

# Analysis of Queueing Systems with Sample Paths and Simulation

Nicky D. van Foreest

October 6, 2017



# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
<b>2</b>	<b>Single-Station Queueing Systems</b>	<b>9</b>
2.1	Exponential Distribution . . . . .	9
2.2	Poisson Distribution . . . . .	20
2.3	Kendall's Notation to Characterize Queueing Processes . . . . .	29
2.4	Construction of Discrete-Time Queueing Processes . . . . .	31
2.5	Construction of the $G/G/1$ Queueing Process in Continuous Time . . . . .	46



# 1 Introduction

**Motivation and Examples** Queueing systems abound, and the analysis and control of queueing systems are major topics in the control, performance evaluation and optimization of production and service systems.

At my local supermarket, for instance, any customer that joins a queue of 4 or more customers get his/her shopping for free. Of course, there are some constraints: at least one of the cashier facilities has to be unoccupied by a server and the customers in queue should be equally divided over the cashiers that are open. (And perhaps there are some further rules, of which I am unaware.) The manager that controls the occupation of the cashier positions is focused on keeping  $\pi(4) + \pi(5) + \dots$ , i.e., the fraction of customers that see upon arrival a queue length longer or equal than 4, very small. In a sense, this is easy enough: just hire many cashiers. However, the cost of personnel may then outweigh the yearly average cost of paying the customer penalties. Thus, the manager's problem becomes to plan and control the service capacity in such a way that both the penalties and the personnel cost are small.

Fast food restaurants also deal with many interesting queueing situations. Consider, for instance, the making of hamburgers. Typically, hamburgers are made-to-stock, in other words, they are prepared before the actual demand has arrived. Thus, hamburgers in stock can be interpreted as customers in queue waiting for service, where the service time is the time between the arrival of two customers that buy hamburgers. The hamburgers have a typical lifetime, and they have to be scrapped if they remain on the shelf longer than some amount of time. Thus, the waiting time of hamburgers has to be closely monitored. Of course, it is easy to achieve zero scrap cost, simply by keeping no stock at all. However, to prevent lost-sales it is very important to maintain a certain amount of hamburgers on stock. Thus, the manager has to balance the scrap cost against the cost of lost sales. In more formal terms, the problem is to choose a policy to prepare hamburgers such that the cost of excess waiting time (scrap) is balanced against the cost of an empty queue (lost sales).

Service systems, such as hospitals, call centers, courts, and so on, have a certain capacity available to serve customers. The performance of such systems is, in part, measured by the total number of jobs processed per year and the fraction of jobs processed within a certain time between receiving and closing the job. Here the problem is to organize the capacity such that the sojourn time, i.e., the typical time a job spends in the system, does not exceed some threshold, and such that the system achieves a certain throughput, i.e., jobs served per year.

Clearly, all the above systems can be seen as queueing systems that have to be monitored and controlled to achieve a certain performance. The performance analysis of such systems can, typically, be characterized by the following performance measures:

1. The fraction of time  $p(n)$  that the system contains  $n$  customers. In particular,  $1 - p(0)$ , i.e., the fraction of time the system contains jobs, is important, as this is a measure of the time-average occupancy of the servers, hence related to personnel cost.
2. The fraction of customers  $\pi(n)$  that 'see upon arrival' the system with  $n$  customers. This measure relates to customer perception and lost sales, i.e., fractions of arriving customers

## 1 Introduction

that do not enter the system.

3. The average, variance, and/or distribution of the waiting time.
4. The average, variance, and/or distribution of the number of customers in the system.

Here the system can be anything that is capable of holding jobs, such as a queue, the server(s), an entire court house, patients waiting for an MRI scan in a hospital, and so on.

It is important to realize that a queueing system can, typically, be decomposed into *two subsystems*, the queue itself and the service system. Thus, we are concerned with three types of waiting: waiting in queue, i.e., *queueing time*, waiting while being in service, i.e., the *service time*, and the total waiting time in the system, i.e., the *sojourn time*.

**Organization** In these notes we will be primarily concerned with making models of queueing systems such that we can compute or estimate the above performance measures. Part of our work is to derive analytic models. The benefit of such models is that they offer structural insights into the behavior of the system and scaling laws, such as that the average waiting time scales (more or less) linearly in the variance of the service times of individual customers. However, these models have severe shortcomings when it comes to analyzing real queueing systems, in particular when particular control rules have to be assessed. Consider, for example, the service process at a check-in desk of KLM. Business customers and economy customers are served by two separate queueing systems. The business customers are served by one server, server A say, while the economy class customers by three servers, say. What would happen to the sojourn time of the business customers if server A would be allowed to serve economy class customers when the business queue is empty? For the analysis of such cases simulation is a very useful and natural approach.

In the first part of these notes we concentrate on the analysis of *sample paths of a queueing process*. We assume that a typical sample path captures the ‘normal’ stochastic behavior of the system. This sample-path approach has two advantages. In the first place, most of the theoretical results follow from very concrete aspects of these sample paths. Second, the analysis of sample-paths carries over right away to simulation. In fact, simulation of a queueing system offers us one (or more) sample paths, and based on such sample paths we derive behavioral and statistical properties of the system. Thus, the performance measures defined for sample paths are precisely those used for simulation. Our aim is not to provide rigorous proofs for all results derived below; for the proofs and further background discussion we refer to ?. As a consequence we tacitly assume in the remainder that results derived from the (long-run) analysis of a particular sample path are equal to their ‘probabilistic counterpart’.

In the second part we construct algorithms to analyze open and closed queueing networks. Many of the sample path results developed for the single-station case can be applied to these networks. As such, theory, simulation and algorithms form a nicely round out part of work. For this part we refer to book of Prof. Zijm; the present set of notes augment the discussion there.

**Exercises** I urge you to make *all* exercises in this set of notes. Many exercises require many of the tools you learned previously in courses on calculus, probability, and linear algebra. Here you can see them applied. Moreover, many of these tools will be useful for other, future, courses. Thus, the investments made here will pay off for the rest of your (student) career. Moreover, the exercises are meant to *illustrate* the material and to force you to *think* about it. Thus, the main text does not contain many examples; the exercises form the examples.

You'll notice that many of these problems are quite difficult, often not because the problem itself is difficult, but because you need to combine a substantial amount of knowledge all at the same time. All this takes time and effort. Next to this, I did not include the exercises with the intention that you would find them easy. The problems should be doable, but hard.

The solution manual is meant to prevent you from getting stuck and to help you increase your knowledge of probability, linear algebra, programming (analysis with computer support), and queueing in particular. Thus, read the solutions very carefully.

As a guideline to making the exercises I recommend the following approach. First read the notes. Then attempt to make a exercises for 10 minutes or so by yourself. If by that time you have not obtained a good idea on how to approach the problem, check the solution manual. Once you have understood the solution, try to repeat the arguments *with the solution manual closed*.

**Symbols** The meaning of the symbols in the margin of pages are as follows:

- The symbol in the margin means that you have to memorize the *emphasized concepts*.
- The symbol in the margin means that this question has a *hint*.
- The symbol in the margin means that this question or its solution requires still some *work on my part*; you can skip it.



**Acknowledgments** I would like to acknowledge dr. J.W. Nieuwenhuis for our many discussions on the formal aspects of queueing theory and prof. dr. W.H.M. Zijm for allowing me to use the first few chapters of his book. Finally, without ? I could not have written these notes.





## 2 Single-Station Queueing Systems

In this chapter, we start with a discussion of the exponential distribution and the related Poisson process, as these concepts are perhaps the most important building blocks of queueing theory. With these concepts we can specify the arrival and service processes of customers, so that we can construct queueing processes and define performance measures to provide insight into the (transient and average) behavior of queueing processes. As it turns out, these constructions can be easily implemented as computer programs, thereby allowing to use simulation to analyze queueing systems. We then continue with developing models for various single-station queueing systems in steady-state, which is, in a sense to be discussed later, the long-run behavior of a stochastic system.<sup>1</sup> In the analysis we use sample-path arguments to count how often certain events occur as functions of time. Then we define probabilities in terms of limits of fractions of these counting processes. Another useful aspect of sample-path analysis is that the definitions for the performance measures are entirely constructive, hence by leaving out the limits, they provide expressions that can be right away used in statistical analysis of (simulations of) queueing systems. Level-crossing arguments will be of particular importance as we use these time and again to develop recursions by which we can compute steady-state probabilities of the queue length or waiting time process.

### 2.1 Exponential Distribution

As we will see in the sections to come, the modeling and analysis of any queueing system involves the specification of the (probability) distribution of the time between consecutive arrival epochs of jobs, or the specification of the distribution of the number of jobs that arrive in a certain interval. For the first case, the most common distribution is the exponential distribution, while for the second it is the Poisson distribution. For these reasons we start our discussion of the analysis of queueing system with these two exceedingly important distributions. In the ensuing sections we will use these distributions time and again.

As mentioned, one of the most useful models for the interarrival times of jobs assumes that the sequence  $\{X_i\}$  of interarrival times is a set of *independent and identically distributed (i.i.d.)* random variables. Let us write  $X$  for the generic random time between two successive arrivals. For many queueing systems, measurements of the interarrival times between consecutive arrivals show that it is reasonable to model an interarrival  $X$  as an *exponentially distributed* random variable, i.e.,

$$P\{X \leq t\} = 1 - e^{-\lambda t} := G(t)$$

The constant  $\lambda$  is often called the *rate*. The reason behind this will be clarified once we relate the exponential distribution to the Poisson process in Section 2.2. In the sequel we often write  $X \sim \exp(\lambda)$  to mean that  $X$  is exponentially distributed with rate  $\lambda$ .

Let us show with simulation how the exponential distribution originates. Consider  $N$  people that regularly visit a shop. We assume that we can characterize the interarrival times  $\{X_k^i, k =$

---

<sup>1</sup>This statement is, admittedly, vague, to say the least.

## 2 Single-Station Queueing Systems

$1, 2, \dots\}$  of customer  $i$  by some distribution function, for instance the uniform distribution. Then, with  $A_0^i = 0$  for all  $i$ ,

$$A_k^i = A_{k-1}^i + X_k^i = \sum_{j=1}^n X_j^i,$$

is the arrival moment of the  $k$ th visit of customer  $i$ . Now the shop owner ‘sees’ the superposition of the arrivals of all customers. One way to compute the arrival moments of all customers together as seen by the shop is to put all the numbers  $\{A_k^i, k = 1, \dots, n, i = 1, \dots, N\}$  into one set, and sort these numbers in increasing order. This results in the (sorted) set of arrival times  $\{A_k, k = 1, 2, \dots\}$  at the shop, and then

$$X_k = A_k - A_{k-1},$$

with  $A_0 = 0$ , must be the interarrival time between the  $k - 1$ th and  $k$ th visit to the shop. Thus, starting from interarrival times of individual customers we can construct interarrival times as seen by the shop.

To plot the *empirical distribution function*, or the histogram, of the interarrival times at the shop, we need to count the number of interarrival times smaller than time  $t$  for any  $t$ . For this, we introduce the *indicator function*:

$$\mathbb{1}_A = \begin{cases} 1, & \text{if the event } A \text{ is true,} \\ 0, & \text{if the event } A \text{ is false.} \end{cases}$$

With the indicator function we define the empirical distribution of the interarrival times for a simulation with a total of  $n \cdot N$  as

$$P_{nN}\{X \leq t\} = \frac{1}{nN} \sum_{k=1}^{nN} \mathbb{1}_{X_k \leq t},$$

where  $\mathbb{1}_{X_k \leq t} = 1$  if  $X_k \leq t$  and  $\mathbb{1}_{X_k \leq t} = 0$  if  $X_k > t$ .

Let us now compare the probability density as obtained for several simulation scenarios to the density of the exponential distribution, i.e., to  $\lambda e^{-\lambda t}$ . As a first example, take  $N = 1$  customer and let the computer generate  $n = 100$  uniformly distributed numbers on the set  $[4, 6]$ . Thus, the time between two visits of this customer is somewhere between 4 and 6 hours, and the average interarrival times  $E\{X\} = 5$ . In a second simulation we take  $N = 3$  customers, and in the third,  $N = 10$  customers. The empirical distributions are shown, from left to right, in the three panels in Figure 2.1. The continuous curve is the graph of  $\lambda e^{-\lambda x}$  where  $\lambda = N/E\{X\} = N/5$ . (In Eq. (??) we show that when 1 person visits the shop with an average interarrival time of 5 hours, it must be that the arrival rate is  $1/E\{X\} = 1/5$ . Hence, when  $N$  customers visit the shop, each with an average interarrival time of 5 hours, the total arrival rate as seen by the shop must be  $N/5$ .) As a second example, we extend the simulation to  $n = 1000$  visits to the shop, see Figure 2.2. In the third example we take the interarrival times to be normally distributed times with mean 5 and  $\sigma = 1$ , see Figure 2.3.

As the graphs show, even when the customer population consists of 10 members, each visiting the shop with an interarrival time that is quite ‘far’ from exponential, the distribution of the interarrival times as observed by the shop is very well approximated by an exponential distribution. Thus, for a real shop, with many thousands of customers, or a hospital, call center, in fact for nearly every system that deals with random demand, it seems reasonable to use the exponential distribution to model interarrival times. In conclusion, the main conditions to use

## 2.1 Exponential Distribution

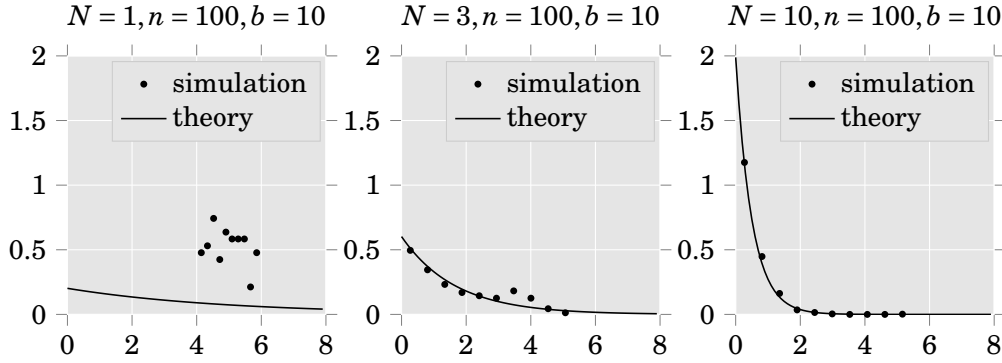


Figure 2.1: The interarrival process as seen by the shop owner. Observe that the density  $\lambda e^{-\lambda x}$  intersects the  $y$ -axis at level  $N/5$ , which is equal to the arrival rate when  $N$  persons visit the shop. The parameter  $n = 100$  is the simulation length, i.e., the number of visits per customer, and  $b = 10$  is number of bins to collect the data.

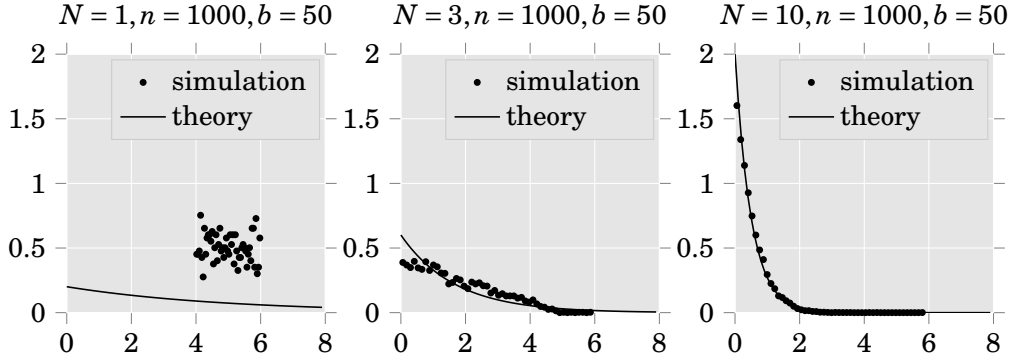


Figure 2.2: Each of the  $N$  customer visits the shop with exponentially distributed interarrival times, but now the number of visits is  $n = 1000$ .

an exponential distribution are: 1) arrivals have to be drawn from a large population, and 2) each of the arriving customers decides, independent of the others, to visit the system.

Another, but theoretical, reason to use the exponential distribution is that an exponentially distributed random variable is *memoryless*, that is,  $X$  is memoryless if it satisfies the property that

$$P\{X > t + h | X > t\} = P\{X > h\}.$$

In words, the probability that  $X$  is larger than some time  $t + h$ , conditional on it being larger than a time  $t$ , is equal to the probability that  $X$  is larger than  $h$ . Thus, no matter how long we have been waiting for the next arrival to occur, the probability that it will occur in the next  $h$  seconds remains the same. This property seems to be vindicated also in practice: suppose that a patient with a broken arm just arrived at the emergency room of a hospital, what does that tell us about the time the next patient will be brought in? Not much, as most of us will agree.

It can be shown that only exponential random variables have the memoryless property. The proof of this fact requires quite some work; we refer the reader to the literature if s/he want to check this, see e.g. [1], Appendix 3.

## 2 Single-Station Queueing Systems

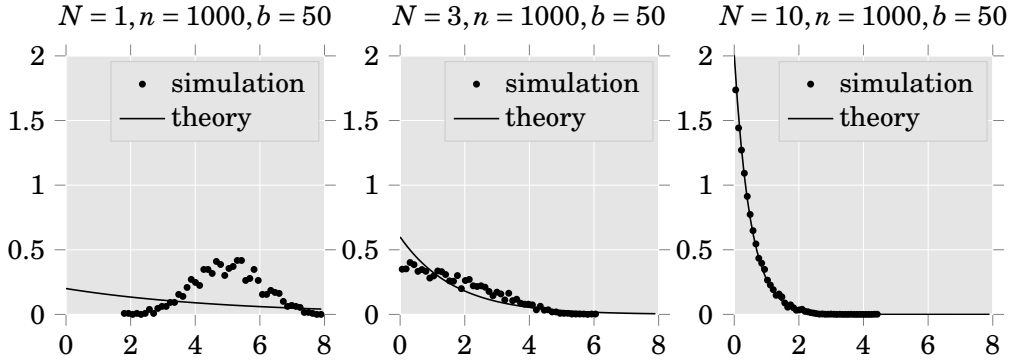


Figure 2.3: Each of the  $N$  customer visits the shop with normally distributed interarrival times with  $\mu = 5$  and  $\sigma = 1$ .

Finally, the reader should realize that it is simple, by means of computers, to generate exponentially distributed interarrival times. Thus, it is easy to use such exponentially distributed random variables to simulate queueing systems.

### Exercises

**Exercise 2.1.1.** (Using conditional probability) We have to give one present to one of three children. As we cannot divide the present into parts, we decide to let ‘fate decide’. That is, we choose a random number in the set  $\{1, 2, 3\}$ . The first child that guesses the number wins the present. Show that the probability of winning the present is the same for each child.

**Exercise 2.1.2.** Assume that the time  $X$  to fail of a machine is uniformly distributed on the interval  $[0, 10]$ . If the machine fails at time  $t$ , the cost to repair it is  $h(t)$ . What is the expected repair cost?

**Exercise 2.1.3.** Show that an exponentially distributed random variable is memoryless.

**Exercise 2.1.4.** If the random variable  $X \sim \exp(\lambda)$ , show that

$$\mathbb{E}\{X\} = \int_0^\infty t \, dF(t) = \int_0^\infty t f(t) \, dt = \int_0^\infty t \lambda e^{-\lambda t} \, dt = \frac{1}{\lambda},$$

where  $f$  is the density function of the distribution function  $F$  of  $X$ .

