Sample Path Analysis and Simulation of Stochastic Systems

Nicky D. van Foreest

November 14, 2026

This work is licensed by the University of Groningen under a Creative Commons Attribution-ShareAlike 4.0 International License.

CONTENTS

1 INTRODUCTION 1
1.1 Motivation and Overview 1
1.2 Algorithmic Patterns 3
1.3 Exponential and Poisson Distribution 12
1.4 Fighting the Exponential Distribution is Futile 18
1.5 Probabilistic Arithmetic 25
2 CONSTRUCTION OF SIMPLE DISCRETE-TIME STOCHASTIC PROCESSES 37
2.1 Waiting for a Psychiatrist 37
2.2 Simulating the Psychiatrists Case 45
2.3 General Behavior of Queueing Systems 48
2.4 Queueing Processes as Regulated Random Walks 52
2.5 Control of Single-item Inventory Systems 53
3 CONSTRUCTION OF SIMPLE CONTINUOUS-TIME STOCHASTIC PROCESSES 60
3.1 Queueing Process in Continuous Time 60
3.2 Kendall's Notation 64
3.3 Simulating Waiting Times 65
4 FUNDAMENTAL TOOLS 72
4.1 Rate, Stability and Load 72
4.2 (Limits of) Empirical Performance Measures 75
4.3 Renewal Reward Theorem 76
4.4 Little's Law 79
4.5 Poisson Arrivals See Time Averages 80
4.6 Level Crossing and Balance Equations 84
4.7 Graphical Summaries 89
5 EXACT MODELS 92
5.1 The $M(n)/M(n)/1$ Queue 92
5.2 Applications of the $M(n)/M(n)/1$ queueing model 96
5.3 $M^B/M/1$ Queue and Rejection Policies 99
5.4 $M/G/1$ Queue: Expected Waiting Time 104
5.5 $M/G/1$ Queue Length Distribution 106
5.6 N-policies for the $M/G/1$ Queue 108
5.7 Analytic Results for the Base-Stock Policy 111
5.8 Analytic Results for the (Q,r) Policy 115
5.9 An algorithm for the (s,S) and (T,S) policies 118
5.10 Hitting problems with applications in single-item inventory control 124
6 APPROXIMATE MODELS 131
6.1 The $G/G/c$ Queue 131
6.2 Server Setups 135
6.3 Server Adjustments 137
6.4 Server Failures 139

7 QUEUEING NETWORKS 142
7.1 $G/G/c$ Queues in Tandem 142
7.2 Deterministic Queueing Networks 144
7.3 Open Single-Class Product-Form Networks 147
7.4 On $\lambda = \gamma + \lambda P$ 150
7.5 Mean Value Analysis of Closed Queueing Networks (UNDER CONSTRUCTION) 151
HINTS 154
SOLUTIONS 159
BIBLIOGRAPHY 203
NOTATION 205
7.6 Arrivals and event processes 205
7.7 Service and capacity 205
7.8 State and performance 206
7.9 Variability 206

1 INTRODUCTION

1.1 MOTIVATION AND OVERVIEW

Queueing and inventory systems are ubiquitous, and their analysis, performance evaluation, control, and optimization are major topics in Operations Research. In this book, we learn to construct models of these systems, evaluate their performance, and compare ways to improve it. Let us illustrate this with two examples.

Fast food restaurants deal with many difficult queueing and inventory situations. For instance, they prepackage hamburgers and put them on a shelf[1] so that customers need not wait for their orders. The lifetime of prepackaged hamburgers is on the order of a few minutes, so when customers do not buy a hamburger in time, some must be scrapped. The question is how many hamburgers to stock so that both the probability of scrapping and that of lost sales[2] are small. Second, fast food restaurants keep an inventory of baked patties not yet turned into hamburgers.[3] Baked patties have a longer lifetime than prepared hamburgers, but since preparing a hamburger from a baked patty takes time, customers may still have to wait. In this case too, the inventory level of these patties needs to be controlled. This does not conclude the matter: the rate of customer demand changes over time, so inventory levels for the different types of hamburgers and patties should vary with the time of day.[4]

As a second example, hospitals (call centers, courts, etc.) have a certain capacity to serve customers. The performance of hospitals is, in part, measured by the total number of jobs served per year and the *sojourn-time* distribution, that is, the distribution of the time a job spends in the system. Here the problem is to organize the system so that the average sojourn time is short without wasting capacity.

In all these examples, the performance of the stochastic system must be monitored and controlled. Typically, the following key performance measures (KPIs) are relevant for queueing systems.[5]

1. The fraction of time $p(n)$ that the system contains n customers. In particular, $p(0)$ is the fraction of time the system is empty, hence measures the fraction of time all servers are idle.
2. The fraction of customers $\pi(n)$ that 'see upon arrival' n customers in the system. This measure relates to customer perception of queue length, and hence to waiting time.
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.

With these examples in mind, let us give an overview of this book.

Chapter 2 shows how to construct rules that govern the dynamic behavior of many different queueing systems and inventory systems

[1] *This is called make-to-stock production.*

[2] *Customers who do not buy because they have to wait.*

[3] *A hamburger is a bun plus a baked patty.*

[4] *Inventory and queueing systems are tightly related by changing perspective. Hamburgers in stock can be interpreted as* customers in queue *waiting for service, where the 'service time' as perceived by a hamburger is the time between the arrivals of two successive customers who buy a hamburger.*

[5] *Inventory systems have similar performance measures, such as the average time items spend on the shelf, the fraction of customers served from stock, and the fraction of lost sales.*

in discrete time. This has several purposes. First, these rules can often be easily implemented in computer code, which we can then use to simulate and observe system behavior over time.[6] The resulting *sample paths*[7] can then be analyzed, visually or statistically. Second, we will use sample-path arguments to develop the theoretical results in later parts of the book.

In Chapter 3 we describe how to construct sample paths of continuous-time stochastic systems.

Once the scope of queueing and inventory theory is clear, the stage is set for a more mathematical treatment of such systems. Despite the power of simulation, structural understanding of stochastic systems is difficult to obtain. Mathematical models, whether exact or approximate, help us reason about and improve such systems.

In Chapter 4 we develop some key results needed for the mathematical analysis of stochastic systems[8] . We use sample paths[9] of stochastic processes to estimate key performance measures, assuming that these paths capture typical stochastic behavior.

In Chapter 5 we use these tools to build exact models for single-station queueing systems. We focus on developing an intuitive understanding of the analytical tools.[10]

In Chapter 6 we use approximations and results from probability theory to study how system parameters, such as service speed, batching rules, and outages, affect performance.

In Chapter 7 we study queueing networks, first in a deterministic and then in a stochastic setting.

IT REMAINS TO discuss the rest of this first chapter. Section 1.2 develops patterns for algorithmic thinking and organizing computational ideas. Section 1.3 introduces the Exponential distribution and relates it to the Poisson distribution. These two are arguably the most important distributions for modeling dynamic stochastic systems. In Section 1.4 we use simulation and some mathematics to explain why these distributions are particularly useful for modeling arrival processes in queueing systems and demand processes for inventory systems. The amount of computational work involved is considerable; in Section 1.5 we discuss fundamental and important algorithms that handle the numerical aspects.

WHILE THE MAIN text contains many examples and derivations, a considerable number of examples are left to the exercises. Also, some exercises check consistency between results derived for different models, thereby revealing important connections between different parts of the text.[11] The exercises are not intended to be easy. Hints and solutions for the exercises are available at the end of the book.

IN THE FINAL STAGES of improving the document I used Claude and Codex to find typos, grammatical and mathematical errors, and to check notational consistency.

[6] *Simulation is generally the best way to analyze practical stochastic systems, as realistic systems seldom yield to mathematics.*

[7] *That is, realizations of these stochastic processes.*

[8] *Queueing systems are one example.*

[9] *In fact, the same sample paths as those obtained through simulation.*

[10] *For most proofs and more extensive results, we refer to the bibliography at the end of the book.*

[11] *Note that while such checks are conceptually simple, the algebra can be tedious.*

1.2 ALGORITHMIC PATTERNS

An important aim of this course is that you develop algorithmic thinking. Here we develop useful and elegant patterns that help us organize computational ideas.[1] You should learn to describe a method as a sequence of precise steps, trace what these steps do, and explain why they work. Here are a few important examples of algorithms we use throughout the course. Study them in detail; they are as important as the mathematical models we develop.

Our pseudocode focuses on the key algorithmic steps. We omit error checking and other implementation details.

It is striking how far three foundational concepts can take us: *dictionaries*, *λ functions*, and *closures*.[2] Mastering these ideas takes time, but with practice they become very useful tools. By combining the same building blocks we can count observations, normalize weights, filter lists, and avoid repeated work in recursive computations. As you read the examples below, look for these recurring patterns: recognizing how the pieces fit together is part of the pleasure of algorithmic thinking.

DICTIONARIES FORM OUR main building block.[3] A dictionary is a mutable function D with a finite domain written as $\text{KEYS}(D)$.[4] If $k \in \text{KEYS}(D)$, $D(k)$ returns its stored value for k.[5] The assignment $D(k) \leftarrow v$ inserts or replaces the value at k in D. The function $\text{VALUES}(D)$ returns the list[6] $[D(k) : k \in \text{KEYS}(D)]$. We assume that $\text{KEYS}(D)$ returns the keys in the order in which they were inserted, and that $\text{VALUES}(D)$ returns their values in the corresponding order.

Here is an example.

```
1: D ← an empty dictionary.
2: D("Lion")                          ▷ Error: the key is absent.
3: D("Lion") ← 15
4: D("Lion")                          ▷ Returns 15.
5: KEYS(D)                            ▷ Returns ["Lion"].
6: VALUES(D)                          ▷ Returns [15].
```

LAMBDA FUNCTIONS are so-called *anonymous* functions.[7] They are useful when a function is too short and simple to need a name. We write

$$\lambda : x \mapsto h(x)$$

for the function that takes x as its argument and returns $h(x)$. For example,

```
1: SQUARE ← λ : x ↦ x · x
2: CUBE ← λ : x ↦ SQUARE(x) · x
```

THE FUNCTION ACCUMULATE applies a function to a running result and each item of a list.[8] By combining this one algorithm with λ-functions, we can define many common functions as one-liners:[9] We call such

[1] *It is also intellectually satisfying to see how to build up concepts from the ground.*

[2] *I use these ideas all the time when working with large language models. They help me choose algorithms for numerical problems and guide the LLMs in organizing code. The LLMs handle all of the programming-language syntax; I remain the architect of the computation.*

[3] *We assume that dictionaries are available to us from the start.*

[4] *In R, the keys are called 'names'.*

[5] *The call $D(k)$ when k is absent results in an error.*

[6] *In contrast to a set, a list allows for duplicates, which we want here.*

[7] *In R, one writes* `\(x) h(x)`.

[8] *In R, the corresponding call is* `Reduce(f, L, init=start)`.

[9] *Recall* $\min \emptyset = +\infty$ *and* $\max \emptyset = -\infty$.

```
SUM ← λ: L ↦ ACCUMULATE(L, λ: (x, y) ↦ x + y, 0)
PROD ← λ: L ↦ ACCUMULATE(L, λ: (x, y) ↦ x · y, 1)
MINIMUM ← λ: L ↦ ACCUMULATE(L, λ: (x, y) ↦ min{x, y}, +∞)
MAXIMUM ← λ: L ↦ ACCUMULATE(L, λ: (x, y) ↦ max{x, y}, −∞)
LENGTH ← λ: L ↦ ACCUMULATE(L, λ: (n, x) ↦ n + 1, 0)
```

```
SUM([1, 2, 3])      ▷ Returns 6.
PROD([1, 2, 3, 4])  ▷ Returns 24.
```

functions as follows.

Here is the implementation.

Algorithm 1.2.1 Accumulate the items of a list.

```
Input: a list L, a function f, and a start value start.
Output: the accumulated value.
function ACCUMULATE(L, f, start)
    m ← start
    for x ∈ L do
        m ← f(m, x)
    return m
```

APPLYING A FUNCTION to each entry of a list is another useful pattern. The function APPLY[10] returns the results in the same order as in the given list. We write APPEND(R, x) for the list obtained by adding x at the end of R.

[10] *In R, the corresponding call is* `lapply(L, f)`.

Algorithm 1.2.2 Apply a function to each entry of a list.

```
Input: a list L and a function f.
Output: the list [f(x) : x ∈ L].
function APPLY(L, f)
    y ← []                          ▷ y is now an empty list.
    for x ∈ L do
        y ← APPEND(y, f(x))
    return y
```

For example, take service times $s = [1, 2, 4, 5]$ and a target $\tau = 3$. The fraction above the target is computed as follows.

```
s, τ ← [1, 2, 4, 5], 3
B ← APPLY(s, λ: x ↦ 𝟙{x > τ})          ▷ B = [0, 0, 1, 1].
fractionLong ← SUM(B) / LENGTH(B)      ▷ fractionLong = 1/2.
```

With SUM and APPLY, it is straightforward to build a function NORMALIZE that turns a list of non-negative weights into a list of probabilities.

Algorithm 1.2.3 Normalize non-negative weights into probabilities.

```
Input: a list w of non-negative weights with positive sum.
Output: a list p of probabilities.
function NORMALIZE(w)
    total ← SUM(w)
    return APPLY(w, λ: x ↦ x/total)
```

FILTERING A LIST keeps the entries that pass a test, in their original order.

Algorithm 1.2.4 Keep the entries of a list that pass a test.

```
Input: a list L and a test h.
Output: the list [x ∈ L : h(x)].
function FILTER(L, h)
    y ← []
    for x ∈ L do
        if h(x) then
            y ← APPEND(y, x)
    return y
```

For the service times above, FILTER selects the service times of jobs that exceed the threshold:

```
FILTER([1, 2, 4, 5], λ: x ↦ x > 3)        ▷ Returns [4, 5].
```

CLOSURES FORM THE next important idea. An example is best to demonstrate this.

Algorithm 1.2.5 Construct a power function with a fixed exponent.

```
Input: an exponent e.
Output: a function that maps b to b^e.
function POWER(e)
    return λ: b ↦ b^e
```

Observe that POWER returns a λ function. This function captures the variable e from its surrounding scope.[11] As a result, the returned function remembers the value of e. In the following example, POWER (2) returns a function with exponent 2 fixed. We name it SQUARE, then call it on a number:

[11] *The* scope *of a variable is the part of a program where it can be used. Here, the λ function is created where e is available.*

```
POWER(1)(9)                    ▷ Returns 9.
SQUARE ← POWER(2)              ▷ SQUARE = λ: b ↦ b^2.
SQUARE(5)                      ▷ Returns 25.
CUBE ← POWER(3)                ▷ CUBE = λ: b ↦ b^3.
CUBE(4)                        ▷ Returns 64.
```

In general terms, an outer function can return an inner function that remembers values from the call to the outer function. The returned inner function is called a *closure*.[12] By calling the outer function and assigning

[12] *It can take some time to get your head around a function that returns another function. Working through a few calls by hand helps.*

its output to a new name, we specialize the outer, general, function. The returned function has the same structure, but with some variables bound to concrete values. In this way, we obtain a new, more specific function from a general computational template.

DICTIONARIES WITH DEFAULT VALUES are a first application of closures. The function MAP in Alg. 1.2.6 uses *default* = 0 unless another default is supplied. Each call creates a private empty dictionary *store* and returns a function F that retains access to it. The same dictionary is available as F.*store*, and its default value as F.*default*. We write F(k) ← v as shorthand for F.*store*(k) ← v.

Algorithm 1.2.6 Construct a map with a configurable default value.

```
1:  Input: a default value default.
2:  Output: a function F.
3:  function MAP(default = 0)
4:      store ← an empty dictionary
5:      function F(k)
6:          if k ∉ KEYS(store) then
7:              store(k) ← default
8:          return store(k)
9:      F.store ← store                ▷ Expose stored entries for later access.
10:     F.default ← default            ▷ Expose the default for later access.
11:     return F
```

The functions KEYS(F) and VALUES(F) return the keys and values of the map F.[13]

[13] *We assumed at the start of this section that dictionaries provide* KEYS *and* VALUES. *Since* F.*store is a dictionary, we can use those operations here.*

Algorithm 1.2.7 Return the keys of a map.

```
1:  Input: a map function F.
2:  Output: its keys.
3:  function KEYS(F)
4:      return KEYS(F.store)
```

Algorithm 1.2.8 Return the values of a map.

```
1:  Input: a map function F.
2:  Output: its values.
3:  function VALUES(F)
4:      return VALUES(F.store)
```

Here is an example to see how to use a map.

```
1:  G ← MAP()                          ▷ G.default = 0.
2:  G(3)                               ▷ Returns 0; stores 0 under key 3.
3:  G(3) ← G(3) + 8                    ▷ G(3) = 8.
4:  G(3)                               ▷ Returns 8.
5:  KEYS(G)                            ▷ Returns [3].
6:  VALUES(G)                          ▷ Returns [8].
```

With a map, it is easy to build a COUNT function that counts the occurrences of each observed value in a list. When a value k is encountered for the first time, *counter*(k) therefore returns zero, and the update sets it to one. Every later occurrence increases the stored value by one. Thus, after all observations, *counter*(k) is the number of times k occurred.

In this and later algorithms, we use the convenient abbreviation $v \xleftarrow{+} 1$ for $v \leftarrow v + 1$.

Algorithm 1.2.9 Count observations in each state.

```
1: Input: a list of observations L.
2: Output: a map of counts counter.
3: function COUNT(L)
4:     counter ← MAP()
5:     for k ∈ L do
6:         counter(k) +← 1
7:     return counter
```

The function MAKEMAP builds a map from two aligned lists. Its default value is zero unless another value is supplied.

Algorithm 1.2.10 Construct a map from lists of keys and values.

```
1: Input: distinct keys keys = [k_1, ..., k_n], values values = [v_1, ..., v_n],
   and a default value.
2: Output: a map M with M(k_i) = v_i and the given default.
3: function MAKEMAP(keys, values, default = 0)
4:     M ← MAP(default)
5:     for i = 1, ..., n do
6:         M(k_i) ← v_i
7:     return M
```

For a map we can filter its stored values by first filtering its list of keys. The result retains the original default value.

Algorithm 1.2.11 Keep the stored entries of a map whose values pass a test.

```
1: Input: a map M and a test h on its values.
2: Output: a map with the same default value and retained entries.
3: function FILTERVALUES(M, h)
4:     S ← FILTER(KEYS(M), λ: k ↦ h(M(k)))
5:     V ← APPLY(S, M)
6:     return MAKEMAP(keys = S, values = V, default = M.default)
```

Notice how little new machinery this algorithm needs. FILTER selects the keys, APPLY retrieves their values, and MAKEMAP puts the pairs into a new map. Once these small operations are understood, we can combine them to express a more complicated idea in just a few steps.

CACHING is a technique to speed up recursive computations tremendously.[14] A recursive function[15] may reach the same state more than

[14] *Caching is also called* memoization.

[15] *We will use recursion on many occasions in this book.*

once. By storing the result after its first computation, we can reuse it on later visits, thereby avoiding the cost of computing it again.

The Fibonacci numbers give a small example of the benefits of caching. They satisfy $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$. Direct recursion computes the same values repeatedly. For example, FIBONACCI(5) computes F_3 twice and F_2 three times.[16]

[16] *Without caching, the number of computations increases exponentially fast as a function of n.*

Algorithm 1.2.12 Compute a Fibonacci number by direct recursion.

```
1: Input: a non-negative integer n.
2: Output: the Fibonacci number F_n.
3: function FIBONACCI(n)
4:     if n ≤ 1 then
5:         return n
6:     return FIBONACCI(n − 1) + FIBONACCI(n − 2)
```

With a cache, we can replace recomputations with lookups, giving an algorithm for the Fibonacci numbers that runs in linear time. In general, a cache is a dictionary with states as keys and their computed results as values. CACHE packages this entire caching pattern into one function. It receives a function and returns a cached version of that function.[17]

[17] *We call this a higher-order function.*

Algorithm 1.2.13 Construct a cached version of a function.

```
1: Input: a function f.
2: Output: a cached version g of f.
3: function CACHE(f)
4:     cache ← an empty dictionary
5:     function G(k)
6:         if k ∉ KEYS(cache) then
7:             cache(k) ← f(k)
8:         return cache(k)
9:     return G
```

Thus, after defining the direct recursive function in Alg. 1.2.12, we obtain its cached version with

$$\text{FIBONACCI} \leftarrow \text{CACHE}(\text{FIBONACCI}).$$

Recursive calls then use the cached binding of FIBONACCI.

To get some feeling for the difference in computation time, suppose a computer can do 10^{10} operations a second. The uncached recursive computation of the 100th Fibonacci number requires on the order of ϕ^{100} operations, where $\phi = (1+\sqrt{5})/2$, giving roughly 8×10^{10} seconds, or 9×10^5 days, under this simplified operation-counting model. Using CACHE(FIBONACCI) instead results in $100/10^10 = 10^{-8}$ seconds.

SORTING is another useful example of an algorithmic pattern. Alg. 1.2.14 implements *quicksort*, a recursive[18] algorithm to carry out the sorting.[19] The algorithm uses the function[20] CHOOSEPIVOT to select an element uniformly from its non-empty input list. Here, + denotes concatenation of lists.

[18] *And very efficient.*

[19] *Think deeply about this algorithm. It is very useful.*

[20] *We assume it is given.*

Algorithm 1.2.14 Sort a list $\mathcal{S}$ in increasing order.

```
1: Input: a list S.
2: Output: the elements of S in increasing order.
3: function QSort(S)
4:     if |S| ≤ 1 then
5:         return S
6:     p ← ChoosePivot(S)
7:     L ← [x ∈ S : x < p]
8:     M ← [x ∈ S : x = p]
9:     R ← [x ∈ S : x > p]
10:    return QSort(L) + M + QSort(R)
```

NOW IT IS TIME to practice. For each *True–False* question, decide whether the statement is true or false.

TF 1.2.1. Consider the following function, which returns a closure.

```
1: function SUBTRACT(b)
2:     function F(a)
3:         return a − b
4:     return F
```

Let G ← SUBTRACT(3). Claim: G(5) = −2.

TF 1.2.2. Consider the following functions.

```
1: function F(w)
2:     return 2w
3: function G(x)
4:     return λ : x ↦ x · F(x)
```

Claim: G(2) returns 8.

TF 1.2.3. Consider the following function, which returns a test for whether a waiting time exceeds a threshold.

```
1: function Exceeds(b)
2:     function F(w)
3:         return 𝟙{w > b}
4:     return F
```

Let LATE ← Exceeds(5). Claim: for waiting times 4, 5, 9, the sum LATE(4) + LATE(5) + LATE(9) equals 2.

TF 1.2.4. Claim: At ???, the algorithm should have d.

```
1: function MAP(d)
2:     store ← an empty dictionary
3:     function F(k)
4:         if k ∉ keys(store) then
5:             store(k) ← ???
6:         return store(k)
7:     F.store ← store
8:     F.default ← d
9:     return F
```

TF 1.2.5. Claim: At ???, the algorithm should have $w_k / total$.

```
1: function NORMALIZE([w_1, ..., w_n])
2:     total ← SUM([w_1, ..., w_n])
3:     for k = 1, ..., n do
4:         p_k ← ???
5:     return [p_1, ..., p_n]
```

TF 1.2.6. Apply the algorithm with threshold $b = 5$ to the times $w = (4,9,2,11,4)$. Claim: The algorithm returns $\bar{w} = 6$ and $q = 2/5$.

```
1: T ← 0, C ← 0
2: for k = 1, ..., n do
3:     T +← w_k
4:     if w_k > b then
5:         C +← 1
6: w̄ ← T/n, q ← C/n
7: return q
```

TF 1.2.7. Consider the following algorithm, which assumes that at least one observation is supplied.

```
1: Input: observations [x_1, ..., x_n], supplied one at a time.
2: Output: average x̄.
3: N ← 0, T ← 0
4: for k = 2, ..., n do
5:     N +← 1
6:     T +← x_k
7: x̄ ← T/N
8: return x̄
```

Claim: The algorithm computes the average of the observations $[x_1, \ldots, x_n]$.

TF 1.2.8. Claim: The algorithm returns the largest observed waiting time. It assumes that at least one waiting time is observed.

```
1: Input: waiting times [w_1, ..., w_n], supplied one at a time.
2: Output: maximum waiting time M.
3: M ← w_1
4: for k = 2, ..., n do
5:     M ← max{M, w_k}
6: return M
```

TF 1.2.9. There are N arrival streams, each containing T arrivals. Let A_i^k be the time of the ith arrival in stream k. Claim: the algorithm below returns the inter-arrival times of the merged stream in chronological order.

```
1: Input: arrival times {A_i^k : 1 ≤ k ≤ N, 1 ≤ i ≤ T}.
2: Output: inter-arrival times (X_2, ..., X_{N·T}).
3: function MergeArrivals({A_i^k})
4:     (A_1, ..., A_{N·T}) ← QSort([A_i^k : 1 ≤ k ≤ N, 1 ≤ i ≤ T])
5:     for 2 ≤ j ≤ N · T do
6:         X_j ← A_j − A_{j−1}
7:     return QSort([X_2, ..., X_{N·T}])
```

TF 1.2.10.

```
1: Input: a list S.
2: Output: the elements of S in increasing order.
3: function QSort(S)
4:     if |S| ≤ 1 then
5:         return S
6:     p ← ChoosePivot(S)
7:     L ← ???
8:     M ← [x ∈ S : x = p]
9:     R ← [x ∈ S : x > p]
10:    return QSort(L) + M + QSort(R)
```

Claim: At ???, the algorithm should have $[x \in S : x \leq p]$.

TF 1.2.11.

```
1: Input: a list S.
2: Output: the elements of S in increasing order.
3: function QSort(S)
4:     if |S| ≤ 1 then
5:         return S
6:     p ← ChoosePivot(S)
7:     L ← [x ∈ S : x < p]
8:     M ← [x ∈ S : x = p]
9:     R ← [x ∈ S : x > p]
10:    return ???
```

Claim: At ???, the algorithm should have QSort($\mathcal{L}$) + $\mathcal{M}$ + QSort($\mathcal{R}$).

The next two questions provide some further practice with the keys and values of maps.

TF 1.2.12. Consider the following instructions.

```
1: p ← MAP(default = 0)
2: p(0) ← 1/4
3: p(1) ← 1/2
4: p(2) ← 1/4
```

Claim: KEYS(p) = $[0,1,2]$ and VALUES(p) = $[1/4, 1/2, 1/4]$.

TF 1.2.13. Consider the following instructions.

```
1: M ← MAP()
2: M(2) ← 5
3: M(5) +← 2
4: M(2) +← 1
```

Claim: KEYS(M) = $[2,5,6]$ and VALUES(M) = $[5,2,6]$.

Exercises require pen and paper.

Ex 1.2.14. For a non-negative integer-valued rv X with $\mathsf{E}\left[X^2\right] < \infty$, write $G(k) = \mathsf{P}\{X > k\}$ and use indicator functions to prove:

1. $\mathsf{E}[X] = \sum_{k=0}^{\infty} G(k)$,
2. $\sum_{i=0}^{\infty} iG(i) = \mathsf{E}\left[X^2\right]/2 - \mathsf{E}[X]/2$.



1.3 EXPONENTIAL AND POISSON DISTRIBUTION

In this section, we make a model for the arrival process that is the most useful for queueing and inventory systems.[1] In particular, we show how the exponential and gamma distributions relate to the Poisson distribution, and we use the memoryless property of job inter-arrival times as our point of departure.

[1] *In queueing systems, an arrival is often called* customer *or* job, *while in inventory systems it is called* demand.

WE START WITH two examples to illustrate why it is reasonable to model the time between two events as memoryless. For instance, assume an hour ago a patient with a broken arm arrived at the emergency department of a hospital. What can we say about the time the next patient with a broken arm will arrive? In other words, does the information that an hour ago a broken arm arrived help to make a better prediction of the next? Similarly, suppose that the average time between the sales of some shirt in a clothing store is 5 minutes. The shop owner informs us that no shirt has been sold in the past 7 minutes. Surely, this knowledge does not imply that the next shirt will necessarily be sold within the next minute.

Let us formalize this. Let X denote the time from the start of observation until the next arrival. If a person does not tell us anything about the past, we write $\mathsf{P}\{X > t\}$ for the probability that no event occurs before time t. However, if the person tells us at time s there was no arrival during $[0,s]$, we write $\mathsf{P}\{X > s+t \mid X > s\}$ for the probability that there will

not be an arrival for an additional amount of time t. If we believe that the information $\{X > s\}$ is immaterial to predicting the lifetime of X after s, then necessarily $\mathsf{P}\{X > s+t \,|\, X > s\} = \mathsf{P}\{X > t\}$.

Definition 1.3.1. A rv[2] X whose cdf[3] satisfies

$$\mathsf{P}\{X > s+t \,|\, X > s\} = \mathsf{P}\{X > t\}, \quad s, t \geq 0, \tag{1.3.1}$$

is said to be *memoryless.*[4]

[2] *rv(s) := random variable(s).*

[3] *cdf := cumulative distribution function.*

[4] *Not all rvs related to lifetime are memoryless. As a counterexample, take X as the lifespan of an arbitrary person born in 1820, and $s = 100$ and $t = 99$ years. Then $\mathsf{P}\{X > 99\}$ was small, but not zero, but $\mathsf{P}\{X > 199 \,|\, X > 100\} = 0$.*

—

The next theorem is fundamental.

Theorem 1.3.2. $X \sim \text{Exp}(\lambda)$ iff X is memoryless.

—

Proof. Observing that $\mathsf{P}\{X \leq t\} = 1 - e^{-\lambda t} \implies \mathsf{P}\{X > t\} = e^{-\lambda t}$,

$$\begin{aligned} \mathsf{P}\{X > s+t \,|\, X > s\} &\overset{1}{=} \frac{\mathsf{P}\{X > s+t\}}{\mathsf{P}\{X > s\}} = \frac{e^{-\lambda(t+s)}}{e^{-\lambda s}} \\ &= e^{-\lambda t} = \mathsf{P}\{X > t\}, \end{aligned}$$

where Step 1 follows from the standard formula for conditional probability and that $\{X > t+s\} \subset \{X > s\}$. The other direction is somewhat more difficult; we skip it. [5] □

[5] *For a proof, see Yushkevich and Dynkin [1969, Appendix 3].*

Note that for $X \sim \text{Exp}(\lambda)$, the following properties hold:[6]

$$\mathsf{E}[X] = \lambda^{-1}, \qquad \mathsf{V}[X] = \lambda^{-2}.$$

[6] *See [1.3.6] for details.*

IF WE HAVE a sequence of inter-arrival times $\{X_i\}_{i=1}^{\infty}$, then we construct arrival times according to the recursive rule

$$A_k = A_{k-1} + X_k, \qquad A_0 = 0. \tag{1.3.2}$$

When inter-arrival times $\{X_k\}$ are iid[7] rvs and distributed as the common rv[8] $X \sim \text{Exp}(\lambda)$, it is clear that the density of the first arrival time is

$$f_{A_1}(t) = f_X(t) = \lambda e^{-\lambda t} = \lambda \frac{(\lambda t)^0}{0!} e^{-\lambda t},$$

where we add the last term to see how a pattern emerges. Using convolution, for the second arrival time,

$$f_{A_2}(t) = \int_0^t f_{A_1}(s) f_X(t-s)\,\mathrm{d}s = \int_0^t \lambda e^{-\lambda s} \lambda e^{-\lambda(t-s)}\,\mathrm{d}s = \lambda \frac{(\lambda t)^1}{1!} e^{-\lambda t}.$$

With induction it is straightforward to extend the pattern and prove that $A_k \sim \text{Gamma}(\lambda, k)$, i.e.,

$$f_{A_k}(t) = \int_0^t f_{A_{k-1}}(s) f_X(t-s)\,\mathrm{d}s = \lambda \frac{(\lambda t)^{k-1}}{(k-1)!} e^{-\lambda t}. \tag{1.3.3}$$

[7] *iid := independent and identically distributed.*

[8] *Henceforth, we write $X_k \overset{\text{iid}}{\sim} X$, or $X_k \overset{\text{iid}}{\sim} \text{Exp}(\lambda)$.*

WHEN THE ARRIVAL times [9] are gamma distributed, the pmf[10] of the

[9] *Arrival times are not the same as inter-arrival times.*

[10] *pmf := probability mass function.*

number $N(t)$ of jobs arriving during $[0,t]$ follows readily by observing that

$$\mathsf{P}\{N(t)=k\} = \mathsf{P}\{A_k \le t < A_{k+1}\} = \mathsf{P}\{A_k \le t\} - \mathsf{P}\{A_{k+1} \le t\}, \tag{1.3.4}$$

that is, to have precisely k arrivals during $[0,t]$, the arrival time A_k of job k must lie in $[0,t]$, but job $k+1$ must arrive after t. For $k \ge 1$, using the cdf of A_k and integration by parts,

$$\begin{aligned}\mathsf{P}\{A_{k+1} \le t\} &= \lambda \int_0^t \frac{(\lambda s)^k}{k!} e^{-\lambda s}\,\mathrm{d}s = \lambda \frac{(\lambda s)^k}{k!}\frac{e^{-\lambda s}}{-\lambda}\Big|_0^t + \lambda \int_0^t \frac{(\lambda s)^{k-1}}{(k-1)!} e^{-\lambda s}\,\mathrm{d}s \\ &= -\frac{(\lambda t)^k}{k!} e^{-\lambda t} + \mathsf{P}\{A_k \le t\}.\end{aligned}$$

Combining this with (1.3.4), we find

$$\mathsf{P}\{N(t)=k\} = \frac{(\lambda t)^k}{k!} e^{-\lambda t}, \tag{1.3.5}$$

that is, $N(t) \sim \mathrm{Pois}(\lambda t)$, i.e., *Poisson distributed* with parameter λt. For $k=0$, the same formula follows directly from $\mathsf{P}\{N(t)=0\} = \mathsf{P}\{X_1 > t\} = e^{-\lambda t}$. It is simple to see from (1.3.5) that[11]

$$\mathsf{E}[N(t)] = \lambda t \qquad\qquad \mathsf{V}[N(t)] = \lambda t.$$

[11] *See [1.3.8] for details.*

Interestingly, we can use (1.3.4) and (1.3.5) to derive the pdf of A_k from the Poisson distribution. From (1.3.5), $\mathsf{P}\{N(t)=0\} = e^{-\lambda t}$ when $k=0$, and as $A_0 = 0$ by definition, the RHS[12] of (1.3.4) becomes $1 - \mathsf{P}\{A_1 \le t\}$. Consequently, $\mathsf{P}\{A_1 \le t\} = 1 - e^{-\lambda t}$, and by differentiating this, the pdf is seen to be $f_{A_1}(t) = \lambda e^{-\lambda t}$. For $k \ge 1$, take the derivative with respect to t of (1.3.4) to see that $\mathrm{d}\mathsf{P}\{N(t)=k\}/\mathrm{d}t = f_{A_k}(t) - f_{A_{k+1}}(t)$. It then follows from (1.3.5) that

[12] *RHS := right hand side, LHS := left hand side.*

$$\lambda \frac{(\lambda t)^{k-1}}{(k-1)!} e^{-\lambda t} - \lambda \frac{(\lambda t)^k}{k!} e^{-\lambda t} = f_{A_k}(t) - f_{A_{k+1}}(t).$$

With recursion, we retrieve (1.3.3).

THE FAMILY OF rvs $N_\lambda = \{N(t), t \ge 0\}$ is a *Poisson process*[13] with *arrival rate* λ. Once we have N_λ, we can define $N(s,t] := N(t) - N(s)$ as the number of customers that arrive in a general time period $(s,t]$.[14]

[13] *A* random process *is a much more complicated mathematical object than a rv: a process is a (possibly uncountable) set of rvs indexed by time, not just* one *rv.*

[14] *Note that* $[0,t]$ *is closed at both ends, but* $(s,t]$ *is open at the left.*

It is important to know that N_λ has *stationary and independent increments*. Stationarity means that the distributions of the number of arrivals are the same for all intervals of equal length, that is, $N(s,t]$ has the same probability distribution as $N(u,v]$ if $t-s = v-u$. Independence means, roughly speaking, that knowing that $N(s,t] = n$ does not help to make any predictions about the value of $N(u,v]$ if the intervals $(s,t]$ and $(u,v]$ do not overlap.

(1.3.5) implies a number of useful properties of the Poisson process. With small o notation[15], if $t \ll 1$,

[15] *A function* $f(h) = o(h)$ *when* $f(h)/h \to 0$ *as* $h \to 0$, *e.g.,* $x^2 = o(x)$.

$$\begin{aligned}&\mathsf{P}\{N(t)=0\} = e^{-\lambda t} = 1 - \lambda t + o(t),\\ &\mathsf{P}\{N(t)=1\} = \lambda t e^{-\lambda t} = \lambda t(1 - \lambda t + o(t)) = \lambda t + o(t),\\ &\mathsf{P}\{N(t)\ge 2\} = 1 - \mathsf{P}\{N(t)=0\} - \mathsf{P}\{N(t)=1\}\\ &\qquad\qquad = 1 - (1-\lambda t) - \lambda t + o(t) = o(t).\end{aligned} \tag{1.3.6}$$

It is important to know that the reverse of the above holds, that is, a stochastic process with stationary and independent increments and satisfying (1.3.6) is necessarily a Poisson process.[16]

[16] *We refer to the literature on (mathematical) probability theory for further background.*

WHEN WE MERGE two independent Poisson processes N_λ and N_μ, e.g., adults and children entering a store[17] , we obtain a new Poisson process $N_{\lambda+\mu}$ with rate $\lambda+\mu$. To see this, we use the above. For $t \ll 1$,

[17] *Or some other feature by which you want to distinguish between customers.*

$$\begin{aligned}
\mathsf{P}\{N_{\lambda+\mu}(t)=0\} &= \mathsf{P}\{N_\lambda(t)=0\}\,\mathsf{P}\{N_\mu(t)=0\} \\
&= (1-\lambda t)(1-\mu t)+o(t) \\
&= 1-(\lambda+\mu)t+o(t), \\
\mathsf{P}\{N_{\lambda+\mu}(t)=1\} &= \mathsf{P}\{N_\lambda(t)=1\}\,\mathsf{P}\{N_\mu(t)=0\} \\
&\quad + \mathsf{P}\{N_\lambda(t)=0\}\,\mathsf{P}\{N_\mu(t)=1\} \\
&= \lambda t(1-\mu t)+(1-\lambda t)\mu t+o(t) \\
&= (\lambda+\mu)t+o(t),
\end{aligned} \tag{1.3.7}$$

and the rest of (1.3.6) follows similarly. So we see that we obtain the same type of equations as in (1.3.6) but with λ replaced by $\lambda+\mu$. Realizing that N_λ and N_μ have stationary and independent increments, it follows that $N_{\lambda+\mu}$ is a Poisson process with rate $\lambda+\mu$.

We can also *split*, or *thin*, a Poisson process into several substreams. For instance, consider a Poisson process N_λ of people passing by a shop. Each customer decides whether to enter by a Bernoulli distributed rv B, with success probability p and failure probability $q=1-p$. These decisions are mutually independent and independent of N_λ. Then the process of customers entering the shop is a Poisson process $N_{\lambda p}$ with rate λp, because, for $t \ll 1$, from (1.3.6),[18]

[18] *Why are there $o(t)$ symbols already after the first equality signs? (Hint, it can happen that $N(t)=2$, but both persons do not enter.)*

$$\begin{aligned}
\mathsf{P}\{N_{\lambda p}(t)=0\} &= \mathsf{P}\{N_\lambda(t)=0\}+\mathsf{P}\{N_\lambda(t)=1\}\,\mathsf{P}\{B=0\}+o(t) \\
&= 1-\lambda t+\lambda t q+o(t)=1-\lambda p t+o(t), \\
\mathsf{P}\{N_{\lambda p}(t)=1\} &= \mathsf{P}\{N_\lambda(t)=1\}\,\mathsf{P}\{B=1\}+o(t)=\lambda t p+o(t), \\
\mathsf{P}\{N_{\lambda p}(t)\geq 2\} &\leq \mathsf{P}\{N_\lambda(t)\geq 2\}=o(t)
\end{aligned}$$

The resulting process $N_{\lambda p}$ has an arrival rate of λp and has stationary and independent increments because N_λ is a Poisson process.

The concepts of *merging* and *thinning* are useful to analyze queueing networks. Suppose the departure stream of a machine splits into two substreams, e.g., a fraction p of the jobs moves on to another machine and the rest $(1-p)$ of the jobs leaves 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. Merging occurs where the output streams of various stations are sent to a common downstream station.

A FINAL INTERESTING property of the Poisson process is the distribution of arrivals over an interval $[0,t]$ given that $N(t)=n$.

Theorem 1.3.3. Conditional on the event $N(t)=n$, $t>0$, the distribution of the arrival times $\{A_i\}_{i=1}^n$ is the same as the distribution of the order statistics of n rvs $U_i \overset{\text{iid}}{\sim} \text{Unif}([0,t])$.[19]

[19] $\text{Unif}(A)$ *is the uniform distribution on the set A.*

—

Proof. We write for convenience $\mathrm{d}t_i = [t_i, t_i + \mathrm{d}t]$, and similarly $\mathrm{d}(t_i - t_{i-1}) = [t_i - t_{i-1}, t_i - t_{i-1} + \mathrm{d}t]$.[20] We provide the proof for $N(t) = 2$; the proof for the general case is nearly identical, but needs more notation.

Consider the Poisson process at $0 \le t_1 < t_2 \le t$. Then,

$$\mathsf{P}\{A_1 \in \mathrm{d}t_1, A_2 \in \mathrm{d}t_2 \,|\, N(t) = 2\} = \frac{\mathsf{P}\{A_1 \in \mathrm{d}t_1, A_2 \in \mathrm{d}t_2, A_3 > t\}}{\mathsf{P}\{N(t) = 2\}}, \quad (\star)$$

because $N(t) = 2 \iff A_2 \le t < A_3$. For the numerator,

$$\begin{aligned}\mathsf{P}\{A_1 \in \mathrm{d}t_1, A_2 \in \mathrm{d}t_2, A_3 > t\} &\overset{1}{=} \mathsf{P}\{X_1 \in \mathrm{d}t_1, X_2 \in \mathrm{d}(t_2 - t_1), X_3 > t - t_2\} \\ &\overset{2}{=} \lambda e^{-\lambda t_1}\,\mathrm{d}t\,\lambda e^{-\lambda(t_2 - t_1)}\,\mathrm{d}t\, e^{-\lambda(t - t_2)} \\ &\overset{3}{=} \lambda^2 e^{-\lambda t}\,\mathrm{d}t^2,\end{aligned}$$

where 1 follows from the recursive relation $A_i = A_{i-1} + X_i$; 2 because the $X_i \overset{\text{iid}}{\sim} \mathrm{Exp}(\lambda)$; 3 is algebra. Because $\mathsf{P}\{N(t) = 2\} = (\lambda t)^2 e^{-\lambda t}/2!$, we find that $\mathsf{P}\{A_1 \in \mathrm{d}t_1, A_2 \in \mathrm{d}t_2 \,|\, N(t) = 2\} = 2!\,\mathrm{d}t^2/t^2$.

As for the iid uniform rvs, $\mathsf{P}\{U_1 \in \mathrm{d}t_1, U_2 \in \mathrm{d}t_2\} = \mathrm{d}t^2/t^2$. However, there are 2! ways that the uniform rvs can hit the set $\mathrm{d}t_1\,\mathrm{d}t_2$, so the probability that the *order statistics* of U_1, U_2 lie in this set is $2!\,\mathrm{d}t^2/t^2$. This equals the conditional probability in $(\star)$. □

[20] *This notation allows us to write* $\mathsf{P}\{X_i \in dt_i\} = \lambda e^{-\lambda t_i}\,dt$ *when* $X_i \sim \mathrm{Exp}(\lambda)$, *and* $\mathsf{P}\{U_i \in dt_i\} = dt/t$ *when* $U_i \sim \mathrm{Unif}([0, t])$.

SUMMARIZING, THE POISSON process N_λ and the exponential distribution are very closely related: a counting process $\{N_\lambda(t)\}$ is a *Poisson process* with rate λ if and only if the inter-arrival times $X_k \overset{\text{iid}}{\sim} \mathrm{Exp}(\lambda)$. Thus, if you find it reasonable to model inter-arrival times as iid memoryless rvs, then the number of arrivals in an interval is necessarily Poisson distributed. Moreover, if you find it reasonable that the occurrence of an event in a small time interval is equally likely over time and independent from one interval to another, then the arrival process is Poisson, and the inter-arrival times are exponential.

TF 1.3.1. Let N be a Poisson process with rate λ and $0 < s < t$. Claim:

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

TF 1.3.2. Assume that $N_a(t) \sim \mathrm{Pois}(\lambda t)$, $N_s(t) \sim \mathrm{Pois}(\mu t)$ and independent. Claim:

$$\mathsf{P}\{N_a(t) + N_s(t) = n\} = e^{-(\mu+\lambda)t} \sum_{i=0}^{n} \frac{(\mu t)^{n-i}}{(n-i)!} \frac{(\lambda t)^i}{i!}.$$

TF 1.3.3. Claim: the algorithm below returns a random variable with distribution $\mathrm{Gamma}(\lambda, k)$.

```
Input: rate λ > 0 and integer k ≥ 1.
Output: value T.
function GENERATEGAMMA(λ, k)
    T ← 0
    for i = 1, ..., k do
        Draw U ~ Unif((0, 1)).
        T ←+ −(1/λ) log U
    return T
```

TF 1.3.4. Claim: the algorithm below returns a random variable with a Poisson distribution with parameter λ.

```
1: Input: parameter λ > 0.
2: Output: count N.
3: function GeneratePoisson(λ)
4:     N ← 0
5:     Draw U ~ Unif((0, 1)).
6:     T ← −log U
7:     while T < λ do
8:         N +← 1
9:         Draw U ~ Unif((0, 1)).
10:        T −← log U
11:    return N
```

TF 1.3.5. The inter-arrival times $\{X_k\}$ are iid and exponentially distributed with mean $1/\lambda$, and $A_k = \sum_{i=1}^{k} X_i$, $A_0 = 0$. Claim:

$$\mathsf{P}\{A_{k+1} \le t\} = -\frac{(\lambda t)^k}{k!} e^{-\lambda t} + \mathsf{P}\{A_k \le t\}.$$

THE EXERCISES REQUIRE pen and paper. All problems have solutions, but not all have hints.

Ex 1.3.6. Show that when $X \sim \mathrm{Exp}(\lambda)$ its mgf[21] is $M_X(t) := \mathsf{E}\left[e^{tX}\right] = \lambda/(\lambda - t)$ for $t < \lambda$, and use this to calculate $\mathsf{E}[X]$ and $\mathsf{V}[X]$.

[21] *mgf := moment generating function.*

Ex 1.3.7. Use mgfs to prove that A_i has density $f_{A_i}(t) = \lambda e^{-\lambda t} \frac{(\lambda t)^{i-1}}{(i-1)!}$.

Ex 1.3.8. When $N \sim \mathrm{Pois}(\lambda)$ and $\alpha > 0$, show that $\mathsf{E}\left[\alpha^N\right] = e^{\lambda(\alpha-1)}$. Use this to see that $M_{N(t)}(s) = \exp(\lambda t(e^s - 1))$. Then find $\mathsf{E}[N(t)]$ and $\mathsf{V}[N(t)]$.

Ex 1.3.9. If $X \sim \mathrm{Exp}(\lambda), S \sim \mathrm{Exp}(\mu)$ and are independent, show that $Z = \min\{X, S\} \sim \mathrm{Exp}(\lambda + \mu)$, hence $\mathsf{E}[Z] = (\lambda + \mu)^{-1}$.

Ex 1.3.10. If the Poisson arrival processes N_λ and N_μ are independent, show that

$$\mathsf{P}\{N_\lambda(t) = 1 \,|\, N_\lambda(t) + N_\mu(t) = 1\} = \frac{\lambda}{\lambda + \mu}.$$

In other words, given that a customer arrived in $[0, t]$, the probability that the customer is of the first type is $\lambda/(\lambda + \mu)$.

Ex 1.3.11. If $X \sim \mathrm{Exp}(\lambda), S \sim \mathrm{Exp}(\mu)$ and are independent, show that

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

IF THE SOMEWHAT heuristic derivations above do not satisfy your sense for rigor, then you should solve the next set of exercises; otherwise you can skip them.

Opt 1.3.12. If the Poisson arrival processes N_λ and N_μ are independent, use moment generating functions to show that $N_\lambda + N_\mu$ is a Poisson process with rate $\lambda + \mu$.

Opt 1.3.13. Show with moment-generating functions that thinning the Poisson process N_λ by means of Bernoulli rvs with success probability p results in a Poisson process $N_{\lambda p}$.



1.4 FIGHTING THE EXPONENTIAL DISTRIBUTION IS FUTILE

Perhaps the mathematical arguments in the preceding section have not fully convinced you that exponential inter-arrival times and Poisson arrival counts arise naturally in many systems. We therefore use a simulation to show how exponential inter-arrival times emerge when many nearly periodic patient streams are combined.

The simulation is based on an internship of one of my master's students with the task of evaluating the scheduling system for patients with inflammatory bowel disease (IBD) at a hospital. Patients with IBD can be categorized as regular and urgent. Regular patients required check-ups approximately every six months, whereas urgent patients needed to be seen within one week, that is, five working days.[1] Each consultation takes 10 minutes. Immediately after a consultation, regular patients scheduled their next visit for approximately six months later. There are about 1000 patients in the system and a working year consists of 200 working days. Given that each patient is seen twice per year, a physician needed to consult with an average of 10 patients daily.[2] The problem was to design a planning procedure that allowed enough flexibility to meet the time constraints for the urgent patients without leading to idle times for the physicians. Of course, new patient intakes occurred, but for simplicity we assume that the patient population remains constant.

[1] *This procedure has been changed recently; however, the ideas of this case apply much more generally.*

[2] *Two visits per year per patient times 1000 patients distributed over 200 days per year is 10 patients per day.*

Planning regular patients six months in advance seems to lead to stable arrival times, but this is not true, as there are many practical disturbances. A significant number of patients rescheduled their appointment date, for example, due to altered vacation plans. There were no-shows and yet other patients relocated to other cities; some died. However, even without such practical problems and even when each patient individually shows relatively predictable behavior, the simulation results below show that *at the population level* the inter-arrival times are well modeled as exponentially distributed rvs.

In conclusion, whether the planner plans six months in advance or does not plan at all, there will not be much difference in the variability of the number of patients arriving per day. In other words, the idea that making appointments half a year in advance will reduce variability by a large amount is simply not true. The verdict must then be clear: When rules do not have an effect, it is best to drop them altogether. It is simply better to send patients a reminder for a new appointment three weeks before a new visit is due, rather than planning six months in advance. Far

fewer replanning actions are needed, and the probability for no-shows decreases.

WE NOW TURN to developing the simulation to show how the exponential distribution arises 'out of thin air'. We focus only on regular patients and neglect urgent patients as this is a relatively small group. Assume that the hospital has just 50 regular patients. For ease, scale the time so that each patient plans the next appointment uniformly in the interval $[0.9, 1.1]$, independent of previous appointments and other patients.[3]

The question of interest is to estimate the distribution of inter-arrival times between two patients *as seen by the planner* of the hospital. Of course, with only one patient this is straightforward, the planner sees $\sim \text{Unif}([0.9, 1.1])$. But what happens for many patients?

Fig. 1.4.1 provides the answer to this question, but we first discuss the algorithms used to produce the figure, as they clarify how to interpret it.

[3] *Like this, the inter-arrival times between two consecutive appointments correspond to half a year (100 working days) plus or minus 10 days (observe that* $[0.9, 1.1] \times 100 = [90, 110]$*).*

THE SIMULATION USES $N = 50$ patients and $T = 10$ half-year periods. For each patient i, the inter-arrival times $X_k^i \sim \text{Unif}([0.9, 1.1])$ are independent, and the arrival times satisfy $A_k^i = A_{k-1}^i + X_k^i$, as in (1.3.2). Alg. 1.4.1 applies these recursions to all patients.

Algorithm 1.4.1 Generate the arrival times of all patients.

```
Input: number of patients N, number of visits T.
Output: patient arrival-time lists [A^1, ..., A^N], where A^i = [A_0^i, ..., A_T^i].
function GeneratePatientArrivals(N, T)
    L ← []
    for 1 ≤ i ≤ N do
        A_0^i ← 0
        A^i ← [A_0^i]
        for 1 ≤ k ≤ T do
            Draw X_k^i ~ Unif([0.9, 1.1]).
            A^i ← Append(A^i, A_{k-1}^i + X_k^i)
        L ← Append(L, A^i)
    return L
```

Next, we build the upper left panel of Fig. 1.4.1. Each row contains a number of dots, and each dot corresponds to an arrival of a patient.[4] As we have 50 patients, we have 50 such rows of dots.

[4] *The x coordinate of the kth dot of patient i is A_k^i and its y coordinate is i.*

We plot the arrivals as seen by the planner as a *jitter plot* at $y = 0$. A jitter plot adds small random noise in the y direction to the data points. By spreading out the points slightly, jittering allows for a clearer visualization of the distribution of individual points. Without the jitter, we would just see small line segments instead of somewhat rectangular shapes.

THE HOSPITAL COMBINES the arrivals of all patients into one ordered sequence. Alg. 1.4.2 sorts the individual arrival times and computes the time between each pair of consecutive arrivals as seen by the hospital.

Using Alg. 1.4.2, we compute the inter-arrival times seen by the hospital and assemble them in 50 bins. On average, we have $\lambda = 50$

arrivals per unit time; thus, the mean inter-arrival time is $1/\lambda = 1/50$. Next, recall that the pdf of the exponential distribution is $\lambda e^{-\lambda x}$. To plot the pdf, we take as support for the plot 5 times the mean time between arrivals, that is, $5/\lambda = 0.1$. We then plot the histogram of the inter-arrival times and the pdf in the upper right panel.

Algorithm 1.4.2 Merge patient arrivals and compute hospital inter-arrival times.

