tmon-p2

October 25, 2023

1 MONTE CARLO TECHNIQUES

Andrés Herencia López-Menchero.

MUTECI 2023/2024.

1.1 PRACTICE 2 - Getting metrics about a M/M/1 model

Modify the code shown on the slides with the example of the M/M/1 car wash model to allow the following efficiency measures to be determined:

- L = Average number of clients in the system
- L_q = Average number of customers in queue
- W = Average time of clients in the system
- W_q = Average time of customers in queue

1.1.1 Code

[1]:
```
import numpy as np
import pandas as pd

def mm1_model(N=100, L=1/7, mu=1/5, seed = 12345, asynthotic = False):

    """
    Simulates an M/M/1 queue system and calculates some performance metrics.

    Args:
    - N (int): Number of time units to simulate. Default = 100.
    - L (float): Average arrival rate (customers per time unit). Default = 1/7.
    - mu (float): Average service rate (customers per time unit). Default = 1/5.
    - seed (int): Seed for saving the random state. Default = 12345.
    - asynthotic (boolean): if True, give the metrics when N tends to infinite.␣
 ↪If false, give the current metrics. Default = False.

    Returns:
    tuple: A tuple containing:
        - model (pandas.DataFrame): A DataFrame that records the system's state␣
 ↪at each time unit. It contains the following parameters:
            * t (float): time index
```

```
            * queue (int): number of customers in the queue in each t.
            * service (int): number of customers served by the system in each t.
            * arrivals (int): total arrivals since simulation has started.
            * stay (float): time that the system maintains its current state.
            * arrival time (float): predicted time of arrival for new customer.
            * service time (float): predicted time of end of service for␣
↪current customer.
        - Ls (float): Average number of customers in the system.
        - Lq (float): Average number of customers in the queue.
        - Ws (float): Average time customers spend in the system.
        - Wq (float): Average time customers spend in the queue.
        - Leff (float): Current lambda obtained after simulation process.

    Raises:
    Exception: An exception is raised if the arrival rate (L) is greater than␣
↪the service rate (mu), indicating that the system has not
    reached the stationary state (process explosive). Additionally, an␣
↪exception is raised when the arguments are not natural numbers or
    when L or mu are higher than 1 or lower than 0.
    """

    if L/mu > 1:
        raise Exception("The system has not reached stationary state. You must␣
↪redefine the parameters of your model.")
    if (N<=0) or type(N) != int:
        raise Exception("The simulation time must be a natural number.")
    if (L<0) or (L>1) or (mu<0) or (mu>1) or type(L) != float or type(mu) !=␣
↪float:
        raise Exception("The parameters of the random distribution are not a␣
↪number or are greater than 1 or lower than 0.")
    if seed < 0:
        raise Exception('The random seed must be a natural number.')

    np.random.seed(seed)
    arrival_time = np.random.exponential(scale=1/L)
    service_time = np.random.exponential(scale=1/mu)
    stay = 0; t = 0; queue = 0; service = 0; arrivals = 0;

    model = pd.DataFrame({
        't': [t],
        'queue': [queue],
        'service': [service],
        'arrivals': [arrivals],
        'stay': [stay],
        'arrival time': [arrival_time],
        'service time': [service_time]
```

```
    })

    while min(arrival_time, service_time) <= N:
        if arrival_time <= service_time:
            t = arrival_time
            if service > 0: # client stays in the queue
                queue += 1
            else: # a client can be served in this moment
                service = 1
            arrivals += 1
            arrival_time = t + np.random.exponential(scale=1/L)
        else:
            t = service_time
            if queue > 0: # client is dispatched from the queue to the server
                queue -= 1
                service_time = t + np.random.exponential(scale=1/mu)
            else: # no one in queue and previous service has finished, system␣
↪at rest
                service = 0
                service_time = arrival_time + np.random.exponential(scale=1/mu)
        stay = min(arrival_time, service_time) - t

        new_register = pd.DataFrame({
            't': [t],
            'queue': [queue],
            'service': [service],
            'arrivals': [arrivals],
            'stay': [stay],
            'arrival time': [arrival_time],
            'service time': [service_time]
        })
        model = pd.concat([model, new_register], ignore_index=True)

    L_eff = 1/np.mean(np.diff(model['arrival time'].unique()))

    queue_time, wait_times = [], []
    for q in range(1,len(model['queue'])):
        if model['queue'][q] > model['queue'][q-1]:
            queue_time.append(model['t'][q])
        elif model['queue'][q] < model['queue'][q-1]:
            wait_times.append(model['t'][q] - queue_time.pop(0))

    Wq = np.mean(wait_times) # only time in the queue

    if asynthotic == True:
        Ws = Wq + 1/mu
        Ls = L_eff * Ws
```

```
        Lq = L_eff * Wq
    else:
        Ws = np.mean(np.diff(model['service time'])) + Wq
        Ls = np.mean(model['queue']) + np.mean(model['service'])
        Lq = np.mean(model['queue'])

    return model, Ls, Lq, Ws, Wq, L_eff
```