**Exercise 2.1.5.** If the random variable  $X \sim \exp(\lambda)$ , show that

$$\mathbb{E}\{X^2\} = \int_0^\infty t^2 \lambda e^{-\lambda t} \, dt = \frac{2}{\lambda^2}.$$

**Exercise 2.1.6.** If the random variable  $X \sim \exp(\lambda)$ , show that the *variance*

$$\mathbb{V}\{X\} = \mathbb{E}\{X\}^2 - (\mathbb{E}\{X\})^2 = \frac{1}{\lambda^2}.$$

Recall in particular this middle term to compute  $\mathbb{V}\{X\}$ ; it is very practical.



**Exercise 2.1.7.** Define the *square coefficient of variation (SCV)* as

$$C_a^2 = \frac{V\{X\}}{(E\{X\})^2}. \quad (2.1)$$

Prove that when  $X$  is exponentially distributed,  $C_a^2 = 1$ . As will become clear later, the SCV is a very important concept in queueing theory. Memorize it as a measure of *relative variability*.

**Exercise 2.1.8.** If  $X$  is an exponentially distributed random variable with parameter  $\lambda$ , show that its moment generating function

$$M_X(t) = E\{e^{tX}\} = \frac{\lambda}{\lambda - t}.$$

**Exercise 2.1.9.** Let  $A_i$  be the arrival time of customer  $i$  and set  $A_0 = 0$ . Assume that the interarrival times  $\{X_i\}$  are i.i.d. with exponential distribution with mean  $1/\lambda$  for some  $\lambda > 0$ . Prove that the density of  $A_i = X_1 + X_2 + \cdots + X_i = \sum_{k=1}^i X_k$  with  $i \geq 1$  is

$$f_{A_i}(t) = \lambda e^{-\lambda t} \frac{(\lambda t)^{i-1}}{(i-1)!}.$$

**Exercise 2.1.10.** Assume that the interarrival times  $\{X_i\}$  are i.i.d. and  $X_i \sim \exp(\lambda)$ . Let  $A_i = X_1 + X_2 + \cdots + X_i = \sum_{k=1}^i X_k$  with  $i \geq 1$ . Use the density of  $A_i$  or the moment generating function of  $A_i$  to show that

$$E\{A_i\} = \frac{i}{\lambda},$$

that is, the expected time to see  $i$  jobs is  $i/\lambda$ .

**Exercise 2.1.11.** If  $X \sim \exp(\lambda)$  and  $S \sim \exp(\mu)$  and  $X$  and  $S$  are independent, show that

$$Z = \min\{X, S\} \sim \exp(\lambda + \mu),$$

hence  $E\{Z\} = (\lambda + \mu)^{-1}$ .

**Exercise 2.1.12.** If  $X \sim \exp(\lambda)$ ,  $S \sim \exp(\mu)$  and independent, show that

$$P\{X \leq S\} = \frac{\lambda}{\lambda + \mu}.$$

**Exercise 2.1.13.** A machine serves two types of jobs. The processing time of jobs of type  $i$ ,  $i = 1, 2$ , is exponentially distributed with parameter  $\mu_i$ . The type  $T$  of job is random and independent of anything else, and such that  $P\{T = 1\} = p = 1 - q = 1 - P\{T = 2\}$ . (An example is a desk serving men and women, both requiring different average service times, and  $p$  is the probability that the customer in service is a man.) What is the expected processing time and what is the variance?

**Exercise 2.1.14.** Try to make Figure 2.2 with simulation.

### Hints

**Hint 2.1.1:** For the second child, condition on the event that the first does not chose the right number.

**Hint 2.1.3:** Condition on the event  $X > t$ .

**Hint 2.1.9:** Check the result for  $i = 1$  by filling in  $i = 1$  (just to be sure that you have read the formula right), and compare the result to the exponential density. Then write  $A_i = \sum_{k=1}^i X_k$ , and compute the moment generating function for  $A_i$  and use that the interarrival times  $X_i$  are independent. Use the moment generating function of  $X_i$ .

**Hint 2.1.10:** Use that  $\int_0^\infty (\lambda t)^i e^{-\lambda t} dt = \frac{i!}{\lambda}$ . Another way would be to use that, once you have the moment generating function of some random variable  $X$ ,  $E\{X\} = \frac{d}{dt}M(t)|_{t=0}$ .

**Hint 2.1.11:** Use that if  $Z = \min\{X, S\} > x$  that then it must be that  $X > x$  and  $S > x$ . Then use independence of  $X$  and  $S$ .

**Hint 2.1.12:** Define the joint distribution of  $X$  and  $S$  and carry out the computations, or use conditioning, or use the result of the previous exercise.

### Solutions

**Solution 2.1.1:** Use the definition of conditional probability ( $P\{A|B\} = P\{AB\}/P\{B\}$ , provided  $P\{B\} > 0$ )

The probability that the first child to guess also wins is  $1/3$ . What is the probability for child number two? Well, for him/her to win, it is necessary that child one does not win and that child two guesses the right number of the remaining numbers. Assume, without loss of generality that child 1 chooses 3 and that this is not the right number. Then

$$\begin{aligned} &P\{\text{Child 2 wins}\} \\ &= P\{\text{Child 2 guesses the right number and child 1 does not win}\} \\ &= P\{\text{Child 2 guesses the right number} | \text{child 1 does not win}\} \cdot P\{\text{Child 1 does not win}\} \\ &= P\{\text{Child 2 makes the right guess in the set } \{1, 2\}\} \cdot \frac{2}{3} \\ &= \frac{1}{2} \cdot \frac{2}{3} = \frac{1}{3}. \end{aligned}$$

Similar conditional reasoning gives that child 3 wins with probability  $1/3$ .

**Solution 2.1.2:** Write for  $F(x) = P\{X \leq x\}$  and  $f(x) = dF(x)/dx$  for the density of  $F$ .

$$\begin{aligned} E\{h(X)\} &= \int_0^{10} E\{h(X)|X=x\} P\{X \in dx\} \\ &= \int_0^{10} E\{h(x)|X=x\} dF(x) \\ &= \int_0^{10} E\{h(x)|X=x\} F(dx) \\ &= \int_0^{10} E\{h(x)|X=x\} f(x) dx \\ &= \int_0^{10} h(x) \frac{dx}{10}. \end{aligned}$$

## 2.1 Exponential Distribution

Here we introduce some notation that is commonly used in the probability literature to indicate the same conceptual idea, i.e,  $P\{X \in dx\} = dF(x) = F(dx) = f(x)dx$ , where the last equality follows from the fact that  $F$  has a density  $f$  everywhere on  $[0, 10]$ .

The concept of conditional expectation is of fundamental importance in probability theory. Any *good* probability book defines this concept as a random variable measurable with respect to some  $\sigma$ -algebra. In this course we will not deal with this elegant idea, due to lack of time.

**Solution 2.1.3:** This is easy, but be sure you can do it.

To see that an exponentially distributed is memoryless, use the definition of conditional probability ( $P\{A|B\} = P\{AB\}/P\{B\}$ , provided  $P\{B\} > 0$ ):

$$P\{X > t+h|X > t\} = \frac{P\{X > t+h, X > t\}}{P\{X > t\}} = \frac{P\{X > t+h\}}{P\{X > t\}} = \frac{e^{-\lambda(t+h)}}{e^{-\lambda t}} = e^{-\lambda h} = P\{X > h\}.$$

**Solution 2.1.4:**

$$\begin{aligned} E\{X\} &= \int_0^\infty E\{X|X=t\}f(t)dt \\ &= \int_0^\infty t\lambda e^{-\lambda t} dt, \quad \text{density is } \lambda e^{-\lambda t} \\ &= \lambda^{-1} \int_0^\infty u e^{-u} du, \quad \text{by change of variable } u = \lambda t, \\ &= -\lambda^{-1} t e^{-t} \Big|_0^\infty + \lambda^{-1} \int_0^\infty e^{-t} dt \\ &= -\lambda^{-1} e^{-t} \Big|_0^\infty = \frac{1}{\lambda}. \end{aligned}$$

**Solution 2.1.5:**

$$\begin{aligned} E\{X^2\} &= \int_0^\infty E\{X^2|X=t\}f(t)dt \\ &= \int_0^\infty t^2 \lambda e^{-\lambda t} dt \\ &= \lambda^{-2} \int_0^\infty u^2 e^{-u} du, \quad \text{by change of variable } u = \lambda t, \\ &= -\lambda^{-2} t^2 e^{-t} \Big|_0^\infty + 2\lambda^{-2} \int_0^\infty t e^{-t} dt \\ &= -2\lambda^{-2} t e^{-t} \Big|_0^\infty + 2\lambda^{-2} \int_0^\infty e^{-t} dt \\ &= -2\lambda^{-2} e^{-t} \Big|_0^\infty \\ &= 2/\lambda^2. \end{aligned}$$

**Solution 2.1.6:** By the previous problems,  $E\{X^2\} = 2/\lambda^2$  and  $E\{X\} = 1/\lambda$ .

**Solution 2.1.7:** By the previous problems,  $V\{X\} = 1/\lambda^2$  and  $E\{X\} = 1/\lambda$ .

**Solution 2.1.8:**

$$\begin{aligned} M_X(t) &= E\{\exp(tX)\} = \int_0^\infty e^{tx} dF(x) = \int_0^\infty e^{tx} f(x) dx = \int_0^\infty e^{tx} \lambda e^{-\lambda x} dx \\ &= \lambda \int_0^\infty e^{(t-\lambda)x} dx = \frac{\lambda}{\lambda-t}. \end{aligned}$$

## 2 Single-Station Queueing Systems

**Solution 2.1.9:** One way to find the distribution of  $A_i$  is by using the moment generating function  $M_{A_i}(t) = E\{e^{tA_i}\}$  of  $A_i$ . Let  $X_i$  be the interarrival time between customers  $i$  and  $i-1$ , and  $M_X(t)$  the associated moment generating function. Using the i.i.d. property of the  $\{X_i\}$ ,

$$\begin{aligned} M_{A_i}(t) &= E\{e^{tA_i}\} = E\left\{\exp\left(t \sum_{j=1}^i X_j\right)\right\} \\ &= \prod_{j=1}^i E\{e^{tX_j}\} = \prod_{j=1}^i M_{X_j}(t) = \prod_{j=1}^i \frac{\lambda}{\lambda - t} = \left(\frac{\lambda}{\lambda - t}\right)^i. \end{aligned}$$

From a table of moment generation functions it follows immediately that  $A_i \sim \Gamma(n, \lambda)$ , i.e.,  $A_i$  is Gamma distributed.

**Solution 2.1.10:**

$$E\{A_i\} = \int_0^\infty t P\{A_i \in dt\} = \int_0^\infty t f_{A_i}(t) dt = \int_0^\infty t \lambda e^{-\lambda t} \frac{(\lambda t)^{i-1}}{(i-1)!} dt.$$

Thus,

$$E\{A_i\} = \frac{1}{(i-1)!} \int_0^\infty e^{-\lambda t} (\lambda t)^i dt = \frac{i!}{(i-1)! \lambda} = \frac{i}{\lambda},$$

where we used the hint.

What if we would use the moment generating function?

$$\begin{aligned} E\{A_i\} &= \left. \frac{d}{dt} M_{A_i}(t) \right|_{t=0} \\ &= \left. \frac{d}{dt} \left(\frac{\lambda}{\lambda - t}\right)^i \right|_{t=0} \\ &= i \left(\frac{\lambda}{\lambda - t}\right)^{i-1} \frac{\lambda}{(\lambda - t)^2} \Big|_{t=0} \\ &= \frac{i}{\lambda} \left(\frac{\lambda}{\lambda - t}\right)^{i-1} \Big|_{t=0} \\ &= \frac{i}{\lambda}. \end{aligned}$$

And indeed, by the fact that  $E\{X + Y\} = E\{X\} + E\{Y\}$  for any r.v.  $X$  and  $Y$ ,

$$E\{A_i\} = E\left\{\sum_{k=1}^i X_k\right\} = i E\{X\} = \frac{i}{\lambda}.$$

We get the same answer.

**Solution 2.1.11:** Use that  $X$  and  $S$  are independent to get

$$\begin{aligned} P\{Z > x\} &= P\{\min X, S > x\} = P\{X > x \text{ and } S > x\} = P\{X > x\} P\{S > x\} \\ &= e^{-\lambda x} e^{-\mu x} = e^{-(\lambda + \mu)x}. \end{aligned}$$



**Solution 2.1.12:** There is more than one way to show that  $P\{X \leq S\} = \lambda/(\lambda + \mu)$ .

Method 1. (I admit that, although the simplest, least technical, method, I did not think of this right away. I am ‘conditioned’ to use conditioning...) Observe first that  $X$  and  $S$ , being exponentially distributed, both have a density. Moreover, as they are independent, we can sensibly speak of the joint density  $f_{X,S}(x,y) = f_X(x)f_S(y) = \lambda\mu e^{-\lambda x}e^{-\mu y}$ . With this,

$$\begin{aligned}
 P\{X \leq S\} &= E\{\mathbb{1}_{X \leq S}\} \\
 &= \int_0^\infty \int_0^\infty \mathbb{1}_{x \leq y} f_{X,S}(x,y) \, dy \, dx \\
 &= \lambda\mu \int_0^\infty \int_0^\infty \mathbb{1}_{x \leq y} e^{-\lambda x} e^{-\mu y} \, dy \, dx \\
 &= \lambda\mu \int_0^\infty e^{-\mu y} \int_0^y e^{-\lambda x} \, dx \, dy \\
 &= \mu \int_0^\infty e^{-\mu y} (1 - e^{-\lambda y}) \, dy \\
 &= \mu \int_0^\infty (e^{-\mu y} - e^{-(\lambda+\mu)y}) \, dy \\
 &= \mu \int_0^\infty (e^{-\mu y} - e^{-(\lambda+\mu)y}) \, dy \\
 &= 1 - \frac{\mu}{\lambda + \mu}
 \end{aligned}$$

This argument is provided in the probability book you use in the first year.

Method 2. Applying a standard conditioning argument

$$P\{X \leq S\} = \int_0^\infty P\{X \leq S | S = s\} \mu e^{-\mu s} \, ds.$$

Now,  $P\{X \leq S | S = s\}$  is a conditional probability distribution. This is a bit of tricky object, but very useful once you get used to it. The tricky part is that  $P\{S = s\} = 0$ . Therefore  $P\{X \leq S | S = s\}$  cannot be defined as  $\frac{P\{X \leq s, S = s\}}{P\{S = s\}}$ . However, if we proceed nonetheless and use the independence of  $S$  and  $X$ , we get

$$P\{X \leq S | S = s\} = \frac{P\{X \leq s, S = s\}}{P\{S = s\}} = \frac{P\{X \leq s\} P\{S = s\}}{P\{S = s\}} = P\{X \leq s\}$$

and thus, indeed,  $P\{X \leq S | S = s\} = P\{X \leq s\}$ . Then,

$$\begin{aligned}
 P\{X \leq S\} &= \int_0^\infty P\{X \leq S | S = s\} \mu e^{-\mu s} \, ds \\
 &= \int_0^\infty P\{X \leq s\} \mu e^{-\mu s} \, ds \\
 &= \int_0^\infty (1 - e^{-\lambda s}) \mu e^{-\mu s} \, ds
 \end{aligned}$$

and we arrive at the integral we have seen above.

So we get the correct answer, but by the wrong method. How can we repair this? As a first step, let's not fix  $S$  to a set of measure zero, but let's assume that  $S \in [s, t]$  for  $s < t$ . Then it follows that

$$\mathbb{1}_{X \leq s} \mathbb{1}_{S \in [s, t]} \leq \mathbb{1}_{X \leq S} \mathbb{1}_{S \in [s, t]} \leq \mathbb{1}_{X \leq t} \mathbb{1}_{S \in [s, t]}$$

## 2 Single-Station Queueing Systems

As a second step, using that  $P\{S \in [s, t]\} > 0$  if  $s < t$  and the independence of  $X$  and  $S$ ,

$$\begin{aligned} P\{X \leq s\} &= \frac{P\{X \leq s\} P\{S \in [s, t]\}}{P\{S \in [s, t]\}} = \frac{P\{X \leq s, S \in [s, t]\}}{P\{S \in [s, t]\}} \\ P\{X \leq t\} &= \frac{P\{X \leq t\} P\{S \in [s, t]\}}{P\{S \in [s, t]\}} = \frac{P\{X \leq t, S \in [s, t]\}}{P\{S \in [s, t]\}} \end{aligned}$$

Now with the result of the first step

$$\begin{aligned} P\{X \leq s\} &= \frac{P\{X \leq s, S \in [s, t]\}}{P\{S \in [s, t]\}} \\ &\leq \frac{P\{X \leq S, S \in [s, t]\}}{P\{S \in [s, t]\}} \\ &= P\{X \leq S | S \in [s, t]\} \\ &\leq \frac{P\{X \leq t, S \in [s, t]\}}{P\{S \in [s, t]\}} \\ &= P\{X \leq t\}. \end{aligned}$$

Hence,

$$P\{X \leq s\} \leq P\{X \leq S | S \in [s, t]\} \leq P\{X \leq t\}.$$

Finally, taking the limit  $t \downarrow s$ , and defining  $P\{X \leq S | S = s\} = \lim_{t \downarrow s} P\{X \leq S | S \in [s, t]\}$ , it follows that

$$P\{X \leq s\} = P\{X \leq S | S = s\}$$

A more direct way to properly define  $P\{X \leq S | S = s\}$  is as follows. For any  $y$  such that  $f_S(y) > 0$ , we can define the conditional probability density function of  $X$ , given that  $S = s$ , as

$$f_{X|S}(x|s) = \frac{f_{X,S}(x, s)}{f_S(s)},$$

where, as before,  $f_{X,S}(x, s)$  is the joint density of  $X$  and  $S$ . Now that the conditional probability density is defined, we can properly define

$$E\{X | S = s\} = \int_0^\infty x f_{X|S}(x|s) dx$$

and also

$$P\{X \leq S | S = s\} = E\{\mathbb{1}_{X \leq S} | S = s\} = \int_0^\infty \mathbb{1}_{x \leq s} f_{X|S}(x|s) dx.$$

Using the definition of  $f_{X|S}(x|s)$  and the independence of  $X$  and  $S$  it follows that

$$f_{X|S}(x|s) = \frac{f_{X,S}(x, s)}{f_S(s)} = \frac{\lambda e^{-\lambda x} \mu e^{-\mu s}}{\mu e^{-\mu s}} = \lambda e^{-\lambda x}$$

from which we get that

$$\begin{aligned} E\{\mathbb{1}_{X \leq S} | S = s\} &= \int_0^\infty \mathbb{1}_{x \leq s} f_{X|S}(x|s) dx \\ &= \int_0^\infty \mathbb{1}_{x \leq s} \lambda e^{-\lambda x} dx \\ &= \int_0^s \lambda e^{-\lambda x} dx \\ &= 1 - e^{-\lambda s}, \end{aligned}$$

that is,

$$P\{X \leq S | S = s\} = E\{\mathbb{1}_{X \leq S} | S = s\} = 1 - e^{-\lambda s} = P\{X \leq s\}.$$

All of these problems can be put on solid ground by using measure theory. We do not pursue these matters any further, but trust on our intuition that all is well.