Input: patient arrival-time lists $[A^1, \ldots, A^N]$.
Output: inter-arrival-time list $X = [X_1, \ldots, X_{N \cdot T}]$.
function MERGEPATIENTARRIVALS($[A^1, \ldots, A^N]$)
    $A_0 \leftarrow 0$
    $(A_1, \ldots, A_{N \cdot T}) \leftarrow \text{QSORT}\left(\left[A_k^i : 1 \le i \le N,\ 1 \le k \le T\right]\right)$
    $X \leftarrow []$
    for $1 \le j \le N \cdot T$ **do**
        $X \leftarrow \text{APPEND}\left(X, A_j - A_{j-1}\right)$
    return X

WHAT CAN WE see in the upper panels of Fig. 1.4.1? In the upper left panel, the arrival times of each patient are at first tightly concentrated around 1, but as time progresses, the arrival times become more spread out. The jitter plot around $y = 0$ also demonstrates this. The rectangles become wider until they almost overlap. In the right upper panel, we see that in comparison to the exponential density, there are relatively many short inter-arrival times. This is expected because all patients start at the same time, so arrival times will be concentrated for the first couple of visits.[5] Therefore, letting all patients start at the same time is not a good model for a hospital that has been serving patients for years.

To compensate for these unnatural start times, we set the initial arrival time of each patient as uniformly distributed on $[0, 1]$. With this change, we run the above code again and plot the results in the lower left and right panels of Fig. 1.4.1. Now we see that the jitter rectangles, that is, the arrivals as seen by the hospital, overlap right from the start, and the exponential distribution is an excellent fit.

[5] *But even with these unrealistic starting times, the exponential distribution is not a bad fit.*

TO CONCLUDE, THE combined inter-arrival times in this simulation are approximately exponentially distributed. This approximation is useful in other systems fed by many mutually independent arrival streams, provided no single stream dominates.[6] To avoid any confusion, here we model the inter-arrival times for one patient as uniformly distributed, but the same phenomenon occurs when the inter-arrival times for one patient have other types of distribution.

The arrival times above are sorted; Alg. 1.2.14 gives an algorithm for sorting a list.

[6] *You need strong arguments for other approximations.*

IT REMAINS TO formalize the observations from the above plots.[7] In fact, we should distinguish between two limiting regimes. Patients recurrently plan appointments, separated around some mean period, that is, roughly every half a year. The upper left panel in Fig. 1.4.1 suggests that if the number of appointments increases, the appointments for each patient

[7] *If you are not interested in probability theory, you can skip the rest of this section.*

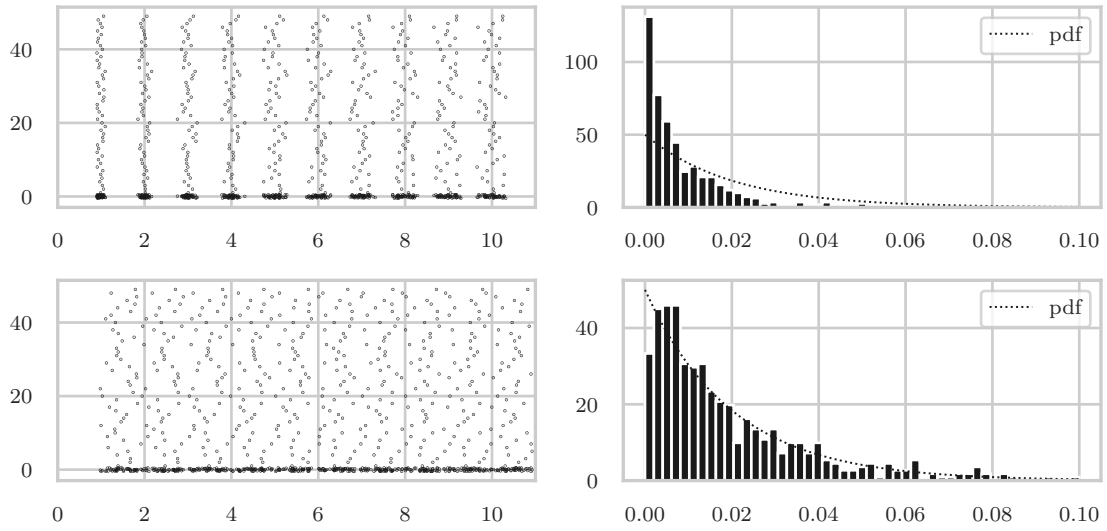


FIG. 1.4.1: How the uniform distribution leads to the exponential distribution.

eventually become uniformly distributed on the period length (here half a year). The lower left panel suggests that if the appointments are uniformly distributed to begin with, they remain uniformly distributed. So, our first step is to prove that, as the number of visits increases, the arrival times modulo the cycle length become uniformly distributed over the cycle. The second claim is that, in the limit of many patients, each arriving at a uniformly distributed time in an interval, the inter-arrival times of all patients together become exponentially distributed, in a scaled sense.

The theorems and proofs are elegant and combine order statistics, the beta distribution, first-step analysis, and some infinitesimal calculus to give the reasoning a heuristic flavor. Finally, we discuss an important smoothing property of taking expectations. This idea is also relevant in data science when moving averages are used.

LET US FIRST show that periodically planned arrivals, with some stochasticity, tend to become uniformly distributed over an interval. To formalize this claim, we consider a discrete-time model. There are on average C days between two successive visits of a patient (for the moment, the time on a day is of no importance). For instance, let the inter-arrival times X_i be such that $\mathsf{P}\{X_i = C-1\} = \mathsf{P}\{X_i = C+1\} = 1/2$. In this case, if the process starts on day 0 and $X_1 = C-1$, the next visit will be on day $C-1$, but if $X_1 = C+1$, the next visit will be on day 1 of the next cycle. Looking at the arrival process like this, the patient *moves on a circle* with states $\{0,1,\ldots,C-1\}$, and with every visit the patient moves with probability $1/2$ one state to the left or to the right. Thus, when $X = C-1$, we can just as well model this as $X = -1$.[8]

Theorem 1.4.3. Assume that the arrival times $\{A_i\}$ live on the circle space $\{0,1,\ldots,C-1\}$, i.e., $A_i = (A_{i-1} + X_i) \bmod C$. If $\{X_k\}_{k=1}^{\infty}$ is a set of iid rvs such that $\mathsf{E}[X_k] < \infty$, $\sup_j \mathsf{P}\{X_k = j\} < 1$, and the greatest common divisor of C and all differences $x - y$ between possible values of X_k is 1,

[8] *There is a small subtlety when the cycle length is even and* $\mathsf{P}\{X = C-1\} = \mathsf{P}\{X = C+1\} = 1/2$. *As an even number is a multiple of* 2, *the odd (even) days can only be visited after an odd (even) number of visits. The problem is easily resolved by assuming, in addition, that* C *is odd or* $\mathsf{P}\{X = 0\} > 0$.

then $\alpha_k(i) = \mathsf{P}\{A_k = i\}$, i.e., the probability that the kth arrival is on the ith day, converges to $1/C$ as $k \to \infty$.

—

Proof. The condition $\mathsf{E}[X_k] < \infty$ restricts attention to increments with a finite mean, although it is not needed for convergence on the finite circle. We need $\sup_j \mathsf{P}\{X_k = j\} < 1$ to exclude trivial rvs.[9]

[9] *A rv X is trivial when $\mathsf{P}\{X = a\} = 1$ for some a.*

For the proof, assume for the moment that $\mathsf{P}\{X_k = 1\} = p \in (0,1)$ and $\mathsf{P}\{X_k = -1\} = q = 1-p$, and let C be odd.[8] By first-step analysis,

$$\alpha_{k+1}(i) = p\alpha_k(i-1) + q\alpha_k(i+1).$$

Clearly, if $\alpha_k(i) = 1/C$ for all i, then $\alpha_{k+1}(i) = 1/C$ also. However, if not all $\alpha_k(\cdot)$ are equal, there must be at least one state i such that $\alpha_k(i)$ is strictly larger than at least one of its neighbors, that is, $\alpha_k(i) \geq \max\{\alpha_k(i-1), \alpha_k(i+1)\}$ and $\alpha_k(i) > \min\{\alpha_k(i-1), \alpha_k(i+1)\}$.[10] But then, on the next visit,

[10] *Observe that, for this argument, it is necessary that C is finite.*

$$\alpha_{k+1}(i) = p\alpha_k(i-1) + q\alpha_k(i+1) < \alpha_k(i).$$

In words, averaging smooths local peaks and troughs, although the global maximum need not decrease after every visit.

For general step distributions, the stated gcd condition ensures that, after sufficiently many visits, there is a positive probability of moving from any position to any other position.[11] Thus, after that many visits, every $\alpha_k(i)$ is replaced by a weighted average of all the previous probabilities, with strictly positive weights. Unless all probabilities are equal, the maximum of these averages is strictly smaller than the previous maximum, and the minimum is strictly larger than the previous minimum. Let $\epsilon > 0$ be the smallest of these finitely many weights. Each such averaging step reduces the difference between the maximum and minimum to at most $1 - C\epsilon$ times its previous value. This difference therefore tends to zero. Since the probabilities sum to one, each $\alpha_k(i)$ converges to $1/C$.

[11] *If you are interested in maths: the positions form a finite irreducible and aperiodic Markov chain; hence some power of its transition matrix has only positive entries.*

Clearly, the proof applies to any finite $C = 1, 2, \ldots$. To see how to generalize the argument to the circle $[0,1)$ on the reals, consider a continuous function f on the unit circle C. As the circle is compact and f continuous, f achieves its maximum $f(\bar{x})$ at $\bar{x}$, say. Suppose for some $\epsilon > 0$ there is set $A_\epsilon(\bar{x}) = \{x \in C : f(x) \leq f(\bar{x}) - \epsilon\}$ such that $\mathsf{P}\{A_\epsilon(\bar{x})\} > 0$. (If there is no such set for any ϵ, then f is a constant.) But then $\mathsf{E}[f(\bar{x} + X)] \leq (f(\bar{x}) - \epsilon)\,\mathsf{P}\{A_\epsilon(\bar{x})\} + f(\bar{x})(1 - \mathsf{P}\{A_\epsilon(\bar{x})\}) < f(\bar{x})$. Again, taking an average over f makes the maximum smaller and the minimum larger. □

Applied to the hospital, after many visits of a patient—each roughly half a year apart— the visits will appear to be uniformly distributed on a 'circle' of half-year length.

There is one point that remains about the lower left panel of Fig. 1.4.1; recall that for this panel we started the simulation with uniformly distributed arrival times. What is the distribution of the arrival times *on the circle* if we start like this? Let us analyze this for the case in which the inter-arrival times are uniform on $[0,1]$ and we take the circle as $C = [0,1)$.

Observe that when $C = [0,1)$, $U + V \bmod 1$, that is, the sum of two iid uniform rvs on the circle, is just the fractional part of $U + V$.

Theorem 1.4.4. Let $Z = \sum_{i=1}^{n} U_i$ for n rvs $U_i \overset{\text{iid}}{\sim} \text{Unif}([0,1))$. The fractional part of Z, i.e., $Z - \lfloor Z \rfloor$, is uniformly distributed on $[0,1)$.[12]

[12] $\lfloor x \rfloor$ *is the integer part of* $x \in \mathbb{R}$.

—

Proof. Realizing that the density of the sum of two uniform rvs on $[0,1]$ is a triangle with base $[0,2]$ and height 1,

$$\begin{aligned}\mathsf{P}\{U + V - \lfloor U + V \rfloor \le x\} &= \mathsf{P}\{U + V \le x\} + \mathsf{P}\{1 \le U + V \le 1 + x\} \\ &= x^2/2 + (1/2 - (1-x)^2/2) = x.\end{aligned}$$

Therefore, the fractional part of the sum of two uniform rvs is uniform on $[0,1]$. We can apply this result successively through the summation in Z. More generally, if $U \sim \text{Unif}([0,1))$ and V is any independent rv, then $(U + V) \bmod 1$ is uniform on $[0,1)$. Conditional on $V = v$, adding v simply rotates the uniform distribution around the circle. Thus, a uniform initial phase remains uniform when the inter-arrival times are distributed on $[0.9, 1.1]$, as in our simulation. □

We now deal with the case with many patients, that is, when the number n of patients goes to infinity. By the above results, we model the arrival times as rvs $\{U_k\}_{k=1}^{n}$ such that $U_k \overset{\text{iid}}{\sim} \text{Unif}([0,1))$. Recall that in the simulation we sorted the arrival times of all patients together to obtain the ordered sequence of arrivals as seen by the hospital. In probabilistic terms, the hospital sees *the order statistics* of $\{U_k\}_{k=1}^{n}$. Write this as $0 = A_0^n < A_1^n < A_2^n < \cdots < A_n^n < A_{n+1}^n = 1$.[13] Define the size of the inter-arrivals as seen by the hospital as $X_k^n = A_k^n - A_{k-1}^n$, $k = 1, \ldots, n+1$.

[13] *We neglect the probability of simultaneous arrivals; these have probability zero.*

As the population size n increases, the size of the first gap, i.e., the properly scaled arrival time of the first patient, becomes increasingly like an exponentially distributed random variable.[14]

[14] *We need to scale with a factor n because the number of patients increases.*

Lemma 1.4.5.

$$\lim_{n\to\infty} \mathsf{P}\{A_1^n \le x/n\} = \lim_{n\to\infty} \mathsf{P}\{X_1^n \le x/n\} = 1 - e^{-x}, \quad x \ge 0.$$

—

Proof. The probability that the smallest of n rvs is less than some x is the same as 1 minus the probability that all n rvs are larger than x. Therefore,

$$\mathsf{P}\{A_1^n \le x/n\} = 1 - \mathsf{P}\{\min\{U_1, \cdots, U_n\} > x/n\} = 1 - (1 - x/n)^n,$$

because $\mathsf{P}\{U_i > x/n\} = 1 - x/n$, and the U_i are iid. The RHS converges to $1 - e^{-x}$ as $n \to \infty$. Finally, from the definition, $X_1^n = A_1^n - A_0^n = A_1^n$. □

Next, we show that all the hospital inter-arrival times have the same density.

Lemma 1.4.6. For $r \in (0,1)$, the density of the kth inter-arrival time is $f_{X_k^n}(r) = n(1-r)^{n-1}$, for $k = 1, \ldots, n+1$.

—

Proof. From the proof of the previous lemma we have $\mathsf{P}\{X_1^n \le r\} = 1-(1-r)^n$, therefore $f_{X_1^n}(r)\,\mathrm{d}r = \mathsf{P}\{X_1^n \in [r, r+\mathrm{d}r]\} = n(1-r)^{n-1}\,\mathrm{d}r$. By symmetry, $X_1^n \sim X_{n+1}^n$. It remains to deal with the intermediate inter-arrivals.

The probability of the event $A_1^n < A_2^n < \cdots < A_{k-2}^n < x$, $A_{k-1}^n \in [x, x+\mathrm{d}x]$, $A_k^n \in [y, y+\mathrm{d}y]$, $y < A_{k+1}^n < \cdots < A_n^n$ is given by

$$f_{A_{k-1}^n, A_k^n}(x,y)\,\mathrm{d}x\,\mathrm{d}y = \frac{n!}{(k-2)!(n-k)!}x^{k-2}(1-y)^{n-k}\,\mathrm{d}x\,\mathrm{d}y,$$

because $k-2$ arrivals must occur before x, $n-k$ after y and one arrival in $[x, x+\mathrm{d}x]$ and one in $[y, y+\mathrm{d}y]$. With this, the probability that $X_k^n \in [r, r+\mathrm{d}r]$ becomes

$$f_{X_k^n}(r)\,\mathrm{d}r = \mathrm{d}r \int_0^{1-r} f_{A_{k-1}^n, A_k^n}(x, x+r)\,\mathrm{d}x,$$

because $X_k^n = A_k^n - A_{k-1}^n = r \implies y = A_k^n = A_{k-1}^n + r = x + r$, and x can lie anywhere in $[0, 1-r]$.

Substituting in this integral the density $f_{A_{k-1}^n, A_k^n}$, but dropping the factorials for the moment for notational ease, we find

$$\begin{aligned}\int_0^{1-r} x^{k-2}(1-r-x)^{n-k}\,\mathrm{d}x &= (1-r)^{n-2}\int_0^{1-r}\left(\frac{x}{1-r}\right)^{k-2}\left(\frac{1-r-x}{1-r}\right)^{n-k}\mathrm{d}x \\ &= (1-r)^{n-1}\int_0^1 x^{k-2}(1-x)^{n-k}\,\mathrm{d}x \\ &= (1-r)^{n-1}\frac{(k-2)!(n-k)!}{(n-1)!},\end{aligned}$$

where we use the normalization constant of the Beta distribution to compute the last integral. Canceling the factorials proves the claim. □

Combining the above two lemmas leads straightaway to the next theorem.

Theorem 1.4.7. Let n jobs arrive at a station such that the arrival times are rvs $U_k \overset{\text{iid}}{\sim} \mathrm{Unif}([0,1])$. Then all inter-arrival times $\{X_k^n\}_{k=1}^{n+1}$ as seen by the server (the hospital) have the same distribution. The common distribution of the scaled inter-arrival times nX_k^n converges to an exponential distribution with rate 1: $\lim_{n\to\infty}\mathsf{P}\{nX_k^n \le x\} = 1-e^{-x}, \quad x \ge 0.$

—

Proof. By Lem. 1.4.6, the rvs $\{X_k^n\}_{k=1}^n$ have the same density $n(1-r)^{n-1}$. By symmetry, X_{n+1}^n has this density too. By Lem. 1.4.5, $\lim_{n\to\infty}\mathsf{P}\{nX_k^n \le x\} = 1-e^{-x}$ for each k. □

1.5 PROBABILISTIC ARITHMETIC

In Section 1.4 we were interested in how the visiting times of a patient would become distributed over periods of half a year after many visits. We can formalize this problem as follows for one patient. Let X_k be the kth inter-arrival time, and write C for the length of the interval in which the patient has to see a doctor; here C is half a year. Then the arrival times modulo C satisfy

$$Z_k = Z_{k-1} + X_k \bmod C, \quad Z_0 = 0,$$

where $X_k \bmod C$ is the remainder of X_k/C. To compute the distribution of Z_k we need to be able to compute the distribution of $Y_k = Z_{k-1} + X_k$, and then find the distribution of $f(Y_k)$ where $f(x) = x \bmod C$. Clearly, this is a very unattractive task to carry out by hand. In this section we develop algorithms that allow a computer to carry out this tedious work.

More generally, later sections require numerical work to gain insight into the behavior of stochastic processes. We therefore develop a small collection of functions for *probabilistic arithmetic*. The emphasis is on reusable algorithmic patterns: how probability distributions are stored, how functions transform and combine random variables, and why these functions work. At the end of this section, we use these algorithms to study the distribution of $\{Z_k\}$ discussed above.

LET US START by making explicit what we mean by a pattern. As a first pattern, consider the + operator. Suppose that we only know how to do addition on the positive integers.[1] Observe that the definition of addition on integers cannot be directly applied to rationals. To add fractions, we need to extend the symbol + to the following procedure: make the denominators of the fractions identical, and then add the numerators.[2] However, this procedure does not work to compute $\sqrt{2} + \sqrt{3}$.[3] To add general real numbers, we need to extend the definition of + yet further.[4] Next, we need to define addition for complex numbers, vectors, matrices, …. In summary, when learning mathematics, we have been silently *overloading* the + operator to many different situations.

In our present case, we *write* $X + Y$ for two independent rvs to mean the convolution:

$$\mathsf{P}\{X + Y = z\} = \sum_i \sum_j p_i q_j \mathbb{1}\{i + j = z\}, \tag{1.5.1}$$

where $p_i = \mathsf{P}\{X = i\}$, $q_j = \mathsf{P}\{Y = j\}$ and the indicator $\mathbb{1}\{A\} = 1$ if the event A occurs, and 0 otherwise. Below we express this as an algorithm.[5]

Thus, the first pattern that helps us to reason in more abstract ways about problems is to use *operator overloading*.

This brings us to a second pattern: observe that the expression $X + Y$ combines two rvs X and Y and then applies some function. Similarly, the operations $X - Y$, XY, $\max\{X, Y\}$ are all *functions applied to two independent rvs*. What we therefore need is a *method* to compute the pmf of a new rv $f(X, Y)$, where f is some general *binary* function.[6]

[1] *For example:* $3 + 5 = 8$.

[2] $1/3 + 1/2 = 2/6 + 3/6 = (2 + 3)/6 = 5/6$.

[3] *$\sqrt{2}$ and $\sqrt{3}$ are not rationals so we cannot make the denominators equal.*

[4] *One way is like this:* $\sqrt{2} + \sqrt{3} = \sup\{x + y : x, y \in \mathbb{Q}, x^2 \le 2, y^2 \le 3\}$.

[5] *For addition there exists a much faster method based on Fast Fourier transforms.*

[6] *A binary function depends on two arguments.*

We identify a third pattern when we realize that to compute $\mathsf{V}[X]$, we need $\mathsf{E}[X^2]$. But $\mathsf{E}[X^2]$ is the expectation of the rv $g(X)$ where $g(x) = x^2$. So, we also need a method to compute the pmf of the rv $g(X)$, where now g is some general *unary* function.[7]

[7] *A unary function depends on just one argument.*

In summary, we want to develop functions that support the following operations on rvs:

1. operator overloading for addition, subtraction, and potentially other arithmetical operations;
2. the application of a unary function to obtain a new rv;
3. the application of a binary function.

We call these functions probabilistic arithmetic, and we will develop, step by step, clear and general algorithms to support this. However, we first need to introduce some computer science concepts.

A USEFUL DATA structure for storing the pmf of a discrete rv is a map, as constructed by MAP in Alg. 1.2.6. Its keys are the possible values of the rv, and the value stored under each key is its probability. Since values outside the support have probability zero, we use the default value zero. We denote the pmf map of an rv X by p_X, so that $p_X(x) = \mathsf{P}\{X = x\}$.

Algorithm 1.5.1 Store two probability mass functions.

```
1: p_X ← MAP()                                   ▷ Initialize p_X.
2: p_X(1) ← 1/2, p_X(2) ← 1/2
3: p_Y ← MAP()
4: p_Y(1) ← 1/2, p_Y(2) ← 1/2
```

It is obvious that the pmf map p_Z for $Z = X + Y$ should satisfy $p_Z(2) = 1/4$, $p_Z(3) = 1/2$, and $p_Z(4) = 1/4$. Let us use (1.5.1) to compute the pmf of $Z = X + Y$.

Algorithm 1.5.2 Compute the pmf of the sum.

```
1: Input: pmf maps p_X, p_Y.
2: Output: pmf map p_Z for X + Y.
3: p_Z ← MAP()
4: for i ∈ [1, 2] do
5:     for j ∈ [1, 2] do
6:         p_Z(i + j) +← p_X(i) · p_Y(j)
7: return p_Z
```

Because p_Z is a map with default value zero, the first contribution to a key is added to zero. Later contributions are added to the value already stored under that key. In particular, the two pairs $(1,2)$ and $(2,1)$ each add $1/4$ to $p_Z(3)$. Thus, the algorithm returns $p_Z(2) = 1/4$, $p_Z(3) = 1/2$, and $p_Z(4) = 1/4$.

THE ABOVE ALGORITHM assumes that mathematically equal keys have the same representation in the computer. This requires care when the

keys are stored as floating-point numbers. Most real numbers cannot be represented exactly with finitely many digits. Consequently, two expressions that are mathematically equal, such as $1/3$ and $1-2/3$, can be stored as slightly different numbers. An implementation of MAP in a real computer language then treats them as different keys and does not combine their probabilities. This is a general problem of computer arithmetic, not a problem specific to a programming language. We avoid it by using exact numbers, such as integers or fractions, whenever possible. Otherwise, we round all keys to an agreed precision before storing them.

The function TOFRACTION of Alg. 1.5.3 replaces a number $x \in \mathbb{R}$ by a nearby fraction, rounded to d decimal places. It first uses denominator 10^d and then reduces the fraction using the greatest common divisor.[8] Keeping this conversion in a separate function hides its details from the algorithms that use it. If we later change the precision or the conversion rule, we only have to change TOFRACTION.

[8] *The greatest common divisor can be computed with Euclid's algorithm. See Euclidean algorithm for its implementation.*

Algorithm 1.5.3 Convert a number to a fraction.

```
1: Input: a number x and precision d.
2: Output: a reduced fraction a/b.
3: function ToFraction(x, d = 10)
4:     q ← 10^d
5:     p ← round(x · q)
6:     g ← GCD(|p|, q)
7:     return p/g, q/g
```

The function MAKEPMF of Alg. 1.5.4 uses the conversion to fractions to construct a pmf map from a map of outcomes and non-negative weights. It first converts every outcome with TOFRACTION. When two outcomes are converted to the same fraction, their weights are added. Finally, NORMALIZEMAP, Alg. 1.5.5, makes the stored weights sum to one. Rounding the outcomes and discarding small weights makes the resulting pmf a numerical approximation to the original distribution.

Algorithm 1.5.4 Construct a probability mass function.

```
1: Input: a finite map w with non-negative values and positive sum.
2: Output: a pmf map p.
3: function MakePMF(w)
4:     r ← Map()
5:     for x ∈ Keys(w) do
6:         r(ToFraction(x)) +← w(x)
7:     return NormalizeMap(r)
```

The function NORMALIZE of Alg. 1.2.3 normalizes a list of weights. For the given map w, we store the weights in a map whose keys are the possible outcomes. It then removes entries whose relative weight does not exceed the default numerical threshold $\epsilon = 10^{-16}$, and then normalizes the remaining weights.

Algorithm 1.5.5 Normalize the values stored in a map.

Input: a finite map w with non-negative values and positive sum; threshold ϵ.
Output: a map p whose values sum to one.
function NORMALIZEMAP($w, \epsilon = 10^{-16}$)
    $T \leftarrow$ SUM (VALUES(w))
    $S \leftarrow$ FILTER (KEYS(w), $\lambda\colon\ k \mapsto w(k)/T > \epsilon$)
    $total \leftarrow$ SUM ($[w(k) : k \in S]$)
    $V \leftarrow$ APPLY (S, $\lambda\colon\ k \mapsto w(k)/total$)
    return MAKEMAP $\big(keys = S, values = V\big)$

WE NOW USE λ functions to compute the different moments of a rv X.

The function EXPECTATION applies f to every possible value of the rv represented by p_X to compute $\mathsf{E}\big[f(X)\big]$.

Algorithm 1.5.6 Compute the expectation of functions of a discrete random variable X.

Input: pmf map p_X, function f.
Output: $\mathsf{E}\big[f(X)\big]$.
function EXPECTATION(p_X, f)
    return SUM $\big(\big[f(x) \cdot p_X(x) : x \in \text{KEYS}(p_X)\big]\big)$

With this, Alg. 1.5.7 computes the mean and variance with calls to simple λ functions. We choose to build the VARIANCE as function, because as λ function it becomes somewhat long.

Algorithm 1.5.7 Compute moments of a discrete random variable.

Input: pmf map p_X.
Output: mean, second moment, variance, and standard deviation of X.
MEAN $\leftarrow \lambda\colon\ p_X \mapsto$ EXPECTATION(p_X, $\lambda\colon\ x \mapsto x$)
SECONDMOMENT $\leftarrow \lambda\colon\ p_X \mapsto$ EXPECTATION(p_X, $\lambda\colon\ x \mapsto x^2$)
function VARIANCE(p_X)
    return SECONDMOMENT(p_X) $-$ MEAN(p_X)^2
STANDARDDEVIATION $\leftarrow \lambda\colon\ p_X \mapsto \sqrt{\text{VARIANCE}(p_X)}$

Another nice example is the cdf; we can use that $F_X(x) = \mathsf{P}\{X \le x\} = \mathsf{E}\,[\mathbb{1}\{X \le x\}]$.

Algorithm 1.5.8 Compute the cdf through expectation.

Input: pmf map p_X.
Output: cdf F_X.
function CDF_SLOW(p_X)
    return $\lambda\colon\ x \mapsto$ EXPECTATION(p_X, $\lambda\colon\ y \mapsto \mathbb{1}\{y \le x\}$)

The function returned by CDF_SLOW(p_X) remembers the map p_X. When called with x, it sums $p_X(y)$ over all keys $y \le x$, giving $F_X(x) = \mathsf{P}\{X \le x\}$. For example:

```
F_X ← CDF_SLOW(p_X)        ▷ Binds F_X to λ: x ↦ E[𝟙{X ≤ x}].
F_X(x)                     ▷ Returns P{X ≤ x}.
```

TO EVALUATE THE CDF at many different values of x, we need not sum the probabilities again at every call. Instead, we compute the cumulative probabilities once and then use SEARCHSORTED, Alg. 1.5.27, to find which cumulative probability to return.[9] First, we sort the support of X in increasing order and list the probabilities in that same order. We then use Alg. 1.5.9 to compute their cumulative sums, which give the cdf at the support points. PREPARECDF, Alg. 1.5.10, implements this as a function.

[9] *SearchSorted takes some explanation. We prove its correctness at the end of the main text.*

Algorithm 1.5.9 Compute the cumulative sums of a list.

```
Input: a list A.
Output: the list C of running totals of A.
function CUMSUM(A)
    C, total ← [], 0
    for a ∈ A do
        total +← a
        C ← APPEND(C, total)
    return C
```

Here is an example how this works. We use M for the map of cumulative probabilities and F for the cdf function.

```
p_X ← MAP()
p_X(3) ← 1/4,   p_X(2) ← 1/3,   p_X(1) ← 5/12
S ← QSORT(KEYS(p_X))                    ▷ S = [1, 2, 3].
P ← [p_X(s_1), p_X(s_2), p_X(s_3)]      ▷ P = [5/12, 1/3, 1/4].
C ← CUMSUM(P)                           ▷ C = [5/12, 3/4, 1].
M ← MAKEMAP(keys = S, values = C)
F ← CDF(p_X)
F(1.1)                                  ▷ F(1.1) = 5/12.
```

Algorithm 1.5.10 Prepare the support and cumulative probabilities of a discrete random variable.

```
Input: pmf map p_X.
Output: cumulative-probability map M.
function PREPARECDF(p_X)
    S ← QSORT(KEYS(p_X))
    C ← CUMSUM([p_X(s_1), ..., p_X(s_n)])
    return MAKEMAP(keys = S, values = C)
```

To find the cdf $\mathsf{P}\{X \le x\}$ from the prepared map M, it now suffices to find how many of its keys are no larger than x. SEARCHSORTED, Alg. 1.5.27, returns this number. We explain how it works at the end of the main text.

The returned value is zero when $x < s_1$. Otherwise it is the largest index k for which $s_k \le x$, and the cdf is simply $M(s_k)$.

The function CDF first prepares the internal cdf map M. It then returns an inner function F. This function retains access to M. Its keys are the support in increasing order because PREPARECDF inserted them in that order. Each call to F therefore only has to run SEARCHSORTED on these keys and retrieve the corresponding value from M.

Algorithm 1.5.11 Construct a cdf that uses search sorted.

```
Input: pmf map p_X.
Output: cdf function F.
function CDF(p_X)
    M ← PREPARECDF(p_X)
    S ← KEYS(M)                          ▷ The keys of M are already sorted.
    function F(x)
        k ← SEARCHSORTED(S, x)
        if k = 0 then
            return 0
        return M(s_k)
    return F
```

The survivor function $\mathsf{P}\{X > x\} = 1 - \mathsf{P}\{X \le x\} = 1 - F(x)$ follows immediately. By binding $\mathrm{CDF}(p_X)$ to F once, we avoid preparing the cdf again at each evaluation of the survivor function.

Algorithm 1.5.12 Compute the survivor function from the cdf.

```
Input: pmf map p_X.
Output: survivor function.
function SURVIVOR(p_X)
    F ← CDF(p_X)
    return λ: x ↦ 1 − F(x)
```

WE CAN NOW ALSO generate simulated realizations of X. Let $F \leftarrow \mathrm{CDF}(p_X)$ and $U \sim \mathrm{Unif}([0,1])$. The inverse-cdf method returns the smallest support value s_k for which $F(s_k) \ge U$. This has the required distribution because

$$\mathsf{P}\left\{F^{-1}(U) \le x\right\} = \mathsf{P}\{U \le F(x)\} = F(x).$$

The function SIMULATE applies SEARCHSORTED to the cumulative probabilities. When $U = c_k$, it returns s_k; otherwise it returns the next support value s_{k+1}. The special case $k = 0$ returns the first support value.

Algorithm 1.5.13 Generate N simulated realizations of a discrete random variable.

Input: pmf map p_X, number of draws N.
Output: N simulated values.
function SIMULATE(p_X, $N = 1$)
    $M \leftarrow$ PREPARECDF(p_X)
    $S \leftarrow$ KEYS(M) ▷ *The keys of M are already sorted.*
    $C \leftarrow$ APPLY(S, M)
    $R \leftarrow []$
    for $j = 1, \ldots, N$ **do**
        Generate $u \sim \text{Unif}([0, 1])$.
        $k \leftarrow$ SEARCHSORTED(C, u)
        if $k = 0$ **then**
            $R \leftarrow$ APPEND(R, s_1)
        else if $c_k = u$ **then**
            $R \leftarrow$ APPEND(R, s_k)
        else
            $R \leftarrow$ APPEND(R, s_{k+1})
    return R

THE TWO REMAINING patterns are binary and unary functions. Supporting binary functions works the same as addition in (1.5.1). The function BINARYTRANSFORM replaces the particular function $x + y$ by a general binary function $f(x, y)$. When several pairs produce the same value z, their probability masses accumulate in the pmf map p_Z.

Algorithm 1.5.14 Construct the pmf of $f(X, Y)$ where X, Y are independent rvs.

Input: pmf maps p_X, p_Y of independent rvs, binary function f.
Output: pmf map of $f(X, Y)$.
function BINARYTRANSFORM(p_X, p_Y, f)
    $p_Z \leftarrow$ MAP()
    for $x \in$ KEYS(p_X) **do**
        for $y \in$ KEYS(p_Y) **do**
            $p_Z(f(x, y)) \xleftarrow{+} p_X(x) \cdot p_Y(y)$
    return MAKEPMF(p_Z)

For a unary function f, let $Z = f(X)$. Its pmf satisfies

$$p_Z(z) = \mathsf{P}\{Z = z\} = \sum_{x \in \text{KEYS}(p_X)} p_X(x)\, \mathbb{1}\{f(x) = z\}. \tag{1.5.2}$$

The function UNARYTRANSFORM implements this rule by applying f to each possible value of X. Different values of X can produce the same value $z = f(x)$, so their probability masses accumulate in the pmf map p_Z.

Algorithm 1.5.15 Construct the pmf of $f(X)$.

```
Input: pmf map p_X, function f.
Output: pmf map of f(X).
function UnaryTransform(p_X, f)
    p_Z ← Map()
    for x ∈ Keys(p_X) do
        p_Z(f(x)) +← p_X(x)
    return MakePMF(p_Z)
```

With these two transform functions, we can now define many useful functions of rvs.

A constant rv takes one value (c) with probability one. Its pmf is useful as the starting point for several of the functions below.

Algorithm 1.5.16 Construct the pmf of a constant rv.

```
Input: value c.
Output: pmf map concentrated at c.
function Constant(c)
    p_c ← Map(), p_c(c) ← 1
    return p_c
```

Addition is the special case of BinaryTransform obtained by taking $f(x, y) = x + y$.

Algorithm 1.5.17 Add two independent rvs.

```
Input: pmf maps p_X, p_Y of independent rvs.
Output: pmf map of X + Y.
function Add(p_X, p_Y)
    return BinaryTransform(p_X, p_Y, λ: (x, y) ↦ x + y)
```

The N-fold convolution of p_X is the pmf of $X_1 + \cdots + X_N$, where the X_k are independent copies of X. For $N = 0$, the sum is zero. We use Add as the accumulator function and Constant(0) as its start value in Alg. 1.2.1.

Algorithm 1.5.18 Construct the pmf of the sum of N independent copies of an rv.

```
Input: pmf map p_X, number of copies N.
Output: pmf map of the sum of N independent copies of X.
function Convolution(p_X, N)
    return Accumulate([p_X : k = 1, ..., N], Add, Constant(0))
```

The unary transform allows us to define $-X$ as an rv: unary minus reflects all outcomes about zero.

Algorithm 1.5.19 Negate an rv.

Input: pmf map p_X.
Output: pmf map of $-X$.
function NEGATE(p_X)
    return UNARYTRANSFORM$\left(p_X, \lambda\colon x \mapsto -x\right)$

We can now implement subtraction as addition of the negative.

Algorithm 1.5.20 Subtract two independent rvs.

Input: pmf maps p_X, p_Y of independent rvs.
Output: pmf map of $X - Y$.
function SUBTRACT(p_X, p_Y)
    return ADD$\left(p_X, \text{NEGATE}\left(p_Y\right)\right)$

The minimum of two independent rvs is another useful binary transform.

Algorithm 1.5.21 Take the minimum of two independent rvs.

Input: pmf maps p_X, p_Y of independent rvs.
Output: pmf map of $\min\{X, Y\}$.
function MINIMUMPMF(p_X, p_Y)
    return BINARYTRANSFORM$\left(p_X, p_Y, \lambda\colon (x, y) \mapsto \min\{x, y\}\right)$

Likewise, we define the maximum of two independent rvs.

Algorithm 1.5.22 Take the maximum of two independent rvs.

Input: pmf maps p_X, p_Y of independent rvs.
Output: pmf map of $\max\{X, Y\}$.
function MAXIMUMPMF(p_X, p_Y)
    return BINARYTRANSFORM$\left(p_X, p_Y, \lambda\colon (x, y) \mapsto \max\{x, y\}\right)$

The positive part satisfies $X^+ = \max\{0, X\}$. We therefore construct a constant rv equal to zero and take its maximum with X.

Algorithm 1.5.23 Take the positive part of an rv.

Input: pmf map p_X.
Output: pmf map of X^+.
function POSITIVEPART(p_X)
    return MAXIMUMPMF$\left(\text{CONSTANT}\left(0\right), p_X\right)$

Likewise, $X^- = \max\{0, -X\} = [-X]^+$, so we first negate X and then take its positive part.

Algorithm 1.5.24 Take the negative part of an rv.

Input: pmf map p_X.
Output: pmf map of X^-.
function NEGATIVEPART(p_X)
    return POSITIVEPART$\left(\text{NEGATE}\left(p_X\right)\right)$

Taking outcomes modulo C is also a unary transform.

Algorithm 1.5.25 Reduce the outcomes of an rv modulo C.

```
Input: pmf map p_X, modulus C.
Output: pmf map of X mod C.
function MODULO(p_X, C)
    return UNARYTRANSFORM(p_X, λ: x ↦ x mod C)
```

WITH THESE FUNCTIONS we can study the content of Th. 1.4.3 numerically. In particular, how quickly do the arrival times on the circle converge to the uniform distribution? Recall that

$$A_k = (A_{k-1} + X_k) \bmod C.$$

We first use ADD, and then use MODULO.

In Alg. 1.5.26 we iteratively compute the arrival-time position $\sum_{i=1}^{n} X_i \bmod C$, where the X_i are mutually independent with $X_i \sim \mathrm{Unif}(\{-1,0,1\})$, and $A_0 = 0$. We track the positions that occur with the smallest and largest probability. Fig. 1.5.1 shows the result.

Algorithm 1.5.26 Track convergence of the arrival-time distribution on a cycle.

```
Input: cycle length C, number of iterations n, and step pmf p_X.
Output: smallest and largest probabilities m_k, M_k.
function ARRIVALCONVERGENCE(C, n, p_X)
    p_{A_0} ← CONSTANT(0)
    m, M ← [], []
    for k = 1, ..., n do
        p_{A_k} ← MODULO(ADD(p_{A_{k-1}}, p_X), C)
        m ← APPEND(m, MINIMUM([p_{A_k}(a) : a = 0, ..., C−1]))
        M ← APPEND(M, MAXIMUM([p_{A_k}(a) : a = 0, ..., C−1]))
    return m, M
```

FINALLY, WE EXPLAIN how SEARCHSORTED works. Given a sorted list A and a number x, it returns the number of entries of A that are less than or equal to x. We first give the algorithm, then provide an example, and finally prove that it returns the stated number.

The algorithm[10] inspects the middle of the current part of the list and then calls itself on either the left or the right part. Each call therefore reduces the part that remains to be searched by about one half, so only about[11] $\log_2 n$ comparisons are needed for a list of length n.

[10] *Like sorting, efficiently searching sorted data is used almost everywhere that computers are used. The ideas behind this algorithm are very important.*

[11] $\log$ *is the natural logarithm,* $\log_2$ *has base 2.*

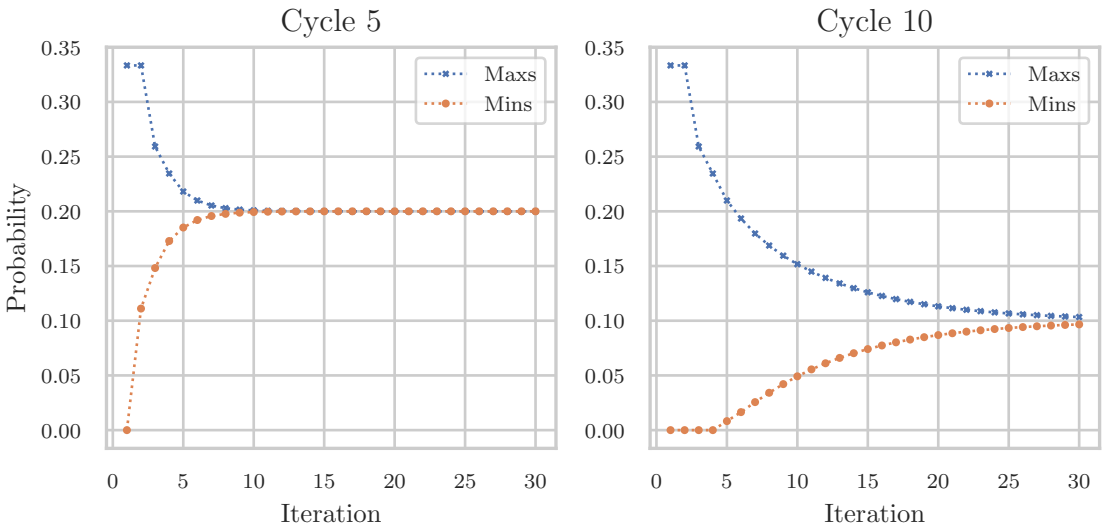


FIG. 1.5.1: Convergence speed of the time between two visits to the hospital. In the left panel the cycle length $C = 5$, in the right $C = 10$. The largest and the smallest probability on the circle converge to $1/C$ (recall Th. 1.4.3). The larger C, the longer it takes for the sequence of visits to converge to uniform inter-arrival times, from the perspective of the hospital. In actual practice, this is not a problem, because patients in the hospital start at arbitrary times, not all at time 0.

Algorithm 1.5.27 Find how many entries of a sorted list are $\le x$.

```
Input: a sorted list A = [a_1, ..., a_n] and a number x.
Output: the number of entries in A that are less than or equal to x.
function SEARCHSORTED(A, x, l = 1, r = n)
    if l > r then
        return r
    m ← ⌊(l + r)/2⌋
    if x < a_m then
        return SEARCHSORTED(A, x, l, m − 1)
    return SEARCHSORTED(A, x, m + 1, r)
```

Consider $A = [1,2,3]$ and $x = 1.1$. Initially, $l = 1$ and $r = 3$, so $m = 2$. Since $x < a_2 = 2$, the next call searches with $l = 1,\ r = 1$. Now $m = 1$. Since $a_1 = 1 \le x$, the next call has $l = 2,\ r = 1$. As $l > r$, it returns $r = 1$: exactly one entry is less than or equal to 1.1.

To prove correctness, we keep track of what each recursive call leaves unchanged.[12] Let k be the number of entries of A that are less than or equal to x.

The first major idea to find a *loop invariant*.[13] For this algorithm, this invariant is $l - 1 \le k \le r$. It[14] holds initially because $l = 1$, $r = n$, and $0 \le k \le n$. If $x < a_m$, sortedness gives $k \le m - 1$, so setting $r = m - 1$ preserves the invariant. Otherwise $k \ge m$, so setting $l = m + 1$ preserves it.[15]

Next, we should show that the algorithm *terminates in a finite number of steps*. Here, each call with $l \le r$ strictly reduces the number of indices still to be searched, so the recursion must terminate.

[12] *You may skip this proof, but I advise against it: it illustrates one of the most important ideas in reasoning about algorithms.*

[13] *A loop invariant is a property that holds initially and remains true after each iteration; here, after each recursive call.*

[14] *Note, the loop invariant is not $l \le k \le m$; the -1 is essential.*

[15] *The key is that with choice, the loop invariant becomes $m \le k \le r$.*

Finally, we need to check what is the *terminal value* of the algorithm. here, the *terminal condition* is $l > r$, and the loop invariant forces $k = r = l - 1$. Thus the returned value r is correct. A bit of thought shows that the cases when no entry or every entry is less than or equal to x are included as terminal values.

TF 1.5.1. Let w be the map with $w(a) = 1$, $w(b) = 2$, and $w(c) = 7$. Apply NORMALIZEMAP$(w, \epsilon = 0.15)$. Claim: the resulting pmf map p satisfies $p(b) = 2/9$.

TF 1.5.2. The function CUMSUM returns the running totals of a list. Claim:

$$\text{CUMSUM}([3, -1, 4, -2]) = [3, 2, 6, 4]\,.$$

TF 1.5.3. Let p_X and p_Y be the pmf maps of independent rvs X and Y. Consider the central part of BINARYTRANSFORM:

```
1: p_Z ← MAP()
2: for x ∈ KEYS(p_X) do
3:     for y ∈ KEYS(p_Y) do
4:         z ← f(x, y)
5:         p_Z(z) +← ???
```

Claim: replacing ??? by $p_X(x)p_Y(y)$ makes p_Z the pmf map of $Z = f(X, Y)$.

TF 1.5.4. Let $A = [a_1, \ldots, a_n]$ be sorted. Consider the recursive function SEARCHSORTED:

```
1: function SEARCHSORTED(A, x, l = 1, r = n)
2:     if l > r then
3:         return r
4:     m ← ⌊(l + r)/2⌋
5:     if x < a_m then
6:         return SEARCHSORTED(A, x, l, m − 1)
7:     return SEARCHSORTED(A, x, ???, r)
```

Claim: at ???, the algorithm should have $m + 1$.

TF 1.5.5. Let the rv X have cumulative distribution function F_X and let $U \sim \text{Unif}(0, 1)$. Claim: $F_X(F_X^{-1}(U)) \sim \text{Unif}(0, 1)$.

TF 1.5.6. Let X and Y be two independent rvs with support $\{0, 1, 2, \ldots\}$ and pmf maps p_X and p_Y, respectively. Claim:

$$\mathsf{P}\{X - Y \le k\} = \sum_{i=0}^{\infty} \sum_{i=0}^{\infty} \mathbb{1}\{i - j \le k\}\, p_X(i) p_Y(j).$$

TF 1.5.7. The function TOFRACTION(x, d) first rounds x to d decimal places and then reduces the resulting fraction. Claim: TOFRACTION$(1/3, 2) = 1/3$.

2

CONSTRUCTION OF SIMPLE DISCRETE-TIME STOCHASTIC PROCESSES

The first step to analyze a stochastic process[16] is to model and simulate it. In this chapter we discuss how to do this by means of discrete-time models of queueing systems. Here, discrete time means that we chop up time into intervals of equal duration, and study how the system behaves from one period to the next.[17]

Section 2.1 and Section 2.2 show how to model and simulate a case from a mental-health department. Section 2.3 uses simulation to obtain general insight into how queueing systems behave. Section 2.4 provides motivation as to why we concentrate on long-run behavior for queueing systems if we want to use mathematics to analyze such systems. Section 2.5 presents suitable recursions for single-item inventories in discrete time, and demonstrates in particular how to design good rules to control stochastic inventory systems.

[16] *In our case, queueing or inventory processes.*

[17] *Such a period can be an hour, or a day; in fact, any amount of time that makes sense in the context of the model. For hospitals, it may be days, for supermarkets it may be 5 minutes.*

2.1 WAITING FOR A PSYCHIATRIST

We start with a case to provide motivation for studying queueing systems. Then we develop a set of fundamental recursions to simulate the case and evaluate the efficacy of several policies to improve the system.

AT A MENTAL-HEALTH department, five psychiatrists perform intakes[1] to determine the best treatment process once patients enter the system. There are complaints about the time patients have to wait for the intake; the desired waiting time is around two weeks, but the realized waiting time is sometimes more than three months. This is unacceptably long, but what to do about it?

[1] *Interviews with new patients.*

The five psychiatrists put forward various suggestions to reduce waiting times.

1. Give all psychiatrists the same weekly capacity to perform intakes. Motivation: Currently, one psychiatrist performs 9 intakes per week, one performs 3, and the other three psychiatrists only perform 1. This is not a problem during weeks when all psychiatrists are present; however, psychiatrists take vacations, visit conferences, etc. Hence, if the psychiatrist with the highest weekly number of intakes goes on leave, the total intake capacity is significantly reduced.

2. Synchronize holidays.[2] Motivation: to reduce the variation in the service capacity, the psychiatrists plan their holidays consecutively. However, perhaps it is better to work at full capacity or not at all.

[2] *Later, with the models of Chapter 6, we will see that this will not work, in fact, synchronization might be detrimental.*

After consultation, another suggestion was proposed:

3. Control the intake capacity as a function of the waiting time. Motivation: in analogy with supermarkets, increase (decrease)

capacity when the queue is long (short). Observe that people often object to such policies because they believe that they have to do more work. However, this reasoning is incorrect when the arrival rate does not depend on the queue length. To see this, suppose that 1000 patients per year need treatment.This number does not depend on whether patients spend time in the queue or not.

We next develop a method to simulate the behavior of the system over time so that we can evaluate the effect of the above suggestions. We use simulation because mathematical analysis is intractable.[3]

[3] *In case you doubt this, try to analyze transient multi-server queueing systems with vacations, of which this is an example.*

LET US START with analyzing the two suggestions of the psychiatrists by making a queueing model. Often, the easiest way to construct queueing processes is to divide time into periods of equal length, and develop recursions that capture the behavior of the queue from one period to the next.[4]

[4] *Using fixed-length periods has the advantage that we do not have to specify inter-arrival times or service times of individual customers. In this case, only the number of arrivals in a period and the number of potential services are relevant.*

Let us define:

- a_k = the number of jobs that arrive *in* period k,
- c_k = the capacity, i.e., the maximal number of jobs that can be served, *during* period k,
- d_k = the number of jobs that depart just before the *end* of period k,
- L_k = the *system size*, i.e., the number of jobs in the system at the *end* of period k.

In what follows, we also call a_k the *size of the batch* arriving in period k. Note that the definition of a_k is 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 in period k; in the latter case, they *cannot* be served in period k.

Since L_{k-1} is the length of the system at the *end* of period $k-1$, it must also be the number of customers at the *start* of period k. Assuming that jobs arriving in period k cannot be served in period k, the number of customers d_k that depart in period k and the number that remain in the system L_k at the end of period k are given by the recursions

$$d_k = \min\{L_{k-1}, c_k\}, \qquad L_k = L_{k-1} - d_k + a_k. \tag{2.1.1}$$

Thus, if we are given L_0 and arrivals $\{a_k\}_{k=1}^n$ and capacities $\{c_k\}_{k=1}^n$, we can obtain $\{d_k\}_{k=1}^n$ and $\{L_k\}_{k=0}^n$.

Typically, we can use company data to model the arrival process $\{a_k\}$ and the capacity $\{c_k\}$, and feed these data into a computer to perform recursions (2.1.1). If insufficient data is available, we build a probability model for $\{a_k\}$ and $\{c_k\}$ and generate random deviates with these distributions.

Another interesting recursion is this:[5]

$$L_k = [L_{k-1} + a_k - c_k]^+. \tag{2.1.2}$$

[5] $x^+ := \max\{x, 0\}$, $x^- := [-x]^+$.

[2.1.9] asks you to prove that this generates a system in which jobs can be served[6] in the period they arrive.

[6] *If there is sufficient capacity.*

CONTINUING WITH THE case of the five psychiatrists, we can now analyze how the first and second suggestion would affect the waiting times.

As a first step in the analysis, we model the arrival process of patients as a Poisson process. The duration of a period is taken to be a week. The average number of arrivals per period, based on company data, is slightly less than 12 per week; in the simulation we set the arrival rate $\lambda = 11.8$ per week.

WE MODEL THE capacity in the form of a matrix such that row i corresponds to the weekly capacity of psychiatrist i:

$$\begin{pmatrix} 1 & 1 & 1 & \dots \\ 1 & 1 & 1 & \dots \\ 1 & 1 & 1 & \dots \\ 3 & 3 & 3 & \dots \\ 9 & 9 & 9 & \dots \end{pmatrix}. \tag{2.1.3}$$

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

With such a matrix, it is easy to construct other capacity schemes. A more balanced scheme would be like this:

$$\begin{pmatrix} 2 & 2 & 2 & \dots \\ 2 & 2 & 2 & \dots \\ 3 & 3 & 3 & \dots \\ 4 & 4 & 4 & \dots \\ 4 & 4 & 4 & \dots \end{pmatrix}. \tag{2.1.4}$$

We include the effects of holidays by setting the capacity of one or some psychiatrists to 0 in a certain week. Let us assume that every week just one psychiatrist is on leave such that their holiday schemes rotate. To achieve this, we set the entries of the matrix like this:

$$\begin{pmatrix} 0 & 2 & 2 & 2 & 2 & 0 & \dots \\ 2 & 0 & 2 & 2 & 2 & 2 & \dots \\ 3 & 3 & 0 & 3 & 3 & 3 & \dots \\ 4 & 4 & 4 & 0 & 4 & 4 & \dots \\ 4 & 4 & 4 & 4 & 0 & 4 & \dots \end{pmatrix}. \tag{2.1.5}$$

Clearly, under this scheme, the average capacity is 12 patients per week. There is another simple holiday scheme: when all psychiatrists take holiday in the same week:

$$\begin{pmatrix} 0 & 2 & 2 & 2 & 2 & 0 & 2 & \dots \\ 0 & 2 & 2 & 2 & 2 & 0 & 2 & \dots \\ 0 & 3 & 3 & 3 & 3 & 0 & 3 & \dots \\ 0 & 4 & 4 & 4 & 4 & 0 & 4 & \dots \\ 0 & 4 & 4 & 4 & 4 & 0 & 4 & \dots \end{pmatrix}. \tag{2.1.6}$$

Of course, random holiday schemes are more likely, but these two are perhaps the simplest that result in the same average capacity.

