Week3_Lab_Discrete_Time_Markov_Chains_part2

October 18, 2018

MOOC: Understanding queues
Python simulations
Week III - Part II: Discrete-time Markov chains

In this second part of the lab, we will study the token bucket mechanism. It uses tokens to ensure that the rate of clients entering a system is limited in order to avoid congestion in the system.

In this context, we will show how a Markov chain behavior can be exhibited and study some properties of the system.

Notes - To display the token bucket and transition diagram in your noteboook , do not forget to also upload the corresponding .png files. - This lab has been inspired by an exercise from the queueing theory book by Bruno Baynat "Théorie des files d'attente: des chaînes de Markov aux réseaux à forme produit", Hermès Science Publications (ISBN:978-2-7462-0120-0).

1 Token Bucket

Let us consider a system to which clients arrive according to a Poisson process with rate $\lambda = 1.5$ arrivals per time unit. To regulate access into the system, a token-based mechanism is set up before the system.

To enter the system, a client must get a token. Tokens are stored in a bucket which can contain up to $K = 3$ tokens. If the bucket is not empty when a client arrives then the client takes one token and enters the system. But if the bucket is empty then the client is lost. One new token is created each $T = 1$ time unit.

We are going to study some properties of this system. For example, we would like to evaluate the loss probability of clients, that is to say the probability that the bucket is empty when a new client arrives.

First we use an analytical approach. It can be remarked that the number of tokens in the bucket just after a new token has been produced is a discrete time Markov chain (DTMC). The loss probability of clients can be derived from the steady state distribution of this DTMC. Then we use simulations to estimate this loss probability and check that the estimated value is close to the theoretical one.

1.1 Questions

We assume again that the capacity of the bucket is limited to $K = 3$ tokens, $T = 1$ and $\lambda = 1.5$.

1) Let α_n be the probability that n clients arrive during a slot of T time units and let β_n be the probability that **at least** n clients arrive during a slot of T time units.

Knowing that the arrival process of clients is a Poisson process, what are the expressions of α_n and β_n?

2) Explain why X is a discrete-time Markov chain. What are the possible values for X_n? In particular is it possible that $X_n = 0$?

3) If $X_n = 1$ what are the possible values for X_{n+1} and what are the corresponding transition probabilities? Same question for $X_n = 2$ and $X_n = 3$. Draw the transition diagram of X and give its transition probability matrix P.

4) Let $[\pi_1, \pi_2, \pi_3]$ denote steady-state probability distribution of X. What is the linear system of equations satisfied by $[\pi_1, \pi_2, \pi_3]$. Complete the code section to compute $[\pi_1, \pi_2, \pi_3]$.

5) What are the two conditions under which a token is lost? What is the probability that a token is lost? Compute this probability.

6) What is the rate of consumed tokens (unit: tokens/s)? What is the rate of clients entering the system? Compute the client loss probability.

7) We now use simulations to estimate this loss probability of clients and check that the estimated value is close to the theoretical one. Consider the code of the function **token_bucket** below. It simulates client arrivals and the operation of the token bucket mechanism. Complete and run the code. Estimate the loss

---

1.2 Answers

1.2.1 Answer to question 1

As the clients arrival process is a Poisson process with rate λ, $\alpha_n = \frac{(\lambda T)^n}{n!} e^{-\lambda T}$. $\beta_n = 1 - \sum_{i=0:n-1} \alpha_i$.

1.2.2 Answer to question 2

For any $n \geq 0$, X_n depends only on X_{n-1} and on the random number of clients that arrive in the time interval $]t_{n-1}, t_n]$. As the clients arrival process is a Poisson process, the number of arrivals on each time slot of length T is independent from each other. Consequently, knowing X_{n-1}, X_n is independent from $X_{n-2}, X_{n-3}, \ldots$ and X is a discrete time Markov chain.

X_n can be equal to 1,2 or 3. It cannot be equal to 0 since X_n is the number of tokens in the bucket just after the production of a new token.

1.2.3 Answer to question 3

If $X_n = 1$ and if no client arrives on $]t_n, t_{n+1}]$ then $X_{n+1} = 2$ as a new token is produced at time t_{n+1}. The corresponding probability is α_0, so $P(X_{n+1} = 2 \mid X_n = 1) = \alpha_0$.

If $X_n = 1$ and it at least one client arrives on $]t_n, t_{n+1}]$ then the first client consumes the token, the other clients are lost, a new token is produced at time t_{n+1} and $X_{n+1} = 1$. The corresponding probability is β_1, so $P(X_{n+1} = 1 \mid X_n = 1) = \beta_1$. One can observe that $\alpha_0 + \beta_1 = 1$ so that the conditional probabilities sum up to 1.