**Solution 2.1.13:** Let  $X$  be the processing (or service) time at the server, and  $X_i$  the service time of a type  $i$  job. Then,

$$X = \mathbb{1}_{T=1}X_1 + \mathbb{1}_{T=2}X_2,$$

where  $\mathbb{1}$  is the indicator function, that is,  $\mathbb{1}_A = 1$  if the event  $A$  is true, and  $\mathbb{1}_A = 0$  if  $A$  is not true. With this,

$$\begin{aligned} E\{X\} &= E\{\mathbb{1}_{T=1}X_1\} + E\{\mathbb{1}_{T=2}X_2\} \\ &= E\{\mathbb{1}_{T=1}\} E\{X_1\} + E\{\mathbb{1}_{T=2}\} E\{X_2\}, \text{ by the independence of } T, \\ &= P\{T=1\}/\mu_1 + P\{T=2\}/\mu_2 \\ &= p/\mu_1 + q/\mu_2 \\ &= p E\{X_1\} + q E\{X_2\}. \end{aligned}$$

(The next derivation may seem a bit long, but the algebra is standard. I include all steps so that you don't have to use pen and paper yourself if you want to check the result.) Next, using that

$$\mathbb{1}_{T=1} \mathbb{1}_{T=2} = 0 \text{ and } \mathbb{1}_{T=1}^2 = \mathbb{1}_{T=1},$$

we get

$$\begin{aligned} V\{X\} &= E\{X^2\} - (E\{X\})^2 \\ &= E\{(\mathbb{1}_{T=1}X_1 + \mathbb{1}_{T=2}X_2)^2\} - \left(\frac{p}{\mu_1} + \frac{q}{\mu_2}\right)^2 \\ &= E\{\mathbb{1}_{T=1}X_1^2 + \mathbb{1}_{T=2}X_2^2\} - \left(\frac{p}{\mu_1} + \frac{q}{\mu_2}\right)^2 \\ &= p E\{X_1^2\} + q E\{X_2^2\} - \left(\frac{p}{\mu_1} + \frac{q}{\mu_2}\right)^2 \\ &= p V\{X_1\} + p(E\{X_1\})^2 + q V\{X_2\} + q(E\{X_2\})^2 - \left(\frac{p}{\mu_1} + \frac{q}{\mu_2}\right)^2 \\ &= p V\{X_1\} + \frac{p}{\mu_1^2} + q V\{X_2\} + \frac{q}{\mu_2^2} - \left(\frac{p}{\mu_1} + \frac{q}{\mu_2}\right)^2 \\ &= p V\{X_1\} + q V\{X_2\} + \frac{p}{\mu_1^2} + \frac{q}{\mu_2^2} - \frac{p^2}{\mu_1^2} - \frac{q^2}{\mu_2^2} - \frac{2pq}{\mu_1\mu_2} \\ &= p V\{X_1\} + q V\{X_2\} + \frac{p(1-p)}{\mu_1^2} + \frac{q(1-q)}{\mu_2^2} - \frac{2pq}{\mu_1\mu_2} \\ &= p V\{X_1\} + q V\{X_2\} + \frac{pq}{\mu_1^2} + \frac{qp}{\mu_2^2} - \frac{2pq}{\mu_1\mu_2} \\ &= p V\{X_1\} + q V\{X_2\} + pq(E\{X_1\} - E\{X_2\})^2. \end{aligned}$$

## 2 Single-Station Queueing Systems

Interestingly, we see that even if  $V\{X_1\} = V\{X_2\} = 0$ ,  $V\{X\} > 0$  if  $E\{X_1\} \neq E\{X_2\}$ . Bear this in mind; we will use these ideas later when we discuss the effects of failures on the variance of service times of jobs.

**Solution 2.1.14:** The source code can be found in `progs/converge_to_exp.py`.

## 2.2 Poisson Distribution

In this section we provide a derivation of, and motivation for, the Poisson process, and clarify its relation with the exponential distribution at the end.

Consider a machine that fails occasionally. Let us write  $N(s, t)$  for the number of failures occurring during a time interval of  $(s, t]$ . We assume, without loss of generality, that repairs are instantaneous. Clearly, as we do not know in advance how often the machine will fail, we model  $N(s, t)$  as a random variable for all times  $s$  and  $t$ .

Our first assumption is that the failure behavior of the machine does not significantly change over time. Then it is reasonable to assume that the expected number of failure is proportional to the length of the interval  $T$ . Thus, it is reasonable to assume that there exists some constant  $\lambda$  such that

$$E\{N(s, t)\} = \lambda(t - s) \quad (2.2)$$

The constant  $\lambda$  is often called the *arrival rate*, or failure rate in this case.

The second assumption is that  $\{N(s, t), s \leq t\}$  has *stationary* and *independent increments*. Stationarity means that the distribution of the number of arrivals are the same for all intervals of equal length. Formally,  $N(s_1, t_1)$  has the same distribution as  $N(s_2, t_2)$  if  $t_2 - s_2 = t_1 - s_1$ . Independence means, roughly speaking, that knowing that  $N(s_1, t_1) = n$ , does not help to make any predictions about  $N(s_2, t_2)$  if the intervals  $(s_1, t_1)$  and  $(s_2, t_2)$  have no overlap.

To find the distribution of  $N(0, t)$ , let us split the interval  $[0, t]$  into  $n$  sub-intervals, all of equal length, and ask: ‘What is the probability that the machine will fail in some given sub-interval.’ By our second assumption, the failure behavior is constant over time. Therefore, the probability  $p$  to fail in each interval should be equal. Moreover, if  $n$  is large,  $p$  must be small, for otherwise (2.2) could not be true. As a consequence, if the time intervals are very small, we can safely neglect the probability that two or more failures occur in one such tiny interval.

As a consequence, then, we can model the occurrence of a failure in some period  $i$  as a Bernoulli distributed random variable  $B_i$  such that  $P\{B_i = 1\}$  and  $P\{B_i = 0\} = 1 - P\{B_i = 1\}$ , and we assume that  $\{B_i\}$  are independent. The total number of failures  $N_n(t)$  that occur in  $n$  intervals is then binomially distributed

$$P\{N_n(t) = k\} = \binom{n}{k} p^k (1 - p)^{n-k}. \quad (2.3)$$

In the exercises we ask you to use this to motivate that, in some appropriate sense,  $N_n(t)$  converges to  $N(t)$  for  $n \rightarrow \infty$  such that

$$P\{N(t) = k\} = e^{-\lambda t} \frac{(\lambda t)^k}{k!}. \quad (2.4)$$

We say that  $N(t)$  is *Poisson distributed* with rate  $\lambda$ , and write  $N(t) \sim P(\lambda t)$ .

Moreover, if there are no failures in some interval  $[0, t]$ , then it must be that  $N(t) = 0$  and the interarrival time  $X_1 = A_1 - 0$  (as  $A_0 = 0$ ) must be larger than  $t$ . Therefore,

$$P\{X_1 > t\} = P\{N(t) = 0\} = e^{-\lambda t} \frac{(\lambda t)^0}{0!} = e^{-\lambda t}.$$

The relations we discussed above are of paramount importance in the analysis of queueing process. We summarize this by a theorem.

**Theorem 2.2.1.** A counting process  $\{N(t)\}$  is a Poisson process with rate  $\lambda$  if and only if the inter-arrival times  $\{X_i\}$ , i.e., the times between consecutive arrivals, are i.i.d. and  $P\{X_1 \leq t\} = 1 - e^{-\lambda t}$ . In other words,  $X_i \sim \exp(\lambda) \Leftrightarrow N(t) \sim P(\lambda t)$

### Exercises

**Exercise 2.2.1.** Show that  $E\{N_n(t)\} = \sum_{i=1}^n E\{B_i\} = np$ .

**Exercise 2.2.2.** What is the difference between  $N_n(t)$  and  $N(t)$ ?


**Exercise 2.2.3.** Show how the binomial distribution (2.3) converges to the Poisson distribution (2.4) if  $n \rightarrow \infty$ ,  $p \rightarrow 0$  such that  $np = \lambda$ .


**Exercise 2.2.4.** If the inter-arrival times  $\{X_i\}$  are i.i.d. and exponentially distributed with mean  $1/\lambda$ , prove that the number  $N(t)$  of arrivals during interval  $[0, t]$  is Poisson distributed.

**Exercise 2.2.5.** Show that if  $N(t) \sim P(\lambda t)$ , we have for small  $h$ ,

1.  $P\{N(h) = n \mid N(0) = n\} = 1 - \lambda h + o(h)$
2.  $P\{N(h) = n + 1 \mid N(0) = n\} = \lambda h + o(h)$
3.  $P\{N(h) \geq n + 2 \mid N(0) = n\} = o(h)$ ,

where  $o(h)$  is a function  $f(h)$  such  $f(h)/h \rightarrow 0$  as  $h \rightarrow 0$ .

**Exercise 2.2.6.** Assume a timer fires at times  $0 = T_0 < T_1 < T_2 < \dots$ , such that  $T_k - T_{k-1} \sim \exp(\lambda)$ . Define  $N(t) = \sum_{k=0}^{\infty} k \mathbb{1}_{T_k \leq t < T_{k+1}}$ , where  $\mathbb{1}$  is the *indicator function*, that is,  $\mathbb{1}_A = 1$  if the event  $A$  is true, and  $\mathbb{1}_A = 0$  if  $A$  is not true. What is the distribution of  $N(t)$ ? 

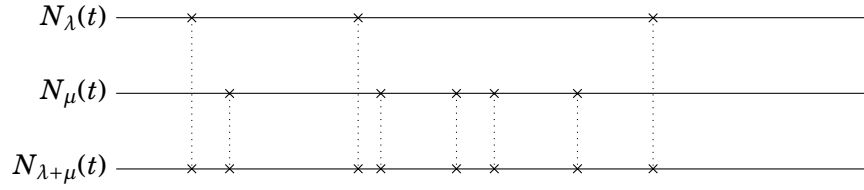
**Exercise 2.2.7.** (I added this question to clarify the next question, you can skip it for year 16/17.) Show that for a Poisson process  $N$ , 

$$P\{N(0, s] = 1 \mid N(0, t] = 1\} = \frac{s}{t}.$$

Thus, if you know that  $N(0, t] = 1$ , the arrival is distributed uniformly on the interval  $[0, t]$ .

**Exercise 2.2.8.** (Merged Poisson streams form a new Poisson process with the sum of the rates.) Assume that  $N_a(t) \sim P(\lambda t)$ ,  $N_s(t) \sim P(\mu t)$  and independent. Show that  $N_a(t) + N_s(t) \sim P((\lambda + \mu)t)$ . The figure below provides a graphical representation of merging (also called superposition) of streams.

## 2 Single-Station Queueing Systems



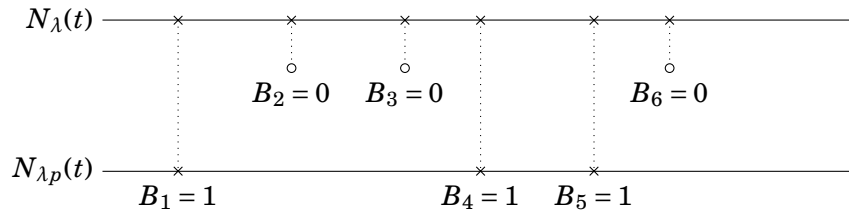
**Exercise 2.2.9.** Assume that  $N_a(t) \sim P(\lambda t)$ ,  $N_s(t) \sim P(\mu t)$  and independent. Use that  $N_a(t) + N_s(t) \sim P((\lambda + \mu)t)$  to conclude

$$P\{N_a(h) = 1, N_s(h) = 0 | N_a(h) + N_s(h) = 1\} = \frac{\lambda}{\lambda + \mu}.$$

Note that the right-hand-side does not depend on  $h$ , hence it holds for any time  $h$ , whether it is small or not.

**Exercise 2.2.10.** Assume that  $N_a(t) \sim P(\lambda t)$ ,  $N_s(t) \sim P(\mu t)$  and independent. What is actually the meaning of the event  $\{N_a(h) = 1, N_s(h) = 0\} \cap \{N_a(h) + N_s(h) = 1\}$ ?

**Exercise 2.2.11.** (A Bernoulli-thinned Poisson process is still a Poisson process) Consider a Poisson process. Split the process in the following way. When a job arrives, throw a coin that lands heads with probability  $p$  and tails with  $q = 1 - p$ . When the coin lands heads, call the job of type 1, otherwise of type 2. Another way of thinning is by modeling the stream of people passing a shop as a Poisson process with rate  $\lambda$ . With probability  $p$  a person decides, independent of anything else, to enter the shop, c.f. the figure below. The crosses at the upper line are passersby in the street. The crosses at the lower line are the customers that enter the shop. The outcome of the  $k$ th throw of a coin is indicated by  $B_k$ . Since  $B_1 = 1$  the first passerby turns into a customer entering the shop; the second passerby does not enter as  $B_2 = 0$ , and so on. Like this, the stream of passerby's is thinned with Bernoulli distributed random variables.



The concept of thinning is particularly useful to analyze queueing networks. Suppose the departure stream of a machine is split into two substreams, e.g., a fraction  $p$  of the jobs move on another machine and the rest  $(1 - p)$  of the jobs leave the system. Then we can model the arrival stream at the second machine as a thinned stream (with probability  $p$ ) of the departures of the first machine.

Show that the Poisson process obtained by thinning the original process is  $\sim P(\lambda t p)$  for any  $t$ .

**Exercise 2.2.12.** Show that  $E\{N(t)\} = \lambda t$  and  $V\{N(t)\} = \lambda t$ . Why is the SCV of  $N(t)$  equal to  $1/\lambda t$ ? Conclude that the relative variability of  $N(t)$  becomes smaller as  $t$  becomes larger. Observe that the SCV of a Poisson distributed random variable is not the same as the SCV of an exponentially distributed random variable.

## Hints

**Hint 2.2.1:** Recall that  $E\{X + Y\} = E\{X\} + E\{Y\}$ . Just as a reminder,  $E\{XY\} \neq E\{X\}E\{Y\}$  in general. Only when  $X$  and  $Y$  are uncorrelated (which is implied by independence), the product of the expectations is the expectation of the products.

**Hint 2.2.3:** First find  $p$ ,  $n$ ,  $\lambda$  and  $t$  are such that the rate at which event occur in both processes are the same. Then consider the binomial distribution and use the standard limit  $(1 - x/n)^n \rightarrow e^{-x}$  as  $n \rightarrow \infty$ .

**Hint 2.2.4:** Realize that  $P\{N(t) = k\} = P\{A_k \leq t\} - P\{A_{k+1} \leq t\}$ .

**Hint 2.2.5:** Think about the meaning of the formula  $P\{N(h) = n | N(0) = n\}$ . It is a conditional probability that should be read like this: given that up to time 0 we have seen  $n$  arrivals (i.e.,  $N(0) = n$ ), what is the probability that just a little later ( $h$ ) the number of arrivals is still  $n$ , i.e.,  $N(h) = n$ ? Then use the definition of the Poisson distribution to compute this probability. Finally, use Taylor's expansion of  $e^x$  to see that  $e^x = 1 + x + o(x)$  for  $|x| \ll 1$ . Furthermore, use that  $\sum_{i=2}^{\infty} x^i/i! = \sum_{i=0}^{\infty} x^i/i! - x - 1 = e^x - x - 1$ .

**Hint 2.2.7:** Use Bayes' law for conditional probability. Observe that

$$\{N(0, s] + N(s, t] = 1\} \cap \{N(0, s] = 1\} = \{1 + N(s, t] = 1\} \cap \{N(0, s] = 1\} = \{N(s, t] = 0\} \cap \{N(0, s] = 1\}.$$

**Hint 2.2.8:** Use a conditioning argument or use probability generating functions (i.e.  $E\{z^X\} = \sum_k z^k P\{X = k\}$ ). In particular, for conditioning, use that  $P\{AB\} = P\{A|B\}P\{B\}$ . More generally, if the set  $A$  can be split into disjoint sets  $B_i$ , i.e.,  $A = \bigcup_{i=1}^n B_i$ , then

$$P\{A\} = \sum_{i=1}^n P\{AB_i\} = \sum_{i=1}^n P\{A|B_i\}P\{B_i\},$$

where we use the conditioning formula to see that  $P\{AB_i\} = P\{A|B_i\}P\{B_i\}$ . Now choose practical sets  $B_i$ .

**Hint 2.2.9:** Suppose we write  $N(t) = N_a(t) + N_s(t)$ . Then

$$P\{N_a(h) = 1, N_s(h) = 0 | N(h) = 1\}$$

is the probability that  $N_a(h) = 1$  and  $N_s(h) = 0$  given that  $N(h) = 1$ . In other words, the question is find out that, given one of the two processes 'fired', what is the probability that  $N_a$  was the one that 'fired'.

**Hint 2.2.11:** Condition on the total number of arrivals  $N(t) = m$  up to time  $t$ . Realize that the probability that a job is of type 1 is Bernoulli distributed, hence when you consider  $m$  jobs in total, the number of type 1 jobs is binomially distributed.

Again use that if the set  $A$  can be split into disjoint sets  $B_i$ , i.e.,  $A = \bigcup_{i=1}^n B_i$ , then

$$P\{A\} = \sum_{i=1}^n P\{A|B_i\}P\{B_i\}.$$

Now choose practical sets  $B_i$ .

## 2 Single-Station Queueing Systems

You might also consider the random variable

$$Y = \sum_{i=1}^N Z_i,$$

with  $N \sim P(\lambda)$  and  $Z_i \sim B(p)$ . Show that the moment generating function of  $Y$  is equal to the moment generating function of a Poisson random variable with parameter  $\lambda p$ .

**Hint 2.2.12:** You might use generating functions here.

### Solutions

**Solution 2.2.1:**

$$\mathbb{E}\{N_n(t)\} = \mathbb{E}\left\{\sum_{i=1}^n B_i\right\} = \sum_{i=1}^n \mathbb{E}\{B_i\} = n \mathbb{E}\{B_i\} = np.$$

**Solution 2.2.2:**  $N_n(t)$  is a binomially distributed random variable with parameters  $n$  and  $p$ . The maximum value of  $N_n(t)$  is  $n$ . The random variable  $N(t)$  models the number of failures that can occur during  $[0, t]$ . As such it is not necessarily bounded by  $n$ . Thus,  $N_n(t)$  and  $N(t)$  cannot represent the same random variable.

**Solution 2.2.3:** Now we like to relate  $N_n(t)$  and  $N(t)$ . It is clear that we at least want the expectations to be the same, that is,  $np = \lambda t$ . This implies that

$$p = \frac{\lambda t}{n},$$

so that

$$\mathbb{P}\{N_n(t) = k\} = \binom{n}{k} \left(\frac{\lambda t}{n}\right)^k \left(1 - \frac{\lambda t}{n}\right)^{n-k}.$$

To see that

$$\lim_{n \rightarrow \infty} \binom{n}{k} \left(\frac{\lambda t}{n}\right)^k \left(1 - \frac{\lambda t}{n}\right)^{n-k} = e^{-\lambda t} \frac{(\lambda t)^k}{k!}.$$

use that

$$\begin{aligned} \binom{n}{k} \left(\frac{\lambda t}{n}\right)^k \left(1 - \frac{\lambda t}{n}\right)^{n-k} &= \frac{n!}{k!(n-k)!} \left(\frac{\lambda t}{n}\right)^k \left(1 - \frac{\lambda t}{n}\right)^n \\ &= \frac{(\lambda t)^k}{k!} \left(\frac{n}{n-\lambda t}\right)^k \frac{n!}{n^k(n-k)!} \left(1 - \frac{\lambda t}{n}\right)^n \\ &= \frac{(\lambda t)^k}{k!} \left(\frac{n}{n-\lambda t}\right)^k \frac{n}{n} \frac{n-1}{n} \cdots \frac{n-k+1}{n} \left(1 - \frac{\lambda t}{n}\right)^n. \end{aligned}$$

Observe now that, as  $\lambda t$  is finite,  $n/(n-\lambda t) \rightarrow 1$  as  $n \rightarrow \infty$ . Also for any finite  $k$ ,  $(n-k)/n \rightarrow 1$ . Finally, we use that  $(1+x/n)^n \rightarrow e^x$  so that  $\left(1 - \frac{\lambda t}{n}\right)^n \rightarrow e^{-\lambda t}$ . The rest is easy, so that, as  $n \rightarrow \infty$ , the above converges to

$$\frac{(\lambda t)^k}{k!} e^{-\lambda t}.$$



**Solution 2.2.4:** We want to show that

$$P\{N(t) = k\} = e^{-\lambda t} \frac{(\lambda t)^k}{k!}.$$

Now observe that  $P\{N(t) = k\} = P\{A_k \leq t\} - P\{A_{k+1} \leq t\}$ . Using the density of  $A_{k+1}$  as obtained previously and applying partial integration leads to

$$\begin{aligned} P\{A_{k+1} \leq t\} &= \lambda \int_0^t \frac{(\lambda s)^k}{k!} e^{-\lambda s} ds \\ &= \lambda \frac{(\lambda s)^k}{k!} \frac{e^{-\lambda s}}{-\lambda} \Big|_0^t + \lambda \int_0^t \frac{(\lambda s)^{k-1}}{(k-1)!} e^{-\lambda s} ds \\ &= -\frac{(\lambda t)^k}{k!} e^{-\lambda t} + P\{A_k \leq t\} \end{aligned}$$

We are done.