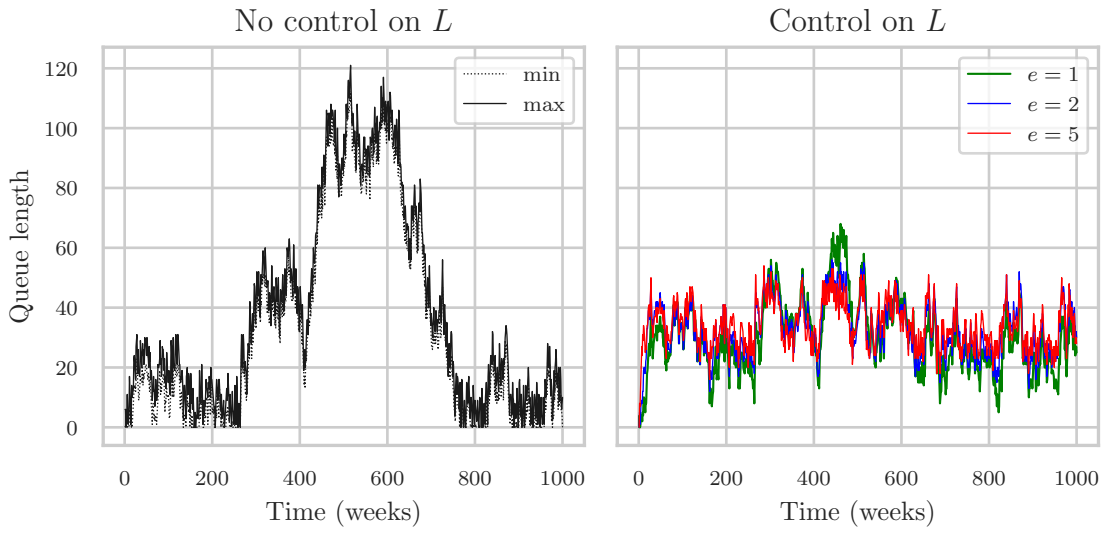


FIG. 2.1.1: The left panel shows the maximum and the minimum queue length under the different holiday and balancing policies on the queue length process $\{L_k\}$. The right panel shows the queue length under the control rule (2.1.7). We see that the fluctuations in queue length are largest when modulating capacity with $e = 1$. After the initial transient, the queues stay positive, so intake capacity is not wasted, but the queues do not make large excursions.

By also applying the same holiday plans to the unbalanced case of (2.1.3), we obtain four different capacity schemes: unbalanced versus balanced capacity, and spread out holidays versus simultaneous holidays. We can now use the recursions (2.1.8) to simulate the queue length process.

The results are shown in Fig. 2.1.1. We plot, for each period, the largest and smallest queue that occurred under all four capacity plans that result from following the first and second suggestions of the psychiatrists. The graphs show that these suggestions hardly affect the behavior of the queue-length process.

This desk study brings us to an important observation: Suggestions 1 and 2 are unlikely to have a positive impact on waiting times. So, rather than setting up an experiment in real life, we should look for other rules.

NOW WE CONSIDER Suggestion 3, which consists of doing more intakes when it is busy and fewer when it is quiet. A simple rule is to let the capacity for week k depend on the queue length at the end of the previous week, for instance,

$$c_k = \begin{cases} 12+e, & \text{if } L_{k-1} \geq 48, \\ 12, & \text{if } 24 < L_{k-1} < 48, \\ 12-e, & \text{if } L_{k-1} \leq 24. \end{cases} \tag{2.1.7}$$

We can take $e = 1$ or 2, or perhaps a larger number; the larger e, the larger the control we exercise.[7] We can, of course, also adapt the thresholds 48 and 24.

Consider three different control levels, $e = 1$, $e = 2$, and $e = 5$; when $e = 1$, one psychiatrist does one additional intake, when $e = 5$, all do an

[7] *Why would it not make much sense to set the lower threshold, here 24, smaller than 12? Hint, do we use the capacity of 12 patients efficiently?*

extra intake in that week. The right panel of Fig. 2.1.1 shows a striking result. The queue does not explode anymore; even taking $e = 1$ has a large effect.

In conclusion, this simulation experiment shows that changing holiday plans or spreading the work more evenly over multiple servers, i.e., psychiatrists, may not significantly affect queueing behavior. However, controlling the service rate as a function of the queue length can dramatically improve the situation.

WE NEXT DISCUSS some general points that need further attention. In (2.1.1) we assume that the jobs that arrive in period k cannot be served in period k. If the situation is such that jobs *can* be served in the same period as they arrive, then (2.1.1) should be changed to[8]

[8] *[2.1.9].*

$$d_k = \min\{L_{k-1} + a_k, c_k\}. \tag{2.1.8}$$

Which of (2.1.1) or (2.1.8) to choose depends on what we need to model; in general, no rule is 'best', and what is 'good' depends essentially on the context. For instance, if we want to be 'on the safe side', then it is perhaps best to use (2.1.1) because with this rule, we overestimate the queue lengths, while with (2.1.8) we underestimate the queue lengths.

Note next that in the computation of d_k we make a fundamental modeling choice: If there are jobs in the system and the server capacity $c_k > 0$, the server will serve jobs. However, there is no formal need to serve jobs, even when there is service capacity available. A simple reason not to serve jobs is when it is too expensive; for example, if there is very little demand for a flight on some day, then the flight can be canceled. In the models we consider, we will *not* allow jobs to wait when there is capacity available; in other words, our service processes are assumed to be *work-conserving*.

The above recursions only construct $\{d_k\}$ and $\{L_k\}$. If we also need information about the *sojourn times*, that is, the time jobs spend in the system, it is necessary to specify the *service discipline*, i.e., a scheduling rule that decides on the order in which jobs in queue are taken into service. In this book, we henceforth assume that jobs move to the server in the order in which they arrive. This is known as *First-In-First-Out (FIFO)*.[9] There are many other scheduling rules, such as Last-In-First-Out; we do not discuss these here.

[9] First-Come, First-Served (FCFS) *is also often used.*

It is quite remarkable that with (2.1.1) the computation of the system length process $\{L_k\}$ is very simple, but it is much harder to compute the sojourn time J_k for the jobs arriving in period k. For a job at a specified position in the batch, Alg. 2.1.1 computes the exact sojourn time. The first and last positions give bounds $J_k^- \le J_k \le J_k^+$ for any job in the batch. We use the algorithm like so:

$$J_k^- \leftarrow \text{SOJOURNTIME}\,(L_{k-1}, [c_k, c_{k+1}, \ldots], 1)\,,$$
$$J_k^+ \leftarrow \text{SOJOURNTIME}\,(L_{k-1}, [c_k, c_{k+1}, \ldots], a_k)\,.$$

To see how this works, suppose that $L_{k-1} = 20$, $a_k = 6$, and $c_k = c = 2$ for all k. During period k, two of the 20 existing jobs depart, leaving 18 ahead

of the arriving batch. Under FIFO, these 18 jobs occupy the next nine periods, so the first two jobs of the batch depart in period $k+10$, the next two in period $k+11$, and the last two in period $k+12$. Measured from the end of the arrival period, their sojourn times are therefore 10, 11, and 12 periods, respectively, giving $J_k^- = 10$ and $J_k^+ = 12$.

Algorithm 2.1.1 Compute the sojourn time of the jth job in an arriving batch, when arrivals cannot be served in their arrival period.

```
1: Input: queue length L, a list c of non-negative integer capacities
   indexed from 1 starting with the arrival period, and position j ≥ 1 of
   the jobs in the arriving batch.
2: Output: sojourn time, measured in periods from the end of the
   arrival period.
3: function SojournTime(L, c, j)
4:     remaining ← max{L − c_1, 0} + j      ▷ Jobs that remain to be served.
5:     m ← 0                                ▷ Track period.
6:     while remaining > 0 do
7:         m +← 1
8:         remaining −← c_{m+1}
9:     return m
```

We note that there is a difference between *waiting time* W and sojourn time J. The former is the time spent by jobs in the queue, the latter is the time in the system, which is the waiting time plus the time at the server. However, in (2.1.1) and (2.1.8) jobs are not really in service; we only count the jobs in the system at the end of a period. Thus, in this model, the number of jobs in the system coincides with the number in queue.

Finally, once the simulation is finished, we compute some performance measures to analyze how the (simulation of the) queueing system behaved. Besides making graphs, we can be interested in the arrival rate, the capacity[10] and mean number in the system,

$$\hat{\lambda} = \frac{1}{n}\sum_{k=1}^{n} a_k \qquad \hat{\mu} = \frac{1}{n}\sum_{k=1}^{n} c_k \qquad L = \frac{1}{n}\sum_{k=1}^{n} L_k.$$

[10] *Also known as the service rate.*

Similarly, we can take the averages over d_k, J_k^- and J_k^+. The variances of these (simulated) rvs can also be of interest. Perhaps the most important measure to check is the *load*, which is the ratio of the rate at which work arrives to the available service capacity. For instance, if $\lambda = 5$ jobs arrive per hour, and each job requires 20 minutes of work (on average), then work arrives at a rate of $5 \times 20/60 = 5/3$ server-hours per hour. Since one server can do only 1 hour of work per hour, we need at least two servers to deal with this number of jobs. With two servers, the load is $(5/3)/2 = 5/6$. For a simulation in discrete time, we define the load as

$$\hat{\rho} = \frac{\hat{\lambda}}{\hat{\mu}} = \frac{\sum_{k=1}^{n} a_k}{\sum_{k=1}^{n} c_k}. \tag{2.1.9}$$

We expect that when $\hat{\rho} > 1$, long queues build up.[11]

[11] *In Section 2.3 we discuss the concept of load more formally.*

IN GENERAL, WITH recursions like (2.1.1) we can carry out numerous what-if analyses.

For example, a hospital considers buying a second MRI scanner because the current one is saturated: it is used from 8 am to 6 pm.[12] If the hospital can identify some additional capacity, it might postpone the purchase of an additional MRI scanner for a few years. Suppose that a percentage of the staff is willing to work from 8 pm to 11 pm, say, and that 30% of the patients[13] are ready to come to the hospital for a scan in the evening, then the capacity would increase by about 3/10 = 30%. To see whether this idea is interesting, we can use (2.1.1) to let the computer make a graph of the influence on L.

[12] *That is, 10 h per working day.*

[13] *Perhaps some staff and patients even prefer to work/have scans in the evening.*

Recursions are not just useful for modeling queueing systems. For instance, a common model in population dynamics has this form

$$N_{t+1} = N_t + B_t + I_t - D_t - E_t,$$

where N_t is the population size at year t, B_t the number of births, I_t the immigration, D_t the deaths, and E_t the emigration.

Yet another use of recursion is to understand how the reflection of infrared light by carbon dioxide affects the temperature of the earth. The atmosphere is modeled as a sequence of layers so that, per layer, the temperature, air density, and chemical composition are considered constant. These properties determine how much infrared light, which is initially reflected by the surface of the earth, reaches outer space. In each of the layers, we need to specify what fraction of infrared photons are transmitted to a higher layer, absorbed, or reflected again to a lower level. Thus, this physical process can be modeled as a queueing system in which photons (customers) go from one layer (station) to another layer (station) or are absorbed (leave the network).[14]

[14] `https://github.com/scienceetonnante/RadiativeForcing/blob/main/RadiativeForcing.py`.

FORMULATING RECURSIONS IS the key step in making useful models; you should practice this a lot.

TF 2.1.1. In a discrete-time queueing system, when job arrivals in period k cannot be served in period k, then $d_k = \min\{L_{k-1}, c_k\}$, $L_k = L_{k-1} - d_k + a_k$.

TF 2.1.2. We have a queueing system in discrete time. Take $c_k = c\,\mathbb{1}\{L_{k-1} > \alpha\}$ as the service capacity in period k, with $\alpha > 0$. Claim: if $c < \alpha$ and $L_0 > 0$, then $L_k > 0$ for all k.

TF 2.1.3. A single machine serves two queues so that jobs in the first queue get priority over jobs in the other queue. Jobs cannot be served in the period they arrive. Claim: these recursions model this situation correctly:

$$\begin{aligned} d_k^1 &= \min\{L_{k-1}^1, c_k\}, & L_k^1 &= L_{k-1}^1 - d_k^1 + a_k^1, \\ c_k^2 &= \min\{L_{k-1}^2, c_k - d_k^1\}, & & \\ d_k^2 &= \min\{L_{k-1}^2, c_k^2\}, & L_k^2 &= L_{k-1}^2 - d_k^2 + a_k^2. \end{aligned}$$

TF 2.1.4. Consider a single server that serves two parallel queues. Queue i has minimal guaranteed service capacity r^i each period, such that $c_k \geq r^1 + r^2$. Extra capacity beyond the reserved capacity is given to queue 1 with priority. Arriving jobs cannot be served in the period they arrive. (An example is a psychiatrist who reserves capacity for different patient groups.) Claim: these recursions model this situation correctly:

$$\begin{aligned} c_k^2 &= \min\{L_{k-1}^2, r^2\}, \\ d_k^1 &= \min\{L_{k-1}^1, c_k - c_k^2\}, & L_k^1 &= L_{k-1}^1 - d_k^1 + a_k^1, \\ d_k^2 &= \min\{L_{k-1}^2, c_k - d_k^1\}, & L_k^2 &= L_{k-1}^2 - d_k^2 + a_k^2. \end{aligned}$$

TF 2.1.5. Consider the control rule in (2.1.7):

$$c_k = \begin{cases} 12 + e, & \text{if } L_{k-1} \geq 48, \\ 12, & \text{if } 24 < L_{k-1} < 48, \\ 12 - e, & \text{if } L_{k-1} \leq 24. \end{cases}$$

Claim: because the upward and downward adjustments both have size e, the average capacity over n periods equals 12.

Ex 2.1.6. *Fast lanes at De Oosterpoort.* The theater *De Oosterpoort* in Groningen wants to increase customer satisfaction during event breaks. In particular, management would like to ensure that visitors receive their drinks before the break ends. At the same time, they aim to increase total sales.

One proposed intervention is the introduction of *fast lanes*. Customers in the regular lane have a mean service time of $\mathsf{E}[S] = 2$ minutes. Customers in the fast lane order from a reduced menu, and their mean service time is $\mathsf{E}[S] - 0.5 = 1.5$ minutes. A fraction α of the customers uses the fast lane, and a fraction β of the staff is assigned exclusively to it; the other customers and staff use the regular lane. Before implementing this change, management would like to simulate the system to understand for which values of α and β fast lanes should be installed.

Consider a discrete-time queueing model with periods of one minute. Let a_k be the total number of customers arriving in minute k, and c_k the number of customers the staff can serve in minute k if all staff work at the regular service rate $1/\mathsf{E}[S]$. Customers cannot be served in the minute in which they arrive. Give the recursions for the departures d_k^r, d_k^f and the queue lengths L_k^r, L_k^f of the regular lane and the fast lane.

For real-life situations the recursions can become quite intricate.[15]

Ex 2.1.7. A nurse takes blood samples in two departments of a hospital. It takes time to walk from one location to another. The nurse serves patients at location A until its queue length falls to a threshold h_A, then walks to location B. At location B, the nurse serves patients until its queue length falls to a threshold h_B, then walks back to A. Walking from one location to the other takes one period, during which the nurse cannot serve patients. Taking $h_A = h_B = 0$ corresponds to serving all waiting patients before switching. For the graph in Fig. 2.1.2, we use $h_A = 40$ and $h_B = 10$.

[15] *This example is a somewhat simplified case from a bachelor thesis.*

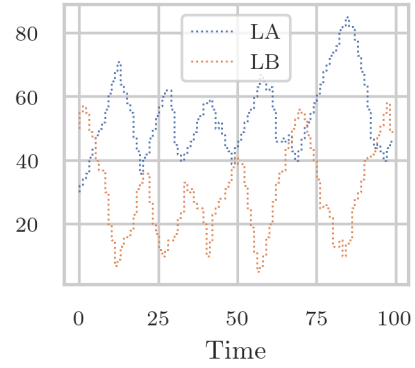


Fig. 2.1.2: A nurse walking between two locations.

Let a_k^i, c_k^i, d_k^i, and L_k^i denote the arrivals, capacity, departures, and queue length at location $i \in \{A, B\}$ in period k. Patients arriving in period k can be served in that same period. Give the recursions that describe the nurse's location and the two queue lengths.

Ex 2.1.8. Each week a civil court receives a random number of new cases. After a case is filed, it must first be processed by a clerk, then reviewed by a judge, and then handled once more by a clerk for the final administrative steps. All workloads are measured in hours. Every newly filed case creates 5 hours of initial clerk work, 3 hours of judge work, and 2 hours of final clerk work. If there is work at both queues of the clerks, they give priority to new cases. The clerks together have 80 hours available per week, and the judge has 34 hours. During week k, a server can only work on the backlog present at the end of week $k-1$. Newly filed cases and work passed downstream during week k become available for service in week $k+1$.

Let a_k be the number of new cases filed in week k. Formulate the weekly recursion relations for all three queues of cases.

Ex 2.1.9. Prove that (2.1.2) produces a system in which jobs can be served in the period they arrive.

2.2 SIMULATING THE PSYCHIATRISTS CASE

In the previous section, we showed how to analyze a case with five psychiatrists acting as five parallel servers processing a queue of patients waiting for an intake. Moreover, we proposed a good rule to control the queue length. In this section, we discuss the algorithms used to produce Fig. 2.1.1.

Before accounting for holidays, the weekly capacities of the five psychiatrists under the unbalanced and balanced plans are, respectively,

$$b^u = (1,1,1,3,9), \qquad b^b = (2,2,3,4,4).$$

To make the case with unbalanced capacity and spread holidays, we take

$$c_{i,k}^{\text{spread}} = b_i^u \mathbb{1}\{i \neq 1 + ((k-1) \bmod 5)\}.$$

For the case with unbalanced capacity and synchronized holidays, we take

$$c_{i,k}^{\text{sync}} = b_i^u \mathbb{1}\{(k-1) \bmod 5 \neq 0\}.$$

The corresponding balanced schemes are obtained by replacing b^u in the two expressions above by b^b. In what follows, let $c_{i,k}$ denote the capacity of psychiatrist i in week k for any one of these four scenarios.

We simulate $n = 1000$ weeks, with independent weekly arrivals $a_k \sim \text{Pois}(11.8)$. For each capacity scenario, the initial queue length is $L_0 = 0$, and the total capacity in week k is $c_k = \sum_{i=1}^5 c_{i,k}$. We use the same generated arrival sequence for all four scenarios. Combining the balanced and unbalanced plans with the spread and synchronized

Algorithm 2.2.1 Compute the queue lengths.

Input: arrivals $(a_1, \ldots, a_n)$, capacities $(c_1, \ldots, c_n)$, and initial queue length L_0.
Output: the list L of queue lengths after each week.
$L \leftarrow []$
for $k = 1, \ldots, n$ **do**
    $d_k \leftarrow \min\{L_{k-1} + a_k, c_k\}$
    $L_k \leftarrow L_{k-1} + a_k - d_k$
    $L \leftarrow \text{APPEND}(L, L_k)$
return L

holiday patterns gives four capacity scenarios. Applying Alg. 2.2.1 to each scenario gives four queue-length trajectories. Their pointwise minimum and maximum are the two curves in the left panel of Fig. 2.1.1.

For the controlled system in the right panel of Fig. 2.1.1, we use the capacity rule in (2.1.7). Alg. 2.2.2 implements the resulting simulation. We apply the control rule separately with $e = 1$, $e = 2$, and $e = 5$. The resulting trajectories are shown in the right panel of Fig. 2.1.1.

Algorithm 2.2.2 Simulate the queue with state-dependent capacity.

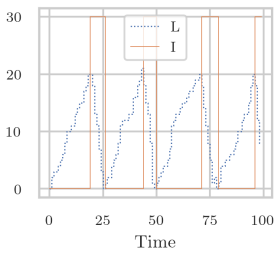
Input: arrivals $(a_1, \ldots, a_n)$, control size e, and initial queue length L_0.
Output: the list L of queue lengths after each week.
$L \leftarrow []$
for $k = 1, \ldots, n$ **do**
    if $L_{k-1} \geq 48$ **then**
        $c_k \leftarrow 12 + e$
    else if $L_{k-1} \leq 24$ **then**
        $c_k \leftarrow 12 - e$
    else
        $c_k \leftarrow 12$
    $d_k \leftarrow \min\{L_{k-1} + a_k, c_k\}$
    $L_k \leftarrow L_{k-1} + a_k - d_k$
    $L \leftarrow \text{APPEND}(L, L_k)$
return L

TF 2.2.1. A server with total capacity c_k in period k serves two parallel queues. Queue i has reserved capacity r^i, where $r^1 + r^2 \leq c_k$, and unused reserved capacity cannot be assigned to the other queue. Capacity beyond the reservations is assigned to queue 1 with priority. Jobs arriving in period k cannot be served until period $k+1$. Claim: The system is described by

$$d_k^1 = \min\{L_{k-1}^1, c_k - r^2\}, \qquad c_k^2 = c_k - d_k^1,$$
$$d_k^2 = \min\{L_{k-1}^2, c_k^2\}, \qquad L_k^i = L_{k-1}^i + a_k^i - d_k^i, \qquad i \in \{1,2\}.$$

In the queueing systems we have analyzed up to now, the server is always present to start serving jobs at the moment they arrive.[1] However, in cases in which there is a significant cost associated with changing the server from idle to busy this condition is typically not satisfied. To

[1] *A queue under an N-policy; the curve labeled I shows* $30I_k$.



reduce the average cost, we can use an N-policy, which works as follows.[2] As soon as the system becomes empty (and the server idle), the server switches off. Then it waits until at least N jobs have arrived, and then it switches on. The server processes jobs until the system is empty again, then it switches off, and remains idle until N new jobs have arrived, and so on, cycle after cycle. Note that under an N-policy, even though the load remains the same, the server has longer busy and idle times.

[2] *N-policies are used to decide when to heat up a large oven.*

Suppose it costs K Euro to set up the server, independent of the time it has been idle, and each job charges h Euros per unit time while in the system. The problem is to find a switching threshold N^* that minimizes the long-term average cost.

TF 2.2.2. Claim: Under an N-policy, the long-run utilization ρ depends on the threshold N.

TF 2.2.3. Claim: Compared with the same system whose server is always available, an N-policy reduces the average waiting time.

TF 2.2.4. Let I_k indicate whether the server is on in period k, and consider the recursion

$$I_k = \mathbb{1}\{L_{k-1} \geq N\} + I_{k-1}\,\mathbb{1}\{0 < L_{k-1} < N\}.$$

Claim: If $L_{k-1} = 0$ and $I_{k-1} = 1$, then the server switches off, so $I_k = 0$.

TF 2.2.5. Claim: Over periods $1, \ldots, T$, the number of switch-on events is

$$\sum_{k=1}^{T} \mathbb{1}\{I_k > I_{k-1}\}.$$

Ex 2.2.6. Derive recursions to model the situation in which jobs that arrive in period k can be served in that period. If the system length hits N (becomes empty) in period k, the machine switches on (off) in period $k+1$. Assume the machine is off at $k = 0$.



Ex 2.2.7. In the setting of the previous exercise, suppose each switch-on event costs K, each period in which the server is on costs β, and each customer in the system costs h per period. Express the total cost over periods $1, \ldots, T$ in terms of I_k and L_k.

Ex 2.2.8. Consider[3] a production network with two production stations in tandem, that is, the jobs processed at station 1 move at the end of period$k$ to station 2. What are the recursions?

[3] *Tandem networks.*

% #+begin$_{\text{exercise}}$ % Consider[4] a paint mixing machine that produces products for two downstream packaging machines$A$ and$B$, each with its own queue. % In the simplest model, the content of the queue at the mixing machine is proportional to the demands λ^A and λ^B for the packaging machines. % Provide the recursions. % #+begin$_{\text{solution}}$ % At the mixing machine $d_k = \min\{L_k, c_k\}$. Therefore, in this very simple model, $a_k^A = d_k\lambda^A/(\lambda^A + \lambda^B)$, and likewise for the other downstream station. % #+end$_{\text{solution}}$ % #+end$_{\text{exercise}}$

[4] *Splitting streams.*

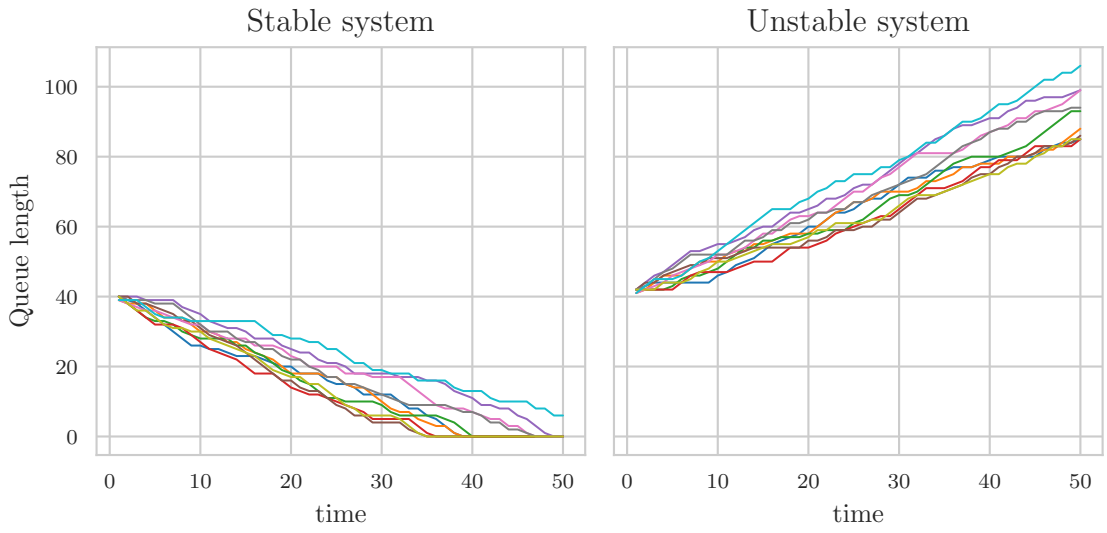


FIG. 2.3.1: Drift of queue-length process. In the left panel, $L_0 = 40$, $\mu = 7$ and $\lambda = 6$; the queue drains at about one job per period. In the right panel, $\mu = 5$, $L_0 = 40$; the queue grows by roughly one job per period. While the queue is large, it has negative drift when $\mu > \lambda$ and positive drift when $\mu < \lambda$.

2.3 GENERAL BEHAVIOR OF QUEUEING SYSTEMS

In this section, we develop a simulation environment for queueing systems to explore their behavior and visualize queue-length dynamics.

THE FIRST ASPECT to address is system stability. Intuitively, when work arrives faster than it can be served, on average, the queue grows; if work arrives slower, the queue remains close to the origin. To illustrate this, Fig. 2.3.1 shows queue-length sample paths for several arrival rates and service capacities.

Let a_k be the number of arrivals and c_k the service capacity in period k. We write

$$\lambda = \mathsf{E}[a_k], \qquad \mu = \mathsf{E}[c_k].$$

We use Alg. 2.2.1 to compute the sample paths; jobs arriving in period k can be served in that same period. We simulate $n = 50$ periods with initial queue length $L_0 = 40$ and mean arrival rate $\lambda = 6$. For each of ten independent sample paths, the arrivals a_k are uniformly distributed on $\{5, 6, 7\}$, while the service capacity is constant at $c_k = 7$, and hence $\mu = 7$. The resulting stable queue-length trajectories are shown in the left panel of Fig. 2.3.1. For the right panel, we repeat the experiment with constant service capacity $c_k = 5$, and hence $\mu = 5$. Since $\mu < \lambda$, the queue has positive drift and is unstable.

HOW ABOUT REDUCING the difference between λ and μ? When μ barely exceeds λ, large fluctuations take longer to die out. To see this in quantitative terms, we run two simulations with smaller values for μ while keeping $\lambda = 6$, and we compute the mean and variance of the number of jobs in the system.

For the first experiment, let $c_k = 6 + B_k$, where $B_k \sim \text{Bern}(1/2)$ independently. Then $\mathsf{E}[c_k] = 6.5$. With $n = 5000$, $L_0 = 0$, and the same arrival distribution as above, the simulation gives a sample mean queue length of 0.45 and a sample variance of 0.64.

For the second experiment, let $c_k = 6 + B_k$, where $B_k \sim \text{Bern}(1/10)$ independently, so $\mu = \mathsf{E}[c_k] = 6.1$. With the same run length, initial queue length, and arrival distribution as above, the sample mean queue length is 4.37 and the sample variance is 25.28. Thus, reducing the excess service capacity from 0.5 to 0.1 substantially increases both the mean and the variance of the queue length.

WOULD VARIABILITY AFFECT the average queue length? We may suspect this because when $a_k = 1$ and $c_k = 1$ for all k and $L_0 = 0$, $L_k = 0$ for all k. If instead $c_i = 5$ when i is a multiple of 5 and 0 otherwise, queues do appear and disappear.[1] Let us use simulation to see how variability can influence the queue-length process. The goal is now to make Fig. 2.3.2.

We first need a way to quantify *relative* variability. Suppose the standard deviation of the service time is 1 minute. If the mean service time is 1 hour, we are inclined to call the service times very regular, while if the mean is just 1 minute, we would call it irregular. To capture this relative effect, we define the *square coefficient of variation (scv)* of the rv X as[2]

$$C_X^2 := \frac{\mathsf{V}[X]}{(\mathsf{E}[X])^2}.$$

Continuing with the simulation, we seek a probability distribution for $\{c_k\}$ with mean μ and scv c^2. A simple way to achieve this is to use a rv X such that $\mathsf{P}\{X = b\} = p$ and $\mathsf{P}\{X = 0\} = 1 - p$, and require that $\mathsf{E}[X] = pb = \mu$ and $C_X^2 = c^2$. Using $\mathsf{V}[X] = \mathsf{E}[X^2] - (\mathsf{E}[X])^2$, we see that $C_X^2 = \mathsf{E}[X^2]/(\mathsf{E}[X])^2 - 1$. Rewriting, we get $\mathsf{E}[X^2] = \mu^2(c^2 + 1)$, but we also know that $\mathsf{E}[X^2] = pb^2$ for this two-point rv. Solving for b and p in terms of μ and c^2 gives $p = 1/(1 + c^2)$ and $b = \mu/p = \mu(1 + c^2)$.

Thus, independent copies of $X = bB$, with $B \sim \text{Bern}(p)$, provide a sequence with mean μ and the required scv. For these experiments, we allow fractional amounts of work: the capacities, departures, and queue lengths need not be integers.

We simulate $n = 5000$ periods with $L_0 = 0$, $\lambda = 6$, and $\mu = 6.5$, and compare scv values 0.5, 1, and 2. In the left panel of Fig. 2.3.2, arrivals are constant, $a_k = 6$, while the service capacities follow the two-point distributions constructed above. In the right panel, both arrivals and service capacities follow such distributions with the same scv. Applying the queue-length algorithm in each case shows that greater relative variability produces larger queue-length fluctuations.

In conclusion, tight capacity or high relative variability leads to large queues. To improve performance, we have basically four options: increase capacity, reduce load (or demand), reduce the scv of the arrivals or of the services.[3]

FINALLY, WE CONSIDER queues in sequence.[4] Such systems are modeled as tandem networks, in which each job visits several stations in a fixed order.

[1] *Observe that the mean service capacity is still one job per period; only the periods in which service is offered changed.*

[2] *The scv is unit-less.*

[3] *Of course, these options are not exclusive; for example, we can try to reduce the variability of the service process and add some capacity at the same time.*

[4] *For example, obtaining grades for a course involves several sequential steps. First, I grade the exams (and handle all changes that might occur during the review). I send a csv file with the grades to the secretary who uploads it. The secretary prints the file for me to sign, after which it is logged by the administration. In abstract terms, there are three stations (me, the secretary, and the back office), and I am involved at least twice, once for the grading and a second time for the signing. More generally, most service systems involve multiple servers visited to complete a service.*

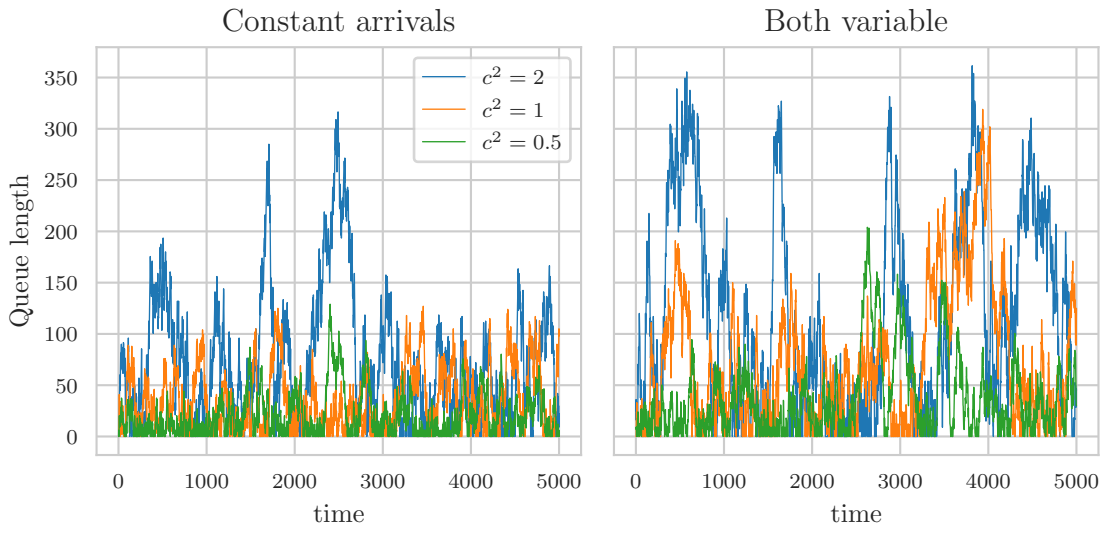


FIG. 2.3.2: The influence of variability on queue length. In the left panel the arrivals are constant, but the number of services per period is variable with scv c^2 as indicated in the legend. In the right panel, we have variability in both arrivals and services. In the second case, the queue length is more variable. (Note that the scales on the y-axes are the same in the left and right panels.)

To gain insight into such systems, we simulate a chain of 4 stations in tandem, which means that jobs arrive at station 1, are then sent to station 2, etc., until they leave the network after station 4. Jobs neither visit the same server twice nor leave the network prematurely.

We simulate $n = 5000$ periods and take $L_0^i = 0$ at every station. The external arrivals at station 1 are independent and uniformly distributed on $\{5,6,7\}$. The service capacities are independent, with

$$c_k^1 \sim \text{Pois}(7), \qquad c_k^2 \sim \text{Pois}(6.1), \qquad c_k^3 \sim \text{Pois}(7.5), \qquad c_k^4 \sim \text{Pois}(7).$$

Thus, station 2 has the smallest mean service capacity.

Let $a_k^1 = a_k$, and for stations $i = 2,3,4$, let the arrivals be the departures from the preceding station. Since these downstream arrivals can be served in the same period, the tandem system satisfies

$$a_k^1 = a_k, \qquad a_k^i = d_k^{i-1},$$
$$d_k^i = \min\left\{L_{k-1}^i + a_k^i, c_k^i\right\}, \qquad L_k^i = L_{k-1}^i + a_k^i - d_k^i.$$

Applying these recursions successively to the four stations produces Fig. 2.3.3. Station 2, which has the smallest mean service capacity, typically has the longest queue. This brings us to an interesting idea: *to improve performance, invest in the bottleneck*. In practice, identifying the bottleneck can be difficult because service capacities are hard to estimate.[5]

TF 2.3.1. A general queueing system is stable when the arrival rate is not higher than the service rate.

TF 2.3.2. If we find the average queue length too large, then we must increase the average service rate to decrease the average queue length.

[5] *The capacity may depend on the person at the station on a certain day. There can also be disturbances due to missing raw materials or machine breakdowns. In short, in practical situations, there are many messy details that disturb the assembly of high-quality data.*

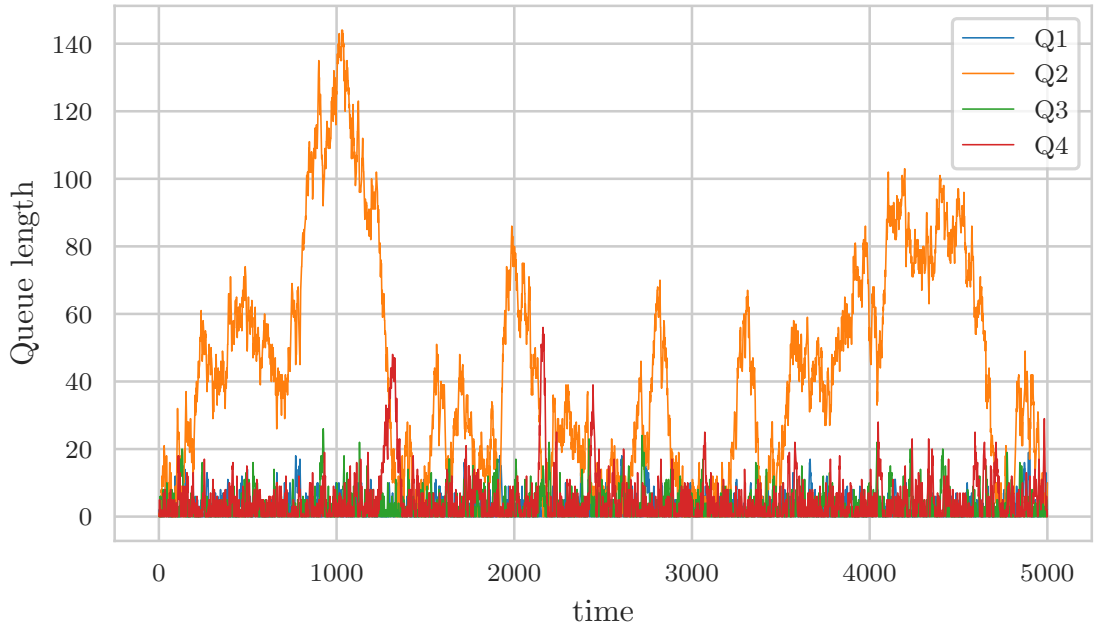


FIG. 2.3.3: A queueing network of 4 stations in tandem.

TF 2.3.3. We consider a discrete-time queueing system with a_k the number of arrivals in period k, and c_k the service capacity. Let $d_k = \min\{L_{k-1}, c_k\}$, $L_k = L_{k-1} - d_k + a_k$. Let all the rvs $\{a_k, c_k\}_{k\geq 1}$ be mutually independent, with $a_k \sim \text{Pois}(2)$, $c_k \sim \text{Pois}(1)$, and $L_0 = 0$. Claim: $\mathsf{P}\{L_{1000} \geq 100\} \leq 1/2$.

TF 2.3.4. Claim: The scv of a rv X is the same as the scv of αX if α is a positive scalar.

TF 2.3.5. The parking meters at the universities are redesigned. A study has shown that the average amount of time individuals need to spend to pay for parking meters has decreased. Claim: As long as the inter-arrival distribution stays the same, the average queue length at the parking meter will be smaller.

TF 2.3.6. Claim: When $X \sim \text{Exp}(\lambda)$ its scv exceeds 1.

TF 2.3.7. Consider a queue with an exponential inter-arrival distribution and a uniform service time distribution on $[0, A]$. Claim: $C_s^2 = \frac{1}{3}$.

Ex 2.3.8. A queueing system is under periodic review, that is, at the end of each period, the queue length is measured. Jobs arriving at period k cannot be served in period k and the system cannot contain more than K jobs; in other words, jobs are blocked when the system is too full. Develop recursions for $\{L_k\}$, the number of jobs lost per period, and the fraction of jobs lost over T periods.

Ex 2.3.9. All items produced by a machine are tested. With probability p an item turns out to be defective and put at the head of the queue. The production time of new and faulty items is equal. Suppose that items

cannot be served in the period in which they arrive, develop a set of recursions to simulate $\{L_k\}$.

Ex 2.3.10. Sometimes items need re-work. Suppose a fraction p of items does not meet the quality requirements after the first pass at a machine, but requires a second pass to repair the problems. Assume that repair jobs need half the service time of new jobs and are served with priority over new jobs. Develop the recursions.

Ex 2.3.11. Consider a network with two production stations in tandem, i.e., all jobs from station 1 move on to station 2, and blocking, i.e., the server at station 1 is not allowed to produce more than the amount M station 2 can maximally contain. Assume that jobs moving from station 1 to station 2 arrive at the end of a period (hence cannot be served in that period), and must fit at the second station before any jobs from station 2 leave. Find recursions.

Ex 2.3.12. Consider the discrete-time model specified by (2.1.1). We assume that a_k is a *batch* of jobs arriving in period k *after* the departures d_k have left. Give an estimator of $\mathsf{P}\{L \le m\}$ for a situation in which the first job of the batch sees $L_{k-1} - d_k$ jobs in the system, the second sees $L_{k-1} - d_k + 1$ jobs, and so on.

2.4 QUEUEING PROCESSES AS REGULATED RANDOM WALKS

In this section, we provide an elegant construction of a queueing process based on a *random walk*.[1] This serves two aims. The first is to show that queueing theory is essentially based on fundamental probabilistic concepts—the random walk—and its close relation to applications in finance, inventory, and insurance. The second goal is to show that it is very hard to characterize the transient behavior of a queueing process analytically. Thus, we will henceforth study queueing systems only in steady state, i.e., their long-run average limits; see Section 4.1.

IN THE CONSTRUCTION of discrete-time queueing processes as set out in Section 2.1, we consider two independent sequences of iid rvs: arrivals $\{a_k\}$ and capacities $\{c_k\}$ per period. The system size then satisfies $L_k = [L_{k-1} + a_k - c_k]^+$. The $[\cdot]^+$ operator complicates computations. Removing it yields a related stochastic process $\{Z_k, k = 0, 1, \ldots\}$ defined by

$$Z_k = Z_{k-1} + a_k - c_k. \tag{2.4.1}$$

In fact, $\{Z_k\}$ is a random walk, since Z_k makes a jump of size $a_k - c_k$, where $\{a_k - c_k\}$ forms a sequence of iid rvs. Observe that $\{Z_k\}$ is 'free'—it can take positive and negative values—while $\{L_k\}$ is restricted to non-negative integers.

It is useful to express $\{L_k\}$ in terms of $\{Z_k\}$. Take $Z_0 = L_0$. From (2.4.1), we have $a_k - c_k = Z_k - Z_{k-1}$, hence

$$L_k = [L_{k-1} + a_k - c_k]^+ = [L_{k-1} + Z_k - Z_{k-1}]^+.$$

[1] *If you don't like math, then you can skip this section.*

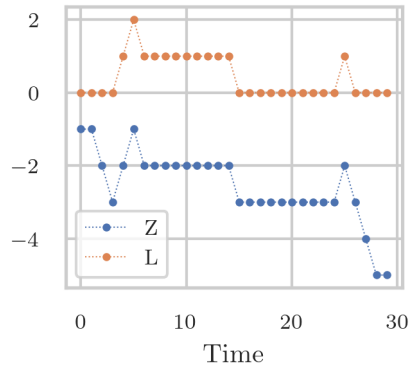


Fig. 2.4.1: An instance of (2.4.1).

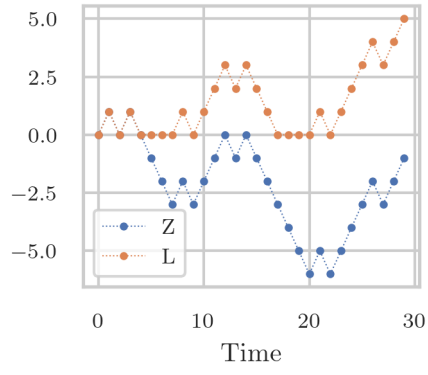


Fig. 2.4.2: An instance of (2.4.3).

[2] *[2.4.1].*

It follows that $L_k - Z_k = \max\{L_{k-1} - Z_{k-1}, -Z_k\}$, and by iterating the recursion,[2]

$$L_k = Z_k - \left(\min_{1\le i\le k} Z_i\right) \wedge 0, \tag{2.4.2}$$

where $a \wedge b := \min\{a, b\}$.

This recursion for L_k yields instructive plots. In Fig. 2.4.1, let $a_k \sim$ Bern (0.3) and $c_k \sim$ Bern (0.4). We clearly see that $L_k = Z_k - \min_{i\le k}\{Z_i\}$. In Fig. 2.4.2, $a_k \sim$ Bern (0.49) so that the random walk satisfies

$$Z_k = Z_{k-1} + 2a_k - 1. \tag{2.4.3}$$

Thus, if $a_k = 1$, the random walk increases by one, while if $a_k = 0$, the random walk decreases by one, so $Z_k \neq Z_{k-1}$ always. This differs slightly from (2.4.1), where $Z_k = Z_{k-1}$, if $a_k = c_k$.

WHAT CAN WE say about the transient (time-dependent) behavior of $\{L_k\}$? To gain insight, suppose $a_k \sim$ Pois (λ) and $c_k \sim$ Pois (μ).

$$Z_k = Z_0 + N_{\lambda,k} - N_{\mu,k}.$$

Writing $Z_0 = m$, one can show[3] that[4]

$$\mathsf{P}_m\{Z_k = n\} = e^{-(\lambda+\mu)k}(\lambda k)^{n-m} \sum_{j=\max\{0,m-n\}}^{\infty} \frac{(\lambda\mu k^2)^j}{j!(n-m+j)!}. \tag{2.4.4}$$

To derive the distribution of L_k from (2.4.2), we need the joint distribution of Z_k and its running minimum, not just the distribution of Z_k. Thus, the calculation above does not yet give $\mathsf{P}\{L_k = n\}$.

Interestingly, $\{L_k\}$ is among the simplest transient queueing system to analyze; most others are far more complex. For this reason, we henceforth focus on the steady-state regime, i.e., the limit as $t \to \infty$.

Opt 2.4.1. Show that L_k satisfies (2.4.2).

Opt 2.4.2. Show that, when $Z_0 = m$, $\mathsf{P}\{Z_k = n\}$ satisfies (2.4.4).

2.5 CONTROL OF SINGLE-ITEM INVENTORY SYSTEMS

In this section, we analyze so-called single-item inventory systems under *periodic review*, which means that decisions to order a replenishment are made at the end of discrete intervals, e.g., daily. As a simple example, think of macaroni boxes on a supermarket shelf. At the end of each period (say, a day), the supermarket uses the remaining stock to decide whether a replenishment is necessary. If so, it orders in the evening a batch of, say, ten boxes to fill the stock. The batch will be delivered by truck at the start of the next day after which a stocker unpacks the batch and places all ten boxes on the shelf.

In general, there are many types of items in inventory, ranging from cheap to expensive; hence different rules are needed[1] to determine *when and how much to order*. We assume that demand is discrete, stochastic, and items are not too perishable[2] so unsold items carry

[3] *[2.4.2].*

[4] *Unlike their sum, the difference of two Poisson rvs is not Poisson.*

[1] *As such, inventory systems are examples of stochastic processes subject to control.*

[2] *Yogurt is acceptable, but fresh fish not.*

over to the next period as on-hand inventory. With these assumptions, we can develop elegant formulas that allow us to simulate the discrete-time dynamics of common inventory-control systems. We then define a number of performance measures to assess their performance. We use the following notation, the meaning of which we explain below:

$$\begin{aligned} Q_k &= \text{size of order placed at the } \textit{start} \text{ of period } k, \\ D_k &= \text{demand arriving } \textit{during} \text{ period } k, \\ I_k &= \text{inventory level at the } \textit{start} \text{ of period } k, \\ I_k^+ &= \text{inventory on hand at the } \textit{start} \text{ of period } k, \\ I_k^- &= \text{backlogged demand at the } \textit{start} \text{ of period } k. \\ P_k &= \text{inventory position at the } \textit{start} \text{ of period } k, \\ l &= \text{fixed lead time.} \end{aligned}$$

We are explicit about the *timing* of each of the variables and distinguish between the start and the end of a period. Quantities measured at the *end* of a period are marked with a prime, for instance I'_k.

THE INVENTORY DYNAMICS can be constructed simply if we are given period demands $\{D_k, k = 1,2,\ldots\}$ and order sizes $\{Q_k, k = 0,1,\ldots\}$, but first we need to address a subtle point. In many inventory systems, there is a delay between *placing* and *receiving* a replenishment order: only after the receipt can the order be used to replenish on-hand stock and serve demand. The time between placing the order and its arrival is called the *lead time* l; the number of orders under way are the *outstanding* orders. We assume l to be constant and independent of the number of outstanding orders.[3] Then, when placing an order of size Q_{k-l} at the start of period $k-l$ so that it arrives at the start of period k, the *inventory level* follows the rules

[3] *If $l = 0$, the order arrives immediately.*

$$I'_{k-1} = I_{k-1} - D_{k-1}, \qquad I_k = I'_{k-1} + Q_{k-l}. \tag{2.5.1}$$

In words, the inventory level at the end of the period is the level at the start of the period minus the period's demand. The level at the start of the next period includes any replenishments.

We call $I_k^+ = \max\{I_k, 0\}$ the *inventory on hand*, as these items can directly meet demand from stock. Next, $I_k^- = \max\{-I_k, 0\}$ is the *backlogged* demand, as it represents demand waiting for future replenishments.

Clearly, if we used the inventory level to determine when and how much to order, we might receive replenishments too late because of the lead time between placing and receiving them. The *inventory position* is a more suitable control variable to compensate for this delay. It updates according to the rules

$$P'_{k-1} = P_{k-1} - D_{k-1}, \qquad P_k = P'_{k-1} + Q_k. \tag{2.5.2}$$

Thus, the inventory position includes outstanding orders directly on the books, so it accounts for orders under way but not yet arrived. Next, we study a number of inventory control policies. Note that these policies use the inventory *position*, not the inventory *level*, to decide when and how much to order.

THE BASE-STOCK POLICY places an order for each unit demanded. Specifically, the size of the order is given by

$$Q_k = S - P'_{k-1},$$

where S is the *order-up-to* level.[4] From (2.5.2) we see that $P_k = S$ for all k.

[4] *The amount ordered depends on the* end-of-period *inventory position.*

A base-stock policy is often used to control the inventories of expensive items when the ordering cost K is small relative to the item's price. For example, when someone buys a washing machine, the customer typically pays for delivery and installation. Moreover, customers rarely buy multiple washing machines at once; hence, any ordering or setup cost must be covered per sale.

AN (s, S) POLICY is appropriate in situations where delivery sizes can vary and order costs are significant. When the inventory position is at or below the *reorder level* s, we place an order up to S, specifically,

$$Q_k = (S - P'_{k-1})\,\mathbb{1}\{P'_{k-1} \le s\}. \tag{2.5.3}$$

Note that the base-stock policy is a special case of the (s, S) policy by setting $S = s + 1$.

Supermarkets could use (s, S) policies to control the inventory of yogurt, for instance, as yogurt has a fixed amount of shelf space, thereby setting the order-up-to level S.

THE (Q, r) POLICY is useful when the ordering cost is relatively large and replenishments arrive in fixed-size batches. The order quantity is a multiple of a minimal order quantity Q such that, after ordering, P_k lies between $r + 1$ and $r + Q$, inclusive.[5] In a formula,

$$Q_k = Q \left\lceil \frac{r + 1 - P'_{k-1}}{Q} \right\rceil \mathbb{1}\{P'_{k-1} \le r\}.$$

[5] *It may seem a bit odd that the* (Q, r) *policy is not called the* (Q, s) *policy, but this is the conventional terminology.*

So, a base-stock policy is a (Q, r) policy with $Q = 1$ and $r = s$.

The (Q, r) policy is used mainly in supermarket supply chains. Factories pack items in batches and ship them on pallets to distribution centers, which then send smaller batches to supermarkets, where staff unpacks and shelves single items. Thus, to reduce ordering costs, the entire supply chain handles batches instead of single units.

FINALLY, THE (T, S) POLICY specifies ordering up to S once every T periods. In a formula:

$$Q_k = (S - P'_{k-1})\,\mathbb{1}\{k \bmod T = 0\}.$$

This policy can be used by, for instance, gas stations at which every three days a truck refills all tanks.

CLEARLY, WHEN GIVEN a sequence of period demands $D_1, D_2, \ldots$ and a chosen inventory control rule, we simulate how the inventory level and position behave over time. It remains to develop a number of KPIs to assess how the inventory control rule performs. We can then search for a good policy by running simulations for various policy parameters and compare their performance using these KPIs.