1.1.2 Model

[2]:
```
[model, Ls, Lq, Ws, Wq, L_eff] = mm1_model(N = 10000, L = 1/7, mu = 1/5, seed =␣
 ↪12345)
model
```

[2]:
```
                t  queue  service  arrivals       stay  arrival time  \
0        0.000000      0        0         0   0.000000     18.576534
1        1.901733      0        0         0  16.674802     18.576534
2       18.576534      0        1         1   1.016207     20.178556
3       19.592742      0        0         1   0.585814     20.178556
4       20.178556      0        1         2   4.193467     26.515054
…             …     …       …         …          …              …
2915  9990.280429      3        1      1459   1.195817  10007.160810
2916  9991.476246      2        1      1459   1.744110  10007.160810
2917  9993.220355      1        1      1459   2.187471  10007.160810
2918  9995.407826      0        1      1459   2.485316  10007.160810
2919  9997.893142      0        0      1459   9.267668  10007.160810

      service time
0         1.901733
1        19.592742
2        19.592742
3        24.372023
4        24.372023
…              …
2915   9991.476246
2916   9993.220355
2917   9995.407826
2918   9997.893142
2919  10009.064305

[2920 rows x 7 columns]
```

1.1.3 Metrics

L = Average number of clients in the system

[3]:
```
round(Ls,4)
```

[3]: 2.8106

L_q = **Average number of customers in the queue**

[4]:
```
round(Lq,4)
```

[4]: 1.9562

W = **Average time of clients in the system (minutes)**

[5]:
```
round(Ws,4)
```

[5]: 19.3866

W_q = **Average time of clients in the queue (minutes)**

[6]:
```
round(Wq,4)
```

[6]: 15.9583

NOTE: There is a difference between the actual metrics and the theorical metrics. This is due to we need a infinite number of time to achieve the theorical results. To obtain the theorical metrics can be approximated via the `asynthotic` param which computes the parameters based on the Little's formulas. These values are:

[7]:
```
mm1, Lt, Lqt, Wt, Wqt, L_eff = mm1_model(N = 10000, asynthotic=True)
print('L: ', Lt, '. Difference with actual value: ',round((Lt/Ls-1)*100,2),'%'
      '\nLq: ', Lqt, '. Difference with actual value: ',round((Lqt/
 ↪Lq-1)*100,2),'%',
      '\nW: ', Wt, '(min). Difference with actual value: ',round((Wt/
 ↪Ws-1)*100,2),'%',
      '\nWq: ', Wqt, '(min). No diference with actual value since it is the␣
 ↪reference to compute the other metrics.',
      '\nLambda (effective): ', L_eff,'. Difference with actual value ',␣
 ↪round(((1/7)/L_eff-1)*100,2),'%')
```

```
L:  3.0613175048468553 . Difference with actual value:  8.92 %
Lq:  2.3309837759820593 . Difference with actual value:  19.16 %
W:  20.9583467383141 (min). Difference with actual value:  8.11 %
Wq:  15.9583467383141 (min). No diference with actual value since it is the
reference to compute the other metrics.
Lambda (effective):  0.14606674577295925 . Difference with actual value  -2.2 %
```