**Solution 2.2.5:** 1.  $P\{N(h) = n | N(0) = n\} = P\{N(h) = 0\} = e^{-\lambda h} (\lambda h)^0 / 0! = e^{-\lambda h} = 1 - \lambda h + o(h)$ , as follows from a standard argument in analysis that  $e^{-\lambda h} = 1 - \lambda h + o(h)$  for  $h$  small.

2.  $P\{N(h) = n + 1 | N(0) = n\} = P\{N(h) = 1\} = e^{-\lambda h} (\lambda h)^1 / 1! = (1 - \lambda h + o(h)) \lambda h = \lambda h - \lambda^2 h^2 + o(h) = \lambda h + o(h)$ .

3.

$$\begin{aligned} P\{N(h) \geq n + 2 | N(0) = n\} &= P\{N(h) \geq 2\} \\ &= e^{-\lambda h} \sum_{i=2}^{\infty} \frac{(\lambda h)^i}{i!} = e^{-\lambda h} \left( \sum_{i=0}^{\infty} \frac{(\lambda h)^i}{i!} - \lambda h - 1 \right) \\ &= e^{-\lambda h} (e^{\lambda h} - 1 - \lambda h) = 1 - e^{-\lambda h} (1 + \lambda h) \\ &= 1 - (1 - \lambda h + o(h))(1 + \lambda h) = 1 - (1 - \lambda^2 h^2 + o(h)) \\ &= \lambda^2 h^2 + o(h) = o(h). \end{aligned}$$

We can also use the results of the previous parts to see that

$$\begin{aligned} P\{N(h) \geq n + 2 | N(0) = n\} &= P\{N(h) \geq 2\} = 1 - P\{N(h) < 2\} \\ &= 1 - P\{N(h) = 0\} - P\{N(h) = 1\} \\ &= 1 - (1 - \lambda h + o(h)) - (\lambda h + o(h)) \\ &= o(h). \end{aligned}$$

**Solution 2.2.6:**  $N(t) \sim P(\lambda t)$ .

**Solution 2.2.7:**

$$\begin{aligned} P\{N(0, s] = 1 | N(0, t] = 1\} &= \frac{P\{N(0, s] = 1, N(0, t] = 1\}}{P\{N(0, t] = 1\}} \\ &= P\{N(0, t] = 1 | N(0, s] = 1\} \frac{P\{N(0, s] = 1\}}{P\{N(0, t] = 1\}} \\ &= P\{N(0, s] + N(s, t] = 1 | N(0, s] = 1\} \frac{e^{-\lambda s} \lambda s}{e^{-\lambda t} \lambda t} \\ &= P\{1 + N(s, t] = 1 | N(0, s] = 1\} e^{-\lambda(s-t)} \frac{s}{t} \\ &= P\{N(s, t] = 0\} e^{-\lambda(s-t)} \frac{s}{t} = e^{-\lambda(t-s)} e^{-\lambda(s-t)} \frac{s}{t} = \frac{s}{t}. \end{aligned}$$

## 2 Single-Station Queueing Systems

**Solution 2.2.8:** First we show how to use conditioning.

$$P\{N_a(t) + N_s(t) = n\} = \sum_{i=0}^n P\{N_a(t) + N_s(t) = n | N_a(t) = i\} P\{N_a(t) = i\}$$

Now, if  $N_a(t) = i$ , then

$$N_a(t) + N_s(t) = n \iff i + N_s(t) = n \iff N_s(t) = n - i.$$

Thus,

$$\begin{aligned} P\{N_a(t) + N_s(t) = n\} &= \sum_{i=0}^n P\{N_s(t) = n - i\} P\{N_a(t) = i\} \\ &= \sum_{i=0}^n \frac{(\mu t)^{n-i}}{(n-i)!} \frac{(\lambda t)^i}{i!} e^{-(\mu+\lambda)t} \\ &= e^{-(\mu+\lambda)t} \sum_{i=0}^n \frac{(\mu t)^{n-i}}{(n-i)!} \frac{(\lambda t)^i}{i!} \\ &= e^{-(\mu+\lambda)t} \frac{1}{n!} \sum_{i=0}^n \binom{n}{i} (\mu t)^{n-i} (\lambda t)^i \\ &= \frac{((\mu + \lambda)t)^n}{n!} e^{-(\mu+\lambda)t}. \end{aligned} \tag{2.5}$$

Now with the probability generating functions.

$$\begin{aligned} M_a(z) &= E\{z^{N_a(t)}\} = \sum_{k=0}^{\infty} z^k P\{N_a(t) = k\} \\ &= \sum_{k=0}^{\infty} z^k e^{-\lambda t} \frac{(\lambda t)^k}{k!} \\ &= e^{-\lambda t} \sum_{k=0}^{\infty} \frac{(z\lambda t)^k}{k!} \\ &= \exp(\lambda t(z - 1)). \end{aligned}$$

Use this, and the similar expression for  $N_s(t)$  and independence to see that

$$M(z) = E\{z^{N(t)}\} = E\{z^{N_a(t)}\} E\{z^{N_s(t)}\} = \exp((\lambda + \mu)t(z - 1)).$$

Finally, since the generating function uniquely characterizes the distribution, and since the above expression has the same form as  $M_1(z)$  but with  $\lambda + \mu$  replacing  $\lambda$ , we can conclude that  $N(t) \sim P((\lambda + \mu)t)$ .

**Solution 2.2.9:** With the above:

$$\begin{aligned}
& P\{N_a(h) = 1, N_s(h) = 0 | N_a(h) + N_s(h) = 1\} \\
&= \frac{P\{N_a(h) = 1, N_s(h) = 0, N_a(h) + N_s(h) = 1\}}{P\{N_a(h) + N_s(h) = 1\}} \\
&= \frac{P\{N_a(h) = 1, N_s(h) = 0\}}{P\{N_a(h) + N_s(h) = 1\}} \\
&= \frac{P\{N_a(h) = 1\} P\{N_s(h) = 0\}}{P\{N_a(h) + N_s(h) = 1\}} \\
&= \frac{\lambda h \exp(-\lambda h) \exp(-\mu h)}{((\lambda + \mu)h) \exp(-(\lambda + \mu)h)} \\
&= \frac{\lambda h \exp(-(\lambda + \mu)h)}{((\lambda + \mu)h) \exp(-(\lambda + \mu)h)} \\
&= \frac{\lambda}{\lambda + \mu}
\end{aligned}$$

**Solution 2.2.10:** This means that, given that an event occurred, the event was an arrival, i.e.,  $N_a$  was the first.

**Solution 2.2.11:** Suppose that  $N_1$  is the thinned stream, and  $N$  the total stream. Then

$$\begin{aligned}
P\{N_1 = k\} &= \sum_{n=k}^{\infty} P\{N_1 = k, N = n\} = \sum_{n=k}^{\infty} P\{N_1 = k | N = n\} P\{N = n\} \\
&= \sum_{n=k}^{\infty} P\{N_1 = k | N = n\} e^{-\lambda} \frac{\lambda^n}{n!} = \sum_{n=k}^{\infty} \binom{n}{k} p^k (1-p)^{n-k} e^{-\lambda} \frac{\lambda^n}{n!} \\
&= e^{-\lambda} \sum_{n=k}^{\infty} \frac{p^k (1-p)^{n-k}}{k!(n-k)!} \lambda^n = e^{-\lambda} \frac{(\lambda p)^k}{k!} \sum_{n=k}^{\infty} \frac{(\lambda(1-p))^{n-k}}{(n-k)!} \\
&= e^{-\lambda} \frac{(\lambda p)^k}{k!} \sum_{n=0}^{\infty} \frac{(\lambda(1-p))^n}{n!} = e^{-\lambda} \frac{(\lambda p)^k}{k!} e^{\lambda(1-p)} \\
&= e^{-\lambda p} \frac{(\lambda p)^k}{k!}.
\end{aligned}$$

We see that the thinned stream is Poisson with parameter  $\lambda p$ . (For notational ease, we left out the  $t$ , otherwise it is  $P(\lambda t p)$ ).

Now consider  $Y = \sum_{i=1}^N Z_i$ . Suppose that  $N = n$ , so that  $n$  arrivals occurred. Then we throw  $n$  coins with success probability  $p$ . It follows that  $Y$  is indeed a thinned Poisson random variable. Model the coins as a generic Bernoulli distributed random variable  $Z$ . We first need

$$E\{e^{sZ}\} = e^0 P\{Z = 0\} + e^s P\{Z = 1\} = (1-p) + e^s p.$$

Suppose that  $N = n$ , then since the  $Z_i$  are i.i.d.,

$$E\{e^{s \sum_{i=1}^n Z_i}\} = \left(E\{e^{sZ}\}\right)^n = (1 + p(e^s - 1))^n$$

Then, using conditioning on  $N$ ,

$$\begin{aligned}
E\{e^{sY}\} &= E\left\{E\left\{e^{s \sum_{i=1}^N Z_i} \mid N = n\right\}\right\} = E\{E\{(1 + p(e^s - 1))^n \mid N = n\}\} \\
&= \sum_{n=0}^{\infty} (1 + p(e^s - 1))^n e^{-\lambda} \frac{\lambda^n}{n!} = e^{-\lambda} \sum_{n=0}^{\infty} \frac{(1 + p(e^s - 1))^n \lambda^n}{n!} \\
&= e^{-\lambda} e^{\lambda(1 + p(e^s - 1))} = e^{\lambda p(e^s - 1)}.
\end{aligned}$$

## 2 Single-Station Queueing Systems

Thus,  $Y$  has the same moment generating function as a Poisson distributed random variable with parameter  $\lambda p$ . Since moment-generating functions specify the distribution uniquely,  $Y \sim P(\lambda p)$ .

**Solution 2.2.12:** I expect that this is easy for you by now. The exercise is just meant to help you recall this.

When a random variable  $N$  is Poisson distributed with parameter  $\lambda$ ,

$$\begin{aligned}
 E\{N\} &= \sum_{n=0}^{\infty} n e^{-\lambda} \frac{\lambda^n}{n!} \\
 &= \sum_{n=1}^{\infty} n e^{-\lambda} \frac{\lambda^n}{n!}, \text{ since the term with } n=0 \text{ cannot contribute} \\
 &= e^{-\lambda} \lambda \sum_{n=1}^{\infty} \frac{\lambda^{n-1}}{(n-1)!} \\
 &= e^{-\lambda} \lambda \sum_{n=0}^{\infty} \frac{\lambda^n}{n!}, \text{ by a change of variable} \\
 &= e^{-\lambda} \lambda e^{\lambda} \\
 &= \lambda.
 \end{aligned}$$

Similarly, using that  $V\{N\} = E\{N^2\} - (E\{N\})^2$ ,

$$\begin{aligned}
 E\{N^2\} &= \sum_{n=0}^{\infty} n^2 e^{-\lambda} \frac{\lambda^n}{n!} \\
 &= e^{-\lambda} \sum_{n=1}^{\infty} n \frac{\lambda^n}{(n-1)!} \\
 &= e^{-\lambda} \sum_{n=0}^{\infty} (n+1) \frac{\lambda^{n+1}}{n!} \\
 &= e^{-\lambda} \lambda \sum_{n=0}^{\infty} n \frac{\lambda^n}{n!} + e^{-\lambda} \lambda \sum_{n=0}^{\infty} \frac{\lambda^n}{n!} \\
 &= \lambda^2 + \lambda.
 \end{aligned}$$

The Poisson *process*  $\{N(t)\}$  is a much more complicated object than the Poisson distributed random variable  $N(t)$ . The process contains it is an *uncountable set* of random variables, whereas  $N(t)$  is just *one* random variable. However, for a fixed  $t$  the element  $N(t)$  is a random variable that is Poisson distributed with parameter  $\lambda t$ . Using the above, the answer of the question follows immediately.

With generating functions we get the result without too much effort. Suppose  $N$  is a Poisson

## 2.3 Kendall's Notation to Characterize Queueing Processes

distributed random variable. Then

$$\begin{aligned}
 \phi(z) &= E\{z^N\} \\
 &= \sum_{k=0}^{\infty} z^k P\{N = k\} \\
 &= \sum_{k=0}^{\infty} z^k \frac{\lambda^k}{k!} e^{-\lambda} \\
 &= e^{-\lambda} \sum_{k=0}^{\infty} \frac{(z\lambda)^k}{k!} \\
 &= e^{-\lambda} e^{z\lambda} \\
 &= e^{(z-1)\lambda}.
 \end{aligned}$$

Observe that  $\phi'(z) = E\{Nz^N\}$ , so that  $\phi'(1) = E\{N\}$ ; also  $\phi''(z) = E\{N(N-1)z^{N-2}\}$ , hence  $\phi''(1) = E\{N(N-1)\}$ . With this,

$$\begin{aligned}
 E\{N\} &= \phi'(1) = \lambda, \\
 E\{N^2\} &= \phi''(1) + E\{N\} = \lambda^2 + \lambda, \\
 V\{N\} &= E\{N^2\} - (E\{N\})^2 = \lambda, \\
 SCV &= \frac{V\{N(t)\}}{(E\{N(t)\})^2} = \frac{\lambda t}{(\lambda t)^2} = \frac{1}{(\lambda t)^2}.
 \end{aligned}$$

Clearly, as  $t$  increases,  $1/\lambda t$  decreases.

Here is a point of confusion for some students. The SCV of an exponentially distributed random variable is 1; the SCV of the related Poisson process  $N_\lambda(t)$  is *not* identically 1 for all  $t$ .

## 2.3 Kendall's Notation to Characterize Queueing Processes

As will become apparent in Sections 2.4 and 2.5, the construction of any queueing process involves three main elements: the distribution of the interarrival times between consecutive jobs, the distribution of the service times of the individual jobs, and the number of servers present to process jobs. In this characterization it is implicit that the interarrival times form a set of i.i.d. (independent and identically distributed) random variables, the service times are also i.i.d., and finally, the interarrival times and service times are mutually independent.

To characterize the type of queueing process it is common to use the *abbreviation*  $A/B/c/K$  where  $A$  is the distribution of the interarrival times,  $B$  the distribution of the services,  $c$  the number of servers, and  $K$  the size of the system. In this notation it is assumed that jobs are served in first-in-first-out (FIFO) order; FIFO scheduling is also often called first-come-first-serve (FCFS).

Let us illustrate the shorthand  $A/B/c/K$  with some examples:

- $M/M/1$ : the distribution of the interarrival times is *Memory-less*, hence exponential, the service times are also *Memoryless*, and there is 1 server. As  $K$  is unspecified, it is assumed to be infinite.
- $M/M/c$ : A *multi-server* queue with  $c$  servers in which all servers have the same capacity. Jobs arrive according to a Poisson process and have exponentially distributed processing times.

## 2 Single-Station Queueing Systems

- $M(n)/M(n)/1$ : the interarrival times are exponential, just as the service times, but the rates of the arrival and service processes may depend on the queue length  $n$ .
- $M/M/c/K$ : interarrival times and process times are exponential, and the *system capacity* is  $K$  jobs. Thus, the queue can contain at most  $K - c$  jobs.<sup>2</sup>
- $M/M/c/c$ : in this system the number of servers is the same as the system capacity, thus the queue length is always zero. This queueing system is useful to model the determine the number of beds in a hospital; the beds act as servers.
- $M^X/M/1$ : Customers arrive with exponentially distributed interarrival times. However, each customer brings in a number of jobs, known as a batch. The number of jobs in each batch is distributed as the random variable  $X$ . Thus, the arrival process of work is *compound Poisson*.
- $M/G/1$ : the interarrival times are exponentially distributed, the service times can have any General distribution (with finite mean), and there is 1 server.
- $M/G/\infty$ : exponential interarrival times, service times can have any distribution, and there is an unlimited supply of servers. This is also known as an *ample* server. Observe that in this queueing process, jobs actually never have wait in queue; upon arrival there is always a free server available.
- $M/D/1 - LIFO$ . Now job service times are *Deterministic*, and the service sequence is last-in-first-out (LIFO).
- $G/G/1$ : generally distributed interarrival and service times, 1 server.

In the sequel we will use Kendall's notation frequently to distinguish the different queueing models. Ensure that you familiarize yourself with this notation.

### Exercises

**Exercise 2.3.1.** What are some advantages and disadvantages of using the Shortest Processing Time First (SPTF) rule to serve jobs?

### Hints

**Hint 2.3.1:** Look up the relevant definitions on wikipedia or ?

### Solutions

- Solution 2.3.1:**
- Advantage: SPTF minimizes the number of jobs in queue. Thus, if you want to keep the shop floor free of jobs (Work In Progress, WIP), then this is certainly a good rule.
  - Disadvantage: large jobs get near to terrible waiting times, and the variance of the waiting time increases. Thus, the  $C_s^2$  is larger than under FIFO. Also, SPTF does not take duedates into account, thus giving a reliable duedate quotation to a customer is hard (near to impossible.)

---

<sup>2</sup>Sometimes, the  $K$  stands for the capacity in the queue, not the entire system. The notation differs among authors. Due to the lack of consistency, I might also not be consistent; be warned.

## 2.4 Construction of Discrete-Time Queueing Processes

In this section we discuss a case as this provides real-life motivation to analyze queueing systems. After describing the case, we develop a set of recursions by which we can construct queueing systems in discrete time. As it turns out, this case is too hard to analyze by mathematical means, so that simulation is the best way forward. Interestingly, the structure of the simulation is very simple. As such, simulation is an exceedingly convincing tool to communicate the results of an analysis of a queueing system to managers (and the like).

At a mental health department five psychiatrists do intakes of future patients to determine the best treatment process for the patients. There are complaints about the time patients have to wait for their first intake; the desired waiting time is around two weeks, but the realized waiting time is sometimes more than three months. The organization considers this to be unacceptably long, but... what to do about it?

To reduce the waiting times the five psychiatrists have various suggestions.

1. Not all psychiatrists have the same amount of time available per week to do intakes. This is not a problem during weeks that all are present. However, psychiatrists tend to take holidays, visit conferences, and so on. So, if the psychiatrist with the most intakes per week would go on leave, this might affect the behavior of the queue length considerably. This raises the question about the difference in allocation of capacity allotted to the psychiatrists. What are the consequences on the distribution and average of the waiting times if they would all have the same weekly capacity?
2. The psychiatrists tend to plan their holidays after each other, to reduce the variation in the service capacity. What if they would synchronize their holidays, to the extent possible, rather than spread their holidays?
3. Finally, suppose the psychiatrists would do 2 more intakes per week in busy times and 2 less in quiet weeks. Assuming that the system is stable, i.e., the service capacity exceeds the demand, then on average the psychiatrists would not do more intakes, i.e., their workload would not increase, but the queue length may be controlled better.

To evaluate the effect of these suggestions on reducing the queueing dynamics we develop a simple simulator and a number of plots. The simulation of a queueing system involves the specification of its behavior over time. The easiest method to construct queueing processes, hence to develop simulations, is to 'chop up' time in periods and develop recursions for the behavior of the queue from period to period. Note that the length of such a period depends on the case for which the model is developed. For instance, to study queueing processes at a supermarket, a period can consist of 5 minutes, while for a production environment, e.g., a job shop, it can be a day, or even a week.