For cost computations, we have to account for the cost of inventory and backlog. Here we choose to charge at the start of the period: each backlogged unit costs b euros, and each unit in stock costs h euros. If there is a cost K per order placed, the average cost up to period n is

$$Kq + hI^+ + bI^-,$$

where the order frequency, the average on-hand inventory, and the average backlog are given by, respectively,

$$q = \frac{1}{n}\sum_{k=1}^{n} \mathbb{1}\{Q_k > 0\}, \qquad I^+ = \frac{1}{n}\sum_{k=1}^{n} I_k^+, \qquad I^- = \frac{1}{n}\sum_{k=1}^{n} I_k^-.$$

Service levels form another set of KPIs to quantify how well demand is met. We discuss three such measures. The *ready rate* is the fraction of periods in which the end-of-period inventory is non-negative, i.e.,

$$\alpha := \frac{1}{n}\sum_{k=1}^{n} \mathbb{1}\{I'_k \geq 0\}.$$

The *fill rate* β measures the fraction of demand satisfied from on-hand stock. In each period, we can serve the demand D_k as long as it does not exceed the on-hand stock I_k^+ :

$$\beta := \frac{\sum_{k=1}^{n} \min\{D_k, I_k^+\}}{\sum_{k=1}^{n} D_k}. \tag{2.5.4}$$

The third measure is *cycle service level*, which is defined as the fraction of cycles[6] in which the inventory was not negative; a cycle is the time between the arrival of two consecutive replenishments. To capture this we use (2.5.1). A cycle ends just in period $k-1$ if in period $k-l$ a replenishment is placed.[7] Thus, the cycle ends in period $k-1$ when $Q_{k-l} > 0$. When in this case $I'_{k-1} \geq 0$, there is still on-hand inventory. Thus, the number of cycles in which the inventory was non-negative is $\sum_{k=l+1}^{n} \mathbb{1}\{Q_{k-l} > 0\}\, \mathbb{1}\{I'_{k-1} \geq 0\}$, The cycle service level then follows from averaging over the total number of cycles $\sum_{k=l+1}^{n} \mathbb{1}\{Q_{k-l} > 0\}$:

$$\alpha_c = \frac{\sum_{k=l+1}^{n} \mathbb{1}\{Q_{k-l} > 0\}\, \mathbb{1}\{I'_{k-1} \geq 0\}}{\sum_{k=l+1}^{n} \mathbb{1}\{Q_{k-l} > 0\}}.$$

The cycle service level is much used in practice because it is simple to measure: when a replenishment arrives, check whether there is still on-hand stock. However, in plain terms, it is a poor performance measure because it completely ignores the amount of unmet demand.[8]

To evaluate a particular pair (s, S), Alg. 2.5.1 applies the inventory recursions to a fixed demand list. We assume that the initial inventory position and level are both S, and that there are no outstanding orders. The function returns the average cost over the simulated periods.

[6] *A cycle consists typically of many periods; it is not a single period.*

[7] *A replenishment placed in period $k-l$ arrives just before period k starts.*

[8] *It is an ideal performance measure for managers: easy to understand, but not very informative. Knowing only that stock was empty provides little information about how much demand was lost.*

Algorithm 2.5.1 Simulate an (s, S) policy and return its average cost.

Input: demand list $d = [d_1, \ldots, d_n]$, parameters s, S, lead time l, costs K, h, b.
Output: average cost c.
function SIMULATESS(d, s, S, l, K, h, b)
    $P_1 \leftarrow S$, $I_1 \leftarrow S$
    $Q_j \leftarrow 0$ for $j \leq 1$
    $c \leftarrow 0$
    for $1 \leq k \leq n$ **do**
        $c \xleftarrow{+} K \cdot \mathbb{1}\{Q_k > 0\} + h \cdot I_k^+ + b \cdot I_k^-$
        $P'_k \leftarrow P_k - d_k$, $I'_k \leftarrow I_k - d_k$
        if $P'_k \leq s$ **then**
            $Q_{k+1} \leftarrow S - P'_k$
        else
            $Q_{k+1} \leftarrow 0$
        $P_{k+1} \leftarrow P'_k + Q_{k+1}$
        $I_{k+1} \leftarrow I'_k + Q_{k+1-l}$
    return c/n

CHOOSING GOOD POLICY PARAMETERS analytically is often difficult.[9] The cost of an (s, S) policy depends on the entire sequence of demands, orders, and inventory levels generated by the recursions, so there is generally no simple expression that we can minimize directly. In practice, s and S are often integer-valued and restricted to reasonable ranges. We can therefore simulate every parameter pair in these ranges and compare their performance.

[9] *O.A. Kilic and N.D. van Foreest,* An intuitive approach to inventory control with optimal stopping, *EJOR, volume 311, 2023.*

A *grid search* evaluates the performance of every option in a finite list and returns the best option. This simple algorithm is useful when the list is not too large.

Algorithm 2.5.2 Find the option with the smallest performance value.

Input: a nonempty list of options $G = [x_1, \ldots, x_n]$, performance function f.
Output: best option $x^\star$ and its value $v^\star$.
function GRIDSEARCH(f, G)
    $x^\star$, $v^\star \leftarrow x_1$, $f(x_1)$
    for $2 \leq i \leq n$ **do**
        $v \leftarrow f(x_i)$
        if $v < v^\star$ **then**
            $x^\star$, $v^\star \leftarrow x_i$, v
    return $x^\star$, $v^\star$

The next algorithm uses Alg. 2.5.1 to search a finite grid of policy parameters. It uses the same demand list for every pair, so that the policies are compared under the same conditions.

Algorithm 2.5.3 Apply grid search to an (s, S) inventory policy.

Input: demand list d, lead time l, costs K, h, b.
Output: best pair $(s^\star, S^\star)$ in the grid and its average cost $v^\star$.
function FINDSS(d, l, K, h, b)
    $sS_policy \leftarrow \lambda : (s, S) \mapsto$ SIMULATESS(d, s, S, l, K, h, b)
    $grid \leftarrow [(s, S) \in [-5, \ldots, 15] \times [-4, \ldots, 30] : s < S]$
    return GRIDSEARCH(sS_policy, $grid$)

THROUGHOUT THE LATER inventory sections, we use the Lighthouse Company case as a running numerical example. The next algorithm collects all its model data in one place. We can run the above algorithms on these data.

Algorithm 2.5.4 Set up the data for the Lighthouse Company case.

Output: demand pmf map p_D, lead time l, holding cost h, backlog cost b, and order cost K.
function LIGHTHOUSEDATA
    $p_D \leftarrow$ MAP()
    $p_D(0)$, $p_D(1)$, $p_D(2) \leftarrow 1/6, 1/5, 1/4$
    $p_D(3)$, $p_D(4)$, $p_D(5) \leftarrow 1/8, 11/120, 1/6$
    l, h, $b \leftarrow 2, 2/3, 20$
    $K \leftarrow 50$

    return p_D, l, h, b, K

TF 2.5.1. Claim: In a single-item inventory system, the inventory level triggers orders.

TF 2.5.2. Claim: In a single-item inventory system, the inventory level depends on the lead time l when $l > 0$.

TF 2.5.3. Claim: In a single-item inventory system, the inventory position and inventory level coincide when the lead time $l = 0$.

TF 2.5.4. The base-stock and (s, S) models assume that the order size is unlimited. Suppose now that it can be maximally Q. Claim: These recursions cover this case.

$$Q_k = \min\{Q, S - P'_{k-1}\}\, \mathbb{1}\{P'_{k-1} \le s\}.$$

TF 2.5.5. Claim: In an (s, S) inventory policy, an order is placed at the end of every period.

Ex 2.5.6. In production environments it is important to decide which products should be make-to-order (MTO) or make-to-stock (MTS). Why can a queueing system model MTO production? What inventory control model would best model a queueing system and what are the parameters?

Ex 2.5.7. Suppose $h \gg b$, i.e., inventory on hand is much more expensive than demand in backlog. For any of the above inventory control models, how would you prefer to perform on the three service levels?

Ex 2.5.8. Which variables (period, order quantity) are fixed, which can vary, in each of the inventory policies we discussed above? Which policy should be able to achieve the lowest cost?

Ex 2.5.9. So far, we assumed that demand in excess of the stock level is backlogged. Modify the recursions of the base-stock model such that it can cope with lost demand. For example, if at the start of period k the inventory level $I_k = 5$ and $D_k = 9$, then 4 items are lost and 5 are fulfilled. How are the (s, S) and (Q, r) policies affected by loss? How is the average cost affected?

Ex 2.5.10. Can-order policies are often used in joint inventory systems in which items can share a delivery (Replenishing supermarkets with trucks forms an example.) Placing an order includes a common setup cost K plus a (much smaller) cost K' for each item included. The can-order policy is characterized by three parameters. Besides the reorder level s_i, item i has a can-order level c_i and an order-up-to level S_i, with $s_i \leq c_i < S_i$. When any item triggers a joint order, include every item whose end-of-period inventory position is at or below its can-order level, and order it up to its order-up-to level. If the end-of-period inventory position $P'_{i,k-1}$ of any item i, $i = 1, \ldots, N$, is at or below its reorder level s_i, this item triggers a joint order at the start of period k. Make the recursions to simulate this situation. Find an expression for the cost up to period n. Explain why searching for good parameters with just simulation is not quite possible for any joint inventory system of reasonable size.

3

CONSTRUCTION OF SIMPLE CONTINUOUS-TIME STOCHASTIC PROCESSES

In Section 3.1 we develop recursions to construct queueing processes in continuous time. This shows that a queueing system can be characterized by a few elements; Section 3.2 introduces useful notation for this purpose. Section 3.3 shows how to implement these recursions and visualize the results. The resulting models remain simple; more realistic queueing situations involving control require further ideas.

3.1 QUEUEING PROCESS IN CONTINUOUS TIME

In Section 2.1 we modeled time as progressing in discrete chunks. However, we can also model queueing systems in continuous time, so that jobs can arrive at any moment in time and have arbitrary service times. In this section, we develop a set of recursions to construct the waiting times of jobs served in their order of arrival.[1]

[1] *That is, according to the FIFO scheduling rule.*

LET US IMAGINE that a machine[2] starts working on a one-hour job at 9 am, and there is no job in the queue. When the next job arrives before 10 am, this second job has to wait in the queue until the first job finishes at 10 am. If, however, this second job arrives after 10 am, it finds the server free, so its service can start immediately upon arrival.

[2] *A single server.*

More generally, suppose that we are given a strictly increasing sequence of *arrival times*[3] $\{A_k\}$ and a set of iid *service times* $\{S_k\}$, so that job k arrives at time A_k and needs an amount S_k of service. The departure times $\{D_k\}$ follow from the recursion[4]

$$\tilde{A}_k = \max\{A_k, D_{k-1}\}, \qquad D_k = \tilde{A}_k + S_k, \qquad D_0 = 0, \tag{3.1.1}$$

[3] *Moments at which the system state is inspected or changes are often called* epochs.

[4] *Note that we assume that job k 'reveals' its required service at the moment it arrives.*

where $\tilde{A}_k$ is the time the service of job k starts[5], and $D_0 = 0$ is meant to start the recursions. Why is this true? The service of job k cannot start before it arrives at A_k or before job $k-1$ leaves the server at D_{k-1}. Thus, the first moment the service of job k can start is $\max\{A_k, D_{k-1}\}$. When started, it takes S_k to get served.

[5] *That is, when this job moves from the queue to the server.*

Once A_k and D_k are known, we compute the *sojourn time*, i.e., the total time in the system, and the *waiting time*, i.e., the time in queue[6], as

$$J_k = D_k - A_k, \qquad W_k = \tilde{A}_k - A_k = J_k - S_k.$$

[6] *But not at the server.*

WHILE IT IS often simple to measure arrival times in practical situations, it is harder to generate $\{A_k\}$ directly with a simulator. For this we let a random number generator generate iid rvs $\{X_k\}$ that represent the *inter-arrival times* between jobs. The arrival times can then be computed recursively from (1.3.2) as $A_k = A_{k-1} + X_k$, $A_0 = 0$. And conversely, from (1.3.2), if $\{A_k\}$ is known, $X_k = A_k - A_{k-1}$. Interestingly, if $\{X_k\}$ and $\{S_k\}$ are given, we need not first compute $\{A_k\}$ and $\{D_k\}$ using (3.1.1) to find the

sojourn times and waiting times. In fact, suppose that job $k-1$ has to wait a time W_{k-1} in queue and spends S_{k-1} in service, then its sojourn time $J_{k-1} = W_{k-1} + S_{k-1}$. If X_k time units elapse between the arrivals of job $k-1$ and k, then job k must wait in queue[7]

$$W_k = [W_{k-1} + S_{k-1} - X_k]^+, \quad W_0 = 0, \quad S_0 = 0, \quad J_k = W_k + S_k. \tag{3.1.2}$$

However, if only J_k is needed, use $J_k = [J_{k-1} - X_k]^+ + S_k$, $J_0 = 0$.

LATER WE WILL need the system state at arbitrary times, not only at arrivals. The number of arrivals $A(t)$ during $[0, t]$ can be computed from $\{A_k\}$ as[8]

$$A(t) := \sum_{k=1}^{\infty} \mathbb{1}\{A_k \le t\} = \max\{k : A_k \le t\}. \tag{3.1.3}$$

Here and below, a maximum over job indices is defined to be 0 if no job satisfies the condition. The function $t \to A(t)$ is right-continuous.[9]

Conversely, we can retrieve the arrival times $\{A_k\}$ from the process $\{A(t)\}$. For example, if $A(s) = k-1$ and $A(t) = k$, then the arrival time A_k of the kth job must lie somewhere in $(s, t]$. Specifically,

$$A_k = \min\{t : A(t) \ge k\}, \quad A(0) = 0.$$

And if $\{A_k\}$ is known, then $X_k = A_k - A_{k-1}$. Thus, the arrival times, the inter-arrival times, and the number of arrivals as a function of time can be derived from one another.

Similarly, for the number of jobs that started service, $\tilde{A}(t)$, and the departures $D(t)$ we have

$$\tilde{A}(t) = \max\{k : \tilde{A}_k \le t\}, \qquad \tilde{A}_k = \min\{t : \tilde{A}(t) \ge k\},$$
$$D(t) = \max\{k : D_k \le t\}, \qquad D_k = \min\{t : D(t) \ge k\}.$$

The *work in the system* $W(t)$ is the time needed to clear the remaining service of all jobs present at time t. To construct $\{W(t)\}$ graphically, we draw lines that start at points (A_k, J_k) and go down with slope -1 until the lines hit the x-axis.[10] If the system is empty, the work in the system remains at zero until an arrival occurs. To express $W(t)$ as a function of time, note that $A(t)$ is the index of the last job to arrive at or before t, thus, $D_{A(t)}$ is the time this job leaves. Therefore,

$$W(t) = [D_{A(t)} - t]^+. \tag{3.1.4}$$

The work in the system is equal to the so-called *virtual waiting time*. For example, if $W(t) = 5$ hours[11] , then a job arriving at t would have a waiting time of 5 hours. Fig. 3.1.1 shows how the above concepts relate to each other.

ONCE WE HAVE the arrival and departure processes, it is easy to compute the *number of jobs in the system* at time t, as illustrated in Fig. 3.1.2,

$$L(t) = A(t) - D(t) + L(0), \tag{3.1.5}$$

[7] *As we start counting jobs for $k = 1, 2, \ldots$, W_0 and S_0 are artificial initial values, both set to zero.*

[8] *Read the RHS as the maximum of all k such that $A_k \le t$.*

[9] *Define $f(x-) = \lim_{y \uparrow x} f(y)$ and $f(x+) := \lim_{y \downarrow x} f(y)$. A function f is right-continuous if $f(x) = f(x+)$ for all x, and left-continuous if $f(x) = f(x-)$ for all x.*

[10] *The slope is -1 because a single server processes one hour of work per hour.*

[11] *It takes 5 hours to complete all jobs in the system at time t.*

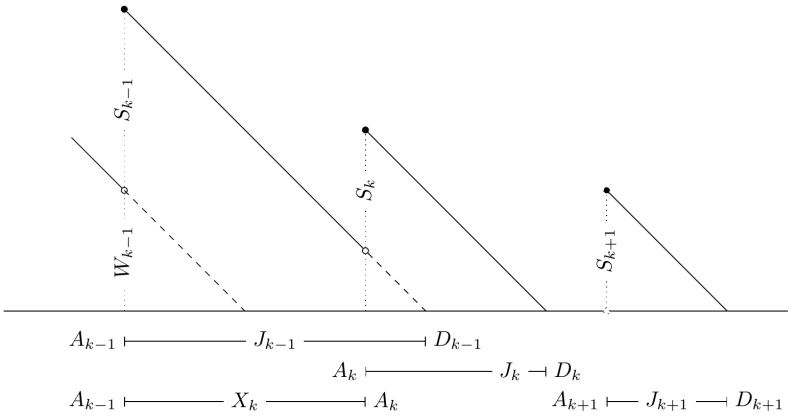


FIG. 3.1.1: The work in the system $W(t)$ as a function of time. Right after job k arrives at A_k, the work in the system satisfies $W(A_k) = J_k$, where the sojourn time J_k is the sum of the waiting time W_k and service time S_k. The server processes work at a rate of one hour per hour so W decreases as the solid lines with slope -1. Once the system is empty, there is nothing to process, and W remains at 0 as long as the server is idle.

where $L(0) = 0$, since the recursions above start with an empty system. Recalling that a job can be either waiting or in service, we distinguish $L(t)$ (in the system), $Q(t)$ (in queue), and $L_s(t)$ (in service):

$$L(t) = Q(t) + L_s(t), \qquad L_s(t) = \tilde{A}(t) - D(t) = L(t) - Q(t).$$

Note that statistics obtained from a sample path $t \to Q(t)$ can be quite different from the samples $k \to Q(A_k-)$ as observed by arriving jobs, and likewise for $L(t)$ and $L(A_k-)$.[12]

[12] *We need to be careful about the left and right limits at jump epochs.*

THE FOLLOWING INDICATOR function

$$I_k(s) := \mathbb{1}\{A_k \le s < D_k\}, \tag{3.1.6}$$

represents the presence of job k at time s.[13] Clearly, $\int_0^t I_k(s)\,\mathrm{d}s$ corresponds to the total time job k spends in the system up to time t while summing over all jobs present at time s gives $L(s)$. Therefore,

[13] $I_k(s) = 1$ *if job k is present in the system at time s and zero otherwise.*

$$J_k = \int_0^\infty I_k(s)\,\mathrm{d}s, \qquad L(t) = \sum_{k=1}^\infty I_k(t). \tag{3.1.7}$$

With this we can relate sojourn times and the number of jobs in the system. When n is the index of the last departure of a simulation, we have on the one hand,[14]

$$\frac{1}{D_n}\sum_{k=1}^n \int_0^{D_n} I_k(s)\,\mathrm{d}s = \frac{1}{D_n}\sum_{k=1}^n J_k,$$

[14] *At time D_n all n jobs have departed.*

while on the other hand,

$$\frac{1}{D_n}\int_0^{D_n} \sum_{k=1}^n I_k(s)\,\mathrm{d}s = \frac{1}{D_n}\int_0^{D_n} L(s)\,\mathrm{d}s = \bar{L}.$$

As the expressions at the LHSs are equal, the sum over all sojourn times divided by the time of the last departure equals the time-average number of jobs in the system:

$$\bar{L} = \frac{1}{D_n}\sum_{k=1}^n J_k. \tag{3.1.8}$$

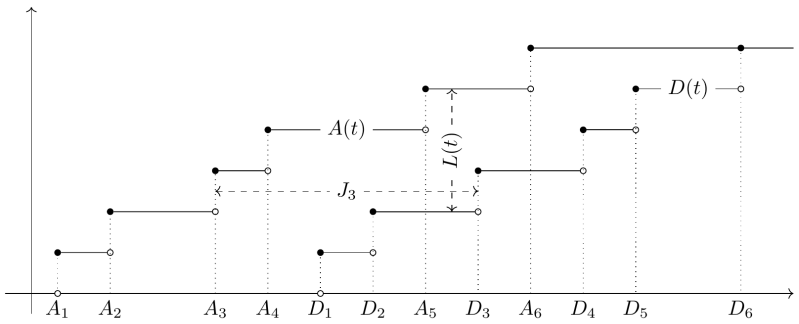


FIG. 3.1.2: Relation between the arrival process $\{A(t)\}$, the departure process $\{D(t)\}$, the number in the system $\{L(t)\}$ and the sojourn times $\{J_k\}$. Observe that the arrival and departure times of different jobs are intertwined.

WE FINISH THIS section with constructing the recursions for the waiting times in a multi-server queue. Jobs line up in queue i, $i = 1,\ldots,c$ that is served by a server working with rate $\mu(i)$. When job k arrives, it sees a vector $w_k = (w_k(1),\ldots,w_k(c))$ of waiting times at the queues. Suppose the job selects the line with the shortest waiting time.[15] Let $e(i)$ denote the ith unit vector[16] , and $\mathbf{1} = \sum_{i=1}^{c} e_i$. Then, in analogy with (3.1.2), the waiting time w_{k+1} updates as[17]

$$s_k = \underset{i\in\{1,\ldots,c\}}{\arg\min}\{w_k(i)\}, \qquad w_{k+1} = [w_k + S_k e(s_k)/\mu(s_k) - X_{k+1}\mathbf{1}]^+, \tag{3.1.9}$$

where s_k is the queue selected by job k. The elapsed time between the arrivals of jobs k and $k+1$ is X_{k+1}.

[15] *This is not necessarily the same as the shortest queue.*

[16] *A vector with a 1 at place i and zeros elsewhere.*

[17] $[\cdot]^+$ *applies element-wise to the vector.*

WHILE THE ABOVE recursions are charming in their simplicity, it is hard to generalize these recursions to more realistic queueing situations involving control. In later sections we discuss how to use discrete-event simulation to analyze such harder cases.

TF 3.1.1. Claim: with our notation, the following mapping is correct:

$$A_k : \mathbb{N} \to \mathbb{R}, \quad \text{job id (integer) to arrival time (real number).}$$

TF 3.1.2. Claim: the number of arrivals $A(t)$ during $[0,t]$ can be defined as $\min\{k : A_k \geq t\}$.

TF 3.1.3. Claim: The waiting time of the third job is correctly represented in Fig. 3.1.3.

TF 3.1.4. Claim: $A_{A(t)} = t$ for all t and $A(A_n) = n$ for all n.

TF 3.1.5. For the $G/G/1$ queue, the virtual waiting time process $\{W(t), t \geq 0\}$ satisfies

$$W(t) = [J(A_{A(t)}) + (A_{A(t)} - t)]^+.$$

Ex 3.1.6. Show that $L(A_k-) > 0 \iff A_k \leq D_{k-1}$.

Ex 3.1.7. Why is $W(A_k) = J_k$?

Ex 3.1.8. Modify (3.1.9) such that job k selects the queue with the shortest sojourn time.

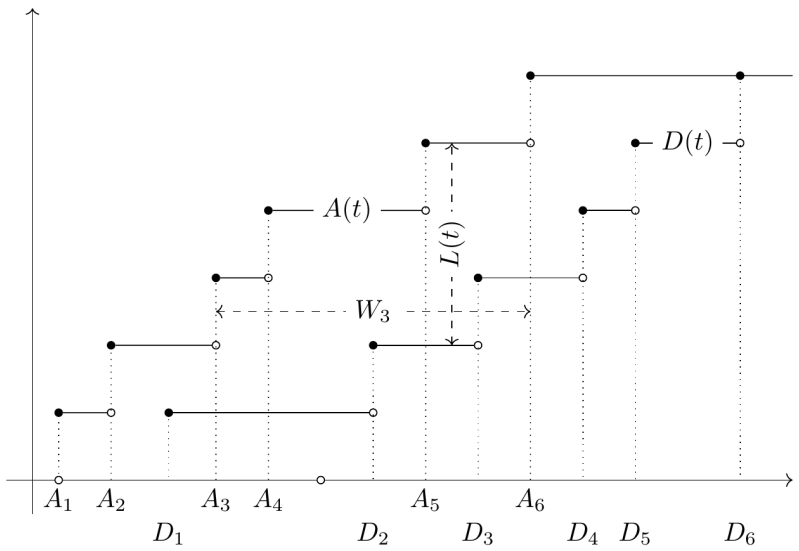


FIG. 3.1.3: The figure for [3.1.3].

3.2 KENDALL'S NOTATION

As is apparent from Section 2.1 and Section 3.1, the construction of a queueing process for a single station involves a few main elements: the distribution of inter-arrival times for jobs, the distribution of service times, the number of servers present to process jobs, and the number of jobs that arrive and can be served simultaneously as a batch.

We will often use *Kendall's abbreviation* $A^B/Y^C/c/K$ to characterize the type of queueing process. Here, A is the distribution of times between consecutive batch arrivals, B the size of the batch[1] at an arrival, Y the distribution of service-batch durations, C the number of items that can be served at once, c the number of servers available, and K the maximal number of items that the system can contain. We assume that the inter-arrival times, service durations, arrival batch sizes, and service batch sizes each form an iid sequence, and that these four sequences are mutually independent. Finally, the service discipline is assumed to be FIFO. When $B \equiv 1$, jobs arrive individually and B is omitted. Similarly, when $C \equiv 1$, jobs are served individually and C is omitted. Likewise, if the system is unbounded, K is omitted. Moreover, service is assumed to be *work-conserving*, which means that a server cannot be idle when there is a job in queue that it can serve.[2] The most important inter-arrival and service distributions are the exponential distribution, denoted with the shorthand[3] M, and the general distribution, denoted G. We write D for a deterministic rv.

[1] *That is, the number of items in the batch.*

[2] *Thus, when service is work-conserving, capacity is not wasted.*

[3] *M := Markov or Memoryless.*

IMPORTANT EXAMPLES INCLUDE the $M/M/1$ queue, with exponentially distributed inter-arrival times and service times; the $M/G/1$ queue, with exponential inter-arrival times and general service times; and the $G/G/c$ queue, where both the inter-arrival and service times are unspecified.

A common model for determining the number of beds required in a hospital ward is the $M/M/c/c$ queue. By the arguments of Section 1.4

patient inter-arrival times are exponentially distributed, and as there are many patients with various reasons for occupying a bed, the recovery times are (roughly) exponentially distributed. There are c beds available, and since each bed serves one patient,[4] the hospital is 'full' when all c beds are occupied. Therefore $K = c$. The hospital's problem is to determine the number c of beds needed such that the blocking probability is below some threshold, say 1%.

[4] *In a hospital patients typically don't share beds.*

The discrete-time model that corresponds to the recursion (2.1.1) can be represented as a $D^B/D^C/1$ queue in which $a_k \sim B$ and $c_k \sim C$.

An inventory under base-stock control with $S = 0$ is a $D^B/D/\infty$ queue: the total demand in a period is B, and each demand is delivered (served) after the lead time, hence it can be seen as if each demand receives its own server.

TF 3.2.1. Consider an $M^2/G/c/K$ queue, with $K \geq c$. Claim: Exactly one of the following statements is false.

- Jobs arrive in pairs of two.
- Service times can be deterministic.
- The queue shared among c parallel servers cannot contain more than $K - c$ jobs.

TF 3.2.2. In the $D/M/1$ queue, jobs have deterministic service times.

TF 3.2.3. Consider a check-in desk at an airport. One desk is dedicated to business customers. However, when this desk is idle (i.e., no business customer in service or in queue), it also serves economy class customers. The other c desks are reserved for economy class customers. The queueing process perceived by the economy class customers can be modeled as an $M/M/(c+1)$ queue.

TF 3.2.4. Claim: The $M/G/1$ queue has Poisson arrivals, general service times, and a single server.

TF 3.2.5. Claim: In an $M/M/c/K$ queue, the K denotes the number of servers.

3.3 SIMULATING WAITING TIMES

Using the recursions of Section 3.1, we can simulate the waiting time of the $G/G/1$ queue. We use this to analyze how various blocking strategies can prevent long waiting times. In passing we will encounter subtle problems with stochastic dependence and with (plotting) pmfs.

THE DYNAMICS OF the single-server queue in continuous time is given by the recursions in (3.1.1).

Suppose that the inter-arrival times $[x_1, \ldots, x_n]$ and service times $[s_1, \ldots, s_n]$ of one sample path are given. The next algorithm computes the arrival, departure, waiting, and sojourn times of all jobs.

Algorithm 3.3.1 Simulation: compute a sample path of a single-server queue.

Input: inter-arrival times $[x_1, \ldots, x_n]$ and service times $[s_1, \ldots, s_n]$.
Output: arrival times A, departure times D, waiting times W, and sojourn times J.
function QUEUESAMPLEPATH($[x_1, \ldots, x_n]$, $[s_1, \ldots, s_n]$)
    $A_0 \leftarrow 0$, $D_0 \leftarrow 0$
    for $k = 1, \ldots, n$ **do**
        $A_k \leftarrow A_{k-1} + x_k$
        $D_k \leftarrow \max\{A_k, D_{k-1}\} + s_k$
        $J_k \leftarrow D_k - A_k$
        $W_k \leftarrow J_k - s_k$
    return A, D, W, J

Once the sample path is available, its average waiting and sojourn times and its time-average number of jobs are

$$\overline{W} = \frac{\text{SUM}([W_1, \ldots, W_n])}{n}, \qquad \overline{J} = \frac{\text{SUM}([J_1, \ldots, J_n])}{n}, \qquad \bar{L} = \frac{\text{SUM}([J_1, \ldots, J_n])}{D_n}.$$

For a threshold T, the fraction of jobs whose waiting time is at least T is

$$q_T = \frac{\text{SUM}([\mathbb{1}\{W_k \geq T\} : k = 1, \ldots, n])}{n}.$$

For example, take independent exponential inter-arrival and service times with rates $\lambda = 1$ and $\mu = 1.1$, respectively. For one sample path of 10 000 jobs, we obtain $\overline{J} = 10.11$, $\overline{W} = 9.20$, and $\bar{L} = 10.167$. The maximum waiting time is 52.12, and $q_5 = 0.59$.

BLOCKING JOBS CAN be a remarkably effective strategy to prevent long and variable waiting times. For the three admission rules below, we use the same sample path and set $T = 5$. One rule, known as *complete acceptance*, accepts job k when $W_k \leq T$ and otherwise blocks it; see Alg. 3.3.2. The algorithm records this decision with the indicator I_k, which is 1 for an accepted job and 0 for a rejected job. Here, W_k is the waiting time job k would have if it were accepted. A rejected job leaves immediately, so its value of W_k records the work it sees, not time it spends in queue. Hence, if job $k-1$ is accepted, $I_{k-1}s_{k-1} = s_{k-1}$ contributes to W_k; if it is rejected, $I_{k-1}s_{k-1} = 0$, so it adds no work to the system. In this way, blocking is implemented without modifying the original service times.

Algorithm 3.3.2 Apply complete acceptance with waiting-time threshold T.

Input: inter-arrival times $[x_1, \ldots, x_n]$, service times $[s_1, \ldots, s_n]$, and threshold T.
Output: candidate waiting times W and acceptance decisions I.
function COMPLETEACCEPTANCE($[x_1, \ldots, x_n]$, $[s_1, \ldots, s_n]$, T)
    $W_1 \leftarrow 0$, $I_1 \leftarrow 1$ ▷ *The system starts empty.*
    for $k = 2, \ldots, n$ **do**
        $W_k \leftarrow [W_{k-1} + I_{k-1} \cdot s_{k-1} - x_k]^+$
        $I_k \leftarrow \mathbb{1}\{W_k \leq T\}$
    return W, I

The fraction of accepted jobs and the average waiting time of accepted jobs are, respectively,

$$a = \frac{\text{SUM}([I_1, \ldots, I_n])}{n}, \qquad \overline{W}_{\text{accepted}} = \frac{\text{SUM}([I_k W_k : k = 1, \ldots, n])}{\text{SUM}([I_1, \ldots, I_n])}.$$

For the sample path above, $a = 0.90$, the average candidate waiting time over all jobs is 2.23, and $\overline{W}_{\text{accepted}} = 1.83$. Moreover, 8995 of the 10 000 jobs are accepted, and the maximum waiting time of an accepted job is 5.00. Thus, about 90% of the jobs are accepted, while the average waiting time drops by a factor of about four and the maximum by more than a factor of ten.

There are other interesting rules to block jobs. A second rule, *complete rejection*, accepts only if $J_k \le T$; see Alg. 3.3.3. Here, $J_k = W_k + s_k$ is the sojourn time job k would have if it were accepted. Thus, unlike complete acceptance, complete rejection also takes the service time of the arriving job into account. As before, $I_{k-1} s_{k-1}$ ensures that a rejected job adds no work to W_k.

Algorithm 3.3.3 Apply complete rejection with sojourn-time threshold T.

1: **Input:** inter-arrival times $[x_1, \ldots, x_n]$, service times $[s_1, \ldots, s_n]$, and threshold T.
2: **Output:** candidate waiting times W, candidate sojourn times J, and acceptance decisions I.
3: **function** COMPLETEREJECTION($[x_1, \ldots, x_n]$, $[s_1, \ldots, s_n]$, T)
4: $\quad W_1 \leftarrow 0$, $J_1 \leftarrow s_1$, $I_1 \leftarrow \mathbb{1}\{J_1 \le T\}$ ▷ *The system starts empty.*
5: $\quad$ **for** $k = 2, \ldots, n$ **do**
6: $\quad\quad W_k \leftarrow [W_{k-1} + I_{k-1} \cdot s_{k-1} - x_k]^+$
7: $\quad\quad J_k \leftarrow W_k + s_k$
8: $\quad\quad I_k \leftarrow \mathbb{1}\{J_k \le T\}$
9: $\quad$ **return** W, J, I

For the arrival and service times above, the average candidate waiting time over all jobs is 1.42, and the average service time of accepted jobs is 0.81. The largest service time among all jobs is 9.41, whereas the largest service time among accepted jobs is 4.92. Perhaps surprisingly, for the same arrival and service times, complete rejection accepts 0.94 of the jobs, compared with 0.90 under complete acceptance. Although complete rejection uses a stricter admission rule, it rejects a job before that job adds work[1] to the system whenever $W_k + s_k > T$. This reduces the waiting times of later jobs, so more of them can be accepted.

[1] *In particular, all jobs with $s_k > T$ are rejected.*

A third rule, *partial acceptance*, admits as much of each job's service requirement as possible without making its sojourn time exceed T; see Alg. 3.3.4. Thus, after job k has waited W_k, at most $T - W_k$ units of its required service time s_k can be accepted. If B_k denotes the accepted part, then

$$B_k = \min\{T - W_k, s_k\}, \qquad J_k = W_k + B_k.$$

Algorithm 3.3.4 Apply partial acceptance with sojourn-time threshold T.

Input: inter-arrival times $[x_1, \ldots, x_n]$, service times $[s_1, \ldots, s_n]$, and threshold T.
Output: waiting times W, accepted service times B, and sojourn times J.
function PARTIALACCEPTANCE($[x_1, \ldots, x_n]$, $[s_1, \ldots, s_n]$, T)
    $W_1 \leftarrow 0$, $B_1 \leftarrow \min\{T, s_1\}$, $J_1 \leftarrow B_1$ ▷ *The system starts empty.*
    for $k = 2, \ldots, n$ **do**
        $W_k \leftarrow [W_{k-1} + B_{k-1} - x_k]^+$
        $B_k \leftarrow \min\{T - W_k, s_k\}$
        $J_k \leftarrow W_k + B_k$
    return W, B, J

For the arrival and service times above, the average waiting time is 1.84, the average sojourn time is 2.64, and the average accepted service time is 0.80. The largest accepted service time is 5.00, as required.

WE NEXT SHOW how to obtain the *distribution* of the waiting time and departure time of job $k = 30$.[2] We use both the probabilistic arithmetic of Section 1.5 and simulation.

[2] *Job $k = 30$ is chosen arbitrarily.*

The inter-arrival and service distributions are as follows. We choose $\mathsf{E}[S] > \mathsf{E}[X]$ to get more widely spread values in the figures.

$$p_X(3) = p_X(4) = p_X(5) = \frac{1}{3}, \qquad p_S(4) = p_S(5) = p_S(7) = \frac{1}{3}, \tag{3.3.1}$$

and both pmfs are zero elsewhere. Thus, $\mathsf{E}[X] = 4$ and $\mathsf{E}[S] = 16/3 \approx 5.33$.

Since the system starts empty, $\mathsf{P}\{W_1 = 0\} = 1$. The function in Alg. 3.3.5 repeatedly applies the probabilistic-arithmetic functions from Section 1.5 to Lindley's recursion. Each iteration uses fresh independent copies of X and S. This is valid because W_{k-1} is independent of S_{k-1} and X_k.

Algorithm 3.3.5 Probabilistic arithmetic: compute the pmf of the waiting time of job n.

Input: inter-arrival-time pmf p_X, service-time pmf p_S, and job number n.
Output: waiting-time pmf p_W of W_n.
function WAITINGTIMEPMF(p_X, p_S, n)
    $p_W \leftarrow$ CONSTANT(0) ▷ *The system starts empty.*
    for $k = 2, \ldots, n$ **do**
        $p_W \leftarrow$ ADD(p_W, p_S)
        $p_W \leftarrow$ SUBTRACT(p_W, p_X)
        $p_W \leftarrow$ POSITIVEPART(p_W)
    return p_W

We compare the distribution obtained by probabilistic arithmetic with an estimate based on simulation. The mean and standard deviation of the waiting time obtained by probabilistic arithmetic are 38.84 and

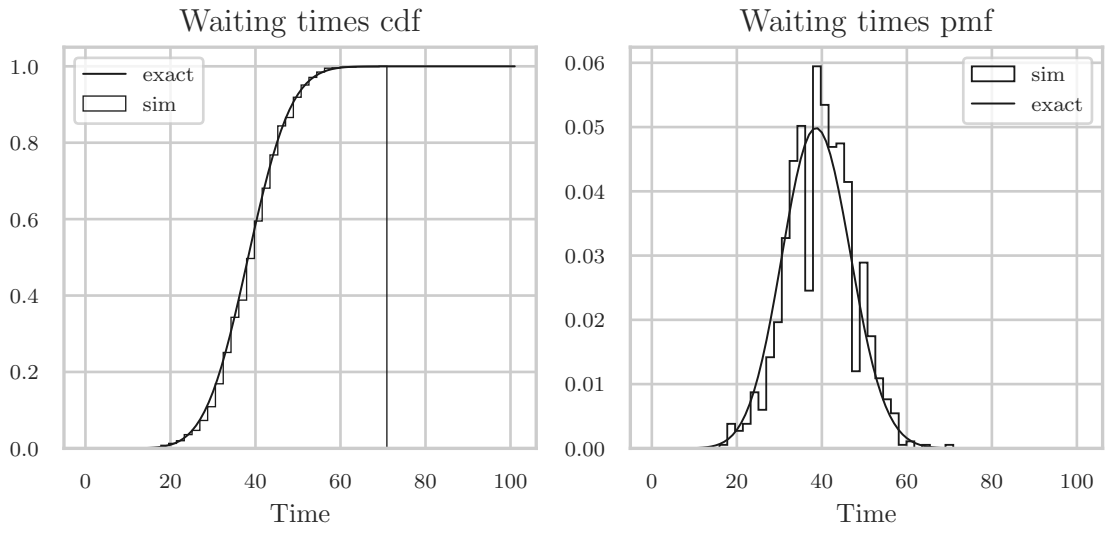


FIG. 3.3.1: The waiting time of job 30 with the inter-arrival and service times specified in the text: theoretical and simulated cdfs on the left, and the theoretical pmf with a simulation histogram on the right. For this unstable queue, the waiting-time distribution shifts to the right as the job number increases.

7.96, respectively. The simulation estimates are 38.78 and 7.97, which agree well. The left panel of Fig. 3.3.1 compares the theoretical and simulated cdfs of W_{30}. The right panel overlays the theoretical pmf and a density-normalized simulation histogram.

LET US NOW compare D_{30} computed with probabilistic arithmetic with the simulated departure time, which we denote by $\hat{D}_{30}$. Recall that $D_{30} = A_{30} + W_{30} + S_{30}$. Thus, to compute D_{30} with probabilistic arithmetic, we need the pmfs of A_{30}, W_{30}, and S_{30}. Since $A_{30} = \sum_{k=1}^{30} X_k$, we obtain its pmf as $p_{A_{30}} \leftarrow \text{CONVOLUTION}\left(p_X, 30\right)$. The function in Alg. 3.3.5 provides the pmf of W_{30}. Finally, S_{30} has the pmf p_S specified in (3.3.1). Combining these three pmfs as if the random variables were independent gives a mean of 164.18 and a standard deviation of 9.22. In contrast, $\hat{D}_{30}$ has simulated mean 164.12 and standard deviation 6.82. The means are close, but the standard deviations differ noticeably. The probabilistic-arithmetic result is wrong.

The reason is that A_k and W_k are negatively correlated. Recall that $A_k = A_{k-1} + X_k$ and $W_k = [W_{k-1} + S_{k-1} - X_k]^+$. If X_k is very large, A_k is large, but W_k is zero (or small), and then $D_k \approx A_k + S_k$. In probabilistic arithmetic, addition of two rvs is only allowed when they are independent.[3] Treating $A_k + W_k + S_k$ as the sum of three independent rvs violates this assumption. But why then is the recursion $W_k = [W_{k-1} + S_{k-1} - X_k]^+$ correct? This is because W_{k-1} *is* independent of S_{k-1} and X_k.

Stochastic dependence is easily overlooked. To avoid making mistakes, check analytical results with simulation. If the results disagree, there is probably an error[4] somewhere.

IN GENERAL, THE cdf behaves more regularly than the pmf, even though the pmf seems easier to interpret. To show this, modify the support of X

[3] *Handling such dependencies requires the modeling of the joint distribution of rvs. The Python package lea supports this.*

[4] *Often not just one error!*

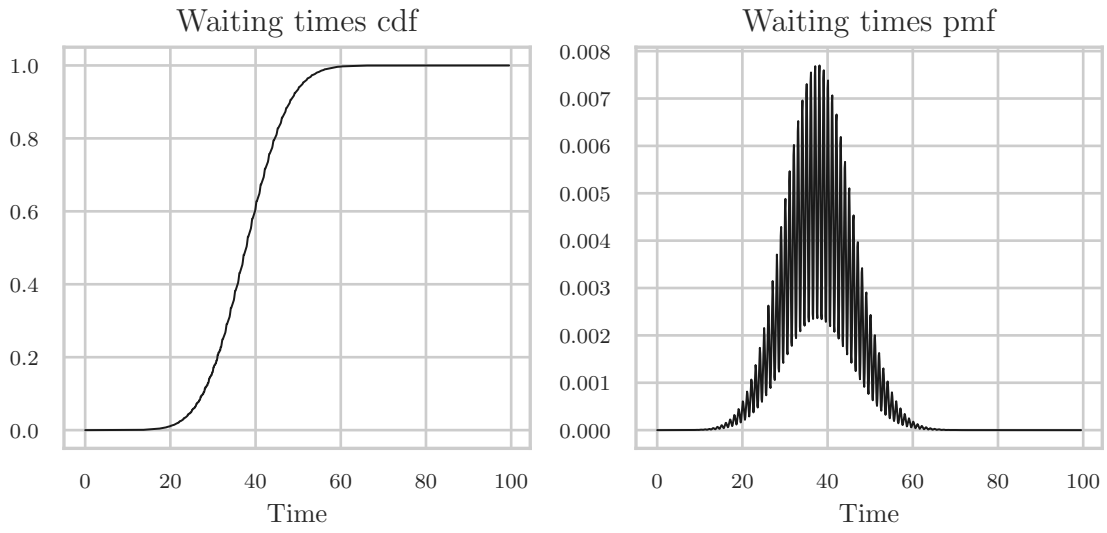


FIG. 3.3.2: The cdf and pmf of the waiting time of job 30 with X with support $\{3, 4.1, 5\}$ instead of $\{3, 4, 5\}$; the rest of the parameters are as in Fig. 3.3.1. Note how pathological the pmf of the theoretical distribution behaves.

to $\{3, 4.1, 5\}$, keep all else the same, and simulate. The resulting pmf in Fig. 3.3.2 differs markedly from that in Fig. 3.3.1!

WE FINISH THIS section with the multi-server queue of (3.1.9). Let $w(i)$ be the time until server i becomes available. During an inter-arrival time x_k, every component of w decreases by x_k, but cannot become negative. Job k then selects a server with the smallest workload. If server i_k has rate $\mu(i_k)$, its service takes $s_k/\mu(i_k)$ time.

Algorithm 3.3.6 Compute a sample path of a multi-server queue.

Input: inter-arrival times $[x_1, \ldots, x_n]$, work requirements $[s_1, \ldots, s_n]$, and server rates $[\mu(1), \ldots, \mu(c)]$.
Output: selected servers i, arrival times A, departure times D, waiting times W, and sojourn times J.
function MULTISERVERQUEUE(x, s, μ)
    $A_0 \leftarrow 0$, $w \leftarrow [0, \ldots, 0]$
    for $k = 1, \ldots, n$ **do**
        $A_k \leftarrow A_{k-1} + x_k$
        $w \leftarrow [w - x_k \cdot \mathbf{1}]^+$
        $i_k \leftarrow \operatorname{argmin}_{j \in \{1, \ldots, c\}} w(j)$
        $W_k \leftarrow w(i_k)$
        $J_k \leftarrow W_k + s_k/\mu(i_k)$
        $D_k \leftarrow A_k + J_k$
        $w(i_k) \overset{+}{\leftarrow} s_k/\mu(i_k)$
    return i, A, D, W, J

The algorithm allows the server rates to differ. Setting $\mu(i) = 1$ for every server gives identical unit-rate servers.

TF 3.3.1. The list $D = [D_0, D_1, \ldots, D_n]$ contains the departure times computed by a queue simulation. Here $D_0 = 0$ is used only for initialization and does not correspond to a job.

```
1: m ← SUM(D) / (n + 1)
```

Claim: m is the mean departure time of the n simulated jobs.

TF 3.3.2. Consider a continuous-time single-server queue with inter-arrival times $\{X_k\}$, service times $\{S_k\}$, waiting times $\{W_k\}$, and sojourn times $\{J_k\}$. Claim: we can compute the waiting times and sojourn times using the recursion:

$$W_k = [J_{k-1} - X_k]^+, \qquad J_k = W_k + S_k.$$

TF 3.3.3. Let p_X and p_S be the pmf maps of the inter-arrival and service times. Consider the following algorithm.

```
1: p_A ← CONSTANT(0)
2: p_D ← CONSTANT(0)
3: for k = 1, ..., n do
4:     p_A ← ADD(p_A, p_X)
5:     p_D ← MAXIMUMPMF(p_D, p_A)
6:     p_D ← ADD(p_D, p_S)
7: return p_D
```

Claim: this algorithm correctly computes the pmf of D_n.

TF 3.3.4. In a $G/G/c$ queue, job k sees the workload $w(j)$ and service rate $\mu(j)$ of each server j. Consider the following selection rule.

```
1: i_k ← arg min_{j ∈ {1,...,c}} w(j)
2: W_k ← w(i_k)
3: J_k ← W_k + s_k / μ(i_k)
```

Claim: this rule selects the server that gives job k the smallest sojourn time.

TF 3.3.5. Claim: In a $G/G/c$ queue simulation, the waiting time of job k can be computed as $W_k = [W_{k-1} + S_{k-1} - X_k]^+$.

4

FUNDAMENTAL TOOLS

With the tools developed in the previous chapters, we can *simulate* very general stochastic processes. While useful, we still need analytical models to make reasoning about such processes simpler; in this chapter we make a start with that.

As we have seen in Section 2.4, characterizing the *transient behavior* of stochastic systems is complicated, so we focus on the *long-run time-average* behavior. Arrival and service rates play a crucial role. For a queue without losses, stability requires that the rate at which work arrives does not exceed the available service capacity. Once stability is established, we can define performance measures such as the long-run average waiting time.

We derive some results fundamental to analyzing any stochastic process. These results are based on sample paths which thereby provide a unifying link between, on the one hand, the construction of a stochastic process in Chapter 2 and Chapter 3, and, on the other hand, the theoretical results that follow here. To illustrate the relations among these concepts, we provide two mind maps at the end of the chapter, see Fig. 4.7.1 and Fig. 4.7.2.

We keep the discussion intuitive and refer to El-Taha and Stidham Jr. [1998] for proofs and further background.

4.1 RATE, STABILITY AND LOAD

In this section, we define the load and stability of a queueing system, which is, arguably, the most important performance measure of such a system. We first develop the concepts for the $G/G/1$ queue, then we consider the $G^B/G^C/c$ queue.

WE START WITH defining the *arrival rate* and *departure rate* as the long-run average number of jobs that arrive and depart along a sample path:

$$\lambda = \lim_{t\to\infty} \frac{A(t)}{t}, \qquad \delta = \lim_{t\to\infty} \frac{D(t)}{t}, \tag{4.1.1}$$

where $A(t)$ and $D(t)$ are discussed in Section 3.1. We assume that both limits exist[1] , are finite, and $0 < \delta \leq \lambda < \infty$.

[1] *The first limit does not necessarily exist if $A(t)$ is some pathological function.*

FOR THE ARRIVAL rate, suppose we are given a sequence of inter-arrival times $X_k \overset{\text{iid}}{\sim} X$. By the strong law of large numbers, $A_n/n = n^{-1}\sum_{k=1}^{n} X_k \overset{a.s.}{\to} \mathsf{E}[X]$ as $n \to \infty$ when $\mathsf{E}[X] < \infty$. The next theorem relates $\mathsf{E}[X]$ to the limit of $A(t)/t$ as $t \to \infty$.

Theorem 4.1.1. Assume that $\lambda \in (0,\infty)$. Then, the existence of one of the next two limits implies the existence of the other, in which case,

$$\frac{A_k}{k} \to \lambda^{-1} > 0 \iff \frac{A(t)}{t} \to \lambda > 0.$$

—

Proof. $\Longrightarrow$: $A_{A(t)}$ is the arrival time of the last job *before* time t and $A_{A(t)+1}$ is the arrival time of the first job *after* time t. Therefore $A_{A(t)} \le t < A_{A(t)+1}$ and, for $A(t) > 0$,

$$\frac{A_{A(t)}}{A(t)} \le \frac{t}{A(t)} < \frac{A_{A(t)+1}}{A(t)} = \frac{A_{A(t)+1}}{A(t)+1}\frac{A(t)+1}{A(t)}.$$

For the LHS, observe that the assumption $A_k/k \to \lambda^{-1} > 0$ as $k \to \infty$ implies that $A_k \to \infty$. This in turn implies that $A(t) := \max\{k : A_k \le t\} \to \infty$ as $t \to \infty$. Next, if k and t are such that $A(t) = k$ then[2] $A_{A(t)} = A_k$, hence, $A_{A(t)}/A(t) = A_k/k$. Consequently, $\lim_{t\to\infty} A_{A(t)+1}/(A(t)+1) = \lim_{t\to\infty} A_{A(t)}/A(t) = \lim_{k\to\infty} A_k/k = \lambda^{-1}$. Finally, since $A(t) \to \infty$ implies that $(A(t)+1)/A(t) \to 1$, we conclude that the LHS and RHS both converge to $1/\lambda$.

$\Longleftarrow$: If $A(t)/t \to \lambda$, then at the epochs A_k we also have $A(A_k)/A_k \to \lambda$. (Observe that the epochs $\{A_k\}$ are just a subset of $\mathbb{R}^+$ while t takes any value in $\mathbb{R}^+$.)[3] Noting that $A(A_k) = k$, it follows that $k/A_k \to \lambda$. □

[2] *[3.1.4].*

[3] *Recall:* $A(t)/t \to \lambda$ *as* $t \to \infty$
$\iff$
$\forall \epsilon > 0 : \exists s : \forall t > s : |A(t)/t - \lambda| < \epsilon.$

Thus, from this theorem it follows that

$$\mathsf{E}[X] = \lim_{n\to\infty} \frac{1}{n}\sum_{k=1}^{n} X_k = \lim_{n\to\infty} \frac{A_n}{n} = \lim_{t\to\infty} \frac{t}{A(t)} = \frac{1}{\lambda}.$$

For the service rate of a *single* server, let S_k denote the required service time of the kth job served, so that $U_n = \sum_{k=1}^{n} S_k$ is the total service time of the first n jobs. Similarly to the definition of $A(t)$, we let $U(t) = \max\{k : U_k \le t\}$ and define the *service*, or *processing*, rate of a single server as

$$\mu := \lim_{t\to\infty} \frac{U(t)}{t}, \quad 0 < \mu < \infty,$$

assuming that this limit exists. This provides us with the relation

$$\mathsf{E}[S] = \lim_{n\to\infty} \frac{1}{n}\sum_{k=1}^{n} S_k = \lim_{n\to\infty} \frac{U_n}{n} = \lim_{t\to\infty} \frac{t}{U(t)} = \frac{1}{\mu}.$$

Hence, the service rate μ of a single server equals $1/\mathsf{E}[S]$.

GIVEN THE ARRIVAL and service rate, we define for the single-server queue the

$$\text{Server load } = \rho := \lambda\,\mathsf{E}[S] = \frac{\lambda}{\mu} = \frac{\mathsf{E}[S]}{\mathsf{E}[X]},$$

and the

$$\text{Server utilization } = \delta\,\mathsf{E}[S] = \frac{\delta}{\mu}.$$

We need to distinguish between these two concepts either when $\lambda > \mu$ or when arriving customers are blocked. In the first situation, the queue in front of the server increases without bound so that $\lambda > \mu = \delta$.[4] In the second situation, the blocked jobs do not enter the system and therefore

[4] *Of course, the departure rate can never exceed the service rate.*

are not served, so $\lambda > \delta$. Bear in mind that the load can exceed 1 (when $\lambda > \mu$), while the utilization is always ≤ 1.

In general we say that a system is *rate-stable* if

$$\lambda = \delta,$$

in words, the system is rate-stable whenever in the long run jobs leave the system just as fast as they enter the system. In this case, load and utilization are equal; we use these terms interchangeably for rate-stable systems. Rate-stability requires that $\lambda \leq \mu$.

It is easy to generalize the definition of the load to the $G^B/G^C/c$ queue. Batches arrive at rate λ. When each batch contains $\mathsf{E}[B]$ jobs each of which requires $\mathsf{E}[S]$ time to serve, the expected work per batch is $\mathsf{E}[B]\,\mathsf{E}[S]$. Similarly, when a server serves, on average, a batch of $\mathsf{E}[C]$ jobs per service time, its service rate is $\mathsf{E}[C]/\mathsf{E}[S]$. Finally, when the station contains c identical servers, the station's capacity is c times as large as that of a single server. Hence, in this more general case,

$$\rho := \frac{\lambda\,\mathsf{E}[S]}{c}\frac{\mathsf{E}[B]}{\mathsf{E}[C]}. \tag{4.1.2}$$

The definition of utilization changes accordingly.

WE CAN USE the memoryless property of the inter-arrival times in the $M/G/1$ queue to show that, when $\rho < 1$, the expected idle and busy times[5] of the $M/G/1$ satisfy

$$\mathsf{E}[I] = \frac{1}{\lambda} \qquad\qquad \mathsf{E}[U] = \frac{\mathsf{E}[S]}{1-\rho}. \tag{4.1.3}$$

[5] *The busy time of a server is the time between an arrival to an empty system until the server becomes idle again.*

For this, observe that since inter-arrival times are memoryless in the $M/G/1$ queue, the expected time to the next arrival is $1/\lambda$. For the busy time, observe that during the service time of the customer starting a busy time[6] , an expected number $\lambda\,\mathsf{E}[S]$ of new jobs arrives. As each of these jobs generates a busy period of its own, each of which is independent and has the same expected duration $\mathsf{E}[U]$, the expected busy time started by the first job must be equal to the expected service time of this job plus the expected busy periods generated by jobs arriving during this service time; hence $\mathsf{E}[U] = \mathsf{E}[S] + \lambda\,\mathsf{E}[S]\,\mathsf{E}[U]$. But from this, $\mathsf{E}[U] = \mathsf{E}[S]/(1-\rho)$, because $\rho = \lambda\,\mathsf{E}[S] < 1$.

