Exercise 4

Write a discrete event simulation program for a blocking system, i.e. a system with m service units and no waiting room. The offered traffic A is the product of the mean arrival rate and the mean service time

1

The arrival process is modelled as a Poisson process. Report the fraction of blocked customers, and a confidence interval for this fraction. Choose the service time distribution as exponential. Parameters: m = 10, mean service time = 8 time units, mean time between customers = 1 time unit (corresponding to an offered traffic of 8 Erlang), 10 x 10.000 customers.

In [ ]:
```
import numpy as np
#import poission
import math
from scipy.stats import poisson
#import exponential
from scipy.stats import expon
import bisect
from discrete_event import Customer, main_loop, confidence_intervals, erlang_b
```

In [ ]:
```
m = 10 #number of servers
s = 8 #mean service time
lam = 1#arrival_intensity
total_customers =10000 #10*10000
A = lam*s
```

In [ ]:
```
#arrival time differences are exponentially distributed
np.random.seed(1)
arrival_interval = lambda : np.random.exponential(1/lam, size = total_customers)
service_time =lambda : expon.rvs(scale = s, size = total_customers)
```

In [ ]:
```
#Amount of people blocked in the system
blocked_1 = main_loop(arrival_interval, service_time, m)
```

In [ ]:
```
print("Blocking probability: ", blocked_1/total_customers)
print("Mean blocking probability: ", np.mean(blocked_1/total_customers))
```

```
Blocking probability:  [0.1293 0.1192 0.117  0.1172 0.1246 0.1248 0.1026 0.1302 0
.1202 0.1262]
Mean blocking probability:  0.12113000000000003
```

In [ ]:
```
#Theoretical blocking probability
print("Theoretical blocking probability",erlang_b(m, A))
```

```
Theoretical blocking probability 0.12166106425295149
```

Answer

According to the discrete event simulation, the fraction of blocked customers is 0.1211 which corresponds well with the theoretical value of 0.1216.

2

The arrival process is modelled as a renewal process using the same parameters as in Part 1 when possible. Report the fraction of blocked customers, and a confidence interval for this fraction for at least the following two cases

In [ ]:
```
# (a) Experiment with Erlang distributed inter arrival times The
#Erlang distribution should have a mean of 1
np.random.seed(1)
inter_arrival = lambda : np.random.gamma(2, 0.5, size = total_customers)
service_time = lambda : expon.rvs(scale = s, size = total_customers)
blocked_erlang = main_loop(arrival_interval, service_time, m)
print("Blocking probability: ", blocked_erlang/total_customers)
print("Mean blocking probability: ", np.mean(blocked_erlang/total_customers))
```

```
Blocking probability:  [0.1293 0.1192 0.117  0.1172 0.1246 0.1248 0.1026 0.1302 0
.1202 0.1262]
Mean blocking probability:  0.12113000000000003
```

Answer

When the inter arrival time is Erlang distributed with mean 1 time unit, the fraction of blocked customers is 0.1211 which does correspond with the theoretical value of 0.1216.

In [ ]:
```
# hyper exponential inter arrival times. The parameters for
#the hyper exponential distribution should be
np.random.seed(1)
p1 = 0.8
λ1 = 0.8333
p2 = 0.2
λ2 = 5.0
s = 8
arrival_interval = lambda : np.random.choice([expon.rvs(scale = 1/λ1), expon.rvs

service_time = lambda : expon.rvs(scale = s, size = total_customers)

blocked_hyperexp = main_loop(arrival_interval,service_time, m)
print("Blocking probability: ", blocked_hyperexp/total_customers)
print("Mean blocking probability: ", np.mean(blocked_hyperexp/total_customers))
```

```
Blocking probability:  [0.3505 0.     0.81   0.474  0.5987 0.0033 0.1212 0.     0
.0347 0.8817]
Mean blocking probability:  0.32741
```

Answer For hyperexponential inter arrival time with mean 1 time unit, the fraction of blocked customers is 0.32741 which does not correspond with the theoretical value of 0.1216.

3

The arrival process is again a Poisson process like in Part 1. Experiment with different service time distributions with the same mean service time and m as in Part 1 and Part 2

a)

Constant service time

In [ ]:
```
# a) Constant service time
np.random.seed(1)
arrival_interval = lambda : np.random.exponential(1/lam, size = total_customers)
service_time = lambda : s*np.ones(total_customers)

blocked_constant = main_loop(arrival_interval,service_time, m)
print("Blocking probability: ", blocked_constant/total_customers)
print("Mean blocking probability: ", np.mean(blocked_constant/total_customers))
```

```
Blocking probability:  [0.1275 0.1158 0.1224 0.1271 0.1214 0.1166 0.1185 0.1242 0
.1255 0.1169]
Mean blocking probability:  0.12159
```

Answer

When the service time is constant, the fraction of blocked customers is 0.12159 which corresponds well with the theoretical value of 0.1216.