Using fixed sized periods has its advantages as it does not require to specify specific inter-arrival times or service times of individual customers. Only the number of arrivals in a period and the number of potential services need to be specified, which is useful since in many practical settings, e.g., production environments, it is easier to provide data in these terms than in terms of inter-arrival and service times. It is, however, necessary to make some careful choices about timing.

Let us *define*



## 2 Single-Station Queueing Systems

$$\begin{aligned}
a_k &= \text{number of jobs that arrive at period } k, \\
c_k &= \text{number of jobs that can be served during period } k, \\
d_k &= \text{number of jobs that depart the queue during period } k, \\
Q_k &= \text{number of jobs in queue at the end of period } k.
\end{aligned} \tag{2.6}$$

In the sequel we also call  $a_k$  the *batch* of job arrivals in period  $k$ . The definition of  $a_k$  is a bit subtle: we may assume that the arriving jobs arrive either at the start or at the end of the period. In the first case, the jobs can be served during period  $k$ , in the latter case, they *cannot* be served during period  $k$ .

Since  $Q_{k-1}$  is the queue length at the end of period  $k-1$ , it must also be the queue length at the start of period  $k$ . Assuming that jobs that arrive in period  $k$  cannot be served in period  $k$ , the number of customers that can depart from the queue in period  $k$  is

$$d_k = \min\{Q_{k-1}, c_k\}, \tag{2.7a}$$

since only the jobs that are present at the start of the period, i.e.,  $Q_{k-1}$ , can be served if the capacity exceeds the queue length. Now that we know the number of departures, the queue at the end of period  $k$  is given by

$$Q_k = Q_{k-1} - d_k + a_k. \tag{2.7b}$$

As an example, suppose that  $c_k = 7$  for all  $k$ , and  $a_1 = 5$ ,  $a_2 = 4$  and  $a_3 = 9$ ; also  $Q_0 = 8$ . Then,  $d_1 = 7$ ,  $Q_1 = 8 - 7 + 5 = 6$ ,  $d_2 = 6$ ,  $Q_2 = 6 - 6 + 4 = 4$ ,  $d_3 = 4$ ,  $Q_3 = 4 - 4 + 9 = 9$ , and so on.

Of course we are not going to carry out these computations by hand. Typically we use company data about the sequence of arrivals  $\{a_k\}_{k=1,2,\dots}$  and the capacity  $\{c_k\}_{k=1,\dots}$  and feed this data into a computer to compute the recursions (2.7). If we do not have sufficient data we make a probability model for these data and use the computer to generate random numbers with, hopefully, similar characteristics as the real data. At any rate, from this point on we assume that it is easy, by means of computers, to obtain numbers  $a_1, \dots, a_n$  for  $n \gg 1000$ , and so on.

We now show how to adapt (2.6) and (2.7) to the case introduced above.

As a first step we model the arrival process of patients as a Poisson process, c.f., Section 2.2. The duration of a period is taken to be a week. The average number of arrivals per period, based on data of the company, was slightly less than 12 per week; in the simulation we set it to  $\lambda = 11.8$  per week. We model the capacity in the form of a matrix such that row  $i$  corresponds to the weekly capacity of psychiatrist  $i$ :

$$C = \begin{pmatrix} 1 & 1 & 1 & \dots \\ 1 & 1 & 1 & \dots \\ 1 & 1 & 1 & \dots \\ 3 & 3 & 3 & \dots \\ 9 & 9 & 9 & \dots \end{pmatrix}$$

Thus, psychiatrists 1, 2, and 3 do just one intake per week, the fourth does 3, and the fifth does 9 intakes per week. The sum over column  $k$  is the total service capacity for week  $k$  of all psychiatrists together.

With the matrix  $C$  it is simple to make other capacity schemes. A more balanced scheme would be like this:

$$C = \begin{pmatrix} 2 & 2 & 2 & \dots \\ 2 & 2 & 2 & \dots \\ 3 & 3 & 3 & \dots \\ 4 & 4 & 4 & \dots \\ 4 & 4 & 4 & \dots \end{pmatrix},$$



## 2.4 Construction of Discrete-Time Queueing Processes

We next include the effects of holidays on the capacity. This is easily done by setting the capacity of a certain psychiatrist to 0 in a certain week. Let's assume that just one psychiatrist is on leave in a week, each psychiatrist has one week per five weeks off, and the psychiatrists' holiday schemes rotate. To model this, we set  $C_{1,1} = C_{2,2} = \dots = C_{1,6} = C_{2,7} = \dots = 0$ , i.e.,

$$C = \begin{pmatrix} 0 & 2 & 2 & 2 & 2 & 0 & \dots \\ 2 & 0 & 2 & 2 & 2 & 2 & \dots \\ 3 & 3 & 0 & 3 & 3 & 3 & \dots \\ 4 & 4 & 4 & 0 & 4 & 4 & \dots \\ 4 & 4 & 4 & 4 & 0 & 4 & \dots \end{pmatrix},$$

Hence, the total average capacity must be  $4/5 \cdot (2 + 2 + 3 + 4 + 4) = 12$  patients per week. The other holiday scheme—all psychiatrists take holiday in the same week—corresponds to setting entire columns to zero, i.e.,  $C_{i,5} = C_{i,10} = \dots = 0$  for week 5, 10, and so on. Note that all these variations in holiday schemes result in the same average capacity.

Now that we have modeled the arrivals and the capacities, we can use the recursions (2.7) to simulate the queue length process for the four different scenarios proposed by the psychiatrists, unbalanced versus balanced capacity, and spread out holidays versus simultaneous holidays. The results are shown in Figure 2.4. It is apparent that suggestions 1 and 2 above do not significantly affect the behavior of the queue length process.

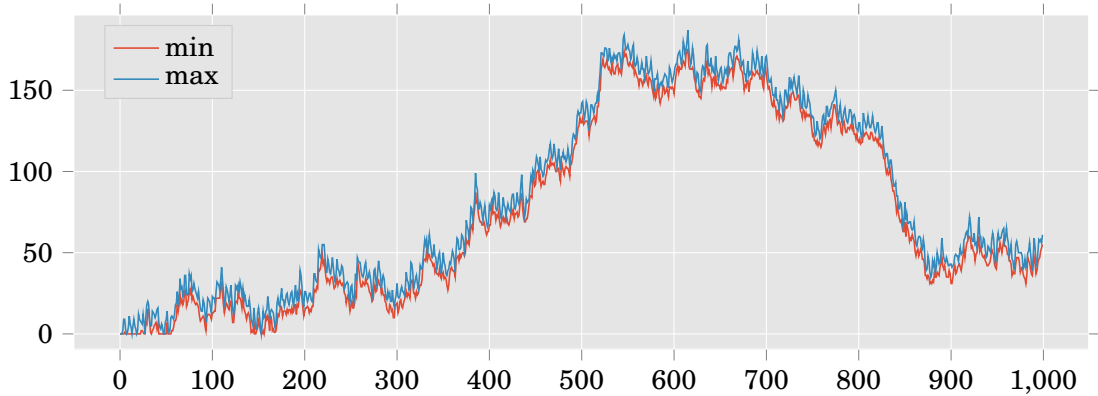


Figure 2.4: Effect of capacity and holiday plans. We plot for time point the maximum and the minimum queue length for each of the policies. We see that the effect of each of these policies is, for all practical purposes, negligible.

Now we consider suggestion 3, which comes down to doing more intakes when it is busy, and do less when it is quiet. A simple rule to implement this is by considering last week's queue  $Q_{n-1}$ : if  $Q_{n-1} < 12$ , i.e., the service capacity of one week, then do  $e$  intakes less. Here,  $e = 1$  or 2, or perhaps a larger number; it corresponds to the amount of control we want to exercise. When  $Q_{n-1} > 24$ , i.e., larger than two weeks of intakes, do  $e$  intakes more. Let's consider three different control levels,  $e = 1$ ,  $e = 2$ , and  $e = 5$ ; thus in the last case all psychiatrists do one extra intake. The previous simulation shows that it is safe to disregard the holiday plans, so just assume a flat service capacity of 12 intakes a week.

Figure 2.5 shows a striking difference indeed. The queue does not explode any more, and already taking  $e = 1$  has a large influence.

## 2 Single-Station Queueing Systems

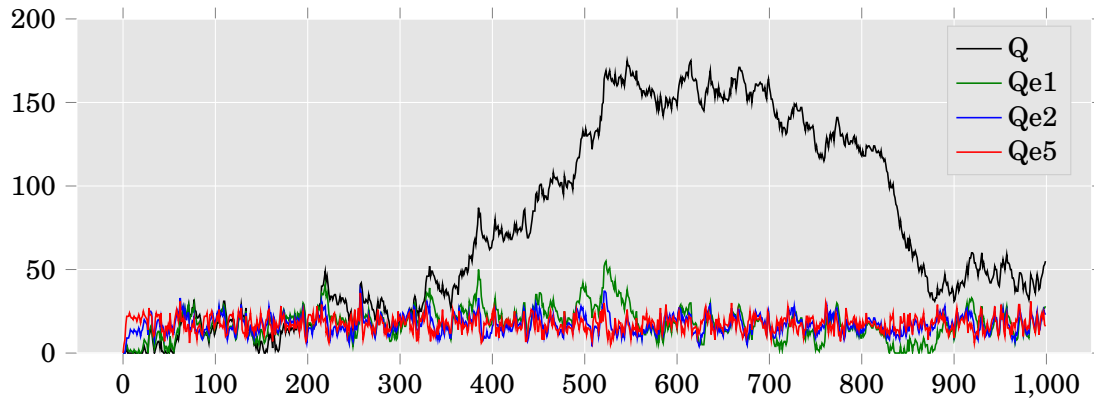


Figure 2.5: Controlling the number of intakes. Clearly, adapting the service rate ‘does wonders’ to control the queue length.

From the simulation experiment we learn that changing holiday plans or spreading the work over multiple servers, i.e., psychiatrists, does not significantly affect the queueing behavior. However, controlling the service rate as a function of the queue length improves the situation quite dramatically.

Thus, we made models for the arrival and service processes and then developed a simple recursion of the type (2.7) to *construct* a queueing process. Finally, for the analysis, we made plots of these processes.

In more abstract terms the study of queueing system is focused on studying the probabilistic properties of the queueing length process and related concepts such as waiting time, server occupancy, fraction of customers lost, and so on. Once we have constructed the queueing process we can compute all performance measures of relevance, such as the average waiting time, c.f. Section ???. If it turns out that the performance of the system is not according to what we desire, we can change parts of the system with the aim to improve the situation and assess the effect of this change. For instance, if the average waiting time is too long, we might add service capacity with the aim to reduce the service times, hence reduce the average waiting time. With simulation it is easy to study the effect of, hence evaluate, such decisions.

Observe that, even with these simple recursions, we can obtain considerable insight into this, otherwise, very complicated controlled queueing process. (If the reader doubts the value of simulation, s/he should try to develop other mathematical methods to analyze multi-server queueing system with vacations, of which this is an example. Warning, do not even attempt: you’ll fail as it is most probably too hard.) The simplicity of the recursions is deceitful, but with these simple recursions we can analyze many practical queueing situations. Together with students the author applied it numerous times, for instance,

- Should a certain hospital invest in a new MRI scanner to reduce waiting times?
- When to switch on and off a tin bath at an electronics component factory?
- What is the effect of reducing the number of jobs in a paint factory?
- Post parcel routing in a post sorting center.
- Controlling inventories at various companies.

- Throughput time analysis at courts.

And so on, and so on. The recursions are indeed astonishingly useful. We therefore urge the reader to practice with this type of queueing modeling. The exercises below provide ample material for this purpose.

In passing we remark that yet more powerful simulations can be carried out with, so-called, event-based simulations. We refer to (? , Section 4.5) for a general description of this technique. The author applied this to quite complicated queueing networks:

- Analysis of the packaging process of a large beer factory.
- Performance analysis of large telecommunication networks
- Optimization of truck routing and queueing for a sugar factory.

Due to lack of time, we decided not to include this.

The reader should understand from the above case that, once we have the recursions, we can analyze the system and make plots to evaluate suggestions for improvement. Thus, getting the recursions is crucial to construct, i.e., model, queueing processes. For this reason, most of the exercises below focus on obtaining recursions for many different queueing systems.

A comment is required about the modeling exercises below. It may be that the recursions you find are not identical to the recursions in the solution; the reason is that the assumptions you make might not be equal the ones I make. I don't quite know how get out of this paradoxical situation. In a sense, to completely specify the model, we need the recursions. However, if the problem statement would contain the recursions, there would be nothing left to practice anymore. Another way is to make the problem description five times as long, but this is also undesirable. So, let's be pragmatic: the aim is that you practice with modeling, and that you learn from the solutions. If you obtain *reasonable* recursions, but they are different from mine, then your answer is just as good.

### Exercises

**Exercise 2.4.1.** What are the consequences of setting  $d_k = \min\{Q_{k-1} + a_k, c_k\}$  rather than the definition (2.7a)?

**Exercise 2.4.2.** (Queue with Blocking) Consider a queueing system under daily review, i.e., at the end of the day the queue length is measured. We assume that at the end of the day no jobs are still in service. We assume that jobs that arrive at day  $n$  cannot be served in day  $n$ . The queue length cannot exceed level  $K$ . Formulate a set of recursions to cover this case.

**Exercise 2.4.3.** (Yield loss) A machine produces items, but a fraction  $p$  of the items produced in a period turn out to be faulty, and have to be made anew. Develop a set of recursions to cover this case.

**Exercise 2.4.4.** (Rework) A machine produces items, but a fraction  $p$  of the items do not meet the quality requirements after the first service but need some extra service time but less than an entirely new arriving job. Make a model to analyze this case. Compare this case with the yield loss problem above.

Let's assume that the repair of a faulty requires half of the work of a new job, and that the faulty jobs are processed with priority over the new jobs. (There are of course many different

## 2 Single-Station Queueing Systems

policies to treat rework; here we make this assumption. For your interest: another possibility is that faulty items are processed at the end of the day. Yet another possibility is that faulty items are collected until there are  $N$ , say, and then the entire batch of  $N$  is repaired.)

**Exercise 2.4.5.** (Cost models) A single-server queueing station processes customers. At the start of a period the server capacity is chosen, so that for period  $k$  the capacity is  $c_k$ . Demand that arrives in a period can be served in that period. It costs  $\beta$  per unit time per unit processing capacity to operate the machine, i.e., to have it switched on. There is also a cost  $h$  per unit time per job in the system. Make a cost model to analyze the long-run average cost for this case.

**Exercise 2.4.6.** (N-policies) A machine can switch on and off. If the queue length hits  $N$ , the machine switches on, and if the system becomes empty, the machine switches off. It costs  $K$  to switch on the machine. There is also a cost  $\beta$  per unit time while the machine is switched on, and it costs  $h$  per unit time per customer in the system. Make a cost model.

**Exercise 2.4.7** (use=false). (Queue with setups) One server serves two parallel queues, one at a time. After serving a batch of 20 jobs of one queue the server moves to the other queue. The change of queue requires one period setup time.

**Exercise 2.4.8.** How would you model (in terms of recursions) a server whose capacity depends on the queue length? Consider, as an example, a rule such that the server only works if the queue is larger than a threshold  $t$ .

**Exercise 2.4.9.** (Fair queueing) One server serves two queues. Each queue receives service capacity in proportion to its queue length. Derive a set of recursions to analyze this situation.

**Exercise 2.4.10.** (Priority queueing) Another interesting situation is a system with two queues served by one server, but such that one queue, queue  $A$  gets priority over the other queue. Again find a set of recursions to describe this case.

**Exercise 2.4.11.** (Queues with reserved service capacity) Consider a single-server that serves two parallel queues. Each queue receives a minimal service capacity every period. Reserved capacity unused for one queue can be used to serve the other queue. Any extra capacity beyond the reserved capacity is given to queue  $A$  with priority. Formulate a set of recursions to analyze this situation.

**Exercise 2.4.12.** (Queue with protected service capacity, lost capacity) Consider a single-server that serves two parallel queues. Each queue receives a minimal service capacity every period. Reserved capacity unused for one queue cannot be used to serve the other queue. Any extra capacity beyond the reserved capacity is given to queue  $A$  with priority. Formulate a set of recursions to analyze this situation.

Let  $r_A$  be the reserved capacity for queue  $A$ , and likewise for  $r_B$ . We assume of course that  $c_k \geq r_A + r_B$ , for all  $k$ .

**Exercise 2.4.13.** (Tandem networks) Consider a production network with two production stations in tandem, that is, the jobs processed by station  $A$  are in the next period to the downstream Station  $B$ . Extend the recursions of (2.7) to simulate this situation.

**Exercise 2.4.14.** (A tandem queue with blocking) Consider a production network with two production stations in tandem with blocking: when intermediate queue, i.e., the queue front of Station  $B$ , exceeds some level  $M$ , then station  $A$  has to stop producing, and when  $Q_k^B < M$  station  $A$  is not allowed to produce more than the intermediate queue can contain. Extend the recursions of (2.7) to simulate this situation.

**Exercise 2.4.15.** (Merging departure streams) Consider another production situation with two machines, A and B say, that send their products to Station C. Derive a set of recursion relations to simulate this system.

**Exercise 2.4.16.** (Estimating the lead time distribution.) Take  $d_k = \min\{Q_{k-1} + a_k, c_k\}$ , and assume that jobs are served in FIFO sequence. Find an expression for the shortest possible waiting time  $W_-(k)$  of a job that arrives at time  $k$ , and an expression for the largest possible waiting time  $W_+(k)$

**Exercise 2.4.17.** (Merging incoming streams) *You can skip this problem for the midterm as I changed the answer quite a bit after the lecture. It is still interesting though.* Consider a single-server queue that servers two customer ‘streams’ in a FIFO discipline. Thus, both streams enter one queue that is served by the server. Let  $\{a_k^a\}$  be the number of arrivals of stream  $a$  in period  $k$  and  $\{a_k^b\}$  be the number of arrivals of stream  $b$ . Find a set of recursions by which it becomes possible to analyze the waiting time distribution of each of the streams. Assume that the service capacity  $c$  is constant for all periods, and that jobs that arrive in period  $k$  can also be served in period  $k$ .



**Exercise 2.4.18.** (Splitting streams) Consider a machine (a paint mixing machine) that produces products for two separate downstream machines A and B (two different paint packaging machines), each with its own queue. Suppose we want to analyze the queue in front of station A. For this we need to know the arrivals to station A, that is, the departures of the mixing station that go to station A. Provide a set of recursions to simulate this system.

**Exercise 2.4.19.** (Inventory control) The recursions used in the exercises above can also be applied to analyze inventory control policies. Consider a production system that can produce maximally  $M_k$  items per week during normal working hours, and maximally  $N_k$  items during extra (weekend and evening hours). Let, for period  $k$ ,

- $D_k$  = Demand in week  $k$ ,
- $S_k$  = Sales, i.e., number of items sold, in week  $k$ ,
- $r_k$  = Revenue per item sold in week  $k$ ,
- $X_k$  = Number of items produced in week  $k$  during normal hours,
- $Y_k$  = Number of items produced in week  $k$  during extra hours,
- $c_k$  = Production cost per item during normal hours,
- $d_k$  = Production cost per item during extra hours,
- $h_k$  = holding cost per item, due at the end of week  $k$ ,
- $I_k$  = On hand inventory level at the end of week  $k$ .

Management needs a production plan that specifies for the next  $T$  weeks the number of items to be produced per week. Formulate this problem as an LP problem, taking into account the inventory dynamics.

**Exercise 2.4.20.** Implement Eqs. 2.7 in a computer program and simulate a simple single-server queueing system.

### Hints

**Hint 2.4.16:** Consider a numerical example. Suppose  $Q_{k-1} = 20$ . Suppose that the capacity is  $c_k = 3$  for all  $k$ . Then a job that arrives in the  $k$ th period, must wait at least  $20/3$  (plus rounding) periods before it can leave the system. Now generalize this numerical example.