[6] *This customer necessarily arrives at an empty system.*

TF 4.1.1. Claim: If for a queueing system $\lim_{t\to\infty} A(t) = \infty$, then the system is not rate stable.

TF 4.1.2. Claim: $\lim_{t\to\infty} \frac{A(t)}{t} = \lim_{n\to\infty} \frac{A_n}{n}$.

TF 4.1.3. Claim: $\lambda = \delta \implies \rho = 1$, so the queue is not stable.

TF 4.1.4. Claim: A queueing system with $\lambda\,\mathsf{E}[S] > 1$ and a single server can still be rate-stable if jobs are sometimes rejected.

TF 4.1.5. Claim: The load $\rho = \lambda\,\mathsf{E}[S]/c$ of a queueing system with c servers is always at most 1.

Opt 4.1.6. Can you make an arrival process such that $A(t)/t$ does not have a limit?



4.2 (LIMITS OF) EMPIRICAL PERFORMANCE MEASURES

In Section 3.1 we use the arrival times $\{A_k\}$ and the service times $\{S_k\}$ to construct the waiting times $\{W_k\}$, the sojourn times $\{J_k\}$, the arrival process $\{A(t)\}$, the departure process $\{D(t)\}$, and the number in the system $\{L(t)\}$. If the queueing system is rate stable, several long-run average performance measures[1] along sample paths exist almost surely.[2]

[1] *Also known as 'steady-state' or 'stationary' limits.*

[2] *Proving the existence of these limits below requires substantial mathematics; see Asmussen [2003] for details. Here we assume these limits exist.*

As the next example demonstrates, we need to distinguish between sampling the system at arrival times and sampling it over time. Suppose that jobs arrive exactly at the start of an hour and require 59 minutes of service time. If we sample the server occupancy at arrival epochs, the server is always idle, whereas over time it is almost always busy. Thus, what jobs see upon arrival *generally differs* from what the server perceives.

ONE SET OF measures focuses on what customers perceive at the moment of arrival.[3] The expectation and distribution function[4] of the waiting time are obtained from, respectively,

$$\mathsf{E}[W] = \lim_{n\to\infty} \frac{1}{n}\sum_{k=1}^{n} W_k, \qquad \mathsf{P}\{W \le x\} = \lim_{n\to\infty} \frac{1}{n}\sum_{k=1}^{n} \mathbb{1}\{W_k \le x\}. \tag{4.2.1}$$

[3] *Such statistics are often described as 'as seen by arrivals'.*

[4] *This is simply counting.*

For the sojourn time J, we use similar definitions.

Since $L(A_k-)$ denotes the number of jobs in the system just before the kth arrival[5], the average and distribution function of the number of jobs in the system as seen by arrivals are given by

$$\mathsf{E}[L] = \lim_{n\to\infty} \frac{1}{n}\sum_{k=1}^{n} L(A_k-), \qquad \mathsf{P}\{L \le m\} = \lim_{n\to\infty} \frac{1}{n}\sum_{k=1}^{n} \mathbb{1}\{L(A_k-) \le m\}. \tag{4.2.2}$$

[5] *Note the A_k-; since $L(t)$ is right-continuous, we focus on what an* arrival *sees just before its arrival.*

A SECOND SET of performance measures follows by tracking the system's behavior over time and taking the *time-average*.[6] Since at time s the number in the system is $L(s) = A(s) - D(s)$, we define the time-average number of jobs[7] and the time-average fraction of time the system contains at most m jobs, respectively, as

$$\mathsf{E}[L] = \lim_{t\to\infty} \frac{1}{t}\int_0^t L(s)\,\mathrm{d}s, \quad \mathsf{P}\{L \le m\} = \lim_{t\to\infty} \frac{1}{t}\int_0^t \mathbb{1}\{L(s) \le m\}\,\mathrm{d}s. \tag{4.2.3}$$

[6] *These performance measures are 'as seen by the server'.*

[7] *Although the symbol is the same, this expectation need not be the same as (4.2.2).*

TF 4.2.1. Claim: If the limit exists, then

$$\frac{1}{t}\sum_{k=1}^{A(t)} \mathbb{1}\{W_k \le x\} \to \mathsf{P}\{W \le w\}, \quad t \to \infty.$$

TF 4.2.2. The departure process $\{D(t)\}$ can be computed from the set $\{D_k\}$ of departure times as

$$D(t) = \sum_{k=1}^{\infty} \mathbb{1}\{D_k \le t\}.$$

TF 4.2.3. Claim: In general, the time-average number of jobs in the system equals the number of jobs seen by arrivals.

TF 4.2.4. Claim: The fraction of time the system contains n or more jobs can be computed as $\mathsf{P}\{L \geq n\} = \lim_{t\to\infty} \frac{1}{t}\int_0^t \mathbb{1}\{L(s) \geq n\}\,\mathrm{d}s$.

TF 4.2.5. Claim: The empirical distribution of the waiting time $\mathsf{P}\{W \leq x\} = \lim_{n\to\infty} \frac{1}{n}\sum_{k=1}^{n} \mathbb{1}\{W_k \leq x\}$ samples at arrival epochs, hence it captures what customers experience.

Ex 4.2.6. Design a queueing system where the time-average number of jobs in the system differs markedly from what arrivals observe.

Ex 4.2.7. If $L(t)/t \to 0$ as $t \to \infty$, can $\mathsf{E}[L]$ still be positive?

4.3 RENEWAL REWARD THEOREM

In a one-dollar store, each item costs \$1. If every minute a customer arrives and each customer buys exactly 10 items, this shop earns money at a rate of \$10 per minute. In another one-dollar store, customers spend variable amounts, but on average \$10 per customer, and customers come in irregularly, but the average inter-arrival time between two customers is 1 minute. The *renewal reward* theorem says that both shops have the same earning rate, even though the second shop perceives variability while the first does not. In this section, we prove this extremely useful theorem and show how to apply it to a famous inventory system.

WE ARE GIVEN a sequence of strictly increasing epochs $0 = T_0 < T_1 < T_2 < \cdots$ such that $T_k \to \infty$ as $k \to \infty$, and a non-decreasing right-continuous (deterministic) function $t \to R(t)$ with $R(0) = 0$. Let $N(t) = \sum_{k=1}^{\infty} \mathbb{1}\{T_k \leq t\}$ count the number of epochs that occurred during $(0, t]$, and $Z_k = R(T_k) - R(T_{k-1})$ be the increment of $R(t)$ during $(T_{k-1}, T_k]$.[1] Fig. 4.3.1 shows graphically how all concepts relate.

[1] *Here Z_k is not necessarily an inter-arrival time between two jobs.*

Theorem 4.3.1 (Renewal Reward Theorem, $R = \alpha Z$)**.** Assume that the limit $\alpha = \lim_{t\to\infty} N(t)/t$ exists with $0 < \alpha < \infty$. Then, the limit $R = \lim_{t\to\infty} R(t)/t$ exists iff $Z = \lim_{n\to\infty} n^{-1}\sum_{k=1}^{n} Z_k$ exists, in which case $R = \alpha Z$.

—

Proof. As $T_{N(t)} \leq t < T_{N(t)+1}$ and $R(\cdot)$ is non-decreasing,

$$R(T_{N(t)}) \leq R(t) \leq R(T_{N(t)+1}).$$

Hence

$$\frac{N(t)}{t}\frac{R(T_{N(t)})}{N(t)} \leq \frac{R(t)}{t} \leq \frac{R(T_{N(t)+1})}{N(t)+1}\frac{N(t)+1}{t}.$$

Since $R(T_{N(t)}) = \sum_{k=1}^{N(t)} Z_k$ and $N(t)/t \to \alpha$, we conclude that $R(t)/t \to R$ iff $(N(t))^{-1}\sum_{k=1}^{N(t)} Z_k \to Z$. By the same arguments as in Th. 4.1.1, if one of these limits exists, so does the other, completing the proof. □

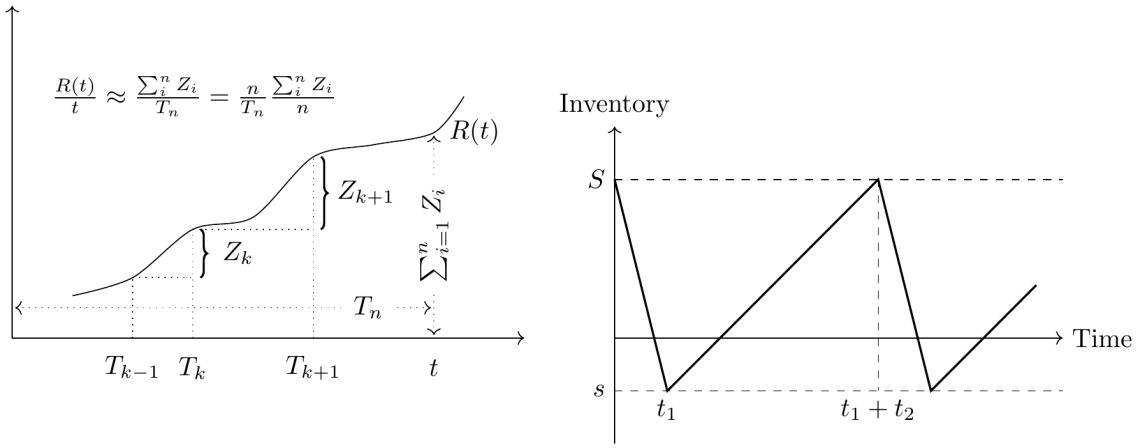


Fig. 4.3.1: The left panel provides a graphical 'proof' of the renewal reward equation $R = \alpha Z$. The right panel shows a sample path of an inventory system with constant production capacity and a fixed demand rate. Since the inventory operates under an (s, S) policy, it moves from cycle to cycle.

THE RENEWAL REWARD theorem finds an easy application in the analysis of the *Economic Production Quantity (EPQ)* model. This is an inventory system in which demand arrives at a fixed rate d and a machine produces items, when on, at a constant rate $r > d$.[2] The inventory is controlled by an (s, S) policy, the lead time $l = 0$, the holding and backlogging cost are h and b per unit per time, and it costs K to switch on the machine. Fig. 4.3.1 shows a sample path of the inventory level. We use the renewal reward theorem (noting that the cumulative cost is non-decreasing) to find the policy parameters[3] that minimize the long-run time-average cost. Clearly, the inventory level repeats over time, so it suffices to study just one cycle that starts and stops when the inventory level hits S. From Fig. 4.3.1 it is simple to see that, respectively,

$$t_1 := \frac{S-s}{d}, \qquad t_2 := \frac{S-s}{r-d}, \qquad T := t_1 + t_2 = \frac{S-s}{d}\frac{r}{r-d},$$

are the time to empty the inventory from level S to s (after which production switches on), the time needed to replenish the inventory from level s to S (after which production switches off), and the cycle time.

Assume next that $s \le 0 < S$.[4] Writing T_+ (T_-) for the amount of time that the inventory is positive (negative) during a cycle, it follows that

$$T_+ = \frac{S}{d} + \frac{S}{r-d}, \qquad T_- = \frac{-s}{d} + \frac{-s}{r-d}, \qquad T = T_+ + T_-,$$

as $s \le 0$. Therefore,[5]

$$\frac{T_+}{T} = \frac{S}{S-s}, \qquad \frac{T_-}{T} = \frac{-s}{S-s}.$$

Since the holding and backlogging cost are proportional to the areas of the corresponding triangles, the average cost of one cycle becomes

$$g = \frac{K}{T} + b\frac{-s}{2}\frac{T_-}{T} + h\frac{S}{2}\frac{T_+}{T}.$$

Substituting the expressions for T_-, T_+, and T in terms of s and S, taking the partial derivatives with respect to s and S and solving for the optimal values by setting $\partial g/\partial s = \partial g/\partial S = 0$ gives[6]

[2] *Stability requires $r \ge d$, but the finite-cycle analysis below requires $r > d$; the case $r = d$ is trivial.*

[3] *That is, the optimal s and S.*

[4] *[4.3.8] asks you to generalize this assumption.*

[5] *This also follows from the geometry of the triangles in the right panel of Fig. 4.3.1.*

[6] *Interestingly, when $r = d$, the cost g^* equals 0. Why is that so?*

$$g^* = \sqrt{2Kd}\sqrt{\frac{hb}{h+b}}\sqrt{1-\frac{d}{r}}, \qquad s^* = -\frac{g^*}{b}, \qquad S^* = \frac{g^*}{h}. \tag{4.3.1}$$

Take the limits $r \to \infty$ and $b \to \infty$ to obtain the *Economic Order Quantity (EOQ)* formula.[7]

[7] *See also [5.8.7].*

$$g^*_{EOQ} = \sqrt{2Kdh} \qquad s^* = 0, \qquad Q^* = \sqrt{\frac{2Kd}{h}},$$

where we write Q^* instead of S^* to indicate that we order in fixed quantities. Observe that the EOQ policy is simultaneously an (s,S) policy, (Q,r) policy, and (T,S) policy.[8]

[8] *Recall the policies of Section 2.5.*

An interesting extension is a system with constraints on cycle length and order size.[9]

[9] *Can you find an optimal policy?*

TF 4.3.1. Consider a machine that fails regularly and needs to be repaired. Assuming that repairs take a negligible amount of time, the time between machine failures has density f_X, and each repair costs c. Claim: The renewal reward theorem tells us that the cost rate equals $\frac{c}{\int_0^\infty x f(x)\,\mathrm{d}x}$.

TF 4.3.2. For a $G/G/1$ queue with $\rho < 1$, write $\mathsf{E}[U]$ for the expected busy time and $\mathsf{E}[I]$ for the expected idle time. Claim:

$$\rho = \mathsf{E}[U]/(\mathsf{E}[U] + \mathsf{E}[I]).$$

TF 4.3.3. An (s,S) policy is a restocking policy where s denotes the size of replenishment orders and S represents the speed at which items are produced.

TF 4.3.4. Claim: In the renewal reward theorem $R = \alpha Z$, the quantity α must be the arrival rate λ.

Ex 4.3.5. Use the renewal reward theorem to relate the load $\rho = \lambda\,\mathsf{E}[S]$ to the limiting fraction of time the server of a stable $G/G/1$ queue is busy.

Ex 4.3.6. We can derive the relation between utilization and the fraction of busy time of the server of a $G/G/1$ queue by considering the inequalities

$$\sum_{k=1}^{A(t)} S_k \geq \int_0^t \mathbb{1}\{L(s) > 0\}\,\mathrm{d}s \geq \sum_{k=1}^{D(t)} S_k. \tag{4.3.2}$$

Explain the inequalities and show that, by taking appropriate limits, we obtain $\lambda\,\mathsf{E}[S] \geq \rho \geq \delta\,\mathsf{E}[S]$.

Ex 4.3.7. Show for the $G/G/1$ queue with $\rho < 1$ that $\rho = \mathsf{E}[U]/(\mathsf{E}[U] + \mathsf{E}[I])$ where $\mathsf{E}[U]$ is the expected busy time and $\mathsf{E}[I]$ the expected idle time.

Ex 4.3.8. For the EPQ model, take T to be the cycle time. Use elementary geometry to show that the average cost $g = K/T + h(s+S)/2$ when $0 \leq s < S$, and $g = K/T - b(s+S)/2$ when $s < S \leq 0$.

4.4 LITTLE'S LAW

There exists a fascinating connection, known as *Little's law*, between the average sojourn time of a job in an input-output system and the long-run time-average number of jobs within that system. To heuristically understand this relationship, imagine a 100 km highway segment between points A and B. Every minute, a car enters the highway and each car maintains a constant speed of 50 km/h. Now, since each car spends 2 hours[1] on the highway, there must be 120 cars[2] on the highway simultaneously. The objective in this section is to demonstrate that, under a few simple conditions, this relation generalizes to stochastic systems, in which case it takes the form $\mathsf{E}[L] = \lambda \mathsf{E}[J]$, i.e., the average number of 'things' in a system is the rate at which these 'things' enter times the average time they spend in the system.[3]

[1] *The sojourn time equals 100 km / 50 km/h = 2 hours.*

[2] *2 h × 60 min/h × 1 car/min = 120 cars.*

[3] *Use the physical dimensions of the components of Little's law to check that* $\mathsf{E}[J] \neq \lambda \mathsf{E}[L]$.

Theorem 4.4.1 (Little's law)**.** If $\delta = \lambda \in (0,\infty)$ and $\frac{1}{n}\sum_{k=1}^{n} J_k \to \mathsf{E}[J] < \infty$ as $n \to \infty$, then the limit $\mathsf{E}[L] := \lim_{t\to\infty} \frac{1}{t}\int_0^t L(s)\,\mathrm{d}s$ exists, and $\mathsf{E}[L] = \lambda \mathsf{E}[J]$.

—

Proof. We give the proof for single-server models.[4] Suppose we charge a job for the time it spends in the system. This can be done at arrival, while present, or upon departure. A comparison of the costs under each of these three charging schemes shows that for some job k

$$J_k \mathbb{1}\{D_k \le t\} \le \int_0^t I_k(s)\,\mathrm{d}s \le J_k \mathbb{1}\{A_k \le t\},$$

where $I_k(s)$ is defined in (3.1.6). Taking the sum over this inequality and using the reasoning that led to (3.1.8) gives[5]

$$\frac{1}{t}\sum_{k=1}^{D(t)} J_k \le \frac{1}{t}\int_0^t \sum_{k=1}^{\infty} I_k(s)\,\mathrm{d}s = \frac{1}{t}\int_0^t L(s)\,\mathrm{d}s \le \frac{1}{t}\sum_{k=1}^{A(t)} J_k.$$

The assumptions imply that $D(t) \to \infty$ as $t \to \infty$. Thus, for the LHS,

$$\lim_{t\to\infty} \frac{1}{t}\sum_{k=1}^{D(t)} J_k = \lim_{t\to\infty} \frac{D(t)}{t}\frac{1}{D(t)}\sum_{k=1}^{D(t)} J_k = \delta \mathsf{E}[J].$$

Likewise, the RHS converges to $\lambda \mathsf{E}[J]$. As $\delta = \lambda < \infty$, the limits of the LHS and RHS are the same. Consequently, the limit $\frac{1}{t}\int_0^t L(s)\,\mathrm{d}s \to \mathsf{E}[L]$ exists and $\lambda \mathsf{E}[J] = \mathsf{E}[L]$. □

[4] *The same result holds for multi-server queues, but then the reasoning is a bit more subtle, because jobs need no longer leave in order of arrival.*

[5] *Observe that* $\sum_{k=1}^{\infty} J_k \mathbb{1}\{D_k \le t\} = \sum_{k=1}^{D(t)} J_k$*, and similarly for* $A(t)$.

The conditions are subtle: assuming only that $\delta = \lambda < \infty$ does not imply that $\lim_{n\to\infty} \frac{1}{n}\sum_{k=1}^{n} J_k < \infty$. For example, consider a single-server queue, take $A_k = k$, and $S_k = 1$ if k is not a square but $S_k = 2$ if k is a square. Then for very large k, $D_k \sim k + \sqrt{k}$. Hence, $A_k/k \to 1$ and $D_k/k \to 1$. However, for large s, $L(s) = A(s) - D(s) \approx \sqrt{s}$, so that $t^{-1}\int_0^t \sqrt{s}\,\mathrm{d}s \to \infty$ as $t \to \infty$.

TF 4.4.1. Claim: Little's law states for a rate-stable $G/G/1$ queue that $\mathsf{E}[J] = \lambda \mathsf{E}[L]$.

TF 4.4.2. Claim: For Little's law to be true for any input-output system the equality

$$\int_0^T L(s)\,\mathrm{d}s = \sum_{k=1}^{A(T)} W_k$$

must hold for all $T \geq 0$.

TF 4.4.3. Consider the server of the $G/G/1$ queue as a system in itself. The time jobs stay in this system is $\mathsf{E}[S]$, and jobs arrive at a rate λ. Claim: It follows from Little's law that the fraction of time the server is busy is $\lambda\,\mathsf{E}[S]$.

TF 4.4.4. Suppose that there are 10 jobs present in the $M/M/1$ queue with arrival rate $\lambda = 3$ and service rate $\mu = 4$ per hour. Claim: The time to clear the system follows from Little's law and is $3 \times 10 = 30$ hours.

TF 4.4.5. Claim: Little's law $\mathsf{E}[L] = \lambda\,\mathsf{E}[J]$ implies that reducing the arrival rate λ always reduces the average number of jobs $\mathsf{E}[L]$ in the system.

Ex 4.4.6. Think of all servers in the (stable) $G/G/c$ queue as being placed in a box. Jobs enter the box at a rate λ and stay for a duration $\mathsf{E}[S]$ in this box. Use Little's law to conclude that the time-average number of busy servers is $\mathsf{E}[L_s] = \lambda\,\mathsf{E}[S]$.

Let's take 'life' itself as an input-output system[6], and ask how many people are born and die in the Netherlands per week.

[6] *An application to actuarial sciences.*

Ex 4.4.7. For simplicity, assume that in 2023 the Netherlands has 16 M inhabitants, and people have an expected lifetime (i.e., the time they remain in the system called 'life') of 80 years. By Little's law, the rate λ of people entering the system (births) is $16M/80 = 0.2M$ per year. For simplicity, if a year has 50 weeks, there are 4000 births per week. If the size of the population remains about constant, also around 4000 people die per week.

As it turns out, CBS records 169521 deaths in 2023, or about 3260 per week when dividing by 52. This is much lower than the estimate, and a more precise computation does not close the gap. Can you explain this paradoxical difference?

4.5 POISSON ARRIVALS SEE TIME AVERAGES

Recall from the example of Section 4.2 that sampling at arrival times can give very different results from sampling over time. Interestingly, as we show here, when jobs arrive as a Poisson process, both sampling methods yield the same value. This result is known as *Poisson arrivals see time-averages* (*PASTA*).

SAY THE SYSTEM is in *state* n at time t when $L(t) = n$. Then define

$$A(n,t) := \sum_{k=1}^{A(t)} \mathbb{1}\{L(A_k-) = n\}$$

as the number of arrivals up to time t that saw the system in state n *upon arrival*.[1] Next, let

[1] *[4.5.6].*

$$Y(n,t) := \int_0^t \mathbb{1}\{L(s) = n\}\, \mathrm{d}s$$

be the total time the system spends in state n during $[0,t]$.

Assume the following limits exist:

$$\lambda := \lim_{t\to\infty} \frac{A(t)}{t}, \qquad \lambda(n) := \lim_{t\to\infty} \frac{A(n,t)}{Y(n,t)}, \tag{4.5.1a}$$

$$p(n) := \lim_{t\to\infty} \frac{Y(n,t)}{t}, \qquad \pi(n) := \lim_{t\to\infty} \frac{A(n,t)}{A(t)}. \tag{4.5.1b}$$

Here, $\lambda(n)$ denotes the arrival rate *when* the system is in state n, $p(n)$ the long-run fraction of time the system spends in state n, and $\pi(n)$ the long-run fraction of *arrivals* that find n customers present.

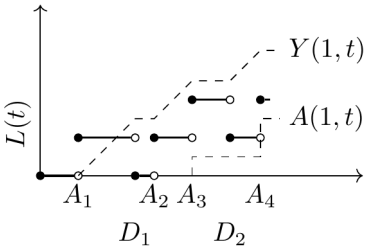


Fig. 4.5.1: We subtracted 1/2 from the graph of $A(1,t)$, otherwise it would overlap with the graph of Y.

Theorem 4.5.1. If all limits in (4.5.1a) exist,

$$\lambda\pi(n) = \lambda(n)p(n).$$

—

Proof. On the one hand,

$$\lim_{t\to\infty} \frac{A(n,t)}{t} = \lim_{t\to\infty} \frac{A(n,t)}{Y(n,t)} \frac{Y(n,t)}{t} = \lambda(n)p(n).$$

On the other,

$$\lim_{t\to\infty} \frac{A(n,t)}{t} = \lim_{t\to\infty} \frac{A(n,t)}{A(t)} \frac{A(t)}{t} = \pi(n)\lambda.$$

□

This result has a simple consequence. Write $\mathscr{S}$ for all states the system can attain.[2] Clearly, any arrival must see the system in some state, hence $\sum_{n\in\mathscr{S}} \pi(n) = 1$. Thus, summing over n in $\lambda\pi(n) = \lambda(n)p(n)$, the overall arrival rate λ satisfies

$$\lambda = \sum_{n\in\mathscr{S}} \lambda(n)p(n). \tag{4.5.2}$$

[2] *When there is no upper bound on L, then $\mathscr{S} = \mathbb{N}$, but when $L(t)$ is bounded by some number K, then $\mathscr{S} = \{0,1,\ldots,K\}$.*

With the above theorem, we can obtain the second result of utmost importance.

Theorem 4.5.2. Suppose $\lambda > 0$. For any state n with $p(n) > 0$, the arrival rate equals the overall arrival rate if and only if the sample probability equals the time-average probability, i.e.,

$$\lambda(n) = \lambda \iff \pi(n) = p(n). \tag{4.5.3}$$

—

Proof. From Th. 4.5.1, $\pi(n) = \lambda(n)p(n)/\lambda$. If $\lambda(n) = \lambda$, the λ's cancel. Conversely, if $\pi(n) = p(n) > 0$, dividing $\lambda\pi(n) = \lambda(n)p(n)$ by $p(n)$ gives $\lambda(n) = \lambda$. □

As the example in the introduction of this section shows, this property is not satisfied in general. However,[3]

[3] *A rigorous proof of this is difficult, see e.g., El-Taha and Stidham Jr. [1998].*

Corollary 4.5.3. If the arrival process is Poisson, then $\lambda(n) = \lambda$, in which case $\pi(n) = p(n)$.

—

BY DISCRETIZING TIME we can give an intuitive explanation of when PASTA holds or fails. If in some period k a job arrives ($a_k = 1$) and the system contains m jobs at the end of period $k-1$, i.e., $L_{k-1} = m$, then we say that the arrival in period k sees m jobs in the system upon arrival. Thus, for k large, $\pi(m) \approx \mathsf{P}\{L_{k-1} = m | a_k = 1\}$. With Bayes' law:

$$\pi(m) \approx \mathsf{P}\{L_{k-1} = m | a_k = 1\} = \frac{\mathsf{P}\{a_k = 1, L_{k-1} = m\}}{\mathsf{P}\{a_k = 1\}}.$$

Observe that L_{k-1} is a function of $\{a_i\}_{i=1}^{k-1}$. If a_k and $\{a_i\}_{i=1}^{k-1}$ are independent, then a_k and L_{k-1} are independent too, and then $\mathsf{P}\{a_k = 1, L_{k-1} = m\} = \mathsf{P}\{a_k = 1\}\,\mathsf{P}\{L_{k-1} = m\}$. By canceling $\mathsf{P}\{a_k = 1\}$ in the fraction, we see that

$$\pi(m) \approx \mathsf{P}\{L_{k-1} = m\} \approx p(m),$$

where the last approximation follows from the fact that $\mathsf{P}\{L_{k-1} = m\}$ is the fraction of periods that $L = m$.[4]

However, PASTA does not hold when a_k depends on L_{k-1}. For example, if $a_k = 0$ when $L_{k-1} > 0$ and $c_k \sim \text{Bern}(d)$ for some small probability d, then $\pi(1) = 0$ while in general $\mathsf{P}\{L_{k-1} = 0\} > 0$.

WITH PASTA, LITTLE'S law and memorylessness we can derive some initial results for the $M/M/1$ queue.[5] By PASTA, an arrival sees on average $\mathsf{E}[L]$ jobs in the system, and the expected time to serve these jobs is $\mathsf{E}[L]\,\mathsf{E}[S]$. This is subtler than it seems: the expected time to serve the jobs in queue is $\mathsf{E}[Q]\,\mathsf{E}[S]$, and by *the memoryless property of* S, the expected remaining time of the job in service (if any) is also $\mathsf{E}[S]$.[6] Next, as the arriving job has to be served itself before leaving, its expected sojourn time $\mathsf{E}[J] = \mathsf{E}[L]\,\mathsf{E}[S] + \mathsf{E}[S]$. With Little's law, $\mathsf{E}[L] = \lambda\,\mathsf{E}[J]$, and therefore,

$$\mathsf{E}[J] = \mathsf{E}[L]\,\mathsf{E}[S] + \mathsf{E}[S] = \lambda\,\mathsf{E}[J]\,\mathsf{E}[S] + \mathsf{E}[S] = \rho\,\mathsf{E}[J] + \mathsf{E}[S].$$

Consequently,

$$\mathsf{E}[J] = \frac{\mathsf{E}[S]}{1-\rho}, \tag{4.5.4a}$$

from which,

$$\mathsf{E}[L] = \lambda\,\mathsf{E}[J] = \frac{\lambda\,\mathsf{E}[S]}{1-\rho} = \frac{\rho}{1-\rho}, \tag{4.5.4b}$$

$$\mathsf{E}[W] = \mathsf{E}[L]\,\mathsf{E}[S] = \frac{\rho}{1-\rho}\,\mathsf{E}[S] \tag{4.5.4c}$$

$$\mathsf{E}[Q] = \lambda\,\mathsf{E}[W] = \frac{\rho^2}{1-\rho}, \tag{4.5.4d}$$

$$\mathsf{E}[L_s] = \mathsf{E}[L] - \mathsf{E}[Q] = \frac{\rho}{1-\rho} - \frac{\rho^2}{1-\rho} = \rho, \tag{4.5.4e}$$

[4] *For mathematically interested students: The assumed strong convergence* $N^{-1}\sum_{k=1}^{N} \mathbb{1}\{L_k = m\} \to p(m)$ *as* $N \to \infty$ *implies the weak convergence* $\mathsf{P}\{L_k = m\} \to p(m)$ *as* $k \to \infty$.

[5] *For the* $M/G/1$ *queue, things are a bit subtler; see [4.5.8] and Section 5.6.*

[6] *This is not true for the* $M/G/1$ *queue.*

[7] *Look this paradox up on the web. You should be aware of such biases in data science.*

[8] *This cute story comes from the book 'Puzzle Math' by G. Gamow and M. Stern,* https://www.arvindguptatoys.com/arvindgupta/puzzlemath.pdf. *The interesting solution can also be found there.*

TO LET YOU think a bit deeper about the PASTA property, and obtaining statistics by sampling in general, consider the next cute riddle. The problem is known as the 'inspection paradox'.[7]

In a small midwestern town[8] there lived a retired railroad engineer named William Johnson. The main line on which he had worked for so many years passed through the town. Mr. Johnson suffered from insomnia and would often wake up at any odd hour of the night and be unable to fall asleep again. He found it helpful, in such cases, to take a walk along the deserted streets of the town, and his way always led him to the railroad crossing. He would stand there thoughtfully watching the track until a train thundered by through the dead of the night. The sight always cheered the old railroad man, and he would walk back home with a good chance of falling asleep.

After a while he made a curious observation; it seemed to him that most of the trains he saw at the crossing were traveling eastward, and only a few were going west. Knowing very well that this line was carrying equal numbers of eastbound and westbound trains, and that they alternated regularly, he decided at first that he must have been mistaken in this reckoning. To make sure, he got his little notebook, and began putting down "E" or "W," depending on which way the first train to pass was traveling. At the end of a week, there were five "E's" and only two "W's" and the observations of the next week gave essentially the same proportion. Could it be that he always woke up at the same hour of night, mostly before the passage of eastbound trains?

Being puzzled by this situation, he decided to undertake a rigorous statistical study of the problem, extending it also to the daytime. He asked a friend to make a long list of arbitrary times such as 9:35 a.m., 12:00 noon, 3:07 p.m., and so on, and he went to the railroad crossing punctually at these times to see which train would come first. However, the result was the same as before. Out of one hundred trains he met, about seventy-five were going east and only twenty-five west. In despair, he called the depot in the nearest big city to find whether some of the westbound trains had been rerouted through another line, but this was not the case. He was, in fact, assured that the trains were running exactly on schedule, and that equal numbers of trains daily were going each way. This mystery brought him to such despair that he became completely unable to sleep and was a very sick man.

Can you reveal the mystery, thereby solving the sleeping problems of Mr. Johnson without prescribing pills?

TF 4.5.1. Claim: The PASTA property implies that $\sum_{n=0}^{\infty} np(n) = \sum_{n=0}^{\infty} n\pi(n)$ for any $G/M/1$ queue.

TF 4.5.2. Consider a stable $M/G/1$ queue. Claim: By the PASTA property, a fraction ρ of the arrivals sees the server occupied, while a fraction $1-\rho$ sees a free server.

TF 4.5.3. Claim: For the $M/G/1$ queue it follows from the PASTA property that

$$\mathsf{E}[J] = \sum_{n=0}^{\infty} \mathsf{E}[W \mid L = n]\,\pi(n) + \mathsf{E}[S].$$

TF 4.5.4. Let $L_S(s)$ be the number of jobs in service at time s in an $M/G/1$ queue. Define

$$\alpha = \lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} L_S(A_k).$$

Claim: PASTA implies that $1-\rho = \alpha$.

TF 4.5.5. Claim: PASTA implies that for any $G/G/1$ queue, $\pi(n) = p(n)$.

Ex 4.5.6. If $\lambda > \delta$, can it happen that $\lim_{t\to\infty} A(n,t)/t > 0$ for some (finite) n?

Ex 4.5.7. When $\lambda \neq \delta$, is $\pi(n) \geq \delta(n)$?

Ex 4.5.8. While (4.5.4c) is true for the $M/M/1$, it is *not true for the* $M/G/1$ *queue*. Why?

4.6 LEVEL CROSSING AND BALANCE EQUATIONS

Consider a sample path of $\{L(t) : t \geq 0\}$ of a queueing or inventory system in which jobs arrive and depart one by one. We say that the sample path *up-crosses level n* at time t when the number of jobs in the system changes from n to $n+1$ due to an arrival, that is, when $L(t-) = n$ and $L(t) = n+1$. The sample path *down-crosses level n* at time t when, due to a departure, $L(t-) = n+1$ and $L(t) = n$. Clearly, the number of up-crossings and down-crossings cannot differ by more than 1 at any time, because a down-crossing of level n can only occur after an up-crossing (or the other way around). This simple idea is a key tool in the analysis of stationary stochastic systems.

SIMILAR TO THE definitions for $A(n,t)$ and $\lambda(n)$, let[1]

$$D(n,t) := \sum_{k=1}^{D(t)} \mathbb{1}\{L(D_k) = n\}, \qquad \mu(n+1) := \lim_{t\to\infty} \frac{D(n,t)}{Y(n+1,t)}, \tag{4.6.1}$$

denote the number of departures up to time t that *leave n jobs behind*, and the departure rate from state $n+1$.

[1] *Note that we write $L(D_k)$ and not $L(D_k-)$: to leave n jobs behind, the system must contain $n+1$ jobs just prior to the departure.*

Theorem 4.6.1. When jobs arrive and depart as single units and the limits (4.5.1a) and (4.6.1) hold,

$$\lambda(n)p(n) = \mu(n+1)p(n+1). \tag{4.6.2}$$

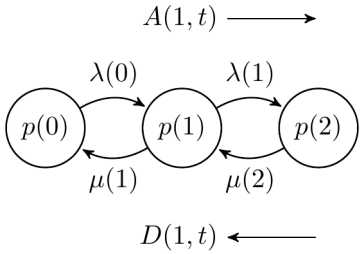


Fig. 4.6.1: Level crossing transitions between states showing arrival and departure rates.

Proof. Since jobs arrive one by one, $|L(t) - L(t-)| \leq 1$ for all $t \geq 0$. This implies that

$$|A(n,t) - D(n,t)| \leq 1, \tag{4.6.3}$$

hence,

$$\lim_{t\to\infty} \frac{A(n,t)}{t} = \lim_{t\to\infty} \frac{D(n,t)}{t}. \tag{4.6.4}$$

The claim follows by noting that

$$\lim_{t\to\infty}\frac{A(n,t)}{t} = \lim_{t\to\infty}\frac{A(n,t)}{Y(n,t)}\frac{Y(n,t)}{t} = \lambda(n)p(n),$$
$$\lim_{t\to\infty}\frac{D(n,t)}{t} = \lim_{t\to\infty}\frac{D(n,t)}{Y(n+1,t)}\frac{Y(n+1,t)}{t} = \mu(n+1)p(n+1).$$

□

SUPPOSE WE CAN specify[2] the arrival and service rates $\lambda(n)$ and $\mu(n)$, then we can easily compute the long-run fraction of time $p(n)$ that the system contains n jobs. To see this, rewrite (4.6.2) as

$$p(n+1) = \frac{\lambda(n)}{\mu(n+1)}p(n). \tag{4.6.5}$$

[2] *In Section 5.1 and onward we will model many queueing situations by making suitable choices for $\lambda(n)$ and $\mu(n)$.*

A simple recursion yields:

$$p(n+1) = \frac{\lambda(n)\lambda(n-1)\cdots\lambda(0)}{\mu(n+1)\mu(n)\cdots\mu(1)}p(0).$$

Assume that the limiting state fractions form a probability distribution and that the following series is finite; then

$$1 = \sum_{n=0}^{\infty} p(n) = p(0)\left(1 + \sum_{n=0}^{\infty}\frac{\lambda(n)\lambda(n-1)\cdots\lambda(0)}{\mu(n+1)\mu(n)\cdots\mu(1)}\right).$$

Thus, we can *normalize* $p(n)$ by setting

$$p(0)^{-1} = 1 + \sum_{n=0}^{\infty}\frac{\lambda(n)\lambda(n-1)\cdots\lambda(0)}{\mu(n+1)\mu(n)\cdots\mu(1)} \tag{4.6.6}$$

Finally, once we have $p(n)$, it is easy to compute the time-average number of jobs in the system and the long-run fraction of time it contains at least n jobs:

$$\mathsf{E}[L] = \sum_{n=0}^{\infty} np(n), \qquad \mathsf{P}\{L \geq n\} = \sum_{i=n}^{\infty} p(i).$$

WITH THE DEFINITIONS of $A(n,t)$ and $D(n,t)$, we establish a relation between $\pi(n)$ and $\delta(n)$, i.e., statistics as obtained by arrivals and departures. Define, analogous to $\pi(n)$,

$$\delta(n) := \lim_{t\to\infty}\frac{D(n,t)}{D(t)}$$

as the long-run fraction of jobs that leave n jobs *behind*. From (4.6.4) and using that $A(n,t) \approx D(n,t)$,

$$\frac{A(t)}{t}\frac{A(n,t)}{A(t)} = \frac{A(n,t)}{t} \approx \frac{D(n,t)}{t} = \frac{D(t)}{t}\frac{D(n,t)}{D(t)}.$$

Taking limits at the left and right, and using (4.1.1), we obtain for the $G/G/c$ queue[3]

$$\lambda\pi(n) = \delta\delta(n). \tag{4.6.7}$$

[3] *Because customers arrive and leave as single units in a (rate-stable) $G/G/c$ queue.*

[4] *Blocked jobs do not enter, hence do not produce up-crossings.*

Hence,

$$\lambda = \delta \implies \pi(n) = \delta(n) \quad \text{for every } n,$$
$$\pi(n) = \delta(n) > 0 \implies \lambda = \delta. \tag{4.6.8}$$

WHEN JOBS CAN be blocked, we must distinguish between jobs that arrive and those that actually enter.[4] To handle this, define

$$A^+(n,t) := \sum_{k=1}^{A(t)} \mathbb{1}\{L(A_k-) = n\}\, \mathbb{1}\{L(A_k) \geq n+1\}, \quad \lambda^+(n) := \lim_{t\to\infty} \frac{A^+(n,t)}{Y(n,t)}, \tag{4.6.9}$$

i.e., $A^+(n,t)$ counts arrivals that up-cross level n, and $\lambda^+(n)$ corresponds to the rate at which jobs *enter* when the system contains n jobs.[5] We should replace $\lambda(n)$ by $\lambda^+(n)$ at all other relevant places, for instance, (4.6.2) becomes

$$\lambda^+(n)p(n) = \mu(n+1)p(n+1). \tag{4.6.10}$$

IT IS IMPORTANT to realize that level-crossing arguments in the simple form as presented above cannot always be applied, since it is not always possible to split the state space into two disjoint subsets by 'drawing a line' between two states. For a more general approach, we focus on a single state and count how often this state is entered and left.

Specifically, define $I(n,t) = A(n-1,t) + D(n,t)$ as the number of times the queueing process enters state n either by an arrival from state $n-1$ or a departure leaving n jobs behind. Similarly, $O(n,t) = A(n,t) + D(n-1,t)$ counts how often state n is left either by an arrival (to state $n+1$) or a departure (to state $n-1$). Just like (4.6.3), it is evident that $|I(n,t) - O(n,t)| \leq 1$, and this implies[6]

$$\lambda(n-1)p(n-1) + \mu(n+1)p(n+1) = (\lambda(n) + \mu(n))p(n). \tag{4.6.11}$$

These equations hold for any $n \geq 1$ and are known as the *balance equations*. These equations are useful for analyzing queueing networks.

LEVEL-CROSSING ARGUMENTS CAN also be applied to continuous systems. For instance, we can find a closed-form expression for the waiting time of the $M/G/1$ queue. However, this is a bit more complicated as the argument hinges on an integral equation. If you do not like developing mathematical skills, you may skip this derivation.

Suppose that the waiting time W of the $M/G/1$ queue has a density f, that is, the cdf $F(x) = \mathsf{P}\{W \leq x\}$ has a derivative $f(x)$ for all $x > 0$.[7,8] Consider some level $x > 0$. By PASTA, the rate of arriving jobs that see a waiting time $y \in [0,x)$ is equal to $\lambda f(y)$.[9] To up-cross level x from state $y < x$, the service time must be at least $x - y$, the probability of which is $G(x-y) = \mathsf{P}\{S > x-y\}$. Adding up the rates from all workloads below x, including the atom at zero, gives the up-crossing rate $\lambda F(0)G(x) + \lambda \int_0^x f(y)G(x-y)\,\mathrm{d}y$. The down-crossing rate equals $f(x)$ because the server drains work at unit rate, so level x is crossed downward whenever the workload is at x. Equating these rates gives $\lambda F(0)G(x) + \lambda \int_0^x f(y)G(x-y)\,\mathrm{d}y = f(x)$ for $x > 0$. For $x = 0$, we need to consider the atom $\mathsf{P}\{W = 0\}$.

[5] *$\lambda(n) > \lambda^+(n)$ when a fraction of jobs is blocked, otherwise $\lambda(n) = \lambda^+(n)$.*

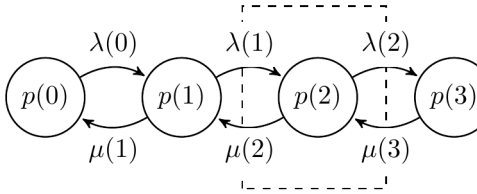


Fig. 4.6.2: Balance equations: counting transitions in and out of a state.

[6] *[4.6.10].*

[7] *A formal proof would first establish that f exists.*

[8] *At $x = 0$, F is only differentiable from the right because $\mathsf{P}\{W = 0\} > 0$, while $\mathsf{P}\{W \leq x\} = 0$ for $x < 0$.*

[9] *Observe that we use the splitting property of Poisson processes.*

In that case, $\lambda \mathsf{P}\{W=0\} = f(0+)$. We can combine these observations into the following level-crossing equation:

$$\lambda \int_0^x G(x-y)\,\mathrm{d}F(y) = f(x). \tag{4.6.12}$$

Using this integral equation

$$\begin{aligned}
\mathsf{E}[W] &= \int_0^\infty x f(x)\,\mathrm{d}x = \lambda \int_0^\infty x \int_0^x G(x-y)\,\mathrm{d}F(y)\,\mathrm{d}x \\
&= \lambda \int_0^\infty \int_0^\infty x\, \mathbb{1}\{y \le x\}\, G(x-y)\,\mathrm{d}x\,\mathrm{d}F(y) \\
&= \lambda \int_0^\infty \int_0^\infty (u+y) G(u)\,\mathrm{d}u\,\mathrm{d}F(y) \\
&\overset{1}{=} \lambda \int_0^\infty \left(\mathsf{E}\left[S^2\right]/2 + y\,\mathsf{E}[S]\right)\mathrm{d}F(y) \\
&\overset{2}{=} \lambda\,\mathsf{E}\left[S^2\right]/2 + \lambda\,\mathsf{E}[S]\,\mathsf{E}[W].
\end{aligned}$$

Step 1 follows from substituting $u = x - y$, and 2 from the fact that $\mathsf{E}[W] = \int_0^\infty y\,\mathrm{d}F(y)$. Assuming $\rho < 1$ and $\mathsf{E}[W] < \infty$, bringing $\lambda\,\mathsf{E}[S]\,\mathsf{E}[W] = \rho\,\mathsf{E}[W]$ to the left and dividing by $1-\rho$ gives

$$\mathsf{E}[W] = \lambda \frac{\mathsf{E}\left[S^2\right]}{2(1-\rho)}.$$

As a last point of interest, let us derive the density of the waiting time for the $M/M/1$ queue. As in this case $S \sim \mathrm{Exp}(\mu)$, f must satisfy

$$f(x) = \lambda \int_0^x e^{-(x-y)\mu}\,\mathrm{d}F(y) = \lambda \int_0^x e^{-(x-y)\mu} f(y)\,\mathrm{d}y + \lambda e^{-x\mu} F(0).$$

Multiplying both sides by $e^{\mu x}$ and defining $g(x) = e^{\mu x} f(x)$, this can be rewritten to $g(x) = \lambda \int_0^x g(y)\,\mathrm{d}y + \lambda F(0)$. Differentiating the LHS and RHS with respect to x gives the differential equation $g'(x) = \lambda g(x)$. Hence, $g(x) = Ae^{\lambda x}$, and therefore, $f(x) = Ae^{-(\mu-\lambda)x}$. At $x = 0$, $f(0) = \lambda \mathsf{P}\{W=0\} = \lambda p(0)$. Since $p(0) = 1-\rho$, it follows that

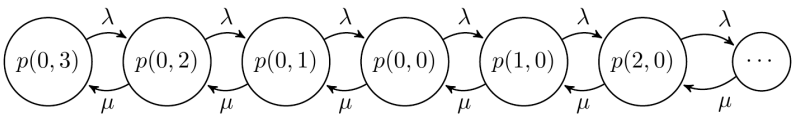
$$f(x) = \lambda(1-\rho)e^{-\mu(1-\rho)x}, \quad x > 0. \tag{4.6.13}$$

Another way to obtain this result is to condition on the number L of jobs found upon arrival, and then using that the total service of these L customers has a gamma density:[10]

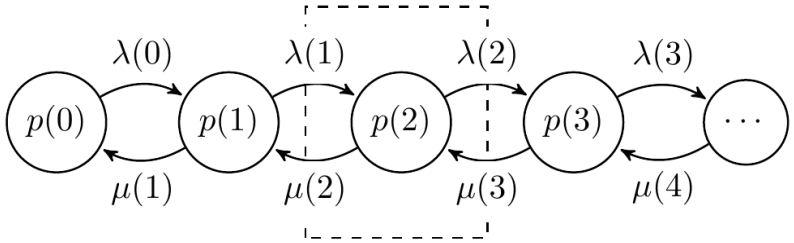
$$\begin{aligned}
f(x) &= \sum_{k=1}^\infty f_{W|L}(x \mid k)\,\mathsf{P}\{L=k\} \\
&= \sum_{k=1}^\infty f_{S_1+\cdots+S_k}(x)\,\mathsf{P}\{L=k\} = (1-\rho)\sum_{k=1}^\infty \rho^k \mu \frac{(\mu x)^{k-1}}{(k-1)!} e^{-\mu x} \\
&= (1-\rho)\lambda e^{-\mu x} \sum_{k=0}^\infty \rho^k \frac{(\mu x)^k}{k!} = (1-\rho)\lambda e^{-\mu(1-\rho)x}.
\end{aligned}$$

[10] *If $L(t) = k > 0$, the new arrival finds one job at the server and $k-1$ in queue. Each of these jobs in the system has to be served before it is the turn of the new job.*

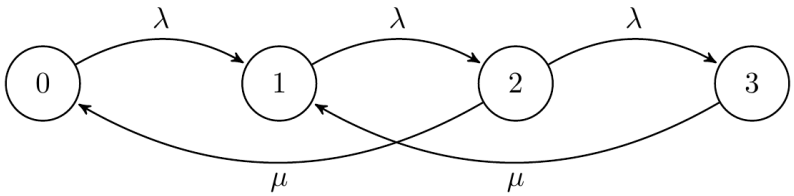
TF 4.6.1. In a large hotel, taxis arrive at a rate of $\lambda = 15$ per hour and riders' parties arrive at a rate of $\mu = 12$ per hour. Whenever taxis are waiting, riders are served immediately upon arrival. Whenever riders are waiting, taxis are loaded immediately upon arrival. A maximum of three taxis can wait at a time (other cabs must go elsewhere). Let $p(i,j)$ be the steady-state probability that there are i parties of riders and j taxis waiting at the hotel. Claim: the transitions are modeled by the graph below.



TF 4.6.2. Claim: To obtain *balance equations*, we do not count the number of up- and down-crossings of a level. Instead, we count how often a box around a state, such as state 2 in the figure below, is crossed from inside and outside.



TF 4.6.3. Consider the $M^2/M^2/1/3$ queue. The graph below shows all relevant transitions.



TF 4.6.4. For the $G/G/1$, the difference between the number of 'out-transitions' and the number of 'in-transitions' is at most 1 for all t. As a consequence,

$$\begin{aligned}
&\text{transitions out} \approx \text{transitions in} \iff \\
&A(n,t) + D(n-1,t) \approx A(n-1,t) + D(n,t) \iff \\
&\frac{A(n,t) + D(n-1,t)}{t} \approx \frac{A(n-1,t) + D(n,t)}{t} \iff \\
&\frac{A(n,t)}{t} + \frac{D(n-1,t)}{t} \approx \frac{A(n-1,t)}{t} + \frac{D(n,t)}{t}.
\end{aligned}$$

Thus, under proper technical assumptions (which you can assume to be satisfied) this becomes for $t \to \infty$,

$$(\lambda(n) + \mu(n))\,p(n) = \lambda(n-1)\,p(n-1) + \mu(n+1)\,p(n+1).$$

We claim that if we specialize this result for the $M/D/1$ queue we get $\lambda(n) = \lambda$ and $\mu(n) = \mu$, hence using PASTA,

$$(\lambda + \mu)\pi(n) = \lambda\pi(n-1) + \mu\pi(n+1).$$

TF 4.6.5. Claim: The level-crossing equation $\lambda(n)\,p(n) = \mu(n+1)\,p(n+1)$ holds for any queueing system in which jobs arrive and depart one at a time.

Ex 4.6.6. Consider the following (silly) queueing process. At times $0, 2, 4, \ldots$ customers arrive, each customer requires 1 unit of service, and there is one server.

1. Find an expression for $A(n,t)$, when $L(0) = 0$.
2. Show that $\pi(0) = 1$ and $\pi(n) = 0$, for $n > 0$.
3. Check that $\lambda\pi(n) = \lambda(n)p(n)$.
4. Find an expression for $Y(n,t)$.
5. Compute $p(n)$ and $\lambda(n)$.
6. Compute $D(n,t)$ and $\mu(n+1)$ for $n \geq 0$.
7. Compute $\lambda(n)p(n)$ for $n \geq 0$, and check $\lambda(n)p(n) = \mu(n+1)p(n+1)$.

Ex 4.6.7. Derive $\mathsf{E}[L] = \sum_{n=0}^{\infty} np(n)$ from (4.2.3).

Ex 4.6.8. Show for the $M/G/1$ that with probability ρ a job leaves behind a busy server.



Ex 4.6.9. Consider a single server that serves one queue and serves only in batches of 2 jobs at a time (so never 1 job or more than 2 jobs). At most 3 jobs fit in the system. Single jobs arrive as a Poisson process with λ. Due to blocking, the entering rate is $\lambda^+(n) = \lambda$ for $n < 3$ and $\lambda^+(3) = 0$. The batch service times are exponentially distributed with mean $1/\mu$, so service completions occur at rate μ in states 2 and 3, and cannot occur in states 0 and 1.

Make a graph of the state-space and show, with arrows, the transitions that can occur and use level-crossing arguments to express the steady-state probabilities $p(n), n = 0, \ldots, 3$ in terms of λ and μ.

Ex 4.6.10. Show (4.6.11) from $|I(n,t) - O(n,t)| \leq 1$.