In [ ]:
```
# Pareto distributed service times with at least k = 1.05 and
#k = 2.05.
np.random.seed(1)
def pareto():
    beta = (k-1)/(k)*8
    Us = np.random.uniform(0, 1, total_customers)
    xs = beta/(Us**(1/k))
    return xs

k = 1.05
service_time = lambda : np.random.pareto(k, total_customers)
service_time = pareto
blocked_pareto_1 = main_loop(arrival_interval, service_time, m)
print("Blocking probability for k= 1.05: ", blocked_pareto_1/total_customers)
print("Mean blocking probability: ", np.mean(blocked_pareto_1/total_customers))
k = 2.05
service_time = lambda : np.random.pareto(k, total_customers)
service_time = pareto
blocked_pareto_2 = main_loop(arrival_interval, service_time, m)
print("Blocking probability for k= 2.05: ", blocked_pareto_2/total_customers)
print("Mean blocking probability: ", np.mean(blocked_pareto_2/total_customers))
```

```
Blocking probability for k= 1.05:  [0.0016 0.0004 0.0023 0.0006 0.0004 0.0006 0.0
008 0.0026 0.0034 0.002 ]
Mean blocking probability:  0.00147
Blocking probability for k= 2.05:  [0.122  0.1216 0.1173 0.1044 0.122  0.1268 0.1
172 0.1195 0.1223 0.113 ]
Mean blocking probability:  0.11861
```

Answer

When the service time is pareto distributed with k=1.05 the mean blocking fraction is 0.00147 which is not at all close to the theoretical value of 0.1216. For k=2.05 the blocking fraction is 0.11861.

To have an accurate mean we change β to be $\beta = \frac{k-1}{k} \cdot 8$, to ensure a mean service time of 8 time units. The result using $k = 1.05$ is heavily skewed towards not rejecting customers. The Pareto distribution with small k is difficult to sample enough large values from, to actually see a mean service time of 8 time units, so we see a lot of small service times, resulting in few blocks. The effect is gone once $k > 2$.

In [ ]:
```
#absolute gaussian distributed service times with mean s and standard deviation .
np.random.seed(1)
service_time = lambda : np.random.normal(s, s/4, size = total_customers)
blocked_gauss = main_loop(arrival_interval, service_time, m)
print("Blocking probability: ", blocked_gauss/total_customers)
print("Mean blocking probability: ", np.mean(blocked_gauss/total_customers))
```

```
Blocking probability:  [0.1291 0.1198 0.1193 0.1172 0.1159 0.1188 0.1258 0.1189 0
.1295 0.1292]
Mean blocking probability:  0.12235
```

Answer

When the service time is normally distributed with mean 8 time units and standard deviation 2 time units, the fraction of blocked customers is 0.12235 which corresponds well with the theoretical value of 0.1216.

4

Compare confidence intervals for Parts 1, 2, and 3 then interpret and explain differences if any.

In [ ]:
```
#show confidence intervals for all the experiments
p = erlang_b(m, A)

bs = np.array([blocked_1, blocked_erlang, blocked_hyperexp, blocked_constant, bl
bs = bs / total_customers
titles = ["Exponential", "Erlang", "Hyper exponential", "Constant", "Pareto k=1.
#print("Theoretical blocking probability",erlang_b(m, A))
#print("Confidence intervals for blocking probability")

for i, b in enumerate(bs):
    print(f"{titles[i]}: ")
    print("CI is:", confidence_intervals(b))
    if p > confidence_intervals(b)[0] and p < confidence_intervals(b)[1]:
        print("Which contains the theoretical value")
    else:
        print("Which does not contain the theoretical value")
    #print("\n")

# print("Part 1: ", confidence_intervals(blocked_1/total_customers))
# if p > confidence_intervals(blocked_1/total_customers)[0] and p < confidence_i
#     print("Which contains the theoretical value")
# else:
#     print("Which does not contain the theoretical value")
# print("Part 2 (Erlang distribution): ", confidence_intervals(blocked_erlang/to
# print("part 3 (Hyper exponential distribution): ", confidence_intervals(blocke
# print("Part 4 (Constant service time): ", confidence_intervals(blocked_constan
# print("Part 5 (Pareto distribution k=1.05): ", confidence_intervals(blocked_pa
```

```
# print("Part 5 (Pareto distribution k=2.05): ", confidence_intervals(blocked_pa
# print("Part 6 (Gaussian distribution): ", confidence_intervals(blocked_gauss/t
```

```
Exponential:
CI is: (0.11640692459344552, 0.12585307540655452)
Which contains the theoretical value
Erlang:
CI is: (0.11640692459344552, 0.12585307540655452)
Which contains the theoretical value
Hyper exponential:
CI is: (0.12340100412897476, 0.5314189958710251)
Which does not contain the theoretical value
Constant:
CI is: (0.11897467394308091, 0.1242053260569191)
Which contains the theoretical value
Pareto k=1.05:
CI is: (0.0008427693757476441, 0.002097230624252356)
Which does not contain the theoretical value
Pareto k=2.05:
CI is: (0.11494051042459581, 0.12227948957540417)
Which contains the theoretical value
Gaussian:
CI is: (0.11916570869423038, 0.12553429130576962)
Which contains the theoretical value
```

All distributions except for hyperexponential and pareto with $k = 1.05$ contain the theoretical value in their confidence interval