In the same way, if $X_n = 2$ it comes that X_{n+1} can be equal to 1, 2 or 3, depending on the number of tokens consumed in the time interval $]t_n, t_{n+1}]$.

$$\begin{aligned} P(X_{n+1} = 1 \mid X_n = 2) &= P(\text{"2 arrivals or more"}) = \beta_2 \\ P(X_{n+1} = 2 \mid X_n = 2) &= P(\text{"1 arrival"}) = \alpha_1 \\ P(X_{n+1} = 3 \mid X_n = 2) &= P(\text{"0 arrival"}) = \alpha_0 \end{aligned}$$

If $X_n = 3$ it is possible that on the next slot a token is lost. If no client arrives then the next token is lost because the bucket is full and $X_{n+1} = 3$. If 1 client arrives then $X_{n+1} = 3$ as well, since one token is consumed and one token is produced. So, $P(X_{n+1} = 3 \mid X_n = 3) = P(\text{"0 or 1 arrival"}) = \alpha_0 + \alpha_1$.

If $X_n = 3$ and 3 clients or more arrive on the next slot then $X_{n+1} = 1$: $P(X_{n+1} = 1 \mid X_n = 3) = \beta_3$. And if $X_n = 3$ and 2 clients arrive on the next slot then $X_{n+1} = 2$ so that $P(X_{n+1} = 2 \mid X_n = 3) = \alpha_2$.

Again, it can be remarked that $\alpha_0 + \alpha_1 + \alpha_2 + \beta_3 = 1$ so that we can be sure that we have not forgotten any possible case.

Transition diagram of the chain:

where the $P_{ij} = P(X_{n+1} = j | X_n = i)$ are given by

$$\begin{array}{lll} P_{11} & = \beta_1 = 1 - \alpha_0 & = 1 - e^{-\lambda T} \\ P_{22} & = \alpha_1 & = (\lambda T)e^{-\lambda T} \\ P_{33} & = \alpha_0 + \alpha_1 & = (1 + \lambda T)e^{-\lambda T} \\ P_{12} = P_{23} & = \alpha_0 & = e^{-\lambda T} \\ P_{21} & = \beta_2 = 1 - \sum_{i=0}^{1} \alpha_i & = 1 - (1 + \lambda T)e^{-\lambda T} \\ P_{32} & = \alpha_2 & = \dfrac{(\lambda T)^2}{2} e^{-\lambda T} \\ P_{13} & = 0 & \text{(since } X_{n+1} \leq X_n + 1 \text{ with probability 1)} \\ P_{31} & = \beta_3 = 1 - \sum_{i=0}^{2} \alpha_i & = 1 - (1 + \lambda T + \dfrac{(\lambda T)^2}{2})e^{-\lambda T} \end{array}$$

1.2.4 Answer to question 4

The steady state distribution $[\pi_1, \pi_2, \pi_3]$ satisfies equations $\sum_{i=1}^{3} \pi_i P_{ij} = \pi_j$ for $j = 1, 2, 3$, that is, $\pi(P - I) = 0$ (with I the identity matrix) and $\sum_{i=1}^{3} \pi_i = 1$:

$$\left\{ \begin{array}{cccc} -e^{-\lambda T}\pi_1 & +[1 - (1 + \lambda T)e^{-\lambda T}]\pi_2 & +[1 - (1 + \lambda T + \dfrac{(\lambda T)^2}{2})e^{-\lambda T}]\pi_3 & = 0 \\ e^{-\lambda T}\pi_1 & +[(\lambda T)e^{-\lambda T} - 1]\pi_2 & +\dfrac{(\lambda T)^2}{2} e^{-\lambda T}\pi_3 & = 0 \\ 0 & +e^{-\lambda T}\pi_2 & +[(1 + \lambda T)e^{-\lambda T} - 1]\pi_3 & = 0 \\ \pi_1 & +\pi_2 & +\pi_3 & = 1 \end{array} \right.$$

Note that the sum of the second and third equations equals the opposite of the first equation. Thus, the first equation is redundant and it can be discarded. Then, looking for the solution of the 3 last equations yields $[\pi_1, \pi_2, \pi_3] = [0.616, 0.255, 0.129]$ (see code below).

```
In [ ]: %matplotlib inline
        from pylab import *

In [ ]: from scipy.linalg import solve
        # To use solve, you can have a look at
```

```
# https://docs.scipy.org/doc/scipy/reference/tutorial/linalg.html#solving-linear-system
T   = 1
   = 1.5
a   = *T
e_a = exp(-a)
A   = array([[e_a,  a*e_a-1, a**2/2*e_a],
             [0,     e_a,     (1+a)*e_a-1],
             [1,     1,       1      ]])
####################################
# give the expression of the steady-state distribution
 = ...
print('pi=',)
####################################

#----------
V1 = [2]
```

1.2.5 Answer to question 5

The token produced at time t_{n+1} is lost if (i) the bucket was full at time t_n and no client has arrived between t_n and t_{n+1}. Since both events are independent, the corresponding probability is $\pi_3\alpha_0$.