4.7 GRAPHICAL SUMMARIES

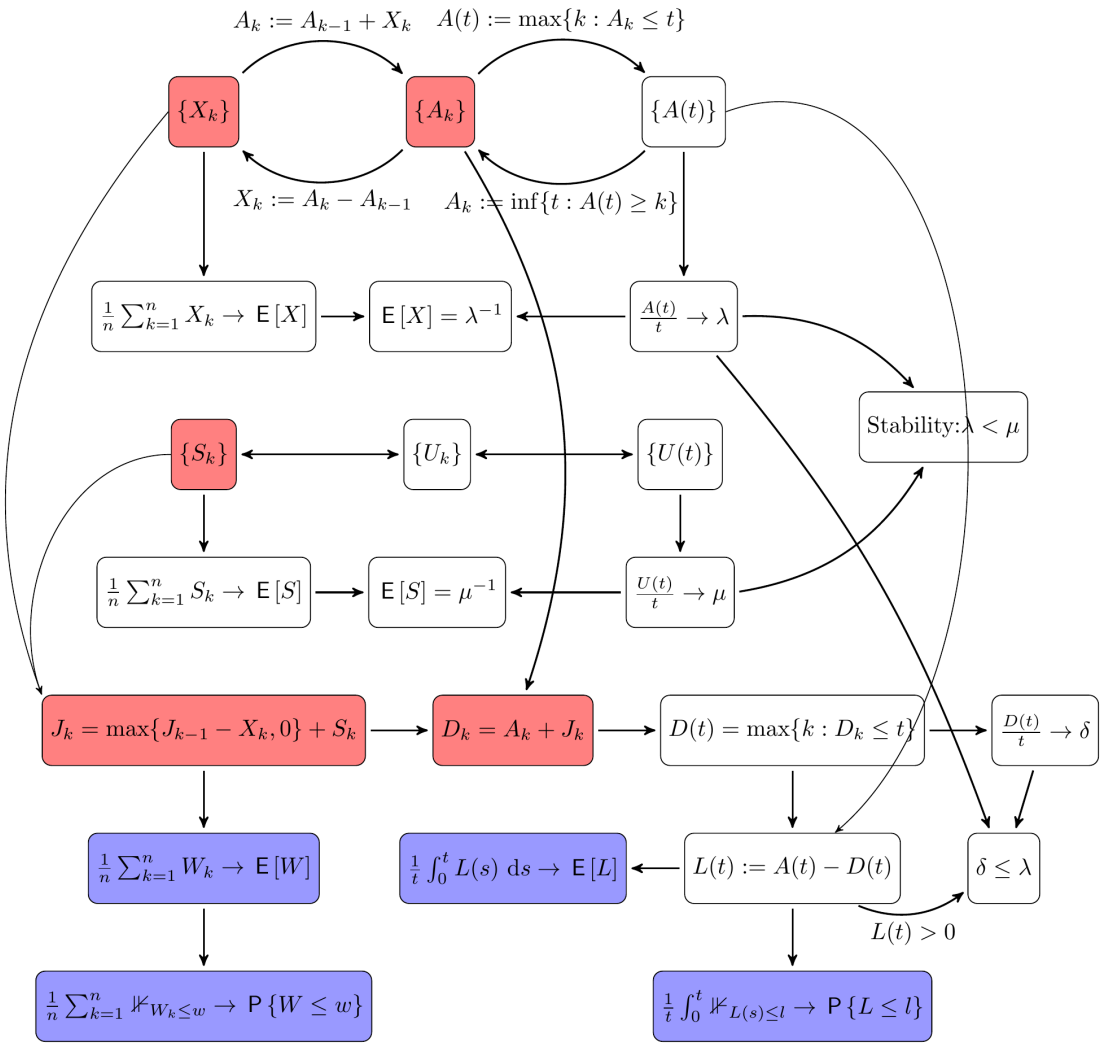


Fig. 4.7.1: An overview of the relations used to construct a single-server queueing process in continuous time; the label $\lambda < \mu$ gives a sufficient stability condition under the usual iid assumptions.

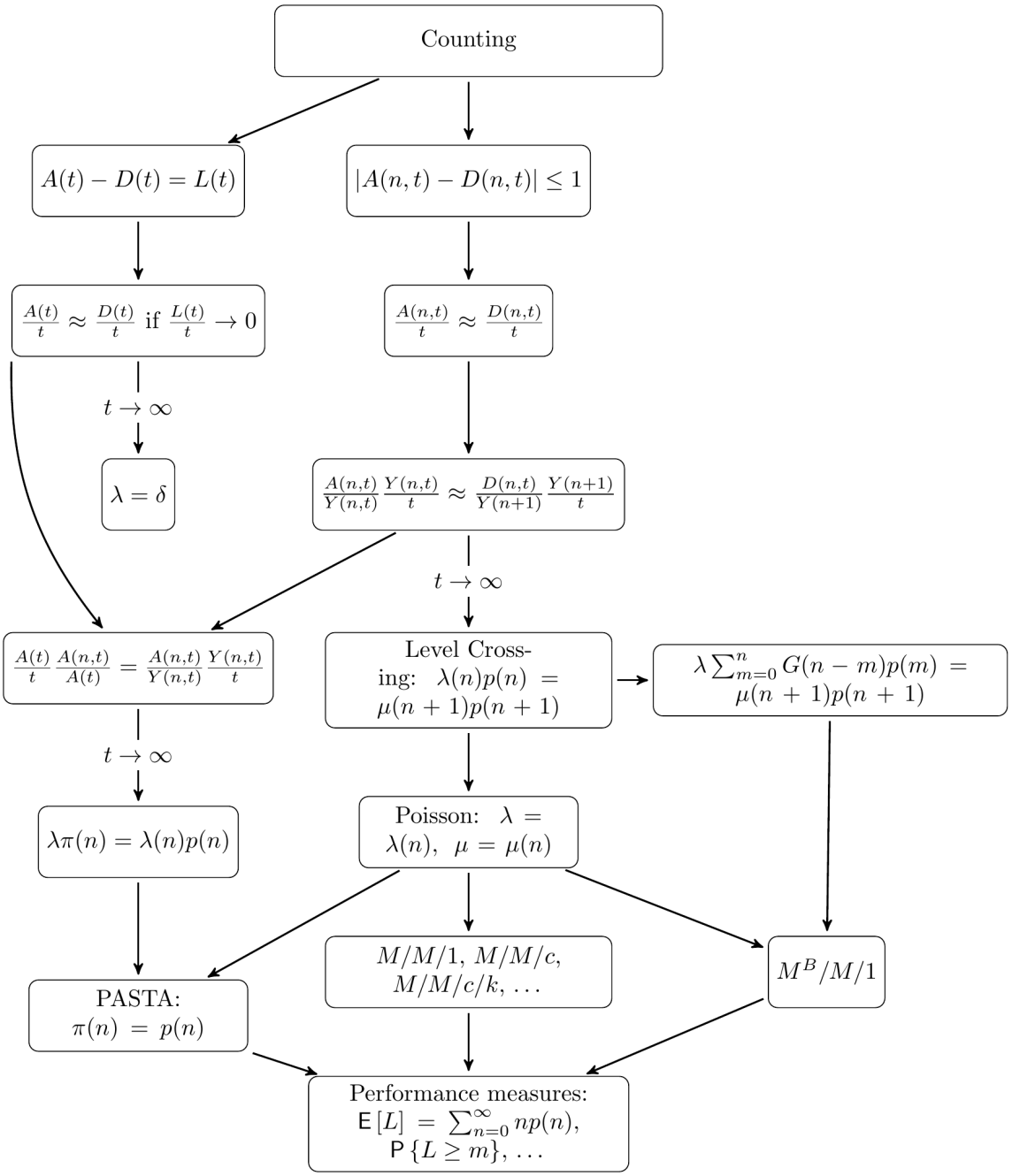


Fig. 4.7.2: An overview of results derived from level-crossing arguments; the box $\mu(n) = \mu$ applies to a busy exponential single server, whereas for $M/M/c$ and $M/M/c/K$ the total departure rate is $\mu(n) = \min(n, c)\mu$.

5

EXACT MODELS

In this chapter we use the concepts of sample-path analysis and level-crossing, developed in Chapter 4, to model and analyze many single-station queueing systems in steady state. Section 5.1 considers several variants of the $M(n)/M(n)/1$ queue. In Section 5.2 we apply the models numerically to production and staffing decisions. Section 5.3 discusses the $M^X/M/1$ queue and blocking policies. This analysis paves the way for deriving the formula for the expected waiting time of the $M/G/1$ queue; Section 5.4 provides the details. Section 5.5 then derives the queue length distribution of the $M/G/1$ queue. Section 5.6 discusses an example of the control of a queueing system. Section 5.7 and Section 5.8 provide models for inventory systems controlled by the base-stock policy or the (Q,r) policy. Section 5.9 develops an algorithm to compute the long-run average cost for the (s,S) and (T,S) inventory policies.

5.1 THE $M(n)/M(n)/1$ QUEUE

In this section, we specify appropriate functions for $\lambda(n)$ and $\mu(n)$ in (4.6.2), or $\lambda^+(n)$ in (4.6.10) in case of blocking, for several of the queueing models discussed in Section 3.2. We emphasize that understanding how to specify the level-crossing equations is more important than deriving closed-form expressions for the performance measures.[1] Further, it is assumed that a system without blocking is stable.[2]

[1] *The code of Section 5.2 deals with the numerical aspects.*

[2] *A system that blocks demand is stable anyway.*

IN THE $M(n)/M(n)/1$ queue the inter-arrival and service times are exponentially distributed conditional on $L=n$, that is,

$$X|L=n \sim \text{Exp}\,(\lambda(n)), \qquad S|L=n \sim \text{Exp}\,\big(\mu(n)\big).$$

Somewhat more specifically, similar to (1.3.6), we require that[3]

$$\begin{aligned} \mathsf{P}\{L(t+\Delta t)=n+1\,|\,L(t)=n\} &= \lambda^+(n)\Delta t + o(\Delta t), \quad n\geq 0,\\ \mathsf{P}\{L(t+\Delta t)=n-1\,|\,L(t)=n\} &= \mu(n)\Delta t + o(\Delta t), \quad n\geq 1. \end{aligned}$$

[3] *In the general case, arrivals need not enter. We therefore consider λ^+ rather than λ.*

Note that, when all arrivals are accepted[4] , we write $\lambda(n)$, but otherwise, we use $\lambda^+(n)$. The next models are all examples of the $M(n)/M(n)/1$ queue with different choices for $\lambda(n)$ (or $\lambda^+(n)$) and $\mu(n)$.

[4] *In which case $\lambda^+(n)=\lambda(n)$.*

THE $M/M/1$ QUEUE has a constant arrival and service rate, so

$$\lambda(n)=\lambda^+(n)=\lambda, \qquad \mu(n)=\mu.$$

The level-crossing equations (4.6.5) result in

$$p(n+1)=\frac{\lambda(n)}{\mu(n+1)}p(n)=\frac{\lambda}{\mu}p(n)=\rho\,p(n)=\rho^{n+1}p(0), \quad \rho=\frac{\lambda}{\mu},$$

and from (4.6.6),

$$p(0)^{-1} = \sum_{n=0}^{\infty} \rho^n = \frac{1}{1-\rho} \quad \Longrightarrow \quad p(n) = (1-\rho)\rho^n, \quad n \geq 0. \tag{5.1.1}$$

With this expression for $p(n)$, it is easy to compute the key performance measures. The load equals the utilization since all jobs are accepted and served. With a bit of algebra,[5]

$$\mathsf{E}[L] = \frac{\rho}{1-\rho}, \qquad \mathsf{E}[L_s] = \rho, \tag{5.1.2}$$

$$\mathsf{E}[Q] = \mathsf{E}[L] - \mathsf{E}[L_s] = \frac{\rho^2}{1-\rho}, \qquad \mathsf{P}\{L > n\} = \rho^{n+1}. \tag{5.1.3}$$

THE $M/M/c$ QUEUE has c identical servers to process jobs, thus

$$\lambda(n) = \lambda^+(n) = \lambda, \qquad \mu(n) = \mu \min\{n, c\}.$$

Closed-form expressions for $\mathsf{E}[Q]$ and $p(n)$ exist[6] , but are not very informative. However, two measures are particularly simple. The load is $\rho = \lambda/c\mu$, and

$$\mathsf{E}[L_s] \overset{1}{=} \sum_{n=0}^{\infty} \min\{n, c\}\, p(n) \overset{2}{=} \frac{\lambda}{\mu},$$

where 1 shows how to compute $\mathsf{E}[L_s]$ as an expectation, and 2 follows from Little's law because jobs spend an average time $\mathsf{E}[S] = 1/\mu$ in service, and jobs arrive at a rate λ at the servers.[7]

THE $M/M/1/K$ QUEUE blocks jobs when $L \geq K$, hence[8]

$$\lambda(n) = \lambda, \qquad \lambda^+(n) = \lambda \mathbb{1}\{n < K\}, \qquad \mu(n) = \mu.$$

This gives $\lambda p(n) = \mu p(n+1)$ for $n < K$, and $p(n) = 0$ for $n > K$, because $\lambda^+(n) = 0$ for $n \geq K$. Thus, in general, $\lambda^+(n)p(n) = \mu(n+1)p(n+1)$, from which follows that

$$\rho = \frac{\lambda}{\mu}, \qquad p(n) = \frac{1-\rho}{1-\rho^{K+1}}\rho^n.$$

There is a subtle point to make here. When the system contains K jobs, the rate at which jobs arrive at the system is $\lambda(K) = \lambda$ because jobs still *arrive* as a Poisson process. However, the rate at which jobs *enter* is $\lambda^+(K) = 0$, consequently,

$$\lambda\pi(K) = \lambda(K)p(K) \neq \lambda^+(K)p(K) = 0.$$

Since $\lambda(K) = \lambda$, we still have[9] $\pi(K) = p(K)$, and therefore the *fraction* of jobs rejected upon arrival is equal to $p(K)$. Note that because jobs are blocked, the utilization is no longer equal to the load $\rho = \lambda/\mu$. Specifically, if we define the mean acceptance rate as

$$\lambda^+ = \sum_{n=0}^{\infty} \lambda^+(n)p(n), \tag{5.1.4}$$

we see for the $M/M/1/K$ queue that $\lambda^+ = \lambda(1 - p(K))$. With this, the utilization becomes

$$\tilde{\rho} = \lambda^+/\mu = \lambda(1-p(K))/\mu = \mathsf{E}[L_s], \tag{5.1.5}$$

where the last equation follows from Little's law.

[5] *[5.1.8].*

[6] *[5.1.9].*

[7] *Note that* $\mathsf{E}[L_s] \neq \rho$.

[8] *Thus,* $\lambda(n) \neq \lambda^+(n)$.

[9] *PASTA.*

[10] *Hence, it combines the* $M/M/c$ *and the* $M/M/1/K$ *queue.*

THE $M/M/c/K$ QUEUE is a multi-server queue with blocking[10] ,

$$\lambda(n) = \lambda, \qquad \lambda^+(n) = \lambda \mathbb{1}\{n < K\}, \qquad \mu(n) = \mu \min\{n, c\}.$$

THE $M/M/c/c$ QUEUE [11] blocks jobs when all of its c servers are occupied,[12]

$$\lambda(n) = \lambda, \qquad \lambda^+(n) = \lambda \mathbb{1}\{n < c\}, \qquad \mu(n) = n\mu.$$

When $n < c$, $\lambda p(n) = (n+1)\mu p(n+1)$, hence

$$p(n+1) = \frac{\lambda}{(n+1)\mu} p(n) = \cdots = \frac{\lambda^{n+1}}{(n+1)!\mu^{n+1}} p(0).$$

The normalizing constant $p(0)$ follows from the constraint $1 = \sum_{n=0}^{c} p(n)$. As there are as many servers as jobs, $\mathsf{E}[Q] = 0$. Moreover, jobs get accepted at a rate $1 - p(c)$, so that by PASTA,

$$\mathsf{E}[L] = \mathsf{E}[L_s] = \frac{\lambda}{\mu}\big(1 - p(c)\big).$$

THE $M/M/\infty$ QUEUE assigns a free server to each arrival,[13]

$$\lambda(n) = \lambda^+(n) = \lambda, \qquad \mu(n) = n\mu.$$

By taking the limit $c \to \infty$ in the expressions of the $M/M/c/c$ queue[14] we obtain

$$p(n) = e^{-\lambda/\mu} \frac{(\lambda/\mu)^n}{n!}, \qquad \mathsf{E}[Q] = 0, \qquad \mathsf{E}[L] = \mathsf{E}[L_s] = \frac{\lambda}{\mu}.$$

We see that the number of busy servers in the $M/M/\infty$ queue is Poisson distributed with parameter $\lambda \mathsf{E}[S]$.

A FINITE CALLING population has a finite number N of jobs. This model is useful for analyzing repair systems and inventory systems of spare parts. To see this, consider a factory with N machines and c mechanics.[15] A machine can be in one of two states: working or failed. When a machine breaks down, it waits at the repair department until repaired. When n machines are in repair, there are still $N - n$ machines running. Thus, if λ is the rate at which an individual machine fails,

$$\lambda(n) = \lambda(N - n), \qquad \mu(n) = \mu \min\{n, c\},$$

A CUSTOMER BALKS when finding the queue too long and deciding not to join. Balking differs from blocking. In the latter case, $\lambda^+(n) = \lambda \mathbb{1}\{n < K\}$; in the former case, a fraction of the customers may already choose not to join the system at a lower level than K. A simple model for balking is

$$\lambda^+(n) = \lambda \frac{[K - n]^+}{K}, \qquad \mu(n) = \mu.$$

TF 5.1.1. Consider the level-crossing analysis of the $M(n)/M(n)/1$ queue. Claim: it is necessary that the inter-arrival times of jobs are iid.

[11] *Also known as the Erlang-B model.*

[12] *Why is it not necessary to write* $\mu(n) = \mu \min\{c, n\}$*?*

[13] *This system is said to have* ample *servers.*

[14] *Or the* $M/M/c$ *queue.*

[15] *When* $c = N$*, we obtain the Ehrenfest model of diffusion, which is used to provide some insight into the second law of thermodynamics.*

TF 5.1.2. For the $M/M/1$ queue, the following reasoning leads to the expected number of jobs in the system.

$$M_L(s) = \mathsf{E}\left[e^{sL}\right] = \sum_{n=0}^{\infty} e^{sn} p(n) = (1-\rho)\sum_{n} e^{sn}\rho^n$$
$$= \frac{1-\rho}{1-e^s\rho},$$

where we assume that s is such that $e^s\rho < 1$. Then,

$$M_L'(s) = (1-\rho)\frac{1}{(1-e^s\rho)^2}e^s\rho.$$

Claim: This implies $\mathsf{E}[L] = M_L'(0) = \rho/(1-\rho)$.

TF 5.1.3. Claim: To model the $M/M/c/c$ queue as an $M(n)/M(n)/1$ queue, we must set $\lambda^+(n) = \lambda$ for all n.

TF 5.1.4. Claim: For the $M/M/c$ queue, $\mathsf{E}[Q] = \sum_{n=0}^{\infty} \max\{n-c, 0\}\, p(n)$, wherein $p(n) = \mathsf{P}\{L = n\}$.

TF 5.1.5. Claim: For the $M/M/c$ queue, $p(n) = p(0)\frac{(\lambda/\mu)^n}{n!}$ for all $n \geq 0$.

Ex 5.1.6. Show that for the $M/M/1$ queue, $\mathsf{E}[Q] = \sum_{n=1}^{\infty}(n-1)\pi(n)$ and use this to find $\mathsf{E}[Q] = \rho^2/(1-\rho)$.

Ex 5.1.7. Derive the steady-state probabilities $p(n)$ for a queue with a finite calling population with N jobs and N servers. What happens if $N \to \infty$?

Ex 5.1.8. Use moment-generating functions to derive $\mathsf{E}[L]$ and $\mathsf{V}[L]$ for the $M/M/1$ queue, and show that $\mathsf{P}\{L > n\} = \rho^{n+1}$.

In the remainder, we derive the KPIs of the above models.[16]

[16] *These exercises are optional, i.e, you may skip them if you don't like algebra.*

Opt 5.1.9. Derive the following expressions for the $M/M/c$ queue:

$$\rho = \frac{\lambda}{c\mu}, \tag{5.1.6a}$$
$$p(n) = \frac{1}{G}\frac{1}{\prod_{k=1}^{n}\min\{c,k\}}(c\rho)^n, \quad n \geq 1, \tag{5.1.6b}$$
$$G = \frac{1}{p(0)} = \sum_{n=0}^{c-1}\frac{(c\rho)^n}{n!} + \frac{(c\rho)^c}{(1-\rho)c!}, \tag{5.1.6c}$$
$$\mathsf{E}[Q] = \sum_{n=c}^{\infty}(n-c)p(n) = \frac{(c\rho)^c}{c!G}\frac{\rho}{(1-\rho)^2} \tag{5.1.6d}$$
$$\tag{5.1.6e}$$

Opt 5.1.10. Check that the performance measures of the $M/M/c$ queue reduce to those of the $M/M/1$ queue if $c = 1$.

Opt 5.1.11. Show that (5.1.5) holds.

Opt 5.1.12. Show that as $K \to \infty$, the performance measures of the $M/M/1/K$ converge to those of the $M/M/1$ queue.

5.2 APPLICATIONS OF THE $M(n)/M(n)/1$ QUEUEING MODEL

In this section we apply the tools developed above to a number of interesting queueing systems. First, we discuss how to use the $M(n)/M(n)/1$ queue to organize a part of a production process. Second, we show how to plan the number of servers at a call center such that the average queue length remains small.

RILEY IS AN operations manager at a factory that produces custom metal parts for various industries. Riley's factory has a single heat treatment furnace to temper the metal parts. Recently, Riley noticed an increase in customer orders, leading to long delays and dissatisfied clients.

1. Should Riley sometimes block jobs to control the waiting time?
2. Should Riley buy another oven to increase the capacity?
3. If Riley bought a second oven, the utilization of both ovens may become quite low. However, as an oven is costly to heat up, it may be favorable to only switch on a second oven when queues are long.

To compare the effect of these different strategies, Riley hires us to model the operation of the oven using queueing theory. We consider the following metrics: average job waiting times, average number of jobs in the system, machine utilization, and job acceptance rate. For ease[1] , we assume that jobs arrive as a Poisson process with a rate λ jobs per hour, and production times are $\sim \text{Exp}(\mu)$ with $1/\mu = \mathsf{E}[S]$.

[1] *At a later stage we can make more complicated models.*

We implemented the level-crossing recursions and the KPI formulas from the previous section for each choice of the arrival and service rates. For the numerical comparison, take $\lambda = 1.9$ and $\mu = 2$ per hour. The results are as follows. Here, $\bar{\lambda}$ is the acceptance rate and ρ is the capacity utilization.

Policy	$\mathsf{E}[W]$	$\mathsf{E}[L]$	$\bar{\lambda}$	ρ	Loss
One oven	9.50	19.00	1.90	0.95	0.00
Capacity $K = 10$	2.04	4.49	1.77	0.88	0.07
Two ovens	0.15	1.23	1.90	0.47	0.00
Controlled two ovens	0.79	2.46	1.90	0.47	0.00

With one oven, the high utilization leads to a long expected waiting time. Limiting the system to $K = 10$ reduces the expected waiting time considerably, at the cost of rejecting about 7% of the jobs. Keeping two ovens running makes the expected waiting time almost negligible, but leaves much capacity unused. Switching on the second oven only when $n \geq 5$ provides a compromise: the expected waiting time is 0.79, while the first and second ovens are busy 81% and 14% of the time, respectively.

As a numerical check, increasing K from 10 to 1000 recovers the results of the $M/M/1$ model without blocking.

LET US NEXT discuss an example of employee planning at a call center. Here we keep the models simple, but we include all necessary steps to solve a realistic planning problem.

We start with specifying the *arrival process*. It is reasonable to model it as a Poisson process, as there are many potential customers, each choosing with a small probability of calling at a certain moment in time. Thus, we only have to characterize the arrival rate. Estimating this for a call center is easy because all calls are registered.

It is common to use a *demand profile* that shows the average number of customers arriving per hour. Then we model the arrivals as a Poisson process with an arrival rate that is constant during a certain hour, as specified by the demand profile.

It is also easy to find the *service distribution*; for ease, we model the service time distribution as exponential with mean 1.5 minutes.

For the *service objective* we prefer to keep the average number of jobs in the system such that $\mathsf{E}[L] \le 5$.[2]

Let us model the situation as an $M/M/1$ queue with one server that works as fast as c single servers combined. Solving for c in the inequality $\mathsf{E}[L] \le 5$ gives[3]

$$c \ge \frac{\mathsf{E}[L]+1}{\mathsf{E}[L]}\lambda\,\mathsf{E}[S] = \frac{6}{5}\lambda\,\mathsf{E}[S] \approx 0.03\lambda,$$

where, with the above estimate, $\mathsf{E}[S] = 1.5/60$ hour. It is easy to apply this formula. For instance, we see in the demand profile Fig. 5.2.1 that $\lambda = 120$ customers per hour between 12 and 13, hence we need $c \approx 3.6 = 4$ employees for that period.

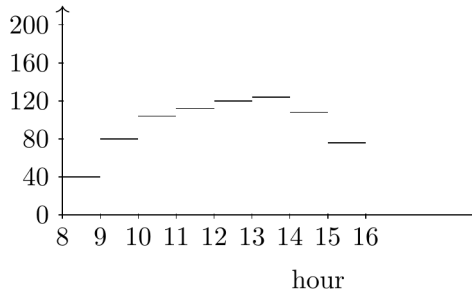


Fig. 5.2.1: A demand profile.

[2] *Other KPIs are also possible to use.*

[3] $\mathsf{E}[L] = \rho/(1-\rho) \implies \rho = \mathsf{E}[L]/(\mathsf{E}[L]+1)$ *and* $\rho = \lambda\,\mathsf{E}[S]/c$.

With this we can find a formula to convert the demand profile into a *load profile* to specify the minimal number of servers per hour needed to meet the service objective.

THE LAST STEP in the planning process is to *cover the load profile with service shifts*. This is typically not easy since shifts have to satisfy all kinds of restrictions with respect to breaks and durations. Moreover, shifts can also have different costs: evening shifts are typically more expensive per hour.

The usual way to solve such covering problems is by means of an integer programming problem. For instance, suppose only 4 *shift plans* are available as shown at the right.[4]

For the optimization, let x_i be the number of shifts of type i and c_i the cost of this type. Then the problem is to solve $\min \sum_i c_i x_i$, such that for all hours t the shifts cover the load, that is, $\sum_i x_i \mathbb{1}\{t \in s_i\} \ge 0.03\lambda_t$. (We write $t \in s_i$ if hour t is covered by shift type i.)

[4] *1.* $++-++$ *2.* $+++-+$ *3.* $++-+++$ *4.* $+++-++$. *Shift plans. A* $+$ *indicates a working hour and* $-$ *a break of an hour.*

TF 5.2.1. Suppose $\frac{\lambda}{\mu} = 2$. Claim: 3 is the minimum number of servers required to make the system stable.

TF 5.2.2. Claim: It is always better to have a single fast server than multiple slow servers which together have the same rate.

TF 5.2.3. A repair/maintenance facility would like to determine how many employees should be working in its tool crib. The service time is exponential, with a mean of 4 minutes, and customers arrive as a Poisson process with a rate of 28 per hour. Claim: With one employee the system is not rate stable.

TF 5.2.4. Consider an $M/M/c/K$ queue that is not rate stable, but jobs sometimes balk. Claim: $\lim_{t\to\infty} \mathsf{P}\{L(t) = K\} = 1$.

TF 5.2.5. Claim: For the $M/M/c$ queue, adding a second server always halves the expected waiting time compared to the $M/M/1$ queue with the same arrival and service rates.

THE NEXT EXERCISES require some modeling skills. Hence, they may be quite hard even though the formulas are simple.

Ex 5.2.6. In the town hall of the municipality, people can, for example, renew their driver's license or passport. For ease, assume that there is just one employee and that the arrivals of customers are not planned. Why is it reasonable to model this queueing system as a $M/M/1$ queue?

Measurements show that $\mathsf{E}[W] = 6$ minutes and $\lambda = 5$ per minute. What are the service rate and utilization? Calculate $\mathsf{E}[Q]$, $\mathsf{E}[L]$.

There should be a sufficient number of seats so that all waiting customers can sit down at least 90 percent of the time. How many seats should there minimally be? What would you advise the municipality?

Ex 5.2.7. The smallest barber shop in town works alone and has two chairs for waiting customers. Six jobs arrive per hour, and the barber needs 12 minutes for a haircut. We model this, for ease, as an $M/M/1/3$ queue. Estimate $\mathsf{E}[Q]$ and the average number of customers served per hour. Then, estimate $\mathsf{E}[Q]$ when the barber hires another location that allows for 3 extra chairs. Quantify the effect on the number of customers served per hour.

Ex 5.2.8. Occasionally there is a pizza stand at the corner of the Verlengde Visserstraat and the Westersingel in Groningen. As people (tend to) behave like sheep, the rate at which people join the queue depends on the actual queue length. When there is nobody waiting, people are somewhat hesitant to order: 'When there are no people waiting, most likely the pizzas are not good.'. However, when the queue is long, people prefer to go elsewhere. Measurements show that the number of arrivals per hour is well captured by the array $\lambda = [10, 10, 20, 20, 5]$, where entry n is the arrival rate when there are n customers in the system. We set $\lambda(n) = 0$ for $n \geq 5$. The pizza oven can contain two pizzas and it takes 2 minutes to bake a pizza. We model the system as a state-dependent queue with two servers.

Calculate the steady-state probabilities and from this the average arrival rate. Then, find $\mathsf{E}[L]$, $\mathsf{E}[Q]$, $\mathsf{E}[J]$ and $\mathsf{E}[W]$.

Ex 5.2.9. Fast food restaurants tend to fry patties before customers arrive. This way, they can serve customers faster, because when a customer arrives while there is still an inventory of prepared patties, the server only has to flip a patty in a bun, while otherwise the customer has to wait for the additional frying time of the patty. Assume the restaurant keeps at most 3 patties on stock. To keep the model simple, assume the restaurant has just one pan to fry a patty and the frying time $S \sim \text{Exp}(\mu)$ with $\mu = 15$ per hour. Customers arrive as a Poisson process with rate $\lambda = 12$ per

hour. When there is a prepared patty upon arrival, the customer is served immediately, otherwise the customer has to wait for a patty.

Explain how to model this system as an $M/M/1$ system. Then compute the average number of patties on stock, the average number of people waiting, and the average waiting time of a patty before it ends up in a bun, and the average time a person has to wait for a hamburger.

You should remember this problem. It is of fundamental importance in inventory control and known as an *inventory system with backlogging*; it is (implicitly) used by nearly any company that keeps inventory.

Ex 5.2.10. [Continuation of [5.2.9]] What would be the impact of allowing four patties on stock?

Ex 5.2.11. [Continuation of [5.2.9]] Suppose that the restaurant does not keep patties in stock. What is the expected waiting time for the customer?

5.3 $M^B/M/1$ QUEUE AND REJECTION POLICIES

In this section, by using the renewal-reward theorem, we derive for the $M^B/M/1$ queue expressions for ρ, $\mathsf{E}[W]$, and $\mathsf{E}[L]$. To compute more difficult performance measures, such as the loss probability $\mathsf{P}\{L > n\}$, we need expressions for the stationary distribution $\pi(n)$. We use level-crossing to derive a recursive scheme which, when combined with a policy to reject batches, can be used to compute these probabilities.

ASSUME THAT BATCHES arrive as a Poisson process with rate λ_B, and their sizes $B_k \overset{\text{iid}}{\sim} B$. Let $S_{k,i}$ denote the service time of the ith job of batch k; assume $S_{k,i} \overset{\text{iid}}{\sim} S$, independently of the batch sizes. By (4.1.2) the utilization $\rho = \lambda_B \mathsf{E}[B]\,\mathsf{E}[S]$; as always, we require $\rho < 1$.

An arriving batch joins the end of the queue (if present). Once it arrives at the head of the queue, the batch moves in its entirety to the server[1] , and then the server processes the jobs one after another until the batch is finished. Writing Q^B for the number of batches in queue and L_s^B for the number of jobs (if any) at the server, the average time $\mathsf{E}\left[W^B\right]$ a batch spends in queue is[2]

$$\mathsf{E}\left[W^B\right] = (\mathsf{E}\left[L_s^B\right] + \mathsf{E}\left[Q^B\right]\mathsf{E}[B])\,\mathsf{E}[S]\,. \tag{5.3.1}$$

Substituting the equality[3] $\mathsf{E}\left[Q^B\right] = \lambda_B \mathsf{E}\left[W^B\right]$ and simplifying yields

$$\mathsf{E}\left[W^B\right] = \frac{\mathsf{E}\left[L_s^B\right]}{1-\rho}\,\mathsf{E}[S]\,. \tag{5.3.2}$$

We can relate the average time $\mathsf{E}[W]$ a *job* spends in queue to $\mathsf{E}\left[W^B\right]$ by observing that $\mathsf{E}\left[L_s^B\right] + \mathsf{E}\left[Q^B\right]\mathsf{E}[B] = \mathsf{E}[L]$,[4] so from (5.3.1)

$$\mathsf{E}\left[W^B\right] = \mathsf{E}[L]\,\mathsf{E}[S] \overset{1}{=} \mathsf{E}[W]\,,$$

where 1 follows from the memorylessness of the service time of the job in service (if any). Thus, to compute $\mathsf{E}[W]$ it remains to find an expression for $\mathsf{E}\left[L_s^B\right]$ in (5.3.2).

[1] *This is the same batching model as in Section 6.2.*

[2] *First the jobs at the server have to be served, then the ones in queue.*

[3] *Little's law.*

[4] *The total number of jobs in the system is the number at the server plus the number in queue. The latter equals the number of batches in queue times the average batch size.*

[5] *Much like the proof of Little's law.*

For this, we use the renewal reward theorem. Define[5] $R(t) := \int_0^t L_s^B(s)\,\mathrm{d}s$, where $L_s^B(s)$ is the number of jobs (of the batch in service) at the server at time s, and $Z_k = R(D_k) - R(D_{k-1})$ where D_k is the departure time of the kth batch. Suppose the kth batch has batch size B_k, then

$$Z_k|B_k = \int_{D_{k-1}}^{D_k} L_s^B(s)\,\mathrm{d}s = B_k S_{k,1} + (B_k - 1)S_{k,2} + \cdots + S_{k,B_k},$$

because B_k jobs are at the server during the service of the first job, $B_k - 1$ during the service of the second, ... Using that the $S_{k,i} \sim S$,

$$\mathsf{E}[Z_k|B_k] = B_k\,\mathsf{E}[S] + (B_k - 1)\,\mathsf{E}[S] + \cdots \mathsf{E}[S] = \frac{B_k(B_k+1)}{2}\,\mathsf{E}[S].$$

As $B_k \sim B$,

$$\mathsf{E}[Z_k] = \mathsf{E}[\mathsf{E}[Z_k|B_k]] = \mathsf{E}\left[\frac{B(B+1)}{2}\right]\mathsf{E}[S] = \frac{\mathsf{E}\left[B^2\right] + \mathsf{E}[B]}{2}\,\mathsf{E}[S].$$

We now have all elements to fill in $R = \alpha Z$. First, $R = \lim_{t\to\infty} R(t)/t = \mathsf{E}\left[L_S^B\right]$. Second, we sample at batch departure moments, so $\alpha = \delta_B$ which equals λ_B by rate stability. Third, $Z = \mathsf{E}[Z_k]$. Combining all this, and using that $\rho = \lambda_B\,\mathsf{E}[B]\,\mathsf{E}[S]$, we obtain

$$\mathsf{E}\left[L_s^B\right] = R = \lambda_B Z = \lambda_B \frac{\mathsf{E}\left[B^2\right]}{2}\,\mathsf{E}[S] + \frac{\rho}{2}. \tag{5.3.3}$$

We can brush this up by realizing that[6]

$$\frac{\mathsf{E}\left[B^2\right]}{(\mathsf{E}[B])^2} = \frac{\mathsf{E}\left[B^2\right] - (\mathsf{E}[B])^2 + (\mathsf{E}[B])^2}{(\mathsf{E}[B])^2} = \frac{\mathsf{V}[B] + (\mathsf{E}[B])^2}{(\mathsf{E}[B])^2} = C_B^2 + 1.$$

[6] *Recall that the scv of a rv B is $C_B^2 = \mathsf{V}[B]/(\mathsf{E}[B])^2$. We assume that $\mathsf{E}\left[B^2\right] < \infty$.*

Substituting this into the above gives the final result

$$\mathsf{E}[W] = \frac{1 + C_B^2}{2}\frac{\rho}{1-\rho}\,\mathsf{E}[B]\,\mathsf{E}[S] + \frac{1}{2}\frac{\rho}{1-\rho}\,\mathsf{E}[S]. \tag{5.3.4}$$

This formula is an important result. It tells us that $\mathsf{E}[W]$ increases when the load ρ or the variability C_B^2 or $\mathsf{E}[B]$ or $\mathsf{E}[S]$ increases (even when the utilization ρ remains the same). As such, it gives us the most important clues about what to change if $\mathsf{E}[W]$[7] has to be reduced.

[7] *Hence, by Little's law, $\mathsf{E}[L]$.*

RATHER THAN USING the time an entire batch spends at the server, we can concentrate on the expected time spent by a single job at the server. This will provide us with a simple expression for $\mathsf{P}\left\{L_s^B = i\right\}$, from which we also can derive (5.3.3).

In this case, let $R_i(t) = \int_0^t \mathbb{1}\left\{L_s^B(s) = i\right\}\mathrm{d}s$ be the total time there are i jobs at the server, so that $R_i(t)/t \to \mathsf{P}\left\{L_s^B = i\right\}$ as $t \to \infty$. By sampling $R(\cdot)$ at batch departure times $\{D_k\}$, see Fig. 5.3.1, we see that

$$Z_k = R_i(D_k) - R_i(D_{k-1}) = S_{k,i}\,\mathbb{1}\{B_k \geq i\},$$

because only when $B_k \geq i$, we can see i jobs at the server for the kth batch, and then $S_{k,i}$ is the time this job occupies the server. With this,

$$Z = \lim_{n\to\infty} \frac{1}{n}\sum_{k=1}^{n} S_{k,i}\,\mathbb{1}\{B_k \geq i\} = \mathsf{E}[S\,\mathbb{1}\{B \geq i\}] = \mathsf{E}[S]\,G(i-1),$$

[8] *Recall: $\mathsf{P}\{A\} := \mathsf{E}[\mathbb{1}\{A\}]$.*

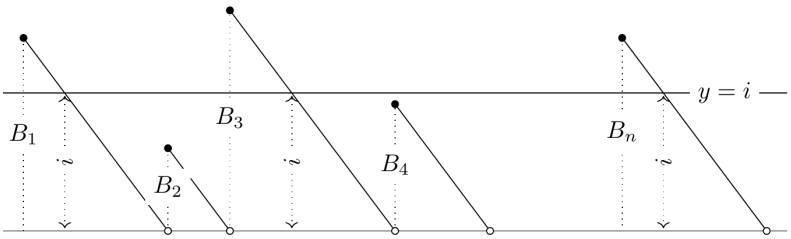


FIG. 5.3.1: A batch crosses the line $y = i$ iff it contains at least i jobs. Thus, during the service of a batch with i or more jobs, there is precisely one service of an i-th job.

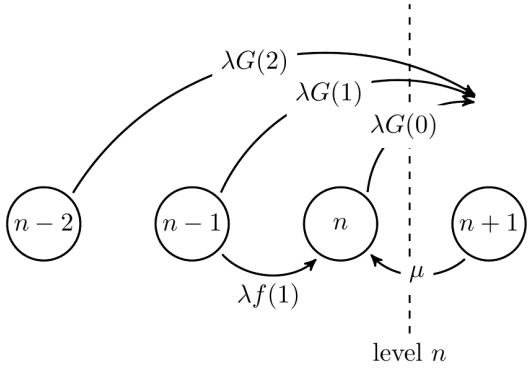


FIG. 5.3.2: When $L(t) = n$, the arrival of any batch crosses level n from below, but when $L(t) = n-1$, only a batch with a size of at least two jobs can realize an up-crossing of level n, and so on.

since $S_{k,i}$ and B_k are independent, and[8] $G(i-1) = \mathsf{E}[\mathbb{1}\{B \geq i\}] = \mathsf{P}\{B > i-1\}$ is the survivor function of B. With the renewal reward theorem and rate stability, $\delta_B = \lambda_B$, we conclude that

$$\mathsf{P}\{L_s^B = i\} = \lambda_B \mathsf{E}[S]\, G(i-1).$$

This expression provides us with another derivation of (5.3.4). Just note that $\mathsf{E}[L_s^B] = \sum_{i=0}^{\infty} i\,\mathsf{P}\{L_s^B = i\} = \lambda_B \mathsf{E}[S] \sum_{i=1}^{\infty} iG(i-1)$ and use [5.3.12] to simplify.

THE SECOND TASK is to use level-crossing to find a recursive formula for the state probabilities $\{\pi(n)\}$.[9] For this we need to generalize our earlier level-crossing equation $\lambda p(n) = \mu p(n-1)$ because when a batch arrives, multiple levels can be up-crossed, see Fig. 5.3.2. We use the PASTA property to express level-crossing in terms of $\{\pi(n)\}$ instead of $\{p(n)\}$.

The down-crossing rate of level n is easy: since jobs are still served one-by-one, we have $\mu\pi(n+1)$.

For the up-crossings of level n we should consider batch arrivals that see $m \leq n$ jobs in the system. Since batches arrive at a rate λ_B, PASTA implies that $\lambda_B \pi(m)$ is the arrival rate of batches that see m upon arrival.[10] When an arriving batch sees m jobs in the system, and level n has to be up-crossed, the batch must contain more than $n-m$ jobs. As $G(n-m)$ is the probability that the batch size is larger than $n-m$ and as batch sizes are independent of arrival times, such batches arrive as a Poisson process with rate $\lambda_B \pi(m) G(n-m)$. Finally, to compute the

[9] *Note: n means the number of jobs in the system, not the number of batches.*

[10] *That is, the total arrival process is thinned with a fraction $\pi(m)$.*

rate at which level n is up-crossed, we merge all these thinned Poisson processes into a new Poisson process with rate $\lambda_B \sum_{m=0}^{n} \pi(m)G(n-m)$.

By level-crossing, the up-crossing and down-crossing rates must match, thus,

$$\lambda_B \sum_{m=0}^{n} \pi(m)G(n-m) = \mu\pi(n+1). \tag{5.3.5}$$

It is an important check[11] to retrieve (5.3.4) from the expression $\mathsf{E}[L] = \sum_n n\pi(n)$ and Little's law.

[11] *[5.3.13]*.

IT REMAINS TO find the normalization constant for π. However, this is not possible in general because (5.3.5) does not lead to a closed form expression for $\pi(n)$.[12] Numerically, though, we can start with $\pi(0) = 1$, apply the recursion, and truncate when the (unnormalized) probabilities become negligible. For a bounded batch size, let K_B be its largest possible value. The function below first computes at least K_B values, so that it does not stop during an initial increase. It then continues until the unnormalized weights become negligible and finally normalizes them.

[12] *Compare* (5.1.1).

Algorithm 5.3.1 Compute the stationary probabilities of an $M^B/M/1$ queue.

Input: rates λ_B, μ, survivor function G, largest batch size K_B, threshold ϵ.
Output: the stationary-probability map π.
function BATCHSTATIONARY(λ_B, μ, G, K_B, $\epsilon = 10^{-10}$)
    $q \leftarrow$ MAP(), $q(0) \leftarrow 1$, $n \leftarrow 0$
    while $n < K_B$ or $q(n) > \epsilon$ **do**
        $q(n+1) \leftarrow \frac{\lambda_B}{\mu} \cdot \text{SUM}\big([q(m) \cdot G(n-m) : m = 0, \ldots, n]\big)$
        $n \xleftarrow{+} 1$
    $\pi \leftarrow$ NORMALIZEMAP(q)
    return π

A different approach is to completely bypass the normalization problem by blocking the demand above some level K. There are three common policies for deciding which jobs in a batch to accept.

1. Complete acceptance: accept all batches that arrive when the system contains $K-1$ or fewer jobs, and reject the entire batch when the system contains K or more jobs. For this case, we make the assumption that the batch size is bounded by the number K_B so that $L(t) \le K - 1 + K_B$.[13]

[13] *[5.3.8]*.

2. Partial acceptance: accept whatever fits of a batch, i.e., up to K, and reject the rest.[14]

[14] *[5.3.9]*.

3. Complete rejection: if a batch does not fit entirely into the system, reject it completely.[15]

[15] *[5.3.10]*.

In [5.3.9]–[5.3.10] we will derive recursions similar to (5.3.5) to compute $\{\pi(n)\}$ for these different blocking models.

TF 5.3.1. For the $M^B/M/1$ queue, let $\mathsf{E}[Q]$ be the expected number of jobs in queue, $\mathsf{E}\left[Q^B\right]$ the expected number of batches in the queue, $\mathsf{E}[W]$ the expected waiting time of a job in queue, λ the rate at which jobs arrive, and λ_B the rate at which batches arrive. Claim: Little's law informs us that $\mathsf{E}\left[Q^B\right]\mathsf{E}[B] = \lambda\,\mathsf{E}[W]$.

TF 5.3.2. For an $M^B/M/1$ queue we know that:

$$\mathsf{E}[W] = \frac{1+C_B^2}{2}\frac{\rho}{1-\rho}\mathsf{E}[B]\,\mathsf{E}[S] + \frac{1}{2}\frac{\rho}{1-\rho}\mathsf{E}[S].$$

Suppose that $\mathsf{V}[B] = 0$. Claim: This formula implies that if the batch size increases but $\mathsf{E}[S]$ decreases such that ρ remains the same, the expected waiting time decreases.

TF 5.3.3. Suppose that the time to process a job has an exponential distribution. Claim: the inter-arrival times at which individual jobs arrive in a $M^B/M/1$ queue are exponentially distributed.

TF 5.3.4. Claim: This algorithm correctly computes the stationary probabilities of an $M^B/M/1/K$ queue with complete rejection.

```
1: function CompleteRejection(K, λ_B, μ, G)
2:     q ← Map(), q(0) ← 1
3:     for n = 0, ..., K − 1 do
4:         q(n+1) ← (λ_B/μ)·Sum([q(m)·[G(n−m) − G(K−m)] : m = 0, ..., n−1])
5:
6:     return NormalizeMap(q)
```

TF 5.3.5. Claim: In the $M^B/M/1$ queue, if the batch size B is deterministic, the waiting time formula reduces to the Pollaczek-Khinchine formula for the $M/M/1$ queue.

Ex 5.3.6. A company operates a machine that receives batches of various sizes. Management would like to know how a reduction in batch size variability would affect average queueing time. Suppose, for the sake of an example, that batch sizes are such that $\mathsf{P}\{B=1\} = \mathsf{P}\{B=2\} = \mathsf{P}\{B=3\} = 1/3$. The batches arrive at the rate $\lambda_B = 1/h$. The average service time for a job is 25 minutes. Calculate by how much $\mathsf{E}[L]$ would decrease if $B \equiv 2$.

Suppose B is geometrically distributed[16] with $\mathsf{E}[B] = 1/p$.

Ex 5.3.7. Compute $\mathsf{E}[L]$ for the $M^B/M/1$ queue when B is geometrically distributed with $\mathsf{E}[B] = 1/p$. Check that if $\mathsf{E}[B] = 1$ the $M/M/1$ queue results.

Ex 5.3.8. Derive a set of recursions analogous to (5.3.5) to compute $\pi(n)$ for the $M^B/M/1/K$ queue with complete acceptance.

Ex 5.3.9. Derive a set of recursions analogous to (5.3.5) to compute $\pi(n)$ for the $M^B/M/1/K$ queue with partial acceptance.

[16] *Sometimes this distribution is called the First Success distribution. There is no consensus on the definition of the geometric distribution, see Wikipedia.*

Ex 5.3.10. Derive a set of recursions analogous to (5.3.5) to compute $\pi(n)$ for the $M^B/M/1/K$ queue with complete rejection.

Opt 5.3.11. Show that when $B \le K_B$, the probabilities $\pi(n)$ decrease geometrically fast.

Opt 5.3.12. Use [1.2.14] to show that $\sum_{i=1}^{\infty} iG(i-1) = (\mathsf{E}\left[B^2\right] + \mathsf{E}[B])/2$.

Opt 5.3.13. Use (5.3.5) in $\mathsf{E}[L] = \sum_{n=0}^{\infty} n\pi(n)$ to derive (5.3.4).

5.4 $M/G/1$ QUEUE: EXPECTED WAITING TIME

The Pollaczek-Khinchine (PK) formula for the average waiting time of an $M/G/1$ queue is a key result in Operations Research. We derive it using the renewal reward theorem, and follow the steps of Section 5.3.

WE START FROM the simple observation that before an arrival gets access to the server, it must first wait for the job in service[1] to complete, then for the queue to clear. From PASTA it follows that $\mathsf{E}[W] = \mathsf{E}[S_r] + \mathsf{E}[Q]\,\mathsf{E}[S]$, where $\mathsf{E}[S_r]$ is the (time-)average remaining service time of the job in service.[2] With Little's law $\mathsf{E}[Q] = \lambda\,\mathsf{E}[W]$ and $\rho = \lambda\,\mathsf{E}[S]$,

$$\mathsf{E}[W] = \mathsf{E}[S_r] + \lambda\,\mathsf{E}[W]\,\mathsf{E}[S] \implies \mathsf{E}[W] = \frac{\mathsf{E}[S_r]}{1-\rho}. \tag{5.4.1}$$

To proceed, we need an expression for $\mathsf{E}[S_r]$.

[1] *If any.*

[2] *The rv S_r is also known as the remaining lifetime and is widely used in maintenance theory and actuarial sciences.*

Theorem 5.4.1. The expected remaining service time as observed by Poisson arrivals is

$$\mathsf{E}[S_r] = \frac{\lambda}{2}\mathsf{E}\left[S^2\right],$$

provided the second moment of S exists.

—

Proof. In Fig. 5.4.1, the service of job k starts at $\tilde{A}_k$, so the job departs at time $D_k = \tilde{A}_k + S_k$. Hence, at any time $s \in [\tilde{A}_k, D_k]$, the remaining service time of job k is $D_k - s$.

More generally, at some arbitrary time s, the number of departures is $D(s)$. Thus, $D(s)+1$ is the index of the first job to depart after time s, and its departure time is $D_{D(s)+1}$. Consequently, the remaining service time at time s is $D_{D(s)+1} - s$, provided that this job is in service, i.e., $\tilde{A}_{D(s)+1} \le s$. The total remaining service time as seen by the server up to time t is therefore

$$R(t) = \int_0^t (D_{D(s)+1} - s)\,\mathbb{1}\left\{\tilde{A}_{D(s)+1} \le s\right\} \mathrm{d}s,$$

and $\lim_{t\to\infty} R(t)/t = \mathsf{E}[S_r]$.

In Fig. 5.4.1, $R(D_k) - R(D_{k-1}) =: Z_k$ equals the area of the triangle. By inspecting $R(\cdot)$ at $\{D_k\}$, we see that $Z = \lim_{n\to\infty} \frac{1}{n}\sum_{k=1}^{n} S_k^2/2 = \mathsf{E}\left[S^2\right]/2$. By the renewal-reward theorem, $R = \delta Z$, but $\delta = \lambda$ by rate-stability. The claim follows now from PASTA. □

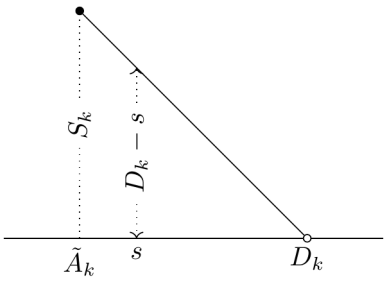


Fig. 5.4.1: Remaining service time.

Noting that

$$\frac{\mathsf{E}\left[S^2\right]}{(\mathsf{E}[S])^2} = \frac{\mathsf{E}\left[S^2\right] - (\mathsf{E}[S])^2 + (\mathsf{E}[S])^2}{(\mathsf{E}[S])^2} = \frac{\mathsf{V}[S] + (\mathsf{E}[S])^2}{(\mathsf{E}[S])^2} = C_s^2 + 1.$$

and substituting this in the expression for $\mathsf{E}[S_r]$ we obtain from (5.4.1) the following result.

Theorem 5.4.2. The expected waiting time in the $M/G/1$ queue equals[3]

$$\mathsf{E}[W] = \frac{1}{2}\frac{\lambda\,\mathsf{E}\left[S^2\right]}{1-\rho} = \frac{1+C_s^2}{2}\frac{\rho}{1-\rho}\,\mathsf{E}[S]. \tag{5.4.2}$$

[3] *The Pollaczek-Khinchine (PK) formula.*

THE POLLACZEK-KHINCHINE EQUATION can also be found as a limiting case of the waiting time of the $M^B/M/1$ queue by interpreting a job in the $M/G/1$ as a batch consisting of many tiny items in the $M^B/M/1$ queue.[4] Suppose that in the $M^B/M/1$ queue the service times of the items are iid with mean $1/\nu$. If F is the cdf of the service times of the $M/G/1$ queue, then we take

[4] *If you are not interested in math, you may skip the next derivation.*

$$\mathsf{P}\{B=k\} = F(k/\nu) - F((k-1)/\nu)$$

as the probability that a batch has size k in an $M^B/M/1$ queue. Note that in the $M^B/M/1$ queue, when a batch has size k, the service time is the sum of k rvs that are $\text{Exp}(\nu)$; thus, the service time of a batch of size k is $\sim \text{Gamma}(\nu,k)$. When $\nu \gg 1$, the expected service time of a batch is

$$\begin{aligned}\mathsf{E}[S_B] &= \mathsf{E}[\mathsf{E}[S_B|B]] = \mathsf{E}\left[\frac{B}{\nu}\right] = \sum_{k=0}^{\infty}\frac{k}{\nu}\mathsf{P}\{B=k\}\\ &= \sum_{k=0}^{\infty}\frac{k}{\nu}\left[F\left(\frac{k}{\nu}\right) - F\left(\frac{k-1}{\nu}\right)\right] \approx \int_0^{\infty} x\,\mathrm{d}F(x) = \mathsf{E}[S].\end{aligned}$$

In words, when ν is large, the batch expected service time in an $M^B/M/1$ queue approximates the expected service time in an $M/G/1$ queue.

In fact, the proof that the expected waiting time of the $M^B/M/1$ converges to the Pollaczek-Khinchine equation when $\nu\to\infty$ is based on the fact that every cdf F, concentrated on $[0,\infty)$, can be approximated (weakly) by the sequence of cdfs[5]

$$F_\nu(x) = F(0) + \sum_{k=1}^{\infty}\left[F\left(\frac{k}{\nu}\right) - F\left(\frac{k-1}{\nu}\right)\right]\int_0^x \frac{\nu^k t^{k-1}}{(k-1)!}e^{-\nu t}\,\mathrm{d}t, \quad \text{as } \nu\to\infty.$$

[5] *See Asmussen, Section III.4, Applied Probability and Queues, 2003.*

That is, the distribution of the service time of a batch whose items have $\text{Exp}(\nu)$ service times approaches the cdf F of the service time of the $M/G/1$ queue. For $\mathsf{E}\left[S^2\right]<\infty$, realize $B=\lceil \nu S\rceil$; the bound $0\le B/\nu - S < 1/\nu$ ensures convergence of the first two scaled batch-size moments to those of S. With $\lambda_B=\lambda$, substituting these moment limits in (5.3.4) gives (5.4.2), since the last term vanishes as $\nu\to\infty$. This leads to (5.4.2).