**Hint 2.4.19:** Formulate the decision variables/controls, the objective and the constraints.

### Solutions

**Solution 2.4.1:** The assumption is that the jobs arrive at the start of period  $k$ , before service in period  $k$  starts, rather than at the end of the period. Therefore the arrivals at period  $k$  can also be served during period  $k$ .

**Solution 2.4.2:** All jobs that arrive such that the queue become larger than  $K$  must be dropped. Thus, the accepted arrivals  $a'_n$  in week  $n$  are such that  $a'_n = \min\{a_n, K - Q_{n-1}\}$ , where  $a_n$  are all arrivals in period  $n$ . The rest of the recursions remain the same.

**Solution 2.4.3:** The amount produced in period  $k$  is  $d_k$ . Thus,  $pd_k$  is the amount lost, neglecting rounding errors for the moment. Thus,  $pd_k$  items have to be fed back to the system in the next period to be remade. Therefore the total amount of arrivals in period  $k + 1$  is  $a'_{k+1} = a_{k+1} + pd_k$ , i.e., the external arrivals plus the extra items. Now use the standard recursions but with the  $\{a'_k\}$  rather than  $\{a_k\}$ .

Can you use these recursions to show that the long-run average service capacity  $n^{-1} \sum_{i=1}^n c_i$  must be larger than  $\lambda(1 + p)$ ?

If you like you can incorporate time-dependent failure rates  $\{p_k\}$  too. Whether this makes practical sense depends on the context of course.

**Solution 2.4.4:** Suppose again that a fraction  $p$  is faulty. Since these faulty items require less processing time than a new job, the service capacity  $c_k$ , i.e., the number of jobs that can be processed in period  $k$ , is a bit bigger; part of the capacity is spent on new jobs but another part is spent on the faulty jobs. By the assumptions above, the repair of a faulty requires half of the work of a new job, and the faulty jobs are processed with priority over the new jobs. Assume queue  $A$  contains the faulty items, and queue  $B$  the new jobs. Then the recursions become:

$$\begin{aligned} d_{k,A} &= \min\{Q_{k-1,A}, 2c_k\}, \text{ as faulty jobs require half of the processing time) } \\ c_{k,B} &= c_k - d_{k,A}/2, \\ d_{k,B} &= \min\{Q_{k-1,B}, c_{k,B}\}, \\ Q_{k,A} &= Q_{k-1,A} + a_{k,A} - d_{k,A}, \\ Q_{k,B} &= Q_{k-1,B} + a_{k,B} - d_{k,B}. \end{aligned}$$

**Solution 2.4.5:** First consider the dynamics of the queue. Since the capacity is chosen at the start of the period:

$$\begin{aligned} d_k &= \min\{Q_{k-1} + a_k, c_k\} \\ Q_k &= Q_{k-1} + a_k - d_k. \end{aligned}$$

## 2.4 Construction of Discrete-Time Queueing Processes

The cost to operate the server during period  $k$  is  $\beta c_k$ . Thus, the total cost up to some time  $T$  for the server must be  $\beta \sum_{k=1}^T c_k$ . In period  $k$  we also have to pay  $hQ_k$ , since  $h$  is the cost per customer per period in the system. Thus, the long-run average cost is

$$\frac{1}{T} \sum_{k=1}^T (\beta c_k + hQ_k).$$

It is an interesting problem to find a policy that minimizes (the expectation of) this cost. The policy is such that the capacity for period  $k$  can be chosen based on the queue length  $Q_{k-1}$  and estimates of the demands  $\hat{d}_k, \hat{d}_{k+1}, \dots$ . This problem is not easy, as far as I can see.

**Solution 2.4.6:** First we need to implement the N-policy. For this we need an extra variable to keep track of the state of the server. Let  $I_k = 1$  if the machine is on in period  $k$  and  $I_k = 0$  if it is off. Then  $\{I_k\}$  must satisfy the relation

$$I_{k+1} = \begin{cases} 1 & \text{if } Q_k \geq N, \\ I_k & \text{if } 0 < Q_k < N, \\ 0 & \text{if } Q_k = 0, \end{cases}$$

and assume that  $I_0 = 0$  at the start, i.e., the machine is off. Thus, we can write:

$$I_{k+1} = \mathbb{1}_{Q_k \geq N} + I_k \mathbb{1}_{0 < Q_k < N} + 0 \cdot \mathbb{1}_{Q_k = 0}.$$

With  $I_k$  it follows that  $d_k = \min\{Q_{k-1}, I_k c_k\}$ , from which  $Q_k$  follows, and so on.

The machine cost for period  $k$  is  $\beta I_k$ , because only when the machine is on we have to pay  $\beta$ , and the queueing cost is  $hQ_k$ . To determine the total switching cost is harder as we need to determine how often the machine has been switched on up to time  $T$ . Observe that the machine is switched on in period  $k$  if  $I_{k-1} = 0$  and  $I_k = 1$ . Thus, whenever  $I_k - I_{k-1} = 1$  the machine is switched on, when  $I_k - I_{k-1} = 0$  the state of the machine remains the same, and if  $I_k - I_{k-1} = -1$  the machine is switched off. In other words  $\max\{I_k - I_{k-1}, 0\}$  captures what we need. The total cost up to time  $T$  becomes:

$$\sum_{k=1}^T (\beta I_k + hQ_k + K \max\{I_k - I_{k-1}, 0\}).$$

**Solution 2.4.7:**

**Solution 2.4.8:** One model could be to let the server only switch on when the queue is larger than some threshold  $t$ , and when the server is on, it works at rate  $c$  per period. In that case,  $c_k = c \mathbb{1}_{Q_{k-1} > t}$ .

**Solution 2.4.9:** Let  $c_k^i$  be the capacity allocated to queue  $i$  in period  $k$ . The fair rule gives that

$$c_k^1 = \frac{Q_{k-1}^1}{Q_{k-1}^1 + Q_{k-1}^2} c = c - c_k^2.$$

Then,

$$\begin{aligned} d_k^1 &= \min\{Q_{k-1}^1, c_k^1\}, \\ Q_k^1 &= Q_{k-1}^1 + a_k^1 - d_k^1, \end{aligned}$$

and likewise for the other queue.

## 2 Single-Station Queueing Systems

**Solution 2.4.10:** The rules below implement a strict priority rule for jobs type A, i.e., jobs sent into queue A.

$$\begin{aligned} d_{k,A} &= \min\{Q_{k-1,A}, c_k\}, \\ c_{k,B} &= c_k - d_{k,A}, \\ d_{k,B} &= \min\{Q_{k-1,B}, c_{k,B}\}, \\ Q_{k,A} &= Q_{k-1,A} + a_{k,A} - d_{k,A}, \\ Q_{k,B} &= Q_{k-1,B} + a_{k,B} - d_{k,B}. \end{aligned}$$

As an aside, another interesting rule to distribute the capacity  $c_k$  over the queues could be based on the principle of *equal division of the contested sum*. This principle is based on game theoretic ideas. Aumann and Maschler applied this principle to clarify certain division rules discussed in the Talmud to divide the legacy among a number of inheritors, each having a different claim size.

**Solution 2.4.11:** First determine how much capacity queue B minimally needs in period  $k$ :

$$c_{k,B} = \min\{Q_{k-1,B}, r_B\}$$

Observe that, since  $c_k \geq r_A + r_B$ , this rule ensures that queue A receives at least its reserved capacity  $r_A$ .

Since queue A is served with priority, we first give all capacity, except what queue B minimally needs, to queue A:

$$d_{k,A} = \min\{Q_{k-1,A}, c_k - c_{k,B}\}.$$

And then we can give any left over capacity to queue B, if needed.

$$d_{k,B} = \min\{Q_{k-1,B}, c_k - d_{k,A}\}.$$

An example is the weekly capacity offered by a psychiatrist at a hospital. Part of the weekly capacity is reserved/allocated/assigned to serve certain patients groups. For instance, each week the psychiatrist does at most five intakes of new patients, provided there are any, and the rest of the capacity is used to treat other patients. The existing patients can also be divided in different groups, each receiving a minimal capacity. If there are less patients of some group, then the capacity can be planned/given to other patient groups.

**Solution 2.4.12:** Queue A can use all capacity, except what is reserved for queue B:

$$d_{k,A} = \min\{Q_{A,k-1}, c_k - r_B\}.$$

Observe that, since  $c_k \geq r_A + r_B$ , this rule ensures that queue A receives at least its reserved capacity  $r_A$ .

Queue B cannot receive more than  $c_k - r_A$ , since  $r_A$  is allocated to queue A, and if queue A does not use all of  $r_A$ , then the surplus is lost. Also, queue B cannot get more than  $c_k - d_{k,A}$  as this is what remains after serving queue A. Thus, letting  $c_{k,B} = \min\{c_k - r_A, c_k - d_{k,A}\} = c_k - \max\{r_A, d_{k,A}\}$ , we see that for queue B:

$$d_{k,B} = \min\{Q_{B,k-1}, c_{k,B}\}.$$



## 2.4 Construction of Discrete-Time Queueing Processes

An example can be the operation room of a hospital. There is a weekly capacity, part of the capacity is reserved for emergencies. It might not be possible to assign this reserved capacity to other patient groups, because it should be available at all times for emergency patients. A result of this is that unused capacity is lost.

In practice it may not be as extreme as in the model, but still part of the unused capacity is lost. ‘Use it, or lose it’, is what often, but not always, applies to service capacity.

**Solution 2.4.13:** Let  $a_k$  be the external arrivals at station A. Then:

$$\begin{aligned} d_k^A &= \min\{Q_{k-1}^A, c_k^A\}, \\ Q_k^A &= Q_{k-1}^A - d_k^A + a_k. \end{aligned} \quad (2.8)$$

The departures of the first station during period  $k$  are the arrivals at station B at the end of period  $k$ , i.e.,  $a_k^B = d_k^A$ . Thus,

$$\begin{aligned} a_k^B &= d_k^A, \\ d_k^B &= \min\{Q_{k-1}^B, c_k^B\}, \\ Q_k^B &= Q_{k-1}^B - d_k^B + a_k^B. \end{aligned} \quad (2.9)$$

**Solution 2.4.14:**

$$\begin{aligned} d_k^A &= \min\{Q_{k-1}^A, c_k^A, M - Q_{k-1}^B\}, \\ Q_k^A &= Q_{k-1}^A - d_k^A + a_k, \\ a_k^B &= d_k^A, \\ d_k^B &= \min\{Q_{k-1}^B, c_k^B\}, \\ Q_k^B &= Q_{k-1}^B - d_k^B + a_k^B. \end{aligned} \quad (2.10)$$

This is a bit subtle: since there is room  $M - Q_{k-1}^B$  at the intermediate buffer and  $d_k^A \leq M - Q_{k-1}^B$ , we know that in the worst case, i.e., when  $c_k^B = 0$ , still  $Q_k^B = Q_{k-1}^B + d_k^A$ . Thus, we are sure that the queue length of the intermediate queue will not exceed  $M$ .

There is still a small problem: What if for the first initial periods  $M < Q_{k-1}^B$ . Then  $M - Q_{k-1}^B < 0$  and then by the specification above,  $d_k^A < 0$ . This is not what we want. Therefore,

$$d_k^A = \min\{Q_{k-1}^A, c_k^A, \max\{M - Q_{k-1}^B, 0\}\}.$$

**Solution 2.4.15:** Realize that Stations A and B have their own arrivals.

$$\begin{aligned} d_k^A &= \min\{Q_{k-1}^A, c_k^A\}, \\ Q_k^A &= Q_{k-1}^A - d_k^A + a_k^A, \\ d_k^B &= \min\{Q_{k-1}^B, c_k^B\}, \\ Q_k^B &= Q_{k-1}^B - d_k^B + a_k^B, \\ a_k^C &= d_k^A + d_k^B, \\ d_k^C &= \min\{Q_{k-1}^C, c_k^C\}, \\ Q_k^C &= Q_{k-1}^C - d_k^C + a_k^C. \end{aligned} \quad (2.11)$$

## 2 Single-Station Queueing Systems

**Solution 2.4.16:** Let's tag the first customer that arrives in period  $k$ . This tagged customer sees  $Q_{k-1}$  customer in the system, hence the tagged customer's service can only start after all  $Q_{k-1}$  customers have been served. Now, if  $Q_{k-1} - c_k > 0$ , there are still people in front of the tagged customer. In fact, as long as  $c_k + c_{k+1} + \dots + c_m < Q_{k-1}$  the number of customers in front of the tagged customer are still in the system.

We can also tag the last customer that arrives in period  $k$ . This customer will certainly have left if  $m$  is such that  $c_k + \dots + c_m > Q_{k-1} + a_k$ .

In formulas the above comes down to the following. A job that arrives in period  $k$  cannot be served before period

$$W_{-,k} := \max \left\{ m : \sum_{i=k}^{k+m} c_i < Q_{k-1} \right\},$$

and it must have been served before period

$$W_{+,k} := \min \left\{ m : \sum_{i=k}^{k+m} c_i \geq Q_k \right\}.$$

Thus, the waiting time of jobs arriving in period  $k$  must lie in the interval  $[W_{-,k}, W_{+,k}]$ .

**Solution 2.4.17:** The behavior of the queue length process is easy:

$$\begin{aligned} d_k &= \min\{Q_{k-1} + a_k^a + a_k^b, c\}, \\ Q_k &= Q_{k-1} + a_k^a + a_k^b - d_k. \end{aligned}$$

To determine the waiting times, observe that any arrival in period  $k$ , independent of the stream, has to wait until all jobs at the start of the period in queue, i.e.,  $Q_{k-1} - d_k$ , are cleared; note that we assume here that the jobs served in period  $k$  depart at the start of the interval. Thus, the minimal waiting time is  $W_{k,-} = \lfloor Q_{k-1}/c \rfloor$ . Similarly, the maximal waiting time is  $W_{k,+} = \lfloor (Q_{k-1} + a_k^a + a_k^b)/c \rfloor$ .

The remaining problem is to make a model to 'distribute' the time between  $W_{k,-}$  and  $W_{k,+}$  over the two streams.

A simple model is to assume that the waiting time is averaged over the jobs. Then each job perceives a waiting time of

$$\frac{W_{k,-} + W_{k,+}}{2}.$$

Another way is to give priority to  $a$  customers, but only for the jobs that arrive in this period. (Hence, this is different from the priority queue. There priority customers can overtake customers that arrived earlier. In the present case this is not allowed, jobs of type  $a$  that arrive in period  $k$  cannot overtake  $b$  jobs that arrived prior to period  $k$ .) Making this completely explicit (so that the recursion can be fed to the computer) requires a bit of work however. It is important to understand the trick we will discuss now because we will use it to model queueing systems with batching. Observe that the first job of the  $a$  stream only has to wait for  $W_{k,-}$ , the second job must wait  $W_{k,-} + 1$ , and so on. Thus, the waiting time  $W_k^a$  for the  $a_k^a$  items is such that

$$W_{k,-}^a := \lfloor Q_{k-1}/c \rfloor \leq W_k^a \leq \lfloor (Q_{k-1} + a_k^a)/c \rfloor =: W_{k,+}^a.$$

Similarly, for the  $b$  jobs must be

$$W_{k,-}^b := \lfloor (Q_{k-1} + a_k^a)/c \rfloor \leq W_k^b \leq \lfloor (Q_{k-1} + a_k^a + a_k^b)/c \rfloor =: W_{k,+}^b.$$

## 2.4 Construction of Discrete-Time Queueing Processes

Note that  $W_{k,+}^a = W_{k,-}^b$ . It is then sensible to set

$$W_k^a = \frac{W_{k,-}^a + W_{k,+}^a}{2},$$

$$W_k^b = \frac{W_{k,-}^b + W_{k,+}^b}{2}.$$

I currently lack time to check whether the above does not contain any off-by-one errors. For future work, implement it in a simulator. . .



**Solution 2.4.18:** 1. Realize that the recursions of Eq (2.7) applied to the queueing situation at the first machine provide us with the total number of departures  $d_k$  during period  $k$ . However, it does not tell us about the type of these departures. Thus, to compute the queue in front of station A, we need to know the number of departures of type A, rather than the total number of departures of the first station.

2. It is reasonable that the number of jobs of type A in queue at the first station is equal to

$$Q_k \frac{\lambda_A}{\lambda_A + \lambda_B}.$$

It is therefore reasonable to assume that the capacity  $c_k$  of the first station is also shared in this proportion to type A and B jobs. Thus, the number of departures to station A is

$$d_k(A) = \frac{\lambda_A}{\lambda_A + \lambda_B} \min\{Q_{k-1}, c_k\}.$$

The rest of the recursions is very similar to what we did in earlier exercises.

**Solution 2.4.19:** The decision variables are  $X_k$ ,  $Y_k$  and  $S_k$  (note, it is not necessary to meet all demand: the production cost and profit may vary per period.) The objective is

$$\max \sum_{k=1}^T (r_k S_k - c_k X_k - d_k Y_k - h_k I_k).$$

The constraints are

$$\begin{aligned} 0 &\leq S_k \leq D_k, \\ 0 &\leq X_k \leq M_k, \\ 0 &\leq Y_k \leq N_k, \\ I_k &= I_{k-1} + X_k + Y_k - S_k. \\ I_k &\geq 0. \end{aligned}$$

**Solution 2.4.20:** Here is an example in python; please read the code to see how I approach the problem. The code is really easy, as it is nearly identical to the mathematical specification. You don't have to memorize the specific syntax of the code, of course, but it is (at least I find it) interesting to see how little code is actually necessary to set things up.

Below I fix the seed of the random number generator to ensure that I always get the same results from the simulator. The arrival process is Poisson, and the number of services is fixed to 21 per period. I use `Q = np.zeros_like(a)` to make an array of the same size as the number of arrival  $a$  initially set to zeros. The rest of the code is nearly identical to the formulas in the text.

## 2 Single-Station Queueing Systems

```
import numpy as np

np.random.seed(3) # fix the seed

labda = 20
mu = 21
```

These are the number of arrivals in each period.

```
a = np.random.poisson(labda, 10)
a

array([21, 17, 14, 10, 22, 22, 17, 17, 19, 21])
```

The number of potential services

```
c = mu * np.ones_like(a)
c

array([21, 21, 21, 21, 21, 21, 21, 21, 21, 21])
```

Now for the queueing recursions:

```
Q = np.zeros_like(a)
d = np.zeros_like(a)
Q[0] = 10 # initial queue length

for k in range(1, len(a)):
    d[k] = min(Q[k - 1], c[k])
    Q[k] = Q[k - 1] - d[k] + a[k]
```

These are the departures for each period:

```
d

array([ 0, 10, 17, 14, 10, 21, 21, 19, 17, 19])
```

The queue lengths

```
Q

array([10, 17, 14, 10, 22, 23, 19, 17, 19, 21])

loss = (Q > 20)
loss

array([False, False, False, False,  True,  True, False, False, False,
       True], dtype=bool)

loss.sum()
```

3

Here I define loss as the number of periods in which the queue length exceeds 20. Of course, any other threshold can be taken. Counting the number of such periods is very easy in python:  $(Q>20)$  gives all entries of  $Q$  such  $Q > 20$ , the function `sum()` just adds them.

Now all statistics:

```
d.mean()

14.800000000000001
```

## 2.4 Construction of Discrete-Time Queueing Processes

```
Q.mean()

17.199999999999999

Q.std()

4.3772137256478576

(Q > 20).sum()

3
```

Since this is a small example, the mean number of departures, i.e., `d.mean()`, is not equal to the arrival rate  $\lambda$ . Likewise for the computation of the mean, variance, and other statistical functions.

