# DISCRETE EVENT SIMULATION

# HOMEWORK 3

# REPORT

İpek Erdoğan

150130102

Note: I added random number generators to my simulation; but not all my plots and the comments. That's why there are 2 python notebooks. One of them is for my final simulation which has an input model and custom random number generators, and one of them is my code which is only for this homework.

## Q1)

**a)** I implemented two random number generators using linear congruential method.

```
In [15]: def linear_random_cab_reorder(seed):
             return ((31*seed+7)%15)+1

In [16]: def linear_random_cab_expire(seed):
             return ((11*seed+7)%5)+7
```

First of them is to generate random numbers between 1 and 15, second of them is to generate random numbers between 7 and 12. We can pass seed as a parameter to these functions. Firstly, I tried to use clock as a seed, but since multiple functions take the same clock value as a parameter at the same time, random generators started to create same values repeatedly. That's why I had to start to pass uniformly random number between 0 and 100 as seed. You can check where did I used these two random generators in my simulation code.

## Q2)

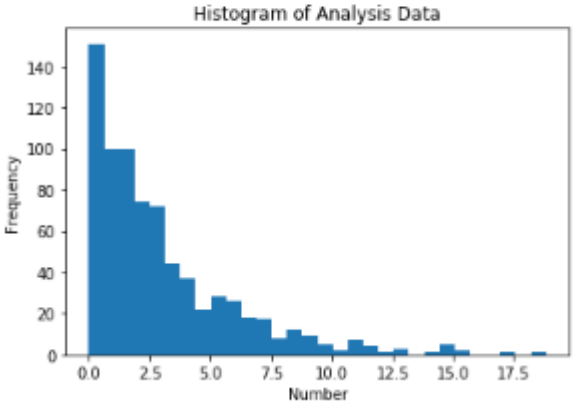
**a)** In the code block down below, you can see how did I implemented the split of our data into analysis and validation groups. (I decided to take ¾ of the data as analysis data and ¼ of the data as validation data.)

```
customer_arrivals=[]
interarrivals=[]
analysis=[]
validation=[]

def interarrival_times():
    with open('customer_arrivals.csv','r') as cusfile:
        for line in cusfile:
            x = re.split('\n+',line.rstrip())
            customer_arrivals.append(x)
        for i in range(len(customer_arrivals)-1):
            interarrivals.append(float(customer_arrivals[i+1][0]) - float(customer_arrivals[i][0]))
        ilen=round(len(interarrivals)/4)
        for i in range(ilen*3):
            analysis.append(interarrivals[i])
        for i in interarrivals[(ilen*3):]:
            validation.append(i)
```

**b)** And for analysis data, here is the visualization(as a histogram):

```
plt.hist(analysis,bins=30)
plt.title('Histogram of Analysis Data')
plt.xlabel('Number')
plt.ylabel('Frequency')
plt.show()
```




**c)** As we can see in the histogram, our data seems like fit in Exponential Distribution. So i decided to use Exponential Distribution for modelling this data.

**d)** As we will use Exponential Distribution, we will need a lambda parameter which we can calculate using Maximum Likelihood Estimation. According to the ML, we can calculate our lambda parameter as down below:

```
#First we need to calculate our lambda parameter with Maximum Likelihood Estimation.
lambdaparameter = len(analysis)/sum(analysis)
```

So, our lambda parameter is 0.3337115397450444.

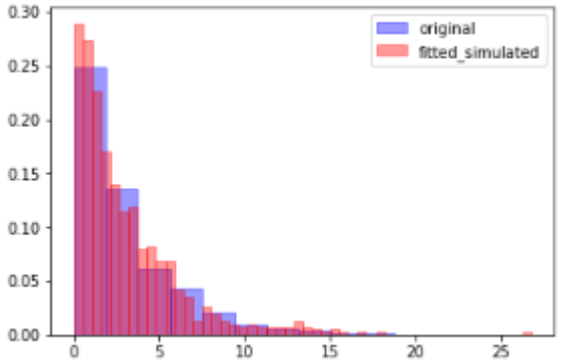
**e)** To generate random numbers for our model according to the Exponential Distribution, we will use Inverse Transform Method. We will take the inverse of the CDF, and then we will pass an x parameter which is uniformly distributed between 0 and 1. Then, we will have a randomly generated number which fits our input model at the end.

```
#Now we need to take the inverse of the CDF of Exponential Distribution (1-e^-lambdaparameter*x)
def inverse(lambdaparameter,x):
    return -math.log(1-x)/lambdaparameter
```

```
#And pass a float from uniform distribution between 0,1 to this inverse function.
def exp_random_generate(lambdaparameter):
    x = random.random()
    return inverse(lambdaparameter,x)
```

## Q3)

For input validation, I preferred to use visualization method. We can see the histogram of both our real data and custom data down below:




We can easily see that our custom generated inputs are fitting with our real data.