TF 5.4.1. Consider the $M/G/1$ queue. Denote by $\tilde{A}_k$ the time job k starts service and by D_k its departure time, $k=1,\ldots,n$. Claim: the expression

$$\sum_{k=1}^{n}\int_0^{D_n}(D_k-s)\,\mathbb{1}\{\tilde{A}_k\le s<D_k\}\,\mathrm{d}s$$

represents the total remaining service time up to $t=D_n$.

TF 5.4.2. Claim: For an $M/D/1$ queue, $\mathsf{E}[W] = \frac{\rho}{1-\rho}\frac{\mathsf{E}[S]}{2}$.

TF 5.4.3. Claim: The PK formula shows that $\mathsf{E}[W]$ is linear in ρ.

TF 5.4.4. Claim: For the $M/G/1$ queue, the expected remaining service time $\mathsf{E}[S_r]$ does not depend on the service time distribution, only on $\mathsf{E}[S]$.

TF 5.4.5. Claim: The derivation of the PK formula uses both PASTA and Little's law.

Ex 5.4.6. Consider a workstation with only one machine. We model the job arrival process as a Poisson process with rate $\lambda = 3$ per day. The average service time $\mathsf{E}[S] = 2$ hours, $C_s^2 = 1/2$, and the store operates 8 hours per day. Show that $\mathsf{E}[W] = 4.5$ h. What would you propose to reduce $\mathsf{E}[W]$ to 2h?

Ex 5.4.7. Compare the expected waiting times in the $M/D/1$ and the $M/M/1$ queues.

Ex 5.4.8. Compute $\mathsf{E}[J]$ for the $M/G/1$ queue with $S \sim U[0,\alpha]$.

Ex 5.4.9. Show that for the $M/G/1$ that $\mathsf{E}[S_r] = \rho\,\mathsf{E}[S_r \mid S_r > 0]$. As a consequence, for the $M/M/1$ queue, $\mathsf{E}[S_r] \neq \mathsf{E}[S]$. Why not?

5.5 $M/G/1$ QUEUE LENGTH DISTRIBUTION

In Section 5.3 we used level-crossing arguments to find a recursion for the stationary distribution $\pi(n)$ of the $M^B/M/1$ queue.[1] Here we find a similar recursion to compute $\pi(n)$ for the $M/G/1$ queue. However, we cannot simply copy the ideas of Section 5.3 to the present situation, because in the $M^B/M/1$ queue the service times of the items are exponential, hence memoryless, while in the $M/G/1$ this is not the case.

[1] *You can skip this if you are not interested.*

IF WE WANT to characterize the state at all moments in time, we need to keep track of the number of jobs in the system *and* the remaining service time of the job in service (if any), because service times are not memoryless. But, by sampling at job departure times $\{D_k\}$,[2] we can restrict the state description to just the number in the system. For ease we write[3] $L_k = L(D_k)$ for the number of jobs in the system left behind at the departure epoch of the kth job.

[2] *So that the remaining service time is guaranteed to be* 0.

[3] *i.e., not* $L(D_k-)$.

We now find a useful recursion for $\{L_k\}$. Assume first that $L_{k-1} > 0$.[4] If Y_k jobs arrive during the service S_k of job k, then $L_k = L_{k-1} - 1 + Y_k$. The situation is slightly more complicated when $L_{k-1} = 0$: we first have to wait for job k to arrive, and then, if Y_k jobs arrive while job k is in service, $L_k = Y_k$. Thus, in general,

[4] *This implies that* $A_k < D_{k-1}$.

$$L_k = [L_{k-1} - 1]^+ + Y_k.$$

With this recursion it is simple to find the level-crossing equations. Write $\delta_k(n) := \mathsf{P}\{L_k = n\}$. The rvs $Y_k \overset{\text{iid}}{\sim} Y$ have pmf $f(j) = \mathsf{P}\{Y = j\}$ and survivor function G.

[5] *Y_k and L_{k-1} are independent by assumption for the $M/G/1$ queue.*

If L_k down-crosses level n, then $L_{k-1} = n+1$; see Fig. 5.5.1 for an example at level $n = 3$. Hence, the probability of down-crossing level n at time D_k is $\delta_{k-1}(n+1)f_0$.[5] To up-cross level n, it is necessary that $L_{k-1} \le n$ and $L_k > n$. When $L_{k-1} = 0$, then $L_k = Y_k$, so we need $Y_k > n$, giving probability $G(n)$. When $L_{k-1} = m \ge 1$, then $L_k = m - 1 + Y_k > n$ requires $Y_k > n+1-m$. The probability of up-crossing is then $\sum_{m=1}^{n} \delta_{k-1}(m)G(n+1-m) + \delta_{k-1}(0)G(n)$.[6]

[6] *Beware off-by-one errors; check!*

It is possible to prove[7] that $\delta(n) = \lim_{k\to\infty} \delta_k(n)$ exists for all n. The up-crossing probability minus the down-crossing probability equals $\mathsf{P}\{L_k > n\} - \mathsf{P}\{L_{k-1} > n\}$, which tends to 0. Thus, taking limits in the two crossing probabilities and noting that $\pi(n) = \delta(n)$, see (4.6.8), gives a recursion for $\{\pi(n)\}$:

[7] *With some Markov chain theory.*

$$\pi(n+1)f(0) = \pi(0)G(n) + \sum_{m=1}^{n} \pi(m)G(n+1-m). \tag{5.5.1}$$

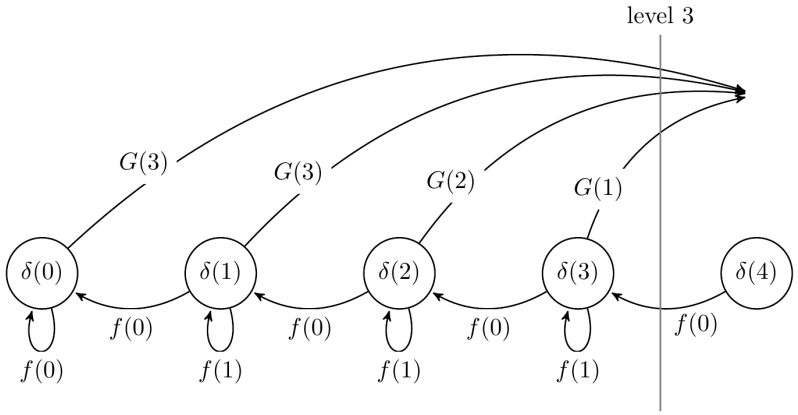


FIG. 5.5.1: Level crossing at departure moments.

FOR THE EVALUATION of the above recursion we can just follow the scheme of Section 5.3, but there is an important difference: here the $\{f(k)\}$ need to be computed.

We use a conditioning argument to find an expression for $\mathsf{P}\{Y = j\}$. Since jobs arrive as a Poisson process, $Y|S = s \sim P(\lambda s)$, i.e.,

$$\mathsf{P}\{Y = j \,|\, S = s\} = e^{-\lambda s}\frac{(\lambda s)^j}{j!}.$$

If $S \equiv s$, that is, a constant, then this is the distribution we are looking for. When F has a density f, then,

$$\mathsf{P}\{Y = j\} = \int_0^\infty \mathsf{P}\{Y = j \,|\, S = s\} f(s)\,\mathrm{d}s = \int_0^\infty e^{-\lambda s}\frac{(\lambda s)^j}{j!} f(s)\,\mathrm{d}s,$$

which can be solved by hand in simple cases, for instance when $S \sim \mathrm{Exp}(\mu)$.[8] When we cannot obtain a closed-form expression for the integral we need numerical methods.

[8] *[5.5.2].*

Ex 5.5.1. If $L(D_{k-1}) = 0$, why is $\mathsf{E}[D_k - D_{k-1}] = \mathsf{E}[X] + \mathsf{E}[S]$?

Ex 5.5.2. If $S \sim \text{Exp}(\mu)$, show that

$$f(j) = \mathsf{P}\{Y_k = j\} = \frac{\mu}{\lambda+\mu}\left(\frac{\lambda}{\lambda+\mu}\right)^j.$$

Ex 5.5.3. If $S \sim \text{Exp}(\mu)$, show that

$$G(j) = \sum_{k=j+1}^{\infty} f(k) = \left(\frac{\lambda}{\lambda+\mu}\right)^{j+1}.$$

Ex 5.5.4. Use the recursion (5.5.1) for an $M/M/1$ queue with $\lambda = 1$ and $\mu = 2$. Compute $\pi(0)$ and $\pi(3)$, and compare the results with the usual $M/M/1$ queue-length distribution.

The next exercise is a nice test of your algebra skills.[9]

Ex 5.5.5. Check that the queue length distribution $\{\pi(n)\}$ of the $M/M/1$ queue satisfies (5.5.1).

[9] *This is a nice exercise to test your algebra skills.*

5.6 N-POLICIES FOR THE $M/G/1$ QUEUE

[2.2.6] motivates the study of the $M/G/1$ queue under an N-policy. We use the renewal-reward theorem to derive the time-average cost for this system by focusing on the average cost and duration of cycles that start and stop each time a job arrives at an empty system. As corollaries we obtain the EPQ formula of Section 4.3 and the PK formula of Section 5.4.

THE EXPECTED CYCLE duration is what we first need. For this, we consider the expected time $T(q)$ to clear the system when the server *starts working* on the first of q jobs present. We present three different ideas to obtain $T(q)$.

The first heuristic idea is this. We know that jobs arrive at a rate λ and are served at a rate $1/\mathsf{E}[S] = \mu > \lambda$. Clearly, the net 'drain rate' of the queue is $\mu - \lambda$; hence we *guess* that[1]

$$T(q) = \frac{q}{\mu-\lambda} = \frac{\mathsf{E}[S]}{1-\lambda\,\mathsf{E}[S]}q = \frac{\mathsf{E}[S]}{1-\rho}q. \tag{5.6.1}$$

[1] *By analogy: when you have a 'queue' of q km to cycle, and your speed is v km/h, then it takes q/v hours to complete the trip.*

Clearly, this reasoning completely ignores the stochasticity in arrival times and services: we cannot guarantee that the result is correct. For the second idea, consider a regular[2] $M/G/1$ queue. When a job arrives to an empty system, it takes a busy time[3] $\mathsf{E}[U]$ to service this job and all the jobs that arrive during the service of this first job. In other words, it takes $\mathsf{E}[U]$ time on average to move from 1 job in the system to 0 jobs, i.e., one job less. But then, when there are q jobs in the system, it must take $T(q) = q\,\mathsf{E}[U]$ units of time to move from state q to state 0. By [4.3.7], $\mathsf{E}[U] = \mathsf{E}[S]/(1-\rho)$, so we again obtain (5.6.1). The third idea is the most powerful. Write Y for the number of jobs that arrive during a service time. Then $T(q)$ satisfies the relation

$$T(q) = \mathsf{E}[S] + \mathsf{E}\big[T(q+Y-1)\big], \quad q \geq 1, \tag{5.6.2}$$

[2] *That is, not controlled by an N-policy.*

[3] *Recall the explanation that led to (4.1.3).*

because first the job in service must leave, and then, when $Y = k$, it takes $T(q+k-1)$ to clear the system. To solve this equation, we substitute the guess $T(q) = aq + b$ and solve for a and b. It is clear[4] that $T(0) = 0$, hence $b = 0$. Substituting aq into (5.6.2) gives $aq = \mathsf{E}[S] + aq + a\mathsf{E}[Y] - a$. Noting[5] that $\mathsf{E}[Y] = \lambda \mathsf{E}[S]$, and solving for a we find (5.6.1). We note that the solution of (5.6.2) is unique once $T(0)$ is fixed.

[4] *If the system is empty, no time is needed to clear it.*

[5] *Poisson arrivals* $\Longrightarrow$ $\mathsf{E}[Y|S=s] = \lambda s \Longrightarrow \mathsf{E}[Y|S] = \lambda S \Longrightarrow \mathsf{E}[Y] = \mathsf{E}[\mathsf{E}[Y|S]] = \lambda \mathsf{E}[S]$.

For the expected cycle duration $C(N)$ as a function of N, observe that, right after the server switches off, we need N independent inter-arrival times to reach level N. As this takes N/λ in expectation, and then the system needs $T(N)$ to become empty again, $C(N) = N/\lambda + T(N)$.

FOR THE CYCLE cost, we proceed as follows. First, consider the expected cost $U(q)$ of waiting that accrues while one job is served and $q-1$ jobs are in queue.[6] When the service starts, the waiting cost for these q jobs is $hq\mathsf{E}[S]$.[7] Next, arrivals during the service of the job also contribute to the cost of waiting. The expected time these arrivals spend in the system equals $\mathsf{E}\left[\int_0^S N(s)\,\mathrm{d}s\right]$, where $N(s) \sim \text{Pois}(\lambda s)$. Conditional on $S = t$, $\mathsf{E}\left[\int_0^S N(s)\,\mathrm{d}s | S = t\right] = \int_0^t \mathsf{E}[N(s)]\,\mathrm{d}s = \int_0^t \lambda s\,\mathrm{d}s = \lambda t^2/2$, hence $\mathsf{E}\left[\int_0^S N(s)\,\mathrm{d}s|S\right] = \lambda S^2/2$, hence $\mathsf{E}\left[\int_0^S N(s)\,\mathrm{d}s\right] = \lambda\,\mathsf{E}\left[S^2/2\right]$. Combining both cost terms, the total cost of waiting during the service of the job is

[6] *Thus, there are q jobs in the system.*

[7] *That is, h times the area of the rectangle with base $\mathsf{E}[S]$ and height q.*

$$U(q) = hq\mathsf{E}[S] + \frac{1}{2}\lambda h \mathsf{E}\left[S^2\right].$$

Let $V(q)$ be the expected queueing costs to clear the system just after a service starts and q jobs are in the system. By analogy with (5.6.2), $V(q)$ must be the solution of

$$V(q) = U(q) + \mathsf{E}\left[V(q+Y-1)\right], \quad q \geq 1. \tag{5.6.3}$$

As $U(q)$ has a term linear in q and a constant term, we substitute a quadratic form $V(q) = aq^2 + bq + c$ instead of a linear relation, assemble terms with the same power in q, and solve for a, b and c. Of course, $c = 0$ since $V(0) = 0$. For a and b we need some work to arrive at[8]

[8] *See [5.6.11]–[5.6.12].*

$$V(q) = \frac{h}{2}\frac{\mathsf{E}[S]}{1-\rho}q^2 + h\frac{1+\rho C_s^2}{2}\frac{\mathsf{E}[S]}{(1-\rho)^2}q.$$

For the cost during the build-up of the queue, we again use a recursive procedure. Write $W(q)$ for the accumulated queueing cost[9] from the moment the server becomes idle up to the arrival time of the qth job (the job that sees $q-1$ jobs in the system upon arrival). Then,[10]

[9] *Here W is not the waiting time in queue.*

[10] *See [5.6.8].*

$$W(q) = W(q-1) + h\frac{q-1}{\lambda} \Longrightarrow W(q) = h\frac{q(q-1)}{2\lambda}.$$

BY COMBINING ALL the above cost factors and using the renewal reward theorem, we find for the long-run average cost[11]

[11] *[5.6.13].*

$$\frac{W(N)+K+V(N)}{C(N)} = h\frac{1+C_s^2}{2}\frac{\rho^2}{1-\rho} + h\rho + h\frac{N-1}{2} + K\frac{\lambda(1-\rho)}{N}. \tag{5.6.4}$$

It is easy to retrieve the Pollaczek-Khinchine formula from this. The LHS in (5.6.4) is the average cost per unit time. Since in the $M/G/1$

queue, each waiting customer pays $h = 1$ per unit time, the LHS corresponds to $\mathsf{E}[L]$. For the RHS, in the $M/G/1$ queue, the setup cost $K = 0$, hence it is optimal to take $N = 1$. Consequently, just the first two terms of the RHS of (5.6.4) remain. The first term is $\mathsf{E}[Q] = \lambda \mathsf{E}[W]$, where $\mathsf{E}[W]$ is given by the PK formula. The second is $\rho = \mathsf{E}[L_s]$. But $\mathsf{E}[L] = \mathsf{E}[Q] + \mathsf{E}[L_s]$, i.e., the LHS and RHS are equal.

A second interesting result is the EPQ formula for a production-inventory system with Poisson arrivals and no backlogging.[12] The optimal production quantity can be found by taking the derivative of (5.6.4) with respect to N, just as if it is a continuous variable. Setting the derivative to zero and solving for N results in, up to rounding,

$$N^* = \sqrt{\frac{2\lambda(1-\rho)K}{h}}.$$

[12] *This is more general than (4.3.1) because demand can arrive as a Poisson process. Although in its present form, backlogging is now not allowed, this is simple to include.*

TF 5.6.1. A machine can be turned on and off. If the queue length reaches N, the machine is turned on, and if the system becomes empty, the machine switches off. Let $I_k = 1$ if the machine is on in period k and $I_k = 0$ if it is off, let L_k be the number of jobs in the system at the end of period k. Claim: the next recursions model this queueing system.

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

$$I_{k+1} = \mathbb{1}\{L_k \geq N\} + I_k \mathbb{1}\{0 < L_k < N\},$$
$$d_k = \min\{L_{k-1}, c_k\},$$
$$L_k = L_{k-1} - (1 - I_k) d_k + a_k.$$

Assume that $I_0 = 0$ at time 0.

TF 5.6.2. We have an $M/G/1$ queue controlled by an N-policy, $\lambda = 1$ and $S_i \equiv 0.5$. Claim: the expected time to clear this queue after reaching N jobs is the same for a $D/D/1$ queue with the same arrival and service rates.

TF 5.6.3. Claim: For the $M/G/1$ queue under an N-policy, the expected time to clear the system starting from q jobs is $T(q) = q\,\mathsf{E}[S]/(1-\rho)$.

TF 5.6.4. Claim: For the $M/G/1$ queue under an N-policy, the optimal N^* does not depend on the service time distribution, only on ρ.

TF 5.6.5. Claim: For the $M/G/1$ queue under an N-policy, setting $N = 1$ and $K = 0$ recovers the Pollaczek-Khinchine formula for $\mathsf{E}[W]$.

Ex 5.6.6. Explain intuitively that the system is rate-stable for any N.

Ex 5.6.7. Why doesn't the utilization ρ depend on N?

Ex 5.6.8. Explain and solve the recursion for $W(q)$.

Ex 5.6.9. Use the memoryless property for the $M/M/1$ queue to show that

$$T(q) = \frac{1}{\lambda+\mu} + \frac{\lambda}{\lambda+\mu} T(q+1) + \frac{\mu}{\lambda+\mu} T(q-1). \tag{5.6.5}$$

Ex 5.6.10. Show that for the $M/M/1$ queue, $V(q)$ satisfies a relation of the form:

$$V(q) = h\frac{q}{\lambda+\mu} + \frac{\lambda}{\lambda+\mu}V(q+1) + \frac{\mu}{\lambda+\mu}V(q-1). \tag{5.6.6}$$

Ex 5.6.11. Simplify $aq^2 + bq = a\mathsf{E}\left[(q+Y-1)^2\right] + b\mathsf{E}\left[q+Y-1\right] + U(q)$, where $U(q) = hq\mathsf{E}[S] + \frac{1}{2}\lambda h\mathsf{E}\left[S^2\right]$, by assembling powers in q to obtain:

$$a = \frac{h}{2}\frac{\mathsf{E}[S]}{1-\mathsf{E}[Y]} = \frac{h}{2}\frac{\mathsf{E}[S]}{1-\rho},$$
$$b(1-\mathsf{E}[Y]) = a(\mathsf{E}\left[Y^2\right] - 2\mathsf{E}[Y] + 1) + \frac{1}{2}h\lambda\mathsf{E}\left[S^2\right].$$

Opt 5.6.12. Derive the expression for $V(q)$ using the previous exercises.

Opt 5.6.13. Derive (5.6.4).

5.7 ANALYTIC RESULTS FOR THE BASE-STOCK POLICY

In this section, we establish a set of formulas to compute performance measures for the single-item base-stock inventory system of Section 2.5.[1]

[1] *This way, we do not have to simulate to see how performance depends on the order-up-to level S.*

WE INTRODUCE SOME notation: The period demands $\{D_i\}_{i=1}^{\infty}$ are assumed to be iid rvs with $0 < \mathsf{E}[D] < \infty$[2] ; for notational ease we take $D_i = 0$ for $i \le 0$. Write

[2] *Why did we not need to make this assumption earlier?*

$$D(i,j] = \sum_{k=i+1}^{j} D_k, \qquad D[i,j] = D_i + D(i,j], \qquad D[i,j) = D[i,j] - D_j,$$

and likewise for $Q(i,j]$ for the total outstanding reorders.

We assume that the lead time l is constant and a multiple of the period duration. Define $F_X(x) = \mathsf{P}\{\sum_{i=1}^{l} D_i \le x\}$; write $X \sim F_X$ for the lead-time demand, $G_X = 1 - F_X$ for the survivor function, and f_X for the pmf. Clearly, $\mathsf{E}[X] = l\,\mathsf{E}[D]$. Since the period demands are iid rvs, all rvs $\{D[k-l,k)\}_{k=l+1}^{\infty}$ have the same distribution F_X.

WE USE THE recursions of Section 2.5 to derive expressions for the long-run performance measures. The first step is to relate the outstanding orders to the demand.

Lemma 5.7.1. Under a base-stock policy with order-up-to level S, $Q(1,k] = D[1,k)$ for all periods $k \ge 1$.

—

Proof. Using the recursions for the inventory position under the base-stock policy repeatedly,

$$\begin{aligned} P_k &= P'_{k-1} + Q_k = P_{k-1} - D_{k-1} + Q_k \\ &= P_{k-1} - D[k-1,k) + Q(k-1,k] \\ &= P_1 - D[1,k) + Q(1,k]. \end{aligned}$$

The proof is finished by noting that $P_k = S$ for all $k \ge 1$. □

We can use this lemma to express I_k in terms of S and the lead-time demand.

Lemma 5.7.2. Under a base-stock policy, if $I_1 = S$ and there are no outstanding orders of period $k \le 1$, then for all periods $k \ge 1$,

$$I_k = S - D[k-l, k), \qquad I'_k = I_k - D_k = S - D[k-l, k].$$

—

Proof. Using the recursion for the inventory level and the assumptions,

$$\begin{aligned} I_k &= I_{k-1} - D_{k-1} + Q_{k-l} \\ &= I_{k-1} - D[k-1, k) + Q(k-l-1, k-l] \\ &= I_1 - D[1, k) + Q(1-l, k-l] \\ &\overset{1}{=} S - D[1, k) + Q(1, k-l] \\ &\overset{2}{=} S - D[k-l, k); \end{aligned}$$

1 uses the assumptions, 2 follows from the above lemma and $D[k-l, k) = D[1, k) - D[1, k-l)$. □

This next theorem, which allows us to define the inventory level as an rv I, follows directly from the above lemmas.

Theorem 5.7.3. The rvs $\{I_k\}_{k=l+1}^{\infty}$ are identically[3] distributed as the rv

$$I = S - X.$$

[3] *But not necessarily independent.*

—

Proof. As observed earlier, each rv $D[k-l, k) \sim X$. As S is constant, the rvs $I_k = S - D[k-l, k) \sim S - X$ for all $k \ge l+1$. □

NOW THAT WE have characterized the distribution of I in terms of the control parameter S and the lead-time demand X, we can find expressions for the performance measures. First, for the ready rate,

$$\alpha \overset{1}{=} \mathsf{P}\{I' \ge 0\} \overset{2}{=} \mathsf{P}\{I - D \ge 0\} \overset{3}{=} \mathsf{P}\{S - X - D \ge 0\} = \mathsf{P}\{X + D \le S\}, \quad (5.7.1)$$

where 1 follows from the definition of α, 2 from $I' = I - D$, 3 from $I = S - X$. Second, for the fill rate, by taking expectations in (2.5.4), we obtain

$$\beta = \frac{\mathsf{E}[\min\{D, I^+\}]}{\mathsf{E}[D]}. \quad (5.7.2)$$

For the third performance measure, the cycle service level, I have not yet tried to find a simple expression in terms of expectations.

The mean inventory level is

$$\mathsf{E}[I] = \mathsf{E}[S - X] = S - \mathsf{E}[X] = S - l\,\mathsf{E}[D].$$

Next, noting that $I^+ - I^- = I$, the average on-hand inventory can be expressed in terms of the average backlog:

$$\mathsf{E}[I^+] = \mathsf{E}[I] + \mathsf{E}[I^-] = S - l\,\mathsf{E}[D] + \mathsf{E}[I^-]. \quad (5.7.3)$$

From the definition of the backlog,

$$\mathsf{E}[I^-] = \mathsf{E}\left[[-I]^+\right] = \mathsf{E}\left[[X-S]^+\right] = \sum_{k=0}^{\infty} (k-S)^+ f(k). \tag{5.7.4}$$

This summation runs to ∞ if the support of D is unbounded. For numerical purposes, we rewrite it as follows.

Lemma 5.7.4. For a non-negative integer order-up-to level S, the average backlog is equal to

$$\mathsf{E}[I^-] = l\,\mathsf{E}[D] - \sum_{j=0}^{S-1} G_X(j).$$

Proof. Since $\sum_{j=0}^{\infty} \mathbb{1}\{j<k\} = k$, we find from (5.7.4)

$$\begin{aligned}\sum_{k=S}^{\infty} (k-S) f(k) &= \sum_{k=S}^{\infty}\sum_{j=0}^{\infty} \mathbb{1}\{j<k-S\}\, f(k) = \sum_{j=0}^{\infty}\sum_{k=S}^{\infty} \mathbb{1}\{k>j+S\}\, f(k)\\ &= \sum_{j=0}^{\infty} G_X(j+S) = \sum_{j=0}^{\infty} G_X(j) - \sum_{j=0}^{S-1} G_X(j).\end{aligned}$$

Since $\sum_{i=0}^{\infty} G_X(i) = l\,\mathsf{E}[D]$, the claim follows. □

Recall from Th. 5.7.3 that I, hence I^+ and I^-, are functions of the reorder level S. Therefore, the average inventory cost $c(S) = h\,\mathsf{E}\left[I^+\right] + b\,\mathsf{E}[I^-]$ is also a function of S.

FOR THE LIGHTHOUSE Company case specified in Alg. 2.5.4, Alg. 5.7.8 computes the ready rate, the fill rate and the cost for various values of S of the base-stock policy. It relies heavily on the algorithms of Section 1.5.

Algorithm 5.7.5 Compute the ready rate of a base-stock policy.

Input: inventory and demand pmf maps p_I, p_D.
Output: ready rate.
function READYRATE(p_I, p_D)
    $p_{I-D} \leftarrow$ SUBTRACT$\left(p_I, p_D\right)$
    return EXPECTATION$\left(p_{I-D}, \lambda\colon z \mapsto \mathbb{1}\{z \geq 0\}\right)$

Algorithm 5.7.6 Compute the fill rate of a base-stock policy.

Input: inventory and demand pmf maps p_I, p_D.
Output: fill rate.
function FILLRATE(p_I, p_D)
    $p_{I^+} \leftarrow$ POSITIVEPART$\left(p_I\right)$
    $p_{\min\{D, I^+\}} \leftarrow$ MINIMUMPMF$\left(p_D, p_{I^+}\right)$
    return MEAN$\left(p_{\min\{D, I^+\}}\right)$/MEAN$\left(p_D\right)$

Algorithm 5.7.7 Compute the average cost of a base-stock policy.

Input: inventory pmf map p_I, costs h, b.
Output: average inventory cost.
function INVENTORYCOST(p_I, h, b)
    $p_{I^+} \leftarrow$ POSITIVEPART(p_I)
    $p_{I^-} \leftarrow$ NEGATIVEPART(p_I)
    return $h \cdot$ MEAN$(p_{I^+}) + b \cdot$ MEAN(p_{I^-})

Algorithm 5.7.8 Evaluate a base-stock policy.

Input: demand pmf map p_D, lead time l, costs h, b, base-stock level S.
Output: ready rate α, fill rate β, and average cost c.
function EVALUATEBASESTOCK(p_D, l, h, b, S)
    $p_X \leftarrow$ CONVOLUTION(p_D, l)
    $p_I \leftarrow$ SUBTRACT$($CONSTANT$(S), p_X)$
    $\alpha \leftarrow$ READYRATE(p_I, p_D)
    $\beta \leftarrow$ FILLRATE(p_I, p_D)
    $c \leftarrow$ INVENTORYCOST(p_I, h, b)
    return α, β, c

S	$\alpha(S)$	$\beta(S)$	$c(S)$
5	0.34	0.33	15.84
6	0.47	0.47	9.42
7	0.60	0.61	5.65
8	0.72	0.74	4.08
9	0.82	0.84	3.54
10	0.89	0.91	3.63
11	0.94	0.95	4.30

Thus, as a function of S, the service levels increase, but the cost first decreases and then increases.[4]

[4] *High service levels do not come for free.*

TF 5.7.1. In a base-stock model, the ready rate is given by $\alpha = \mathsf{P}\{I - D \geq 0\}$. Claim:

$$\alpha = \mathsf{P}\{S - X - D \geq 0\} = F_{X+D}(S).$$

TF 5.7.2. Claim: In a base-stock model with lead-time demand X, the backlog is given by $\mathsf{E}\left[[S - X]^+\right]$.

TF 5.7.3. Claim: Increasing the order-up-to level S always decreases the average inventory cost $c(S) = h\mathsf{E}\left[I^+\right] + b\mathsf{E}[I^-]$.

TF 5.7.4. Claim: For the base-stock model, $\mathsf{E}[I] = S - l\,\mathsf{E}[D]$ can be negative.

TF 5.7.5. Claim: Under a base-stock policy, $I^+ - I^- = I$.

Ex 5.7.6. Suppose $l = 0$. What values do X, $\mathsf{E}[I]$, $\mathsf{E}\left[I^+\right]$ and $\mathsf{E}[I^-]$ take?

Ex 5.7.7. Why does the definition of β follow from (2.5.4)?

[5] *A function f is coercive if* $\lim_{x \to \pm\infty} f(x) = \infty$.

The cost function $S \mapsto c(S)$ is convex and, provided $h > 0$ and $b > 0$, coercive.[5]

Ex 5.7.8. Prove that there exists an order-up-to level S^* that minimizes the average cost.



5.8 ANALYTIC RESULTS FOR THE (Q,r) POLICY

In this section, we extend the analysis of a single-item inventory system under a base-stock policy to a (Q,r) inventory system.

THERE IS A convenient way to relate the (Q,r) system to a number of base-stock systems. To see this, imagine two animals, a squirrel that observes the (Q,r) system at the end of each demand period, and a bear that only wakes up when the inventory position right at the start of the interval equals some fixed i, where i must lie in the set $\{r+1,\dots r+Q\}$; if not, the bear hibernates. Clearly, from the bear's point of view, the system behaves as a base-stock system with order-up-to level i, because whenever the bear wakes, the inventory position is i, and the position just before the bear falls asleep is $i-D$. More generally, we can associate a different bear with each position in the set $\{r+1,\dots r+Q\}$, so exactly one bear is awake at any time. To combine the statistics observed by the bears and the statistics observed by the squirrel, we need, for each i, the long-run fraction of periods in which the inventory position P equals i. For instance, if the squirrel knows[1]

$$p(i) = \lim_{T\to\infty} \frac{1}{T} \sum_{k=1}^{T} \mathbb{1}\{P_k = i\},$$

for all $i \in \{r+1,\dots,r+Q\}$, then it can retrieve the ready rate $\sum_{i=r+1}^{Q+r} p(i)\alpha(i)$ from the ready rate $\alpha(i)$ observed by the bear with order-up-to level i.[2]

Theorem 5.8.1. Assume that the greatest common divisor of Q and the possible demand sizes is 1 (in particular, $\mathsf{P}\{D=1\} > 0$ suffices). Under the (Q,r) policy, $p(i) = \mathsf{P}\{P=i\} = 1/Q$ for $i = r+1,\dots,r+Q$, in the long-run time limit. Consequently, I is distributed as $P - X$.

—

Proof. For simplicity take $Q = 3, r = 0$, so that the inventory position P cycles between the levels $1,2$ and 3.[3] We apply level-crossing arguments, as follows. Write $f_i = \mathsf{P}\{D \in \{i, i+3, i+6, \dots\}\}$ for the probability that the period demand equals a multiple of 3 plus i, $i = 0,1,2$. Suppose that, at the start of period k, the position $P_k = 3$. Since we order in multiples of $Q = 3$, the probability $\mathsf{P}\{P_{k+1} = 3 | P_k = 3\} = f_0$, i.e., equal to the probability that D_k is a multiple of 3. Likewise, $\mathsf{P}\{P_{k+1} = 2 | P_k = 3\} = f_1$, and $\mathsf{P}\{P_{k+1} = 1 | P_k = 3\} = f_2$. Next, since we again order in multiples of Q, $\mathsf{P}\{P_{k+1} = 1 | P_k = 2\} = f_1$, and similarly for the other possibilities. Therefore, we can use level crossing to conclude that in the long run, $(f_1 + f_2)p(3) = f_2 p(2) + f_1 p(1)$. However, because of the (Q,r) ordering rule, we can cyclically change $p(1), p(2)$ and $p(3)$ in this equality. By symmetry, the uniform distribution is a solution. The gcd condition

[1] *We assume that the limit over all sample paths $\{P_k\}$ exists almost surely, so that we can invoke the strong law of large numbers.*

[2] *We label the ready rate of the base-stock model by the order-up-to level.*

[3] *The proof is the same for general Q and r, but needs a bit more notation.*

makes the position process irreducible, so this is the unique long-run time distribution: $p(i) = 1/3$ for all i.

For the second claim, condition on the inventory position. When $P = i$, the system behaves as a base-stock system with order-up-to level i, so by Th. 5.7.3, the conditional rv $I \mid P = i \sim i - X$. Since P and X are independent (the position depends on past demands, while X is the lead-time demand over future periods), it follows that $I \sim P - X$. □

WITH THE ABOVE theorem the ready rate for the (Q, r) model can be computed with the formula

$$\alpha = \frac{1}{Q} \sum_{i=r+1}^{r+Q} \alpha(i) = \frac{1}{Q} \sum_{i=r+1}^{r+Q} \mathsf{P}\{X + D \le i\}, \tag{5.8.1}$$

where $\alpha(i)$ is the ready rate of the base-stock model with reorder level i.

For the average inventory level, we write $I(i)$ for the inventory level of the base-stock model with order-up-to level i, and get

$$\mathsf{E}[I] = \frac{1}{Q} \sum_{i=r+1}^{r+Q} \mathsf{E}[I(i)] = \frac{1}{Q} \sum_{i=r+1}^{r+Q} (i - \mathsf{E}[X]) = \frac{Q+1}{2} + r - \mathsf{E}[X].$$

The expected number of backorders $\mathsf{E}[I^-]$ and expected on-hand inventory $\mathsf{E}\left[I^+\right]$ can be found by equivalent summations:

$$\mathsf{E}\left[I^+\right] = \frac{1}{Q} \sum_{i=r+1}^{r+Q} \mathsf{E}\left[I^+(i)\right], \qquad \mathsf{E}[I^-] = \frac{1}{Q} \sum_{i=r+1}^{r+Q} \mathsf{E}[I^-(i)],$$

using (5.7.3) and (5.7.4) of the base-stock model.

Finally, with h and b the holding and backorder cost per item per period, the total average cost becomes

$$c(Q, r) = K \frac{\mathsf{E}[D]}{Q} + h\mathsf{E}\left[I^+\right] + b\mathsf{E}[I^-].$$

With regard to the first term, note that $\mathsf{E}[D]$ is the average demand per period, so $\mathsf{E}[D]/Q$ is the order frequency.

It remains to find optimal values for r and Q. Fast algorithms exist to compute optimal r and Q, but we omit this. A simple workaround is to perform a complete grid search over a reasonable set of pairs of r and Q. Another simple heuristic is to use the EOQ[4] formula to set $Q = \sqrt{2\mathsf{E}[D]\,K/h}$ and then search for r such that the total average cost is minimal.[5]

With similar methods we can search for an r such that the α or β service level criteria are met.

The cost is the sum of the ordering cost and the inventory cost.

[4] *See [5.8.7], and set* $d = \mathsf{E}[D]$.

[5] *For that value of* Q*!*

Algorithm 5.8.2 Compute the average cost of a (Q, r) policy.

1: **Input:** inventory and demand pmf maps p_I, p_D, costs h, b, K, order size Q.
2: **Output:** average cost.
3: **function** QRCOST(p_I, p_D, h, b, K, Q)
4: $\quad c_{\text{order}} \leftarrow K \cdot \text{MEAN}\left(p_D\right)/Q$
5: $\quad c_{\text{inventory}} \leftarrow \text{INVENTORYCOST}\left(p_I, h, b\right)$
6: $\quad$ **return** $c_{\text{order}} + c_{\text{inventory}}$

The function EVALUATEQR implements the preceding formulas. It uses Th. 5.8.1 to construct the uniform pmf of the inventory position and then reuses the functions developed for the base-stock model.

Algorithm 5.8.3 Compute the performance of a (Q, r) policy.

Input: demand pmf map p_D, lead time l, costs h, b, K, and policy parameters Q, r.
Output: ready rate α, fill rate β, and average cost c.
function EVALUATEQR(p_D, l, h, b, K, Q, r)
    $p_X \leftarrow$ CONVOLUTION$\left(p_D, l\right)$
    $p_P \leftarrow$ MAP()
    for $i = r+1, \ldots, r+Q$ **do**
        $p_P(i) \leftarrow 1/Q$
    $p_I \leftarrow$ SUBTRACT$\left(p_P, p_X\right)$
    $\alpha \leftarrow$ READYRATE$\left(p_I, p_D\right)$
    $\beta \leftarrow$ FILLRATE$\left(p_I, p_D\right)$
    $c \leftarrow$ QRCOST$\left(p_I, p_D, h, b, K, Q\right)$
    return α, β, c

FOR THE LIGHTHOUSE Company case specified in Alg. 2.5.4, we compute the performance of a (Q, r) policy with the above formulas and the algorithms of Section 1.5. To select the policy parameters, we apply the general grid-search algorithm of Alg. 2.5.2 to the exact cost $c(Q, r)$.

Algorithm 5.8.4 Find a low-cost (Q, r) policy for the Lighthouse Company.

Input: policy cost function $c(Q, r)$.
Output: best (Q, r) on the specified grid and its cost.
$G \leftarrow [(Q, r) : Q \in [1, \ldots, 30],\ r \in [-5, \ldots, 15]]$
$qr_cost \leftarrow \lambda : (Q, r) \mapsto c(Q, r)$
$(Q^\star, r^\star), c^\star \leftarrow$ GRIDSEARCH$\left(qr_cost, G\right)$
return $(Q^\star, r^\star), c^\star$

For every displayed value of Q, the table gives the value of r with the lowest cost in the grid, together with the resulting ready rate, fill rate, and average cost.

Q	r	$\alpha(Q,r)$	$\beta(Q,r)$	$c(Q,r)$
16	5	0.899	0.906	14.003
17	5	0.905	0.911	13.864
18	5	0.911	0.916	13.777
19	5	0.915	0.921	13.734
20	5	0.920	0.925	13.729
21	5	0.923	0.928	13.756
22	5	0.927	0.931	13.811
23	5	0.930	0.934	13.890
24	4	0.905	0.909	13.972

Over the entire grid, the lowest cost is attained at $(Q^\star, r^\star) = (20, 5)$, with $c^\star \approx 13.73$.

TF 5.8.1. Claim: The (Q,r) inventory policy can be analyzed by comparing it with multiple base-stock systems.

TF 5.8.2. Claim: In a (Q,r) policy system, the inventory position P at the beginning of an interval must always be equal to the reorder level r.

TF 5.8.3. Suppose that the demand in every period is $D \equiv 2$ and the lead time is three periods. Claim: The lead-time demand is $X = D_1 + D_2 + D_3 = 6$.

TF 5.8.4. The fill rate for the (Q,r) inventory system is defined as

$$\beta = \mathsf{E}\left[\min\left\{D, I^+\right\}\right] / \mathsf{E}[D].$$

Claim: Applying MINIMUMPMF to the pmf maps of the independent rvs D and I^+, and then dividing its mean by $\mathsf{E}[D]$, gives β.

TF 5.8.5. Claim: The order frequency of the (Q,r) policy is equal to $Q/\mathsf{E}[D]$.

TF 5.8.6. Claim: The average cost under a (Q,r) policy is

$$c(Q,r) = K\frac{Q}{\mathsf{E}[D]} + h\mathsf{E}\left[I^+\right] + b\mathsf{E}[I^-].$$

Ex 5.8.7. Perhaps the most important formula in practical inventory management is the Economic Order Quantity (EOQ) that determines the optimal order quantity in case demand is constant, i.e., $D \equiv d$, the lead time $l = 0$ and backlogging is not allowed. For order quantities with $\gcd(d,Q) = 1$, use the (Q,r) model to derive the EOQ relation $Q^2 = 2dK/h$.

5.9 AN ALGORITHM FOR THE (s,S) AND (T,S) POLICIES

We develop an algorithm by which we can compute the long-run average cost for a single-item inventory system under periodic review and controlled by the (s,S) or the (T,S) policy. This algorithm finds its importance in its underlying ideas: hitting (or stopping) problems, the renewal reward theorem, and dynamic programming. We start with the (T,S) policy, then move on to the (s,S) policy, and then extend to a combination of the other two, the (T,s,S) policy, to be introduced presently.

RECALL THAT UNDER the (T,S) policy, we place a replenishment order every T periods such that the inventory position P equals S after the replenishment. When the inventory level I equals x, the inventory cost is given by

$$c(x) = bx^- + hx^+.$$

By conditioning on $P = x$, and using[1] that $I = P - X$, it follows that

[1] *See Th. 5.7.3.*

$$L(x) = \mathsf{E}[c(I)|P = x] = \mathsf{E}[c(x - X)] = \sum_{k=0}^{\infty} f_k c(x-k) \tag{5.9.1}$$

is the expected inventory cost per period when $P = x$.

Let us write $V(n,x)$ for the expected cost until reordering if we start at position $P = x$ and there are n, $0 \le n \le T$, periods remaining. By reasoning analogously as in Section 5.6, $V(n,x)$ satisfies the recursion

$$V(n,x) = \begin{cases} L(x) + \mathsf{E}\,[V(n-1, x-D)]\,, & n > 0, \\ K, & n = 0. \end{cases}$$

To see this, observe that when $P = x$ and $n > 0$, we first pay $L(x)$ for keeping inventory during one period, then a demand D arrives, and for the next period we have one less period to go until ordering. When $n = 0$, we pay the order cost K to replenish to S. Note that this recursion always terminates because $V(n,\cdot)$ is expressed in terms of $V(n-1,\cdot)$, so always one period less to go.

Since a (renewal) cycle starts at level S, the expected cycle cost is $V(T,S)$ and the expected cycle duration is T periods. The renewal reward theorem implies that the long-run average cost is given by $V(T,S)/T$.

To facilitate the development of the other models, we introduce some important definitions[2] that are used in the above recursion: $L(x)$ is called the *running* cost, K the *hitting* cost, T is the *hitting* time, and $\{n : n = 0\}$ is the *hitting* set.[3]

[2] *Of dynamic programming.*

[3] *We use the adjectives* hitting *and* stopping *interchangeably.*

WITH THIS TERMINOLOGY we can easily structure the analysis of the (s,S) policy. In view of (2.5.3), when the inventory position P enters (or hits) the stopping set $\{i : i \le s\}$ we place an order and pay a stopping cost K. When $P > s$, we pay the running cost $L(x)$ and continue for another period. Hence, the expected cost until stopping satisfies the recursion

$$V(s,x) = \begin{cases} L(x) + \mathsf{E}\,[V(s, x-D)]\,, & x > s, \\ K, & x \le s. \end{cases}$$

For the hitting time we find

$$T(s,x) = \begin{cases} 1 + \mathsf{E}\,[T(s, x-D)]\,, & x > s, \\ 0, & x \le s, \end{cases}$$

because the running cost is 1 if we continue for another period, and when the process hits the stopping set, the time stops counting, so the stopping cost is 0. Note that this recursion is *structurally identical* to that of V.

To find an algorithm, we expand the recursion for V at $x > s$ as

$$\begin{aligned} V(s,x) &= L(x) + \mathsf{E}\,[V(s, x-D)] = L(x) + \sum_{k=0}^{\infty} f_k V(s, x-k) \\ &= L(x) + f_0 V(s,x) + f_1 V(s, x-1) + \cdots . \end{aligned}$$

Clearly, $V(s,x)$ appears at the LHS and RHS, so $V(s,x)$ is expressed in terms of itself, that is, circular.[4] The way out is to isolate $V(s,x)$ by rewriting this as

$$(1 - f_0) V(s,x) = L(x) + f_1 V(s, x-1) + f_2 V(s, x-2) + \cdots . \tag{5.9.2}$$

[4] *This is different from the recursion for the (T,S) policy.*

[5] *This is not restrictive, because $\mathsf{P}\{D = 0\} = 1$ implies that there is no demand, in which case there is no need to keep inventory.*

Thus, assuming that[5] $f_0 < 1$,

$$V(s,x) = (1-f_0)^{-1}\left(L(x) + \sum_{k=1}^{\infty} f_k V(s,x-k)\right). \tag{5.9.3}$$

We can now use one algorithm to compute V and T by adjusting the running and stopping cost accordingly.

Finally, when the expected cycle cost $V(s,S)$ and expected hitting time $T(s,S)$ have been computed, the long-run average cost is $V(s,S)/T(s,S)$.

BY COMBINING THE two policies above we obtain the (T,s,S) policy. The recursion for $V(n,s,x)$ specifies the behavior:

$$V(n,s,x) = \begin{cases} L(x) + \mathsf{E}\left[V(n-1,s,x-D)\right], & x > s, \\ K, & x \le s \text{ or } n = 0. \end{cases} \tag{5.9.4}$$

The structure of the recursion for $T(n,s,x)$ is the same, but with running cost 1 and stopping cost 0.

To see how the (s,S) and (T,s,S) policy relate, realize that when T becomes very large, the set $set i : i \le s$ will be hit with overwhelming probability long before the time-to-go has become 0. Therefore, the average cost under the (s,S) policy must be $V(s,S)/T(s,S) = \lim_{n\to\infty} V(n,s,S)/T(n,s,S)$. By similar reasoning, $V(T,S)/T = \lim_{s\to-\infty} V(T,s,S)/T(T,s,S)$.

IT IS INTERESTING to understand why we can obtain the closed-form expression $T(q) = q/(\mu-\lambda)$ in (5.6.1), while for the (s,S) and (T,s,S) policies we need an algorithm. The difference is that in the earlier situation[6] $T(q) = 1 + \mathsf{E}\left[T(q+Y-1)\right]$ while in the case of inventory control, the inventory position can jump down by more than one unit when $\mathsf{P}\{D \ge 2\} > 0$. Thus, the behavior of v and T just above s is more complicated. Moreover, in the queueing situation, the queue length can never be negative, while in the inventory situation the level I can take negative values. As a consequence, the cost function $L(x)$ is not linear[7] , hence the idea to substitute $T(q) = \alpha q + \beta$ and solve for α and β will not work in general.

[6] *So, possibly big steps upward, but only single steps downward.*

[7] *But still convex.*

Next, we note that deriving a recursive algorithm for the cost under the (Q,r) policy seems not so attractive. To see why, observe that the stopping cost is given by $V(r,x) = V(r,x+Q)$, that is, it is no longer constant such as K, but dependent on other values of v. Therefore, the simplifying trick that led to (5.9.2) cannot be applied here.

THE RECURSIONS LEAD directly to algorithms. We first construct the running-cost function. For a position x, the inventory level has the same distribution as $x - X$, so we can reuse the functions developed for the base-stock model.

Algorithm 5.9.1 Construct the running-cost function for an inventory model.

```
Input: demand pmf map p_D, lead time l, costs h, b.
Output: running-cost function L.
function RunningCost(p_D, l, h, b)
    p_X ← Convolution(p_D, l)
    function L(x)
        p_I ← Subtract(Constant(x), p_X)
        return InventoryCost(p_I, h, b)
    return L
```

The function TsSHittingValue returns the function V for the (T, s, S) policy. It uses the Alg. 1.2.13 algorithm to speed up the recursion.

Algorithm 5.9.2 Construct a hitting-value function for a (T, s, S) policy.

```
Input: demand pmf map p_D, running-cost function R, and stopping-cost function H.
Output: a hitting-value function V.
function TsSHittingValue(p_D, R, H)
    function Stop?(n, s, x)
        return x ≤ s or n = 0
    function V(n, s, x)
        if Stop?(n, s, x) then
            return H(x)
        return R(x) + Sum([p_D(d) · V(n − 1, s, x − d) : d ∈ Keys(p_D)])
    V ← Cache(V)
    return V
```

For the (s, S) policy, the recursion does not reduce a time index. We therefore remove the term for $D = 0$ from the sum and divide by $1 - p_D(0)$, as in (5.9.3). We select positive demand values once before evaluating the recursion.

Algorithm 5.9.3 Construct a hitting-value function for an (s, S) policy.

```
Input: demand pmf map p_D, running-cost function R, and stopping-cost function H.
Output: a hitting-value function V.
function sSHittingValue(p_D, R, H)
    positiveKeys ← Filter(Keys(p_D), λ: d ↦ 𝟙{d > 0})
    function Stop?(s, x)
        return x ≤ s
    function V(s, x)
        if Stop?(s, x) then
            return H(x)
        z ← Sum([p_D(d) · V(s, x − d) : d ∈ positiveKeys])
        return (R(x) + z)(1 − p_D(0))
    V ← Cache(V)
    return V
```

The function SSAVERAGECOST provides the direct interface for an (s, S) policy. It uses the hitting-value function once with the inventory costs and once with a running cost of one to obtain the expected cycle cost and cycle duration.

Algorithm 5.9.4 Compute the long-run average cost of an (s, S) policy.

Input: demand pmf map p_D, lead time l, costs h, b, K, levels s, S.
Output: long-run average cost.
function SSAVERAGECOST(p_D, l, h, b, K, s, S)
    $L \leftarrow$ RUNNINGCOST$\big(p_D, l, h, b\big)$
    $V \leftarrow$ SSHITTINGVALUE$\big(p_D, L, \lambda\colon x \mapsto K\big)$
    $\tau \leftarrow$ SSHITTINGVALUE$\big(p_D, \lambda\colon x \mapsto 1, \lambda\colon x \mapsto 0\big)$
    return $V(s, S)/\tau(s, S)$

The corresponding interface for a (T, s, S) policy differs only in the hitting-value function and its arguments.

Algorithm 5.9.5 Compute the long-run average cost of a (T, s, S) policy.

Input: demand pmf map p_D, lead time l, costs h, b, K, levels T, s, S.
Output: long-run average cost.
function TSSAVERAGECOST(p_D, l, h, b, K, T, s, S)
    $L \leftarrow$ RUNNINGCOST$\big(p_D, l, h, b\big)$
    $V \leftarrow$ TSSHITTINGVALUE$\big(p_D, L, \lambda\colon x \mapsto K\big)$
    $\tau \leftarrow$ TSSHITTINGVALUE$\big(p_D, \lambda\colon x \mapsto 1, \lambda\colon x \mapsto 0\big)$
    return $V(T, s, S)/\tau(T, s, S)$

The (T, S) policy follows by taking $s = -\infty$ in TSSAVERAGECOST.

THE TWO HITTING-VALUE functions above have the same basic structure. Each accepts a running-value function R and a stopping-value function H. With $R = L$ and $H(x) = K$, the returned function computes the expected cost until stopping. With $R(x) = 1$ and $H(x) = 0$, it computes the expected time until stopping. The average-cost functions divide the first quantity by the second.

The policies differ in their stopping rules. For the (T, s, S) policy, the process stops when $x \le s$ or $n = 0$. For the (s, S) policy, it stops when $x \le s$. Moreover, because D can be zero, the latter recursion contains $V(s, x)$ on both sides. Isolating this term produces the factor $1 - p_D(0)$ in (5.9.3).

An important point is that TSSHITTINGVALUE and SSHITTINGVALUE return an inner function V, rather than a number. This function retains access to p_D, R, H, and *cache* created by the enclosing function. Only when we call $V(T, s, S)$ or $V(s, S)$ do we obtain a numerical value. Thus, V is a closure; see [5.9.6] for a simpler example.

LET US EXAMINE a few numerical examples.

We can check that for T large, the (T, s, S) policy and the (s, S) have the same average cost.

For the (T, S) policy, setting T to a large value is costly, as expected.

FOR THE LIGHTHOUSE Company case specified in Alg. 2.5.4, take $s = 16$ and $S = 20$. For the (s,S) policy, the algorithms give the following cycle statistics and long-run average cost.

Expected hitting time	Expected cycle cost	Average cost
2.2865088	72.04810304	31.51009217

For the (T,s,S) policy, the average cost approaches that of the (s,S) policy as T increases.

T	Average cost
1	60.3
2	38.59665072
10	31.51019163

For the (T,S) policy, taking T too large is costly, as expected.

T	Average cost
1	60.3
2	34.54166667
10	35.83618568
20	193.77674739

TF 5.9.1. Claim: For the (s,S) policy, the expected cost until stopping $V(s,x)$ satisfies the same recursion as the one for the (T,S) policy.

TF 5.9.2. Claim: The next formula is a correct implementation to compute the expected time to hit s under the (s,S) policy:

$$\begin{cases} (1-f_0)^{-1}(1+\mathsf{E}[T(s,x-D)]), & x > s, \\ 0, & x \le s, \end{cases}$$

TF 5.9.3. Let p_D be the demand pmf, L the running-cost function, and K the ordering cost. Consider the calls

```
V_sS ← sSHITTINGVALUE(p_D, L, λ: x ↦ K)
V_TS ← TsSHITTINGVALUE(p_D, L, λ: x ↦ K)
```

Claim: $V_{sS}(s,S)$ is the expected cost until ordering under the (s,S) policy, and $V_{TS}(T,-\infty,S)$ is the expected cost until ordering under the (T,S) policy.