Now I am going to run the same code, but for a larger instance.

```
num = 1000
a = np.random.poisson(labda, num)
c = mu * np.ones_like(a)
Q = np.zeros_like(a)
d = np.zeros_like(a)
Q[0] = 10 # initial queue length

for k in range(1, len(a)):
    d[k] = min(Q[k - 1], c[k])
    Q[k] = Q[k - 1] - d[k] + a[k]

d.mean()

20.178000000000001

Q.mean()

28.420000000000002

Q.std()

10.155865300406461

(Q > 30).sum()/num * 100

34.200000000000003
```

I multiply with 100 to get a percentage. Clearly, many jobs see a long queue, the mean is already some 24 jobs. For this arrival rate a service capacity of  $\mu = 21$  is certainly too small.

Next, an example with the same arrival pattern but with a Poisson distributed number of jobs served per day, so the service rate is still 21, but the period capacity  $c$  is random variable with Poisson distribution with  $P(21)$

```
c = np.random.poisson(mu, num)
Q = np.zeros_like(a)
d = np.zeros_like(a)
Q[0] = 10 # initial queue length

for k in range(1, len(a)):
    d[k] = min(Q[k - 1], c[k])
    Q[k] = Q[k - 1] - d[k] + a[k]
```

## 2 Single-Station Queueing Systems

```
d.mean()

20.148

Q.mean()

42.518000000000001

Q.std()

22.841096208369684

(Q > 30).sum()/num * 100

62.399999999999999
```

Compared to the case with constant service capacity, i.e.,  $c = 21$  in each period, we see that the average waiting time increases, just as the number of periods in which the queue length exceeds 30. Clearly, variability in service capacity does not improve the performance of the queueing system, quite on the contrary. In later sections we will see why this is so.

You can try to implement some of the other cases of the exercises in this section and analyze them in the same way. Hopefully you understand from this discussion that once you have the recursions to construct/simulate the queueing process, you are ‘in business’. The rest is easy: make plots, do some counting, i.e., assemble statistics, vary parameters for sensitivity analysis, and so on.

## 2.5 Construction of the G/G/1 Queueing Process in Continuous Time

In the previous section we considered time in discrete ‘chunks’, minutes, hours, days, and so on. For given numbers of arrivals and capacity per period we use a set of recursions (2.7) to compute the departures and queue length per period. Another way to construct a queueing system is to consider inter-arrival times between consecutive customers and the service times each of these customers require. With this we obtain a description of the queueing system in continuous time. The goal of this section is to develop a set of recursions for the G/G/1 single-server queue.

Assume we are given, as basic data, the *arrival process*  $\{A(t); t \geq 0\}$ : the number of jobs that arrived during  $[0, t]$ . Thus,  $\{A(t); t \geq 0\}$  is a *counting process*.

From this arrival process we can obtain various other interesting concepts, such as the arrival times of individual jobs. Specially, if we know that  $A(s) = k - 1$  and  $A(t) = k$ , then the arrival time  $A_k$  of the  $k$ th job must lie somewhere in  $(s, t]$ . Thus, from  $\{A(t)\}$ , we can define

$$A_k = \min\{t : A(t) \geq k\}, \quad (2.12)$$

and set  $A_0 = 0$ . Once we have the set of arrival times  $\{A_k\}$ , the *inter-arrival times*  $\{X_k, k = 1, 2, \dots\}$  between consecutive customers can be constructed as

$$X_k = A_k - A_{k-1}. \quad (2.13)$$

## 2.5 Construction of the G/G/1 Queueing Process in Continuous Time

Often the basic data consists of the inter-arrival times  $\{X_k; k = 1, 2, \dots\}$  rather than the arrival times  $\{A_k\}$  or the number of arrivals  $\{A(t)\}$ . Then we construct the arrival times as

$$A_k = A_{k-1} + X_k,$$

with  $A_0 = 0$ . From the arrival times  $\{A_k\}$  we can, in turn, construct the arrival process  $\{A(t)\}$  as

$$A(t) = \sum_{k=1}^{\infty} \mathbb{1}_{A_k \leq t}, \quad (2.14a)$$

where  $\mathbb{1}$  is the indicator function. Thus, in the above we count all arrivals that occur up to time  $t$ . Another, equivalent, way to define  $A(t)$  is

$$A(t) = \max\{k : A_k \leq t\}. \quad (2.14b)$$

Clearly, we see that from the inter-arrival times  $\{X_k\}$  it is possible to construct  $\{A_k\}$  and  $\{A(t)\}$ , and the other way around, from  $\{A(t)\}$  we can find  $\{A_k\}$  and  $\{X_k\}$ . Figure 2.6 shows these relations graphically. To memorize it may be helpfull to write it like this:

$$\begin{aligned} A_k : \mathbb{N} &\rightarrow \mathbb{R}, & \text{job id (integer) to arrival time (real number),} \\ A(t) : \mathbb{R} &\rightarrow \mathbb{N}, & \text{time (real number) to number of jobs (integer).} \end{aligned}$$

To compute the departure times  $\{D_k\}$  we proceed in stages. The first stage is to construct the *waiting time in queue*  $\{W_{Q,k}\}$  as seen by the arrivals. In Figure 2.6 observe that the waiting time of the  $k$ th arrival must be equal to the waiting time of the  $k-1$ th customer plus the amount of *service time* required by job  $k-1$  minus the time that elapses between the arrival of job  $k-1$  and job  $k$ , unless the server becomes idle between jobs  $k-1$  and  $k$ . In other words,

$$W_{Q,k} = [W_{Q,k-1} + S_{k-1} - X_k]^+, \quad (2.15)$$

where  $[x]^+ = \max\{x, 0\}$ . If we set  $W_{Q,0} = 0$ , we can compute  $W_{Q,1}$  from this formula, and then  $W_{Q,2}$  and so on.

The time job  $k$  leaves the queue and moves on to the server is

$$D_{Q,k} = A_k + W_{Q,k},$$

because a job can only move to the server after its arrival plus the time it needs to wait in queue. Note that we here explicitly use the FIFO assumption.

Right after the job moves from the queue to the server, its service starts. Thus,  $D_{Q,k}$  is the epoch at which the service of job  $k$  starts. After completing its service, the job leaves the system. Hence, the *departure time of the system* is

$$D_k = D_{Q,k} + S_k.$$

The *sojourn time*, or *waiting time in the system*, is the time a job spends in the entire system. With the above relations we see that

$$W_k = D_k - A_k = D_{Q,k} + S_k - A_k = W_{Q,k} + S_k, \quad (2.16)$$

where each of these equations has its own interpretation.

## 2 Single-Station Queueing Systems

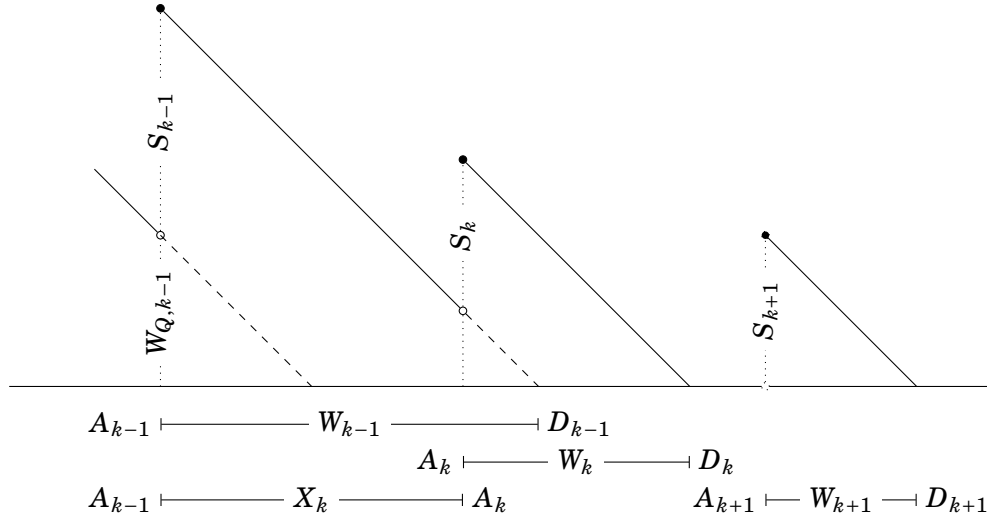


Figure 2.6: Construction of the  $G/G/1$  queue in continuous time. The sojourn time  $W_k$  of the  $k$ th job is sum of the work in queue  $W_{Q,k}$  at its arrival epoch  $A_k$  and its service time  $S_k$ ; its departure time is then  $D_k = A_k + W_k$ . The waiting time of job  $k$  is clearly equal to  $W_{k-1} - X_k$ . We also see that job  $k+1$  arrives at an empty system, hence its sojourn time  $W_{k+1} = S_{k+1}$ . Finally, the virtual waiting time process is shown by the lines with slope  $-1$ .

A bit of similar reasoning gives another recursion for  $W_k$ :

$$\begin{aligned} W_{Q,k} &= [W_{k-1} - X_k]^+, \\ W_k &= W_{Q,k} + S_k = [W_{k-1} - X_k]^+ + S_k. \end{aligned} \quad (2.17)$$

from which follows a recursion for  $D_k$

$$D_k = A_k + W_k. \quad (2.18)$$

This in turn specifies the departure process  $\{D(t)\}$  as

$$D(t) = \max\{k; D_k \leq t\} = \sum_{k=1}^{\infty} \mathbb{1}_{D_k \leq t}.$$

Once we have the arrival and departure processes it is easy to compute the *number of jobs in the system* at time  $t$  as

$$L(t) = A(t) - D(t) + L(0), \quad (2.19)$$

where  $L(0)$  is the number of jobs in the system at time  $t = 0$ ; typically we assume that  $L(0) = 0$ . Thus, if we were to plot  $A(t)$  and  $D(t)$  as functions of  $t$ , then the difference  $L(t)$  between the graphs of  $A(t)$  and  $D(t)$  tracks the number in the system, see Figure 2.7.

The number in the system as seen by the  $k$ th arrival is defined as

$$L_k = L(A_k), \quad (2.20)$$

since the  $k$ th job arrives at time  $A_k$  and  $\{L(s)\}$  is right-continuous.

Observe that in a queueing system, jobs can be in queue or in service. For this reason we distinguish between the number in the system  $L(t)$ , the number in queue  $L_Q(t)$ , and the number



## 2.5 Construction of the G/G/1 Queueing Process in Continuous Time

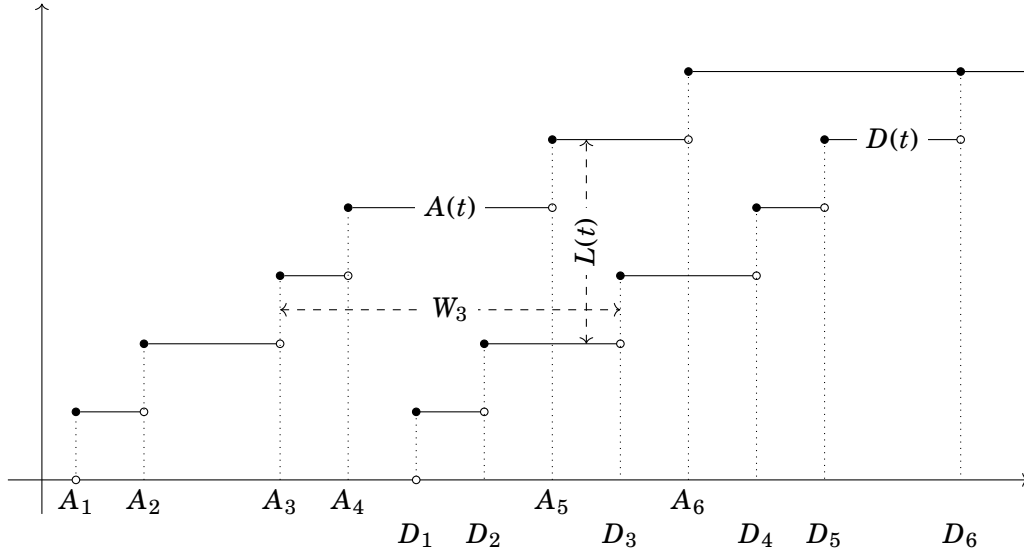


Figure 2.7: Relation between the arrival process  $\{A(t)\}$ , the departure process  $\{D(t)\}$ , the number in the system  $\{L(t)\}$  and the waiting times  $\{W_k\}$ .

of jobs in service  $L_s(t)$ . If we know  $D_Q(t)$ , i.e. the number of jobs that departed from the queue up to time  $t$ , then

$$L_Q(t) = A(t) - D_Q(t)$$

must be the number of jobs in queue. The above expressions for  $L(t)$  and  $L_Q(t)$  then show that the number in service must be

$$L_s(t) = D_Q(t) - D(t) = L(t) - L_Q(t).$$

Finally, the *virtual waiting time process*  $\{V(t)\}$  is the amount of waiting that an arrival would see if it would arrive at time  $t$ . To construct  $\{V(t)\}$ , we simply draw lines that start at points  $(A_k, W_k)$  and have slope  $-1$ , unless the line hits the  $x$ -axis, in which case the virtual waiting time remains zero until the next arrival occurs. Thus, the lines with slope  $-1$  in Figure 2.6 show (a sample path of) the virtual waiting time.

Observe that, just as in Section 2.4, we have obtained a set of recursions by which we can run a simulation of a queueing process of whatever length we need, provided we have a sequence of inter-arrival times  $\{X_k\}$  and service times  $\{S_k\}$ . A bit of experimentation with computer programs such as *R* or python will reveal that this is easy.

### Exercises

**Exercise 2.5.1.** 1. Assume that  $X_1 = 10$ ,  $X_2 = 5$ ,  $X_3 = 6$  and  $S_1 = 17$ ,  $S_2 = 20$  and  $S_3 = 5$ , compute the arrival times, waiting times in queue, the sojourn times and the departure times for these three customers.

**Exercise 2.5.2.** Suppose that  $X_k \in \{1, 3\}$  such that  $P\{X_k = 1\} = P\{X_k = 3\}$  and  $S_k \in \{1, 2\}$  with  $P\{S_k = 1\} = P\{S_k = 2\}$ . If  $W_{Q,0} = 3$ , what are the distributions of  $W_{Q,1}$  and  $W_{Q,2}$ ?

**Exercise 2.5.3.** If  $S \sim U[0, 7]$  and  $X \sim U[0, 10]$ , where  $U[I]$  stands for the uniform distribution concentrated on the interval  $I$ , compute  $P\{S - X \leq x\}$ .

## 2 Single-Station Queueing Systems

**Exercise 2.5.4.** What are the meanings of  $A_{A(t)}$  and  $A(A_n)$ ?

**Exercise 2.5.5.** Would it make sense to define  $A(t) = \min\{k : A_k \geq t\}$ ?

**Exercise 2.5.6.** Define  $L_Q(t)$  as the number of job in queue, and  $L_s(t)$  as the number of jobs in service. Likewise, let  $D_Q(t)$  be the number of jobs that departed from the queue up to time  $t$ .

1. Why don't we need separate notation for  $D_s(t)$ , the number of jobs that departed from the server?
2. Is  $D_Q(t) \leq D(t)$  or  $D_Q(t) \geq D(t)$ ?
3. Why is  $L(t) = L_Q(t) + L_s(t)$ ?
4. Express  $L_Q(t)$  and  $L_s(t)$  in terms of  $A(t)$ ,  $D_Q(t)$  and  $D(t)$ .
5. Consider a multi-server queue with  $m$  servers. Suppose that at some  $t$  it happens that  $D_Q(t) - D_s(t) < m$  even though  $A(t) - D_s(t) > m$ . How can this occur?

**Exercise 2.5.7.** Show that  $L(t) = A(t) - D(t)$  implies that

$$L(t) = \sum_{k=1}^{\infty} \mathbb{1}_{A_k \leq t < D_k}.$$

Show also that if  $L(A_k) > 0$ , i.e., the system contains at least one job at the time of the  $k$ th arrival, then  $A_k \leq D_{k-1}$ , i.e., job  $k$  arrives before job  $k-1$  departs.

**Exercise 2.5.8.** Can you derive the following (algorithmic efficient) procedure to compute the number of jobs in the system as seen by arrivals,

$$L_k = L_{k-1} + 1 - \sum_{i=k-1-L_{k-1}}^{k-1} \mathbb{1}_{D_i < A_k}?$$

Why do we take  $i = k-1-L_{k-1}$  in the sum, and not  $i = k-2-L_{k-1}$ ?

**Exercise 2.5.9.** Provide a specification of the virtual waiting time process  $\{V(t)\}$  for all  $t$ .

**Exercise 2.5.10.** Another set of recursions to compute the arrival and departure times for a given set of inter-arrival and service times is the following:

$$\begin{aligned} A_k &= A_{k-1} + X_k, \\ D_k &= \max\{A_k, D_{k-1}\} + S_k. \end{aligned} \tag{2.21}$$

Now the computation of the waiting times is trivial:  $W_k = D_k - A_k$ .

1. Why do the recursions Eq. (2.21) work?
2. (Very difficult) Extend the above recursions to a situation in which one queue is served by two servers.
3. (Impossible?) Extend the above recursions to a situation in which one queue is served by  $m > 2$  servers.

## 2.5 Construction of the G/G/1 Queueing Process in Continuous Time

**Exercise 2.5.11.** Implement the above recursions in excel or some other computer program such as R, python, or julia, and check the results of the previous exercise.

**Exercise 2.5.12.** (Multiple queues, for the interested) Suppose one server serves two queues, such that jobs in queue A are served with priority over jobs in queue B. Assume that service is not-preemptive. Can you develop a similar set of recursions for each of the queues, or, otherwise, an algorithm?

**Exercise 2.5.13.** (LIFO queue) Can you develop a set of recursions or algorithm for a LIFO (Last-In-First-Out) queue?

### Hints

**Hint 2.5.2:** Use that  $W_{Q,1} = [W_{Q,0} + S_1 - X_1]^+$  and likewise for  $W_{Q,2}$ .

This first year probability. It should be elementary for you. Ensure you can do it.

**Hint 2.5.3:** This appears to be trivial... , but I didn't find it easy... In fact, I found it a major headache, hence it is good to try yourself before looking at the answer. Check your probability book if you don't know what to do.

**Hint 2.5.5:** Compare this to the definition in (2.14).

**Hint 2.5.7:** Use Boolean algebra. Write, for notational ease,  $A = \mathbb{1}_{A_k \leq t}$  and  $\bar{A} = 1 - A = \mathbb{1}_{A_k > t}$ , and define something similar for  $D$ . Then show that  $A - D = A\bar{D} - \bar{A}D$ , and show that  $\bar{A}D = 0$ . Finally sum over  $k$ .

**Hint 2.5.9:** Make a plot of the function  $A_{A(t)} - t$ . What is the meaning of  $V(A_{A(t)})$ ? What is  $V(A_{A(t)} + A_{A(t)} - t)$ ?

### Solutions

**Solution 2.5.1:** The intent of this exercise is to make you familiar with the notation.

BTW, such simple test cases are also very useful to test computer code. The numbers in the exercise are one such simple case. You can check the results by hand; if the results of the simulator are different, there is a problem.

