

CLT:

- Provides an actual bound and not an approximation
- Works for all N
- Con: It's loose in general

$$-1.96 * \frac{\sigma}{\sqrt{N}} + \mu \leq X_0 \leq 1.96 * \frac{\sigma}{\sqrt{N}} + \mu$$
$$0.551 \leq X_0 \leq 0.612$$

Bootstrapping:

- Gives full distribution without assuming normality
- Con: Approximation when N is finite, hard to determine accuracy
- Con: Requires the availability of all N measurements

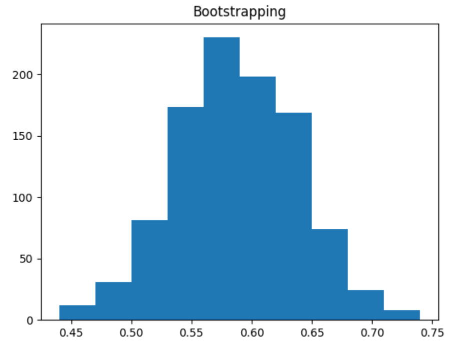


Figure 1: Code included in appendix

Q2

user1: 0.416 | **user2**: 0.399 | **user3**: 0.334

Q3

Using marginalisation and summing all the probabilites to get Z_n:

$$P(X_0 = 1)P(U_0) + P(X_1 = 1)P(U_1) + P(X_2 = 1)P(U_2) + P(X_3 = 1)P(U_3)$$

$$0.582(0.09742...) + 0.416(0.40468...) + 0.399(0.23529...) + 0.334(0.26260...)$$

$Z_n = 0.4066392298682297$

Q4

$$P(U_n = 0|Z_n > 10ms) = P(E|F) = \frac{P(F|E)P(E)}{P(F)}$$

- $P(F|E) = 0.582$ (from Q1)
- $P(E) = 0.09742483650256$ (from top line of dataset)
- $P(E^c) = 1 - P(E) = 1 - 0.09742483650256 = 0.902575163$
- $P(F|E^c) = 1 - P(F|E) = 1 - 0.582 = 0.418$
- $P(F) = P(F|E) * P(E) + P(F|E^c) * P(E^c)$

$$P(U_n = 0|Z_n > 10ms) = \frac{0.582 * 0.09742483650256}{(0.582 * 0.09742483650256) + (0.418 * 0.902575163)}$$

$P(U_n = 0|Z_n > 10ms) = 0.1306547741392685$

Q5

Code included in appendix For my simulation the result of Z_n tends to be much higher every run, around 0.9 compared to the estimate in $(Q3)$. In my simulation I generated 1000 requests for each of the 4 users and specified a request to be defined as anywhere between 0 and 100 ms. The estimate of Z_n is higher because it depends on the methodology of my simulation.

```python
import matplotlib.pyplot as plt
from random import randrange, choices

USER_PROBS = {
    'user0': 0.09742483650256,
    'user1': 0.40468106772459,
    'user2': 0.23529265941813,
    'user3': 0.26260143635472
}

def prob_X(lst, i, xi_arr):
    user_times = [x[i] for x in lst]
    user_gt10 = list(filter(lambda x: x > 10, user_times))
    x = len(user_gt10) / len(user_times)
    xi_arr.append(x)
    print("Prob(X_"+str(i) + " = 1) for user"+str(i) + ": " + str(x))

# For part of q1(c)
def bootstrapping(user_times):
    x_arr = []
    for i in range(0, 1000):
        sample = choices(user_times, k=100)
        user_gt10 = list(filter(lambda x: x > 10, sample))
        x = len(user_gt10) / len(sample)
        x_arr.append(x)
    plt.hist(x_arr)
    plt.title("Bootstrapping")
    plt.show()

# For Q3
def Zn(xi_arr):
    zn = (xi_arr[0] * USER_PROBS["user0"]) + (xi_arr[1]* USER_PROBS["user1"]) + (xi_arr[2] * USER_PROBS["user2"]) + (xi_arr[3] * USER_PROBS["user3"])
    print("Zn = " + str(zn))

# For Q5
def stochastic_sim():
    user0_times = []
    user1_times = []
    user2_times = []
    user3_times = []
    user_requests = []
    for x in range(0, 100):
        user0_times.append(randrange(0, 100))
```

```python
            user1_times.append(randrange(0, 100))
            user2_times.append(randrange(0, 100))
            user3_times.append(randrange(0, 100))
        user_requests.append(user0_times)
        user_requests.append(user1_times)
        user_requests.append(user2_times)
        user_requests.append(user3_times)
        xi_arr = []
        for i in range(0, 4):
            user_gt10 = list(filter(lambda x: x > 10, user_requests[i]))
            x = len(user_gt10) / len(user_requests[i])
            xi_arr.append(x)
        print("Stochastic sim")
        Zn(xi_arr)

def frequencies(values):
    frequencies = {}
    for v in values:
        if v in frequencies:
            frequencies[v] += 1
        else:
            frequencies[v] = 1
    return frequencies

def probabilities(sample, freqs):
    probs = []
    for k,v in freqs.items():
        probs.append(round(v/len(sample),5))
    return probs

if __name__ == "__main__":
    lst = []
    xi_arr = []
    with open("dataset.txt") as f:
        next(f)
        for line in f:
            lst.append([int(x) for x in line.split()])

    ## Q1(a) ##
    user_times = [x[0] for x in lst]
    freqs = frequencies(user_times)
    probs = probabilities(user_times, freqs)
    x_axis = list(set(user_times))
    plt.bar(x_axis,  probs, width=1)
    plt.title("PMF of user0 Times")
    plt.show()
    ## Q1(a) ##
```

```

    ## Q1(b) ##
    prob_X(lst, 0, xi_arr)
    ## Q1(b) ##

    ## Q1(c) ##
    bootstrapping(user_times)
    ## Q1(c) ##

    ## Q2 ##
    prob_X(lst, 1, xi_arr)
    prob_X(lst, 2, xi_arr)
    prob_X(lst, 3, xi_arr)
    ## Q2 ##

    ## Q3 ##
    Zn(xi_arr)
    ## Q3 ##

    ## Q4 ##
    bayes = (USER_PROBS["user0"]*0.582) / ((USER_PROBS["user0"]*0.582) + (0.418*0.902575163))
    print("Q4: " + str(bayes))
    ## Q4 ##

    ## Q5 ##
    stochastic_sim()
    ## Q5 ##
```