In [ ]:
```
######################
# Enter P_loss_token
P_loss_token = ...
######################
V2 = P_loss_token
```

1.2.6 Answer to question 6

The rate of tokens production is $1/T$. As these tokens are lost with probability $\pi_3\alpha_0$, the rate of consumed tokens is $(1-\pi_3\alpha_0)/T$. As each client entering the system consumes one token, the rate of clients entering the system is also $\lambda_e = (1-\pi_3\alpha_0)/T$. Thus, the rate of clients lost is $\lambda - \lambda_e$ and the loss probability of clients is $(\lambda - \lambda_e)/\lambda$.

In [ ]:
```
######################
# Enter P_loss_client
P_loss_client = ...
######################
V3 = P_loss_client
```

1.2.7 Answer to question 7

In [ ]:
```
      = 1.5  # intensity of clients arrivals at the bucket
T     = 1.   # generation period of tokens
Bsize = 3    # bucket size

def token_bucket(x0=1, Tmax=100):
    x       = [x0]  # bucket states at instants 0, t_1, t_2, ...
```

```
                    # (just after the production of tokens)
    x_aux   = x[-1] # bucket state between t_n and t_{n+1}
    y       = []    # clients indicator: 1 --> enter the system, 0-->rejected)
                    # y permits
           = [(-1/)*log(rand())] # times of client arrivals
    n       = 0     # number of elapsed intervals of duration T
    while [-1]<Tmax:
        # generate tokens until next client arrival:
        while [-1]> n*T+T:
            n +=1 # increment time index
            # update the bucket state (add a new token in the bucket
            # if it is not full yet):
            x.append(minimum(x_aux+1,Bsize))
            x_aux = x[-1]
        # check whether the new client enters the system or not:
        if x_aux>0: # the new client enters the system
            x_aux = x_aux-1
            y.append(1)
        else:       # the new client is rejected
            y.append(0)
        ####################################
        # generate time of arrival of a new client:
        .append(...)
        ####################################
     = [:-1] # discard time of arrival > Tmax
    return x, y,

x, y,  = token_bucket()
figure(figsize=(12,4))
subplot(121)
step(T*arange(len(x)),x)
axis(ymin=0,ymax=4)
yticks([1,2,3],[1,2,3])
title("Bucket state")
subplot(122)
plot(,y,'r.')
axis(ymin=-.5,ymax=1.5)
yticks([0,1],[0,1])
title("Clients acceptation (0: discarded, 1: accepted)")

#------
x,y, = token_bucket(Tmax=10**4)
#########################
# Supply the estimate of clients loss probability
V4    = ...
#########################
print("Estimated clients loss probability = ",V4)
```

Remark It can be noticed that from the point of view of tokens, the bucket is a $D/M/1/K$ queue (with $K = 3$). Indeed, tokens arrive at deterministic times nT and the time necessary to consume 1 token has distribution $Exp(\lambda)$. The $D/M/1/K$ queue is not a Markovian queue: the number of tokens in the bucket is not a continuous time Markov chain. However, at times $t_n = (nT)^+$, the number of tokens in the bucket is a discrete time Markov chain and we used this property to solve the problem.

2 Conclusion

This token bucket example illustrates that it is sometimes possible to compute quite simply some parameters such as the steady state distribution from a simulation, while analytical derivation can be somewhat more tedious. However, perhaps you noticed that writing a function that correctly simulates the phenomenon under study requires a precise understanding of its evolution mechanisms and care at programming. You will have more opportunities to simulate other Markovian evolutions in the next labs.

3 Your answers for this notebook

```
In [ ]: print("-----------------------------------------------------\n"
              +"VALIDITY OF RESULTS SUPPLIED FOR WEEK III - Part II:\n"
              +"-----------------------------------------------------")
        results  = dict()
        for k in range(1,5):
            results["V"+str(k)] = "NO"
        try:
            if abs(V1-.13)<.01:    results["V1"] =  "OK"
        except: pass
        try:
            if abs(V2-.029)<.001:  results["V2"] =  "OK"
        except: pass
        try:
            if abs(V3-.35)<.01:    results["V3"] =  "OK"
        except: pass
        try:
            if abs(V4-.35)<.02:     results["V4"] =  "OK"
        except: pass

        for key,val in results.items():
            print(key,': ',val)
```