**Solution 2.5.2:** First find the distribution of  $Y_k := S_{k-1} - X_k$  so that we can write  $W_{Q,k} = [W_{Q,k-1} + Y_k]^+$ . Use independence of  $\{S_k\}$  and  $\{X_k\}$ :

$$P\{Y_k = -2\} = P\{S_{k-1} - X_k = -2\} = P\{S_{k-1} = 1, X_k = 3\} = P\{S_{k-1} = 1\}P\{X_k = 3\} = \frac{1}{4}.$$

Dropping the dependence on  $k$  for ease, we get

$$P\{Y = -2\} = P\{S - X = -2\} = P\{S = 1, X = 3\} = P\{S = 1\}P\{X = 3\} = \frac{1}{4}$$

$$P\{Y = -1\} = P\{S = 2\}P\{X = 3\} = \frac{1}{4}$$

$$P\{Y = 0\} = P\{S = 1\}P\{X = 1\} = \frac{1}{4}$$

$$P\{Y = 1\} = P\{S = 2\}P\{X = 1\} = \frac{1}{4}.$$

## 2 Single-Station Queueing Systems

With this

$$\begin{aligned} P\{W_{Q,1} = 1\} &= P\{W_{Q,0} + Y_0 = 1\} = P\{3 + Y_0 = 1\} = P\{Y_0 = -2\} = \frac{1}{4} \\ P\{W_{Q,1} = 2\} &= P\{3 + Y = 2\} = P\{Y = -1\} = \frac{1}{4} \\ P\{W_{Q,1} = 3\} &= P\{3 + Y = 3\} = P\{Y = 0\} = \frac{1}{4} \\ P\{W_{Q,1} = 4\} &= P\{3 + Y = 4\} = P\{Y = 1\} = \frac{1}{4}. \end{aligned}$$

And, then

$$\begin{aligned} P\{W_{Q,2} = 1\} &= P\{W_{Q,1} + Y = 1\} = \sum_{i=1}^4 P\{W_{Q,1} + Y = 1 \mid W_{Q,1} = i\} P\{W_{Q,1} = i\} \\ &= \sum_{i=1}^4 P\{i + Y = 1 \mid W_{Q,1} = i\} \frac{1}{4} = \sum_{i=1}^4 P\{Y = 1 - i \mid W_{Q,1} = i\} \frac{1}{4} \\ &= \frac{1}{4} \sum_{i=1}^4 P\{Y = 1 - i\} = \frac{1}{4} (P\{Y = 0\} + P\{Y = -1\} + P\{Y = -2\}) = \frac{3}{16}. \end{aligned}$$

Typing the solution becomes boring..., let's use the computer.

```
from lea import Lea

W = 3
S = Lea.fromVals(1, 2)
X = Lea.fromVals(1, 3)

# This is WQ1
W = Lea.fastMax(W + S - X, 0)
W

1 : 1/4
2 : 1/4
3 : 1/4
4 : 1/4

# This is WQ2
W = Lea.fastMax(W + S - X, 0)
W

0 : 3/16
1 : 3/16
2 : 4/16
3 : 3/16
4 : 2/16
5 : 1/16
```

Great! Our handiwork matches with the computer's results.

**Solution 2.5.3:** Let us write  $f(x)$  for the density of  $P\{S - X \leq x\}$ , i.e.,  $P\{S - X \leq x\} = \int_{-\infty}^x f(y) dy$ . (A formal point, why does  $P\{S - X \leq x\}$  actually have a density? You can skip this detail if you

## 2.5 Construction of the G/G/1 Queueing Process in Continuous Time

are not interested in mathematical details, nevertheless, it is interesting to think about.) With conditioning,

$$f(x) = \int_0^{10} P\{S - X = x | X = u\} P\{X \in du\}$$

This notation is, admittedly, quite subtle (and to get it right requires much more work than we can do in this course). Intuitively,  $P\{X \in du\}$  means the (infinitesimal) probability that  $X \in [u, u + du]$ . Since  $X$  is uniform on  $[0, 10]$ ,  $P\{X \in [s, t]\} = (t - s)/10$  for  $s, t \in [0, 10]$  and  $s < t$ . Now, taking this literally, let  $s = u$  and  $t = u + du$ , it follows that,

$$P\{X \in [u, u + du]\} = \frac{u + dt - u}{10} = \frac{du}{10}.$$

Using this in the above expression for  $f(x)$  gives

$$f(x) = \frac{1}{10} \int_0^{10} P\{S - X = x | X = u\} du.$$

Now use that  $S$  and  $X$  are independent so that

$$\begin{aligned} P\{S - X = x | X = u\} &= \frac{P\{S - X = x, X = u\}}{P\{X = u\}} = \frac{P\{S - u = x, X = u\}}{P\{X = u\}} \\ &= \frac{P\{S - u = x\} P\{X = u\}}{P\{X = u\}} = P\{S = u - x\}. \end{aligned}$$

Observe that we just use the formulas we are used to from conditioning on sets with positive probability. Recall,  $P\{A|B\} = P\{AB\}/P\{B\}$  only when  $P\{B\} > 0$ . But here  $P\{X = x\} = 0$  for all  $x$ , since  $x$  consists of a single point and  $X$  is uniformly distributed. Thus, we divide by 0! To get the mathematical details right, requires quite a bit more work than we can do in this course. Interestingly, in this case it works, as  $P\{X = u\}$  appears in the numerator and denominator, hence can be cancelled. In general, be careful when using such shortcuts; it can go wrong easily.

With this,

$$f(x) = \frac{1}{10} \int_0^{10} P\{S - u = x\} du = \frac{1}{10} \int_0^{10} P\{S = x + u\} du.$$

Next, interpret  $P\{S = y\}$  as the density of  $P\{S \leq y\}$  so that  $P\{S = y\} = \mathbb{1}_{y \in [0, 7]}/7$ . Therefore, while using that in the above integral that  $x$  is fixed,

$$P\{S = x + u\} = \frac{1}{7} \mathbb{1}_{-x \leq u \leq 7-x},$$

from which

$$\begin{aligned} f(x) &= \frac{1}{70} \int_0^{10} \mathbb{1}_{-x \leq u \leq 7-x} du \\ &= \frac{1}{70} \int_{-\infty}^{\infty} \mathbb{1}_{0 \leq u \leq 10} \mathbb{1}_{-x \leq u \leq 7-x} du \\ &= \frac{1}{70} \int_{-\infty}^{\infty} \mathbb{1}_{\max\{0, -x\} \leq u \leq \min\{10, 7-x\}} du. \end{aligned}$$

First we make some simple observations. If  $x > 7$  then  $\min\{10, 7 - x\} < 0$ . This is smaller than  $\max\{0, -x\}$ . Thus,  $f(x) = 0$  for  $x \geq 7$ . Likewise, when  $x \leq -10$ ,  $f(x) = 0$ . Also, when  $x \in [-3, 0]$  the

## 2 Single-Station Queueing Systems

max and min overlap. Thus, all in all,

$$f(x) = \begin{cases} 0, & \text{if } x \leq -10, \\ \frac{x+10}{70} & \text{if } x \in [-10, -3], \\ \frac{1}{10} & \text{if } x \in [-3, 0], \\ \frac{7-x}{70} & \text{if } x \in [0, 7], \\ 0, & \text{if } x > 7, \end{cases}$$

The above is the answer; but, given the amount of effort I had to put into getting it, I want to check it. For this I used Wolframalpha. You can skip the rest of the solution below if you are not interested in how to do this (I will not ask this at the exam). However, to expand your skill set, I advise you to take notice of it anyway. Besides that, Wolframalpha is a great site. Please check it out if you haven't done so up to now.

So, I went to Wolframalpha, and this is what I typed:

```
\int_{-\infty}^{\infty} \text{Boole}[0 \leq u \leq 10] \text{Boole}[-x \leq u \leq 7-x] du,
```

so, once you know  $\text{\LaTeX}$  you can use wolframalpha. Wolframalpha turned it to

```
integral_{(-infinity)^infinity} \text{Boole}[0 \leq u \leq 10] \text{Boole}[-x \leq u \leq 7-x] du
```

For your convenience, I also include the following code from Wolframalpha

```
Integrate[Boole[Max[0, -x] <= u <= Min[10, 7 - x]], {u, -Infinity, Infinity}]
```

I got the same results as Wolframalpha, but certainly not for free.

**Solution 2.5.4:**  $A(t)$  is the number of arrivals during  $[0, t]$ . Suppose that  $A(t) = n$ . This  $n$ th job arrived at time  $A_n$ . Thus,  $A_{A(t)}$  is the arrival time of the last job that arrived before or at time  $t$ . In a similar vein,  $A_n$  is the arrival time of the  $n$ th job. Thus, the number of arrivals up to time  $n$ , i.e.,  $A(A_n)$ , must be  $n$ .

**Solution 2.5.5:** Suppose  $A_3 = 10$  and  $A_4 = 20$ . Take  $t = 15$ . Then  $\min\{k : A_k \geq 15\} = 4$  since  $A_3 < t = 15 < A_4$ . On the other hand  $\max\{k : A_k \leq t\} = 3$ .

**Solution 2.5.6:** 1. Because  $D_s(t) = D(t)$ . Once customers leave the server, their service is completed, and they leave the queueing system.

2. All customers that left the system must have left the queue. Thus,  $D_Q(t) \geq D(t)$ .

3. Jobs in the system are in queue or in service.

4.  $L_Q(t) = A(t) - D_Q(t)$ .  $L_S(t) = D_Q(t) - D(t)$ . This is in line with the fact that  $L(t) = L_Q(t) + L_S(t) = A(t) - D(t)$ .

5. In that case, there are servers idling while there are still customers in queue. If such events occur, we say that the server is not work-conservative.

**Solution 2.5.7:** 1.

$$\begin{aligned} L(t) &= A(t) - D(t) \\ &= \sum_{k=1}^{\infty} \mathbb{1}_{A_k \leq t} - \sum_{k=1}^{\infty} \mathbb{1}_{D_k \leq t} \\ &= \sum_{k=1}^{\infty} [\mathbb{1}_{A_k \leq t} - \mathbb{1}_{D_k \leq t}]. \end{aligned}$$

## 2.5 Construction of the G/G/1 Queueing Process in Continuous Time

Write for the moment  $A = \mathbb{1}_{A_k \leq t}$  and  $\bar{A} = 1 - A = \mathbb{1}_{A_k > t}$ , and likewise for  $D$ . Now we can use Boolean algebra to see that  $\mathbb{1}_{A_k \leq t} - \mathbb{1}_{D_k \leq t} = A - D = A(D + \bar{D}) - D = AD + A\bar{D} - D = A\bar{D} - D(1 - A) = A\bar{D} - D\bar{A}$ . But  $D\bar{A} = 0$  since  $D\bar{A} = \mathbb{1}_{D_k \leq t} \mathbb{1}_{A_k > t} = \mathbb{1}_{D_k \leq t < A_k}$  which would mean that the arrival time  $A_k$  of the  $k$ th job would be larger than its departure time  $D_k$ . As  $A\bar{D} = \mathbb{1}_{A_k \leq t < D_k}$

$$\begin{aligned} L(t) &= \sum_{k=1}^{\infty} [\mathbb{1}_{A_k \leq t} - \mathbb{1}_{D_k \leq t}] \\ &= \sum_{k=1}^{\infty} \mathbb{1}_{A_k \leq t < D_k}. \end{aligned}$$

Boolean algebra is actually a really nice way to solve logical puzzles. If you are interested you can find some examples on my homepage.

2. In a sense, the claim is evident, for, if the system contains a job when job  $k$  arrives, it cannot be empty. But if it is not empty, then at least the last job that arrived before job  $k$ , i.e., job  $k-1$ , must still be in the system. That is,  $D_{k-1} \geq A_k$ . A more formal proof proceeds along the following lines. Using that  $A(A_k) = k$  and  $D(D_{k-1}) = k-1$ ,

$$\begin{aligned} L(A_k) > 0 &\Leftrightarrow A(A_k) - D(A_k) > 0 \Leftrightarrow k - D(A_k) > 0 \Leftrightarrow k > D(A_k) \\ &\Leftrightarrow k-1 \geq D(A_k) \Leftrightarrow D(D_{k-1}) \geq D(A_k) \Leftrightarrow D_{k-1} \geq A_k, \end{aligned}$$

where the last relation follows from the fact that  $D(t)$  is a counting process, hence monotone non-decreasing.

**Solution 2.5.8:** Let  $L_k = L(A_k-)$ , i.e., the number of jobs in the system as ‘seen by’ job  $k$ . It must be that  $L_k = k-1 - D(A_k)$ . To see this, assume first that no job has departed when job  $k$  arrives. Then job  $k$  must see  $k-1$  jobs in the system. In general, if at time  $A_k$  the number of departures is  $D(A_k)$ , then the above relation for  $L_k$  must hold. Applying this to job  $k-1$  we get that  $L_{k-1} = k-2 - D(A_{k-1})$ .

For the computation of  $L_k$  we do not have to take the departures before  $A_{k-1}$  into account as these have already been ‘incorporated in’  $L_{k-1}$ . Therefore,

$$L_k = L_{k-1} + 1 - \sum_{i=k-1-L_{k-1}}^{k-1} \mathbb{1}_{D_i < A_k}.$$

Suppose  $L_{k-1} = 0$ , i.e., job  $k-1$  finds an empty system at its arrival and  $D_{k-1} > A_k$ , i.e., job  $k-1$  is still in the system when job  $k$  arrives. In this case,  $L_k = 1$ , which checks with the formula. Also, if  $L_{k-1} = 0$  and  $D_{k-1} < A_k$  then  $L_k = 0$ . This also checks with the formula.

The reason to start at  $k-1-L_{k-1}$  is that the number in the system as seen by job  $k$  is  $k-1-D(A_k)$  (not  $k-2-D(A_k)$ ). Hence, the jobs with index from  $k-1-L_{k-1}, k-L_{k-1}, \dots, k-1$ , could have left the system between the arrival of job  $k-1$  and job  $k$ .

**Solution 2.5.9:** There is a funny way to do this. Recall from a previous exercise that if  $A(t) = n$ , then  $A_n$  is the arrival time of the  $n$ th job. Thus, the function  $A_{A(t)}$  provides us with arrival times as a function of  $t$ . When  $t = A_{A(t)}$ , i.e., when  $t$  is the arrival time of the  $A(t)$ th job, we set  $V(t) = V(A_{A(t)}) = W_{A(t)}$  i.e., the virtual waiting time at the arrival time  $t = A_{A(t)}$  is equal to the

## 2 Single-Station Queueing Systems

waiting time of the  $A(t)$ th job. Between arrival moments, the virtual waiting time decreases with slope 1, until it hits 0. Thus,

$$V(t) = [V(A_{A(t)}) + (A_{A(t)} - t)]^+ = [W_{A(t)} + (A_{A(t)} - t)]^+$$

The notation may be a bit confusing, but it is in fact very simple. Take some  $t$ , look back at the last arrival time before time  $t$ , which is written as  $A_{A(t)}$ . (In computer code these times are easy to find.) Then draw a line with slope  $-1$  from the waiting time that the last arrival saw.

- Solution 2.5.10:** 1. Of course, the service of job  $k$  cannot start before it arrives. Hence, it cannot leave before  $A_k + S_k$ . Therefore it must be that  $D_k \geq A_k + S_k$ . But the service of job  $k$  can also not start before the previous job, i.e. job  $k-1$ , left the server. Thus job  $k$  cannot start before  $D_{k-1}$ . To clarify it somewhat further, define  $S'_k$  as the earliest start of job  $k$ . Then it must be that  $S'_k = \max\{A_k, D_{k-1}\}$ —don't confuse the earliest start  $S'_k$  and the service time  $S_k$ —and  $D_k = S'_k + S_k$ .
2. I found this not easy, to say the least. . . The problem is that in a multi-server queueing systems, unlike for single-server queues, jobs can overtake each other: a small job that arrives after a very large job can still leave the system earlier. After trying for several hours, I obtained an inelegant method. A subsequent search on the web helped a lot. The solution below is a modification of N. Krivulin, 'Recursive equations based models of queueing systems'.

The recursions for the two-server system are this:

$$\begin{aligned} A_k &= A_{k-1} + X_k, \\ C_k &= \max\{A_k, D_{k-2}\} + S_k, \\ M_k &= \max\{M_{k-1}, C_k\}, \\ D_{k-1} &= \min\{M_{k-1}, C_k\}. \end{aligned}$$

Here,  $C_k$  is the completion time of job  $k$ , and  $\{D_k\}$  is a sorted list of departure times. Thus,  $D_k$  is the  $k$ th departure time; recall this is not necessarily equal to the completion time  $C_k$  of the  $k$ th job (as jobs may overtake each other). To understand the other equations, we reason like this. By construction,  $C_k > D_{k-m}$  (as  $S_k > 0$ ). Therefore, when we arrived at time  $C_k$ ,  $(k-m)$  jobs must have departed. Moreover, by construction,  $M_k$  tracks the latest completion time of all  $k$  jobs, hence,  $M_{k-m+1}$  is the latest completion time of the first  $k-m+1$  jobs. Therefore, if  $C_k > M_{k-1}$ , job  $k$  must leave later than the latest of the jobs in  $\{1, 2, \dots, k-1\}$ . Hence, the latest departure time of the jobs in  $\{1, 2, \dots, k-1\}$  jobs must be  $M_{k-1}$ . If however,  $C_k < M_{k-1}$ , then job  $k$  leaves earlier than the latest of the jobs in  $\{1, 2, \dots, k-1\}$ . As  $C_k > D_{k-2}$ , it must be that  $C_k > M_{k-2}$ , because  $D_{k-2}$  is latest departure of the jobs in  $\{1, 2, \dots, k-2\}$ , and this is also equal to  $M_{k-2}$ . As a consequence, if  $C_k < M_{k-1}$ , job  $k$  is also the first job that leaves after  $D_{k-2}$  (provided of course that  $C_{k+1} < C_k$ ). Thus, all in all  $D_{k-1} = \min\{M_{k-1}, C_k\}$ .

3. Similar reasoning for the  $G/G/m$  queue seems to lead to the following.

$$\begin{aligned} A_k &= A_{k-1} + X_k, \\ C_k &= \max\{A_k, D_{k-m}\} + S_k, \\ M_k &= \max\{M_{k-m+1}, C_k\}, \\ D_{k-m+1} &= \min\{M_{k-m+1}, C_k\}, \end{aligned}$$



but it is not correct. Can you find a counter example?

**Solution 2.5.11:** You can check the files on github in the progs directory.

- `mm1.py` is the python implementation
- `mm1.R` is the R implementation
- `mm1.jl` is the julia implementation

Take your pick, and start playing with it. These examples are meant to be simple to understand, not necessarily super efficient.

**Solution 2.5.12:** This question came up in class. I don't think this can be constructed as a straightforward recursion, and a search on the web lead to nothing. In case you can find a recursion, please let me know.

While a straightforward recursion may not exist, it is not really difficult to code this queueing discipline. The key idea is to put all jobs into one queue, but sort the elements in the queue in order of priority. Every time the server becomes empty, it checks the head of the queue. If the queue is empty, it wait until the next arrival occurs. Otherwise, it starts the service of the first job in line. When a new job arrives, then identify its priority first, and then put it at the end of the jobs of the same priority.

This type of sorting is known as lexicographic sorting, which is what we do when we build a dictionary. First we sort words in order of first letter, then the second letter, and so on. In case of sorting the queue, we first have to sort in order of priority, then, within the class of jobs with the same priority, sort in ascending order of arrival time.

In python the code is like this, where I include the FIFO case for comparison. As always, it is not obligatory to memorize the code.

```
class Fifo(Queue):
    def __init__(self):
        self.queue = SortedSet(key = lambda job: job.arrivalTime)

class Priority(Queue): # a priority queue
    def __init__(self, numServers = 1):
        self.queue = SortedSet(key = lambda job: job.p)
```

Interestingly, once we have a proper environment to carry out simulations, changing the queueing discipline is a one-liner!.

**Solution 2.5.13:** Again, I don't know a good set of recursions, but an algorithm is straightforward. Rather than sorting the jobs in queue in ascending order of arrival time, just sort them in descending order of arrival time.

```
class Lifo(Queue):
    def __init__(self):
        self.queue = SortedSet(key = lambda job: -job.arrivalTime)
```

Another interesting queueing rule is to sort in increasing job size;

```
class SPTF(Queue): # shortest processing time first
    def __init__(self):
        self.queue = SortedSet(key = lambda job: job.serviceTime)
```

Do you see how sort in descending order of job size (it just a matter of putting a minus sign at the right place)?