TF 5.9.4. Let p_D be the demand pmf and suppose that $p_D(0) < 1$. Consider the following recursion for the expected cost until ordering under an (s,S) policy.

```
function V(s, x)
    if x ≤ s then
        return K
    return L(x) + SUM([p_D(d) · V(s, x − d) : d ∈ KEYS(p_D), d > 0])
```

Claim: this recursion correctly computes $V(s,x)$.

TF 5.9.5. Claim: When $P = x$ and the leadtime $l > 0$, the inventory cost under the (T, s, S) policy is given by

$$c(x) = bx^- + hx^+.$$

Ex 5.9.6. Consider the following functions to illustrate how functions like `hitting_time` and `hitting_cost` can be understood.

```
def exp_apply(exponent):
    def apply(base):
        return base ** exponent
    return apply

lion = exp_apply(2)
tiger = exp_apply(3)
```

Without running the code, what do you think will be printed by:

```
print(exp_apply(1)(9))
print(lion(5))
print(tiger(4))
```



5.10 HITTING PROBLEMS WITH APPLICATIONS IN SINGLE-ITEM INVENTORY CONTROL

In a single-item inventory under period review and controlled by an (s, S) policy, every time the inventory position becomes s or lower, a replenishment up to state S is placed at cost K.[1] When the position is at $x > s$, the inventory cost is $L(x)$, see (5.9.1). Typically, we are interested in the long-run average cost when the position starts at S and stops when getting at or below s. By the renewal reward theorem, it suffices to compute the expected cost $v(s, S)$ and expected time $T(s, S)$ of one cycle that starts at S and stops at $x \le s$. The long-run average cost is then given by $v(s, S)/T(s, S)$.

[1] *If you don't like challenges, you can skip this section.*

Both $v(s, S)$ and $T(s, S)$ can be found as the solution of a hitting problem. The stopping set is $x \le s$. When in state $x > s$, the continuation (or running) cost for v is, as said, $L(x)$, and 1 for T, because it costs 1 period to stay 1 period in state x. The stopping cost for v is the ordering cost K, and it is 0 for T, because once stopped, time no longer progresses.

Below we discuss a general algorithm to numerically solve such hitting problems.

A BIT OF theory helps to set up this algorithm. We start with a discrete Markov chain $X = \{X_k, k \ge 0\}$ with a reasonably small state space, e.g., at most several thousand states.[2] The state space is split into two disjoint sets C and D. When $X_k \in C$, the process continues, while if $X_k \in D$, it stops. Define the stopping time $\tau = \inf\{k : X_k \in D\}$, and assume that the transition probabilities $P(x, y)$ of X are such that the chain gets absorbed in D in finite expected time, that is, $\mathsf{E}_x[\tau] < \infty$ for all possible initial

[2] *Otherwise we should resort to other numerical methods.*

states x. We write $\tilde{L}(x) = L(x)\mathbb{1}\{x \in C\} + K\mathbb{1}\{x \in D\}$ for the cost associated with spending a period or stopping in state x. The interesting question is to ask: What is the expected cost until the process stops?[3]

[3] *For suitable conditions on c, see for instance, Dynkin and Yushkevich 1968.*

Focus first on some state y, then, when starting in x, $\mathsf{E}_x\left[\sum_{t=0}^{\tau} \mathbb{1}\{X_t = y\}\tilde{L}(y)\right]$ is the expected cost for spending time in, or hitting, state y. With this, we write the total cost until stopping as

$$v(x) = \sum_y \mathsf{E}_x\left[\sum_{t=0}^{\tau} \mathbb{1}\{X_t = y\}\tilde{L}(y)\right].$$

By splitting the expectation and taking the summation over t out of the expectation we obtain a very useful expression for v:

$$v(x) = \sum_y g(x,y)\tilde{L}(y),$$

where g, known as Green's kernel, is given by

$$g(x,y) = \sum_{t=0}^{\infty} \mathsf{E}_x\left[\mathbb{1}\{X_t = y, t \le \tau\}\right].$$

It represents the expected number of visits to y until absorption when the chain starts at x. In terms of probabilities, g can be written as[4]

[4] *Use that* $\mathsf{E}[\mathbb{1}\{A\}] = \mathsf{P}\{A\}$ *for an indicator event* A.

$$g(x,y) = \sum_{t=0}^{\infty} \mathsf{P}_x\{X_t = y, t \le \tau\}. \tag{5.10.1}$$

Finally, if X can start at some state x with probability μ_x, the expected number of visits to y becomes

$$g(y) = \sum_x \mu_x g(x,y). \tag{5.10.2}$$

THE FOLLOWING SEEMINGLY simple, and certainly short, algorithm computes g as defined in (5.10.2).

Algorithm 5.10.1 Compute expected visits to states before absorption.

```
1:  Input: a map of initial probabilities μ, transitions P(x), a stopping
    rule STOP?, cutoff ε.
2:  Output: expected visits g.
3:  function GREEN(μ, P, STOP?, ε)
4:      g ← MAP()
5:      while KEYS(μ) ≠ ∅ do
6:          states ← KEYS(μ)
7:          for x ∈ states do
8:              p ← μ(x)
9:              g(x) +← p
10:             μ(x) ← 0
11:             if ¬STOP?(x) then
12:                 for (y, p_xy) ∈ P(x) do
13:                     μ(y) +← p · p_xy
14:     μ ← FILTERVALUES(μ, λ: p ↦ p > ε)
15:     return g
```

Before explaining how this works, we discuss this function's arguments. The input map μ maps a state x to the initial probability $\mu(x)$ of that state. For instance, when the process starts (almost) surely at $X_0 = x$, we set $\mu(x) \leftarrow 1$. Next, $P(x)$ returns a list of pairs (y, p_{xy}), where y is a state to which X can move from x and p_{xy} is the probability of that transition. The rule STOP?(x) is true when the chain should stop (is absorbed) at x, and false otherwise. Finally, ε is a small number that prevents keeping states that have negligible probability of being accessed.

So, intuitively speaking, how does this algorithm work? As long as not all probability mass has been absorbed[5] , a for loop cycles over the states x currently in the map. The map g counts the expected number of visits to a state x and is updated in every round of the loop. We fix the range of the for loop by storing KEYS(μ) in *states*, because the keys of μ can change in the body of the loop. When x is processed, $\mu(x)$ is the probability mass waiting to be processed at x. In accordance with (5.10.1), we add this probability to the expected total number of visits $g(x)$ and assign it to p for ease.

[5] *That is,* KEYS(μ) *is not yet empty.*

We set $\mu(x) \leftarrow 0$ for two reasons. When the chain is absorbed (stopped) at x, we continue with the next state to be checked in the **for** loop. In this case, the total probability mass should decrease by the absorption probability p, which is indeed achieved by setting $\mu(x) \leftarrow 0$. If the chain does not stop at x, the probability p should be distributed over all states y to which X can transition. If $y \neq x$, then $\mu(y) \overset{+}{\leftarrow} p \cdot p_{xy}$, because y could have been accessed from other states too. When $y = x$, the probability of staying in x is $p \cdot p_{xy}$.[6] Since we earlier set $\mu(x) \leftarrow 0$, we can also use $\mu(x) \overset{+}{\leftarrow} p \cdot p_{xx}$ for state x. In other words, the same update works for all next states, without singling out x.

[6] *Recall that the expected number of consecutive periods to stay in state x is* $\sim \text{Geom}(p_x)$.

Once the for loop completes, μ is replaced by a map containing only states with probability greater than ε.

There remains, however, a fundamental question: why is it allowed to change μ in the for loop itself, and does it really converge to the Green kernel we discussed above?

TO REALLY UNDERSTAND why the GREEN function works, we need to discuss some further ideas of discrete Markov chains. Recall the interpretation of the Green kernel $g(x, y)$ as the expected number of times y will be visited up to absorption and when starting in $X_0 = x$. For the stopped process, set $P(x, y) = 0$ whenever $x \in D$. Thus a stopping state contributes one visit but has no outgoing transitions. Clearly, using the Markov property, g should satisfy the recursive expression

$$\begin{aligned} g(x,y) &= \mathsf{E}_x\left[\mathbb{1}\{X_0 = y\}\right] + \mathsf{E}_x\left[g(X_1, y)\right] \\ &\overset{1}{=} \mathbb{1}\{x = y\} + \sum_z \mathsf{E}_x\left[\mathbb{1}\{X_1 = z\}\right] g(z, y) \\ &\overset{2}{=} \mathbb{1}\{x = y\} + \sum_z P(x,z) g(z,y); \end{aligned}$$

in 1 we use that $g(z, y)$ is a constant, hence can be taken out of the expectation, in 2 that $\mathsf{E}_x[\mathbb{1}\{X_1 = z\}] = P(x, z)$. Casting this in matrix format makes this structure clearer:

$$G = I + PG. \tag{5.10.3}$$

Then, by using this recursive structure, and noting that X will be absorbed in some set D in expected finite time and P is finite so that P^t converges geometrically fast to 0 elementwise as $t \to \infty$,

$$G = I + PG = I + P(I + PG) = I + P + P^2 G = \sum_{t=0}^{n} P^t + P^{n+1} G \to \sum_{t=0}^{\infty} P^t.$$

And, if we define recursively $\mu^t = \mu^{t-1} P = \mu P^t$, so that $\mu_y^t = \sum_x \mu_x P^t(x, y)$,

$$g(y) = \sum_x \mu_x \sum_{t=0}^{\infty} P^t(x, y) = \sum_{t=0}^{\infty} \mu_y^t.$$

The advantage of this algorithm over solving $G = I + PG$ with matrix methods is as follows. The latter is a matrix equation for which we need to specify the size of G. However, the states that have a reasonable probability of being visited are not yet known. Thus, solving $G = I + PG$ may include many more or less superfluous states, which is numerically wasteful.

LET US DEVELOP a few intermediate algorithms that apply the above ideas to compute $g(y)$. Note that μ is the starting distribution of X_0. The key part updates g and μ as $g^{k+1} = g^k + \mu^k$, and $\mu^{k+1} \leftarrow \mu^k P$. Thus, after k iterations, g equals $\sum_{t=0}^{k-1} \mu^t$ and μ equals μ^k. In the update of μ we prune on too small events. As μ^k becomes smaller and smaller as k increases, eventually, all elements of μ become less than ε, and the while loop terminates.

Algorithm 5.10.2 Compute expected visits by updating visits and probability mass separately.

```
1:  Input: initial probability map μ, transitions P(x), stopping rule STOP?,
    cutoff ε.
2:  Output: expected visits g.
3:  function GREENTWOPASS(μ, P, STOP?, ε)
4:      g ← MAP()
5:      while KEYS(μ) ≠ ∅ do
6:          for x ∈ KEYS(μ) do                    ▷ Compute g^{t+1} = g^t + μ^t.
7:              g(x) +← μ(x)
8:          next ← MAP()
9:          for x ∈ KEYS(μ) do                    ▷ Compute next = μ^t P.
10:             if ¬STOP?(x) then
11:                 for (y, p_xy) ∈ P(x) do
12:                     next(y) +← μ(x) · p_xy
13:         μ ← FILTERVALUES(next, λ: p ↦ p > ε)   ▷ Retain mass for μ^{t+1}.
14:     return g
```

We can combine the updating of g and *next* in one for loop because these maps do not affect each other in the for loop. Observe that the assignment $p \leftarrow \mu(x)$ avoids looking up $\mu(x)$ twice.

Algorithm 5.10.3 Compute expected visits with one loop for both updates.

```
1:  Input: initial probability map μ, transitions P(x), stopping rule STOP?, cutoff ε.
2:  Output: expected visits g.
3:  function GreenCombined(μ, P, STOP?, ε)
4:      g ← Map()
5:      while Keys(μ) ≠ ∅ do
6:          next ← Map()
7:          for x ∈ Keys(μ) do
8:              p ← μ(x)
9:              g(x) +← p                          ▷ Update g^{t+1}.
10:             if ¬STOP?(x) then
11:                 for (y, p_xy) ∈ P(x) do        ▷ Accumulate μ^t P in next.
12:                     next(y) +← p · p_xy
13:         μ ← FilterValues(next, λ: p ↦ p > ε)   ▷ Retain mass for μ^{t+1}.
14:     return g
```

Observe that in these two algorithms, the **while** loop *first* updates $g^{t+1} = g^t + \mu^t$, and *then* computes $\mu^t P$ and prunes small entries to obtain μ^{t+1}. In other words, the entire vector μ^{t+1} is computed before adding it to g^{t+1} in the next pass of the **while** loop.

THE NEXT PHASE in the analysis of GREEN is to see why it is allowed to update μ within the **for** loop, that is, why GREEN gives the same value for $g(y)$ as the above two algorithms. The next theorem answers this.

Theorem 5.10.4. Let P be a matrix such that $G(I-P) = I$ has a solution G, compare (5.10.3). Assume $\varepsilon = 0$, so no positive probability mass is pruned. Here k counts individual state updates, rather than passes through the **while** loop. If $g^0 = 0$ and $\mu^0 = \mu$, then $g^k \to \mu G$ when the update rules of GREEN

$$p = \mu_x^k, \qquad g^{k+1} = g^k + p\delta_x, \qquad \mu^{k+1} = \mu^k - p\delta_x + p\delta_x P,$$

are applied sequentially for all x in the support of μ^k.

—

Proof. We first show that the quantity $H^k = g^k + \mu^k G$ remains the same for all $k \geq 0$ under the update rules. Using substitution of the update rules,

$$\begin{aligned} H^{k+1} &= g^{k+1} + \mu^{k+1} G && = g^k + p\delta_x + \big(\mu^k - p\delta_x + p\delta_x P\big)G \\ &= g^k + p\delta_x + \mu^k G - p\delta_x (I-P)G \\ &= g^k + \mu^k G = H^k, \end{aligned}$$

because $(I-P)G = I$, cf. (5.10.3). Then, by recursion, $g^k + \mu^k G = g^0 + \mu^0 G = \mu G$ since $g^0 = 0$ and $\mu^0 = \mu$. Since the stopped chain is transient and

every state with pending mass is eventually processed, $\mu^k \to 0$. The invariant $g^k + \mu^k G = \mu G$ then gives $g^k \to \mu G$. □

Without pruning, GREEN gives the same Green kernel as the other algorithms. With $\varepsilon > 0$, it gives an approximation.

WE HAVE SOME final remarks.

- The advantage of GREEN is that we do not have to make a guess of the states that will be visited by X with reasonable probability. When using matrix methods to compute G such that $G(I - P) = I$, we need to expand P to many states that are irrelevant (up to ε).
- In inventory control, the update over P is often a fixed list. In such cases, it is quite a bit faster to update $\mu(x - i) \xleftarrow{+} p \cdot \mathsf{P}\{D = i\}$ for each demand value i than to pass P.
- The difference between the, say, regular approaches and GREEN can be compared to a Jacobi versus Gauss-Seidel updating scheme. In my experiments with inventory control, the GREEN algorithm converges in about half the time of the other schemes.

LET US DEMONSTRATE the GREEN algorithm for the (s, S) inventory control algorithm, see Alg. 5.10.5. It obtains the Lighthouse Company data from Alg. 2.5.4, constructs the running cost $L(x)$, and sets $\mu^0(S) = 1$ and zero elsewhere. It then defines the stopping and transition rules and calls GREEN. The expected visits $g(x)$ give the cycle length, cycle cost, and average cost.

Algorithm 5.10.5 Compute the average cost of an (s, S) policy using GREEN.

```
Input: reorder level s, order-up-to level S, cutoff ε.
Output: average cost averageCost.
(p_D, l, h, b, K) ← LIGHTHOUSEDATA()
p_X ← CONVOLUTION(p_D, l)
c ← λ: z ↦ b·z⁻ + h·z⁺
L ← λ: x ↦ EXPECTATION(p_X, λ: u ↦ c(x − u))
μ ← CONSTANT(S)
STOP? ← λ: x ↦ x ≤ s
P ← λ: x ↦ [(x − i, p_D(i)) : i ∈ KEYS(p_D)]
g ← GREEN(μ, P, STOP?, ε)
T(s, S) ← Σ_{x>s} g(x)
v(s, S) ← Σ_{x>s} g(x)·L(x) + K·Σ_{x≤s} g(x)
averageCost ← v(s, S)/T(s, S)
```

In a second example, we consider an inventory system that accepts returns. These returns can be seen as negative demand. In that case, the regular algorithm to compute the average cost under an (s, S) policy no longer works, but GREEN has no problems with such cases. The return probabilities below are the normalized values of the weights used in the reproducibility code. We reuse the stopping rule and start with a fresh point mass at S.

Algorithm 5.10.6 Compute the average cost when customers may return items.

Input: p_D, s, S, c, K, ε, STOP? from the previous example.
Output: average cost with returns *averageCostReturns*.
$p_R \leftarrow \text{MAP}()$
$p_R(0) \leftarrow 8/25, \quad p_R(1) \leftarrow 8/25$
$p_R(2) \leftarrow 6/25, \quad p_R(3) \leftarrow 3/25$
$p_{D-R} \leftarrow \text{SUBTRACT}(p_D, p_R)$
$p_{X'} \leftarrow \text{CONVOLUTION}(p_{D-R}, 2)$
$L' \leftarrow \lambda\colon x \mapsto \text{EXPECTATION}\left(p_{X'}, \lambda\colon u \mapsto c(x-u)\right)$
$P' \leftarrow \lambda\colon x \mapsto \left[(x-i, p_{D-R}(i)) : i \in \text{KEYS}(p_{D-R})\right]$
$\mu \leftarrow \text{CONSTANT}(S)$
$g' \leftarrow \text{GREEN}(\mu, P', \text{STOP?}, \varepsilon)$
$T' \leftarrow \sum_{x>s} g'(x)$
$v' \leftarrow \sum_{x>s} g'(x) \cdot L'(x) + K \cdot \sum_{x \le s} g'(x)$
averageCostReturns $\leftarrow v' / T'$

Opt 5.10.1. In the pruning step of GREEN, why not retain a state x whenever $\mu(x) > \varepsilon$ or STOP?(x) is false?

6

APPROXIMATE MODELS

In Section 6.1 we introduce Sakasegawa's formula, which approximates the expected waiting time in a $G/G/c$ queue. With some license, one could say that the philosophy behind lean manufacturing and the Toyota production system rests on principles derived from this formula. We then illustrate how to use Sakasegawa's formula to estimate waiting times in three queueing settings where the service process is interrupted. In the first case, Section 6.2, the server produces jobs from different families with a change-over time required to switch from one family to another. Such setups reduce the server's available time, thereby increasing the load. To counter this, the server produces in batches of fixed size. In the second case, Section 6.3, the server occasionally requires small adjustments, for example to prevent production quality from degrading below a certain level. Such adjustments typically do not occur during a job's service, but can take place *between* any two jobs. Consequently, the number of jobs served between adjustments (or setups) varies, unlike the situation in which batch sizes are fixed. In the third example, Section 6.4, quality issues or breakdowns may occur *during* a job's service. These increase service-time variability, leading to longer expected queueing times.

Along the way, we use some interesting results from probability theory and the Poisson process.

6.1 THE $G/G/c$ QUEUE

In this section, we introduce Sakasegawa's formula, which gives an approximation for the expected waiting time for a $G/G/c$ queue. We discuss the insights it offers and show how to estimate the squared coefficient of variation of the inter-arrival time from data.

SAKASEGAWA'S FORMULA PROVIDES a reasonable approximation[1] for $\mathsf{E}[W]$ of the $G/G/c$ queue, and has the form

$$\mathsf{E}[W] = \frac{C_a^2 + C_s^2}{2} \frac{\rho^{\sqrt{2(c+1)}-1}}{1-\rho} \frac{\mathsf{E}[S]}{c}, \tag{6.1.1}$$

where the utilization[2] of the station is $\rho = \lambda \mathsf{E}[S]/c \in [0,1)$, and $C_a^2 = \mathsf{V}[X]/(\mathsf{E}[X])^2$ and $C_s^2 = \mathsf{V}[S]/(\mathsf{E}[S])^2$ are the scv of the inter-arrival time and service time, respectively.

Note that Sakasegawa's formula for the $G/G/c$ queue reduces to the PK formula for $\mathsf{E}[W]$ for the $M/G/1$ queue; just fill in $c = 1$ and set $C_a^2 = 1$. Part of the intuition behind Sakasegawa's formula is based on the following idea. To generalize the formula for $\mathsf{E}[W]$ from the $M/M/1$ queue to that of the $M/G/1$ queue, we replace $(1+1)/2$ by $(1+C_s^2)/2$; the result remains exact. Now, to go from the $M/G/1$ queue to the $G/G/1$

[1] *There is no known exact expression for the $G/G/c$ queue.*

[2] *Here is a subtle point. When there are multiple machines, the utilization of each machine is not necessarily equal to ρ. For instance, if there is a preference to choose the 'left-most' machine whenever it is free, then the utilization of this machine is larger than ρ. Only when the machines have the same speed and* routing *is such that each machine receives a fraction λ/c of jobs, all machines in one station have the same utilization.*

queue, replace $(1+C_s^2)/2$ by $(C_a^2+C_s^2)/2$; this is no longer exact, but the approximation remains good for most practical purposes. For the details behind the power with c, we refer to Sakasegawa's paper.

IT IS CRUCIAL to memorize the insights this formula offers. First, we see that $\mathsf{E}[W] \sim (1-\rho)^{-1}$. Consequently, when ρ is large, the waiting time is (very) long. And, not only is $\mathsf{E}[W]$ large, it is also extremely *sensitive* to the actual value of ρ. Clearly, such situations must be avoided and therefore, when trying to improve a queueing system, the first focus should be on reducing ρ.

Second, $\rho = \lambda \mathsf{E}[S]/c$. Thus, when ρ must be made smaller, we have just three options.[3] We can reduce the arrival rate λ of jobs, for instance, by blocking demand or sending it elsewhere, such as to another machine.[4] We can make $\mathsf{E}[S]$ smaller by replacing a server with a faster one or by shifting some of the processing steps of a job to other servers, thereby making job sizes smaller. Finally, we can add servers, i.e., making c larger.[5] If technically possible, adapting c to λ when $\mathsf{E}[S]$ cannot be easily changed is a very effective mechanism for controlling waiting times.

Third, $\mathsf{E}[W] \sim C_a^2$ and $\mathsf{E}[W] \sim C_s^2$, which implies that when the job inter-arrival or service times are very variable, $\mathsf{E}[W]$ is large. Thus, after ensuring that ρ is sufficiently small, it becomes important to concentrate on reducing C_a^2 and C_s^2.[6]

Finally, $\mathsf{E}[W] \sim \mathsf{E}[S]/c$. This means that from the perspective of a job in queue, average job service times are c times as short as 'their own' service times.

CLEARLY, SAKASEGAWA'S EQUATION requires an estimate[7] of C_a^2 and C_s^2. Now it is not always easy in practice to determine the actual service-time distribution, since service times are often estimated by a planner rather than measured. Similarly, the actual arrival moments of jobs are often not registered, only the date, or perhaps the hour, that a customer arrived. Hence, it is often not possible to directly estimate C_a^2 and C_s^2 from the available information.

However, when arrivals form a renewal process and the logging periods are long compared with the inter-arrival times, we can use the counts $\{a_n\}$ to estimate C_a^2 by the variance-to-mean ratio below.[8] Many short periods alone do not ensure that this approximation is accurate.

$$C_a^2 \approx \frac{\tilde{\sigma}^2}{\tilde{\lambda}},$$

where

$$\tilde{\lambda} = \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} a_i, \quad \tilde{\sigma}^2 = \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} a_i^2 - \tilde{\lambda}^2.$$

We derive this result in steps.[9]

First, recall some results from earlier sections. Let the inter-arrival time X have mean $1/\lambda$ and variance σ^2, so that

$$C_a^2 = \frac{\mathsf{V}[X_i]}{(\mathsf{E}[X_i])^2} = \frac{\sigma^2}{1/\lambda^2} = \lambda^2\sigma^2.$$

[3] *And not more!*

[4] *And if customers have a choice, they will take their own measures, simply by going elsewhere.*

[5] *This is precisely what you see in supermarkets.*

[6] *It works the other way too: systems with low variability can deal with higher load.*

[7] *If you are not interested in statistics, you can skip this part.*

[8] *This can also be used to estimate C_s^2 from completion counts during uninterrupted busy time.*

[9] *The arguments are based on a book by Cox on renewal theory. If you don't like to extend your knowledge of statistics, you can skip this derivation.*

Let A_k be the arrival time of the kth arrival and $A(t)$ the number of arrivals up to time t. By the definition of $A(t)$, we have the following useful relation between $A(t)$ and A_k :

$$\mathsf{P}\{A(t) < k\} = \mathsf{P}\{A_k > t\}.$$

Since the inter-arrival times have finite mean and second moment by assumption, we use the central limit theorem to see that

$$\lim_{k\to\infty} \frac{A_k - k/\lambda}{\sigma\sqrt{k}} = N(0,1),$$

where $N(0,1)$ is a standard normal rv with distribution $\Phi(\cdot)$. Similarly, for an α yet to be determined,[10]

$$\frac{A(t) - \lambda t}{\alpha\sqrt{t}} \to N(0,1).$$

For large t, renewal theory gives $\mathsf{E}[A(t)] \sim \lambda t$ and $\mathsf{V}[A(t)] \sim \alpha^2 t$.[11]

Using that $\mathsf{P}\{N(0,1) \le y\} = \mathsf{P}\{N(0,1) > -y\}$, we get

$$\begin{aligned}\Phi(y) &\approx \mathsf{P}\left\{\frac{A_k - k/\lambda}{\sigma\sqrt{k}} \le y\right\} \approx \mathsf{P}\left\{\frac{A_k - k/\lambda}{\sigma\sqrt{k}} > -y\right\} \\ &= \mathsf{P}\left\{A_k > \frac{k}{\lambda} - y\sigma\sqrt{k}\right\}.\end{aligned}$$

We can use this relation between the distributions of $A(t)$ and A_k to see that $\mathsf{P}\{A_k > t_k\} = \mathsf{P}\{A(t_k) < k\}$ where we define for ease

$$t_k = \frac{k}{\lambda} - y\sigma\sqrt{k}.$$

With this we get,

$$\begin{aligned}\Phi(y) &\approx \mathsf{P}\{A_k > t_k\} = \mathsf{P}\{A(t_k) < k\} \\ &= \mathsf{P}\left\{\frac{A(t_k) - \lambda t_k}{\alpha\sqrt{t_k}} < \frac{k - \lambda t_k}{\alpha\sqrt{t_k}}\right\}.\end{aligned}$$

Since $(A(t_k) - \lambda t_k)/\alpha\sqrt{t_k} \to N(0,1)$ as $t_k \to \infty$, the above implies that

$$\frac{k - \lambda t_k}{\alpha\sqrt{t_k}} \to y,$$

as $t_k \to \infty$. Using the above definition of t_k, the LHS of this equation can be written as

$$\frac{k - \lambda t_k}{\alpha\sqrt{t_k}} = \frac{\lambda\sigma\sqrt{k}}{\alpha\sqrt{k/\lambda - y\sigma\sqrt{k}}} y.$$

Since $t_k \to \infty$ is implied by (and implies) $k \to \infty$, we therefore want that α is such that

$$\frac{\lambda\sigma\sqrt{k}}{\alpha\sqrt{k/\lambda - y\sigma\sqrt{k}}} y \to y,$$

as $k \to \infty$. This is precisely the case when

$$\alpha = \lambda^{3/2}\sigma.$$

[10] *This is a common trick: suppose that there exists a constant to take of the scaling, and then try an extra relation to 'ferret out' the scaling constant.*

[11] *These moment asymptotics require a separate argument from convergence in distribution.*

Finally, for t large, or, by the same token, k large,

$$\frac{\sigma_k^2}{\lambda_k} = \frac{\mathsf{V}[A(t)]}{\mathsf{E}[A(t)]} \approx \frac{\alpha^2 t}{\lambda t} = \frac{\alpha^2}{\lambda} = \frac{\lambda^3\sigma^2}{\lambda} = \lambda^2\sigma^2 = C_a^2,$$

where the last equation follows from the above definition of C_a^2.

TF 6.1.1. Sakasegawa's approximation for the waiting time in a $G/G/c$ queue is

$$\mathsf{E}[W] = \frac{C_a^2 + C_s^2}{2} \frac{\rho^{\sqrt{2(c+1)}-1}}{c(1-\rho)} \mathsf{E}[S].$$

We claim that it is exact for the $M/G/1$ queue.

TF 6.1.2. Claim: For a $G/G/1$ queue, if $\rho = 0.95$ then a 5\

TF 6.1.3. Sakasegawa's approximation for the waiting time in a $G/G/c$ queue is

$$\mathsf{E}[W] = \frac{C_a^2 + C_s^2}{2} \frac{\rho^{\sqrt{2(c+1)}-1}}{c(1-\rho)} \mathsf{E}[S].$$

Claim: For a stable $D/D/1$ queue $C_a^2 = 0$ and $C_s^2 = 0$, therefore $\mathsf{E}[W] = 0$.

Ex 6.1.4. Consider a single-server queue at which every minute a customer arrives, precisely at the first second and $S \equiv 50$ s. What are ρ, $\mathsf{E}[L]$, C_a^2, and C_s^2?

Ex 6.1.5. Consider the same single-server system as in [6.1.4], but now the customer service time is stochastic: with probability 1/2 a customer requires 1 minute and 20 seconds of service, and with probability 1/2 the customer requires only 20 seconds of service. What are ρ, C_a^2, and C_s^2?

Ex 6.1.6. In a workplace, when a mechanic has used a specific high precision tool, the tool has to be inspected by a microscope for potential failures. The inter-arrival times of the tools at the microscope is $X \sim \text{Unif}([3,9])$ minutes, and the inspection time S is such that $\mathsf{E}[S] = 5$ minutes and $\mathsf{V}[S] = 4$ minutes squared. Use Sakasegawa's formula to estimate $\mathsf{E}[W]$. Next, when tools arrive as a Poisson process, what would be $\mathsf{E}[W]$? Discuss your findings.

Ex 6.1.7. A machine serves two types of jobs. The service time of jobs of type T, $T = 1,2$, is exponentially distributed with parameter μ_T. The type T of a job is random and independent of anything else, and such that $\mathsf{P}\{T=1\} = p = 1-q = 1-\mathsf{P}\{T=2\}$. (An example is a desk serving adults and children, both of which require different average service times, and p is the probability that an arriving customer is an adult.) Show that the expected service time and variance are given by

$$\mathsf{E}[S] = p\,\mathsf{E}[S_1] + q\,\mathsf{E}[S_2]$$
$$\mathsf{V}[S] = p\,\mathsf{V}[S_1] + q\,\mathsf{V}[S_2] + pq(\mathsf{E}[S_1] - \mathsf{E}[S_2])^2.$$

Thus, even if $\mathsf{V}[S_1] = \mathsf{V}[S_2] = 0$, still $\mathsf{V}[S] > 0$ if $\mathsf{E}[S_1] \neq \mathsf{E}[S_2]$. Mixing different job types increases variability; hence queueing times.

6.2 SERVER SETUPS

In some cases, machines must be set up before they can start producing jobs. Consider, for example, a machine that paints jobs in red and blue.[1] When the machine requires a color change, it may be necessary to clean up the machine, which takes time. Another example is an oven that needs a temperature change when different job types require different production temperatures. Service operations form another setting with setup times: When servers (personnel) have to move from one part of a building to another, the time spent moving cannot be spent serving customers.

In all such cases, the setups, or change-over times, consume a significant amount of time; in fact, setup times of an hour or more are not uncommon. To avoid overloading the server, it is necessary to produce in batches such that a server first processes a batch of jobs of one type[2] , then performs a setup to serve a batch of another type, and so on.

Here we use Sakasegawa's formula to model the effect of setup times on the average sojourn time of jobs. In the main text, we provide a list of elements required to compute $\mathsf{E}[J]$; in [6.2.6] we illustrate in detail how to carry out the computations.

ASSUME A SINGLE machine produces runs of red and blue jobs arriving at a rate λ_r and λ_b. The scv of the inter-arrival times of both job types is given by C_a^2. When the server changes from one color to the other, it needs a setup time. Once the server is set up for the correct color, we assume for simplicity that for all jobs the *net service times* $\sim S_0$.[3] Finally, the setup times $R_k \sim R$ with mean $\mathsf{E}[R]$ and variance $\mathsf{V}[R]$, and are assumed to be independent of the service times of the job.

The sojourn time of a job in the *entire* system is built as follows. First, we assemble the jobs into batches of the same color. For simplicity, we take B constant and the same for both colors.[4] Once B jobs of one color have arrived, the batch is complete, and the *batch enters* a queue; thus we consider a queue with batches, not single jobs. After some time, the batch reaches the head of the queue, and its service starts as soon as the machine becomes free. To serve a batch, the machine first performs a setup and then processes each job individually until the batch is finished. Once a job's service is completed, it leaves the server.

It remains to build a quantitative model of this queueing system.

WHEN A JOB of color i arrives, the expected time for its batch to be formed is given by[5]

$$\frac{B-1}{2\lambda_i}, \quad i \in \{r, b\}. \tag{6.2.1}$$

WHEN THE BATCH is complete, the batch joins the queue, so we next compute the average time batches spend in queue.[6] For this we can use Sakasegawa's formula; we only have to find expressions for each of its components.

The total arrival rate of jobs is $\lambda = \lambda_b + \lambda_r$. Thus, $\lambda_B = \lambda / B$ is the arrival rate of *batches*.

[1] *In realistic problems there can be tens of families. Often, the setup time depends on the sequence in which the families are produced, for example, a switch in color from white to black might take less cleaning time than from black to white. The problem then includes determining a production sequence that minimizes the sum of the setup times to produce the families, and finding suitable batch sizes that minimize the average waiting times.*

[2] *Producing one type of job is often called a 'run'.*

[3] *It is a simple exercise to generalize the notation to allow service times to depend on color.*

[4] *In general, B should depend on the arrival rate of the job type when the arrival rates vary considerably between the families.*

[5] *[6.2.7].*

[6] *Note that we shift interpretation: batches (not individual jobs) form the queue.*

The expected service time of a batch is

$$\mathsf{E}[S_B] = \mathsf{E}[R] + B\,\mathsf{E}[S_0]. \tag{6.2.2}$$

With the batch arrival rate and the expected batch service time, the load becomes $\rho = \lambda_B \mathsf{E}[S_B]$

It is essential that B is sufficiently large to ensure that $\rho < 1$. With (6.2.2) this leads to the condition that B must be larger than some minimal batch size,

$$B > B_m = \frac{\lambda \mathsf{E}[R]}{1 - \lambda \mathsf{E}[S_0]}.$$

Now that we have identified the arrival rate and the service times of the batches, it remains to find expressions for the scv of the batch interarrival times $C^2_{a,B}$ and the batch service times $C^2_{s,B}$. It is easy to derive that[7]

[7] [6.2.8] [6.2.9].

$$C^2_{a,B} = \frac{C^2_a}{B}, \qquad C^2_{s,B} = \frac{\mathsf{V}[S_B]}{(\mathsf{E}[S_B])^2}, \tag{6.2.3}$$

where

$$\mathsf{V}[S_B] = \mathsf{V}[R] + B\,\mathsf{V}[S_0].$$

Observe that now we have all components for Sakasegawa's formula.

It can be useful to convert the effects of the setup time into an *effective service time* of individual jobs. Since there are B jobs per batch, by dividing by B we see

$$\mathsf{E}[S] = \frac{\mathsf{E}[S_B]}{B} = \mathsf{E}[S_0] + \frac{\mathsf{E}[R]}{B}, \quad \mathsf{V}[S] = \frac{\mathsf{V}[S_B]}{B} = \mathsf{V}[S_0] + \frac{\mathsf{V}[R]}{B}. \tag{6.2.4}$$

Note in particular that the variance of the effective service time is larger than the variance of the net service time.

IT REMAINS TO find a rule to determine what happens to a job after it has been processed. If the job has to wait until all the jobs in the batch are served, the expected time it spends at the server is $\mathsf{E}[R] + B\,\mathsf{E}[S_0]$. However, if the job can leave right after being served, the expected time at the server is[8]

[8] [6.2.10].

$$\mathsf{E}[R] + \frac{B+1}{2}\mathsf{E}[S_0]; \tag{6.2.5}$$

the setup takes $\mathsf{E}[R]$, on average $\frac{B-1}{2}$ jobs are served before the job itself, and then the job requires $\mathsf{E}[S_0]$ of service. Our model is complete!

WE CAN OBTAIN a number of important insights from the above model. Using the code from [6.2.6] we plot the sojourn time $\mathsf{E}[J_r]$ of a red job for various values of B in the figure at the right.

First, as B increases, we see a sharp decrease of $\mathsf{E}[J_r]$. The reason for this is that the load ρ decreases as a function of B. Since $\mathsf{E}[W] \sim (1-\rho)^{-1}$, it is essential to avoid critically loading the server.

When B becomes quite large, we see that the sojourn time increases linearly. This follows immediately from (6.2.1) and (6.2.5), because the time to (dis)assemble batches is linear in B.

Finally, observe that the graph is not symmetric around the minimum. It is much worse to take B too small than too large. More generally,

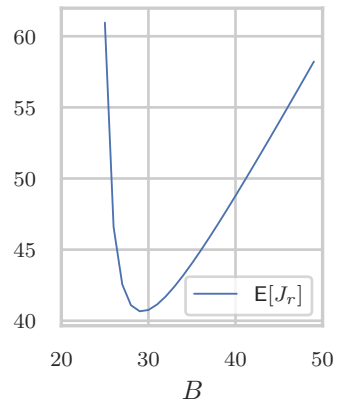


Fig. 6.2.1: The sojourn time of the red jobs as a function of the batch size B.

when designing systems, we can use such graphs to understand how sensitive the system is to variation and measurement errors.

TF 6.2.1. Jobs arrive at a rate λ and are assembled in batches of size B. The average time a job waits until the batch is complete is $\mathsf{E}[W] = \frac{B-1}{2\lambda}$.

TF 6.2.2. Claim: for a $G^B/G/1$ queue, the waiting time of a job can be reduced by making batches larger, as this means that there are fewer batches in the queue.

TF 6.2.3. Claim: In a system with setup times between batches, the load decreases when the batch size B increases.

TF 6.2.4. Claim: In a system with setup times between batches, batching reduces the variability of the arrival process; see (6.2.3).

TF 6.2.5. Claim: In a system with setup times between batches, increasing the batch size B always reduces the average sojourn time of a job.

Before dealing with technical derivations, let us first see how to apply the model.[9]

[9] *Before dealing with technical derivations, let us first see how to apply the model.*

Ex 6.2.6. Jobs arrive at $\lambda = 3$ per hour at a machine with $C_a^2 = 1$; service times are exponential with an average of 15 minutes. Assume $\lambda_r = 0.5$ per hour, and hence $\lambda_b = 2.5$ per hour. Between two batches, the machine requires a clean-up of 2 hours, with a standard deviation of 1 hour. First find the smallest batch size that can be allowed, then compute the average time a red job spends in the system in the case $B = 30$ jobs.

Ex 6.2.7. Show that the average time a job has to wait to fill the batch (to which this job belongs) is given by (6.2.1).

Ex 6.2.8. Explain that the scv of the batch inter-arrival times is given by (6.2.3).

Ex 6.2.9. Show that $C_{s,B}^2$ takes the form as in (6.2.3).

Ex 6.2.10. Show that, when jobs can leave right after being served, the time at the server is given by (6.2.5)

6.3 SERVER ADJUSTMENTS

In Section 6.2 we study the effect of setup times between batches of jobs. There are, however, other reasons to interrupt the server while it is serving jobs in queue. For example, when a knife on a cutting machine becomes blunt, an operator has to stop the machine to replace the knife. Another example is a medical doctor who has to make many unexpected phone calls between seeing two patients. All such small (maintenance) tasks can be carried out in between jobs, but it is often hard to predict when they are required. Therefore, the number of jobs[1] served between any two tasks[2] is no longer constant. But we know from Sakasegawa's formula that randomness in service times affects queueing times, and we

[1] *Compare this to a batch of jobs.*

[2] *And compare a task to a setup.*

also know from (6.2.4) that the batch size affects the service time. Hence, such unplanned tasks must make sojourn times longer.

We refer to this type of interruption as a server *adjustment*. It is important to realize that setups and adjustments are *non-preemptive outages*, i.e., they occur *between* jobs, not *during* job service times. Section 6.4 deals with the latter set of outages.

In this section, we develop a simple model to understand the impact of adjustments occurring randomly between jobs; we use the same notation as in Section 6.2 and follow the same line of reasoning. With this model, we can analyze a number of trade-offs, such as making fewer, but longer adjustments, or planning adjustments instead of just waiting until it becomes necessary at an unexpected moment.[3]

[3] *[6.3.7].*

LET US WRITE F_i for the Bernoulli rv that, when equal to 1, indicates that job i requires an adjustment, and, when 0, the adjustment is not necessary. We assume that $F_i \sim F$ such that $\mathsf{P}\{F=1\} = p = 1 - \mathsf{P}\{F=0\}$. Thus, with constant probability p an adjustment occurs between any two jobs. As a consequence, the number of jobs served between two consecutive adjustments is a geometric rv B with $\mathsf{E}[B] = 1/p$. We further assume that the adjustment times $R_i \sim R$ with mean $\mathsf{E}[R]$ and variance $\mathsf{V}[R]$. Finally, we assume that the net service times $\{S_{0,i}\}$, $\{R_i\}$ and $\{F_i\}$ are independent.

Thus, whereas in Section 6.2 the batch sizes were constant between two setups, they are here geometrically distributed.

TO COMPUTE $\mathsf{E}[W]$ with Sakasegawa's formula, we only have to find out how the adjustments affect the mean $\mathsf{E}[S]$ and scv C_s^2 of the effective service time. As adjustments do not influence the job arrival process, λ and C_a^2 remain the same.

In the exercises below we show that the average effective service time is

$$\mathsf{E}[S] = \mathsf{E}[S_0] + p\,\mathsf{E}[R] = \mathsf{E}[S_0] + \frac{\mathsf{E}[R]}{\mathsf{E}[B]}, \tag{6.3.1}$$

and its variance is

$$\mathsf{V}[S] = \mathsf{V}[S_0] + p\,\mathsf{V}[R] + p(1-p)(\mathsf{E}[R])^2. \tag{6.3.2}$$

By dividing $\mathsf{V}[S]$ by $(\mathsf{E}[S])^2$ we obtain C_S^2.

Assuming there is one server, we can now fill in Sakasegawa's formula!

BEFORE CONSIDERING SOME examples, let us compare the results of this model with those of Section 6.2. The expected effective service time is the same: an amount $\mathsf{E}[R]/\mathsf{E}[B]$ is added to the net service time $\mathsf{E}[S_0]$. However, the impact on the variance is different. By comparing (6.2.4) to (6.3.2) we see that the latter has an extra (positive) term. This supports our intuition: Unexpected interruptions have a larger effect on variability than expected (planned) interruptions.

The first few exercises demonstrate how to apply the above; the rest are concerned with deriving (6.3.1) and (6.3.2).

TF 6.3.1. The number of jobs served between two consecutive adjustments follows a Poisson distribution.

TF 6.3.2. A repair shop is considering buying a new CNC machine. There are two options available, the Yamazaki and the Haas. The Haas is less reliable and breaks down every week on average. However, it can be easily fixed by running through the control checklist, which takes 60 minutes. The Yamazaki breaks down about once a month, but it is much more complicated to fix and takes 3.5 hours. The time till a breakdown is memoryless and the machines have the same service rate. Claim: The Yamazaki is better from a queueing perspective.

TF 6.3.3. Claim: In a system with random adjustments between jobs, the effective service time has the same variance as in a system with planned setups and the same average batch size.

TF 6.3.4. Claim: In a system with random adjustments between jobs, making fewer but longer adjustments (same total adjustment time on average) reduces the waiting time.

TF 6.3.5. Claim: In a system with random adjustments between jobs, if the probability of an adjustment is constant $p \in (0,1)$ for each job, the number of jobs served between two consecutive adjustments follows a geometric distribution.

Ex 6.3.6. Jobs arrive as a Poisson process with a rate $\lambda = 9$ per working day. The machine works two 8 hour shifts a day. Work not processed on a day is carried over to the next day. The job service times are 1.5 hours, on average, with a standard deviation of 0.5 hours. Outages occur on average between 30 jobs. The average duration of an outage is 5 hours and has a standard deviation of 2 hours. Compute $\mathsf{E}[J]$.

Ex 6.3.7. In the setting of [6.3.6] we can perhaps choose to make an adjustment every 20 jobs. For simplicity, assume that adjustments never occur at random. Also, assume that an adjustment takes less time, for instance 4.5 hours, and is constant. What is $\mathsf{E}[J]$ now?

The easiest way to derive the expressions for $\mathsf{E}[S]$ and $\mathsf{V}[S]$ is to use Adam's and Eve's rule.[4]

Ex 6.3.8. Derive the expressions for $\mathsf{E}[S]$ and $\mathsf{V}[S]$.

[4] $\mathsf{V}[Y] = \mathsf{E}[\mathsf{V}[Y|X]] + \mathsf{V}[\mathsf{E}[Y|X]]$.

6.4 SERVER FAILURES

In Section 6.2 and Section 6.3 we assumed that servers are never interrupted while serving a job. However, this assumption is not always satisfied; for example, a machine may fail in the middle of processing a job. In this section, we develop a model[1] to calculate the influence on the mean waiting time of such *preemptive outages*, again based on Sakasegawa's formula for the $G/G/1$ queue.

As in Section 6.3, we only have to derive expressions for the expectation and variance of the effective service time S. The other components of Sakasegawa's formula are not affected.

[1] *There is not much point in combining the models of Section 6.2–Section 6.4 into one large model. For example, when a machine can be set up while a part is being repaired, there may not be time wasted on setup. Everything depends on specific strategies to combine outages.*

We assume that a failure can occur at any time during the service time of a job. The repair times $R_i \sim R$ have common mean $\mathsf{E}[R]$ and finite second moment $\mathsf{E}[R^2]$. Supposing that N interruptions occur during the net job service time S_0, the effective service time becomes[2]

$$S = S_0 + S_N = S_0 + \sum_{i=1}^{N} R_i. \tag{6.4.1}$$

[2] *In an insurance context, if we interpret $\{R_i\}$ as a set of claims, then $\sum_{i=1}^{N} R_i$ is the total claim size of N claims. Likewise, in an inventory system, this sum is the total demand of N customers.*

In general, N is a random number, so we need to adjust for this in the computation of the mean and variance of S.

IT IS COMMON to assume that the time between two interruptions is memoryless with mean $1/\lambda_f$, where λ_f is the *failure rate*. Defining the MTTF[3] as $m_f = 1/\lambda_f$, then the *availability* is given by

[3] *MTTF := mean time to fail.*

$$A := \frac{m_f}{m_f + \mathsf{E}[R]},$$

from which follows easily that [4]

$$A = (1 + \lambda_f \mathsf{E}[R])^{-1}. \tag{6.4.2}$$

[4] *[6.4.7].*

Next, because by assumption the time between two failures is $\sim \mathrm{Exp}(\lambda_f)$ so that $N \sim \mathrm{Pois}(\lambda_f S_0)$ if S_0 is given,

$$\mathsf{E}[N] = \mathsf{E}[\mathsf{E}[N|S_0]] = \mathsf{E}[\lambda_f S_0] = \lambda_f \mathsf{E}[S_0]. \tag{6.4.3}$$

Therefore,

$$\begin{aligned}\mathsf{E}[S] &= \mathsf{E}[S_0] + \mathsf{E}[N]\,\mathsf{E}[R] = \mathsf{E}[S_0] + \lambda_f \mathsf{E}[S_0]\,\mathsf{E}[R] \\ &= \mathsf{E}[S_0]\,(1 + \lambda_f \mathsf{E}[R]) = \mathsf{E}[S_0]\,/\,A.\end{aligned}$$

With this expression for the expected effective service time, the load becomes

$$\rho = \lambda \mathsf{E}[S] = \lambda \frac{\mathsf{E}[S_0]}{A}.$$

Clearly, $A \in (0, 1]$, so the load increases due to failures.

WITH SOME MORE work, we obtain[5] from (6.4.1),

$$\mathsf{V}[S] = \frac{\mathsf{V}[S_0]}{A^2} + \lambda_f \mathsf{E}[R^2]\,\mathsf{E}[S_0]. \tag{6.4.4}$$

[5] *[6.4.8].*

If the repair times are exponentially distributed, it is possible to find a neat expression for the scv of the effective service time:[6]

$$C_s^2 = C_0^2 + 2A(1 - A)\frac{\mathsf{E}[R]}{\mathsf{E}[S_0]}, \tag{6.4.5}$$

where C_0^2 is the scv of S_0.

[6] *[6.4.10].*

TF 6.4.1. The normal service time of a job, without interruptions, is given by S_0. The durations of interruptions are given by $R_i \sim R$. If N interruptions occur, the effective service time will then be

$$S = S_0 + \sum_{i=1}^{N} R_i.$$

Then all steps in the computation below are correct:

$$\mathsf{E}\left[\sum_{i=1}^{N} R_i\right] = \mathsf{E}\left[\sum_{n=0}^{\infty} \mathbb{1}\{N = n\}\right] \mathsf{E}\left[\sum_{i=1}^{n} R_i\right] = \mathsf{E}[N]\,\mathsf{E}[R]$$

TF 6.4.2. Assume that server failures are memoryless and occur at a rate of 6 per hour. Moreover, assume that the expected time to fix a failure is 18 minutes. An investment can be made which will result in a failure rate of 18 per hour and an expected repair time of 6 minutes per failure. Claim: This is a good investment.

TF 6.4.3. Claim: Server failures reduce the availability A but do not affect the effective service time $\mathsf{E}[S]$.

TF 6.4.4. Claim: If the net service time S_0 is deterministic, then the effective service time $S = S_0 + \sum_{i=1}^{N} R_i$ is also deterministic.

TF 6.4.5. Claim: In the model of this section, the scv of the effective service time satisfies $C_s^2 \geq C_0^2$, where C_0^2 is the scv of the net service time.

Ex 6.4.6. Suppose that we have a machine with memoryless failure behavior, with a mean time-to-fail of $m_f = 3$ hours. Regular service times are deterministic with an average of $S_0 = 10$ minutes, jobs arrive as a Poisson process with a rate of $\lambda = 4$ per hour. Repair times are exponential with a mean duration of $\mathsf{E}[R] = 30$ minutes. What is the average sojourn time?

Ex 6.4.7. Derive (6.4.2).

Ex 6.4.8. Derive (6.4.4) using Adam and Eve's rule.

Ex 6.4.9. Show that

$$C_s^2 = \frac{\mathsf{V}[S]}{(\mathsf{E}[S])^2} = C_0^2 + \frac{\lambda_f \mathsf{E}\left[R^2\right] A^2}{\mathsf{E}[S_0]},$$

Ex 6.4.10. Assuming that the repair times R are exponentially distributed, show (6.4.5).

Ex 6.4.11. How is the renewal reward theorem used in (6.4.2) to find an expression for the availability A?

7

QUEUEING NETWORKS

In this chapter we first study how variability propagates through tandem networks of $G/G/c$ queues in Section 7.1. We then explore deterministic queueing networks in Section 7.2, which, despite their simplicity, already pose interesting challenges. Next, we analyze open networks of $M/M/c$ stations in Section 7.3. The analysis of such networks involves an equation of the type $\lambda = \gamma + \lambda P$, where λ and γ are row vectors and P is a substochastic matrix.[7] In Section 7.4, we concentrate on the solution of this equation. Finally, Section 7.5 analyzes closed queueing networks using mean value analysis.

We point out that the techniques developed in this chapter extend (way) beyond just queueing theory; they are worth memorizing. Thus, while this chapter closes our journey through the study of queueing systems, it is a first step toward a much longer journey into the diverse applications of probability theory. For further reading, we recommend the following books.

- You can find very nice discussions of networks of $M/M/\infty$, chemical reactions, population dynamics, and Petri nets in Baez and Biamonte [2019], which is freely available on arXiv.
- Simple queueing networks (networks that satisfy the so-called local balance) can be modeled as electrical networks. For this, see Doyle and Laurie Snell [1984], which you can download for free from the homepage of Doyle.
- In more general terms, queueing systems or networks are examples of Markov processes. A particularly nice book on these topics is Norris [1997]. The material of this chapter can be couched in the theory of martingales and optimal stopping, which find many applications in areas such as inventory theory, decision theory, and quantitative finance.

[7] *A substochastic matrix is a nonnegative matrix whose row sums are less than or equal to* 1. *For the open networks considered here, we also require all eigenvalues to have modulus less than* 1.

7.1 $G/G/c$ QUEUES IN TANDEM

OFTEN, QUEUES APPEAR in sequence. In such cases, customers are interested in the mean sojourn (and waiting times) of their jobs in the entire chain. For this reason, we need a formula for dealing with networks.

Consider two $G/G/1$ stations in tandem.[1] Suppose we have the financial means to reduce the variability of the service times at one of the stations, but not at both. Clearly, we would like to improve the one that has the most impact on the total sojourn time in the line.

For the waiting time of the first machine, we can use Sakasegawa's formula, but to apply this to the second machine, we need $C_{a,2}^2$, i.e., the scv of the inter-arrival times at the second station. Now, noting that the output of the first machine forms the input of the second machine, it is

[1] *'In tandem' means 'in line', i.e., one station after the other.*

clear that $C_{a,2}^2 = C_{d,1}^2$, where $C_{d,1}^2$ is the scv of the *inter-departure* times of the *first* station.

Consider the inter-departure times of a $G/G/1$ queue. Suppose that the utilization ρ is very high. Then the server will seldom be idle, so that most of the inter-departure times are equal to the service times of the first station. However, if the utilization is low, the server will be idle most of the time and the inter-departure times must be approximately equal to the inter-arrival times of the first station. By interpolating between these two extremes, we obtain the following approximation for the scv of the inter-departure times of a $G/G/c$ queue:

$$C_{d,1}^2 \approx 1 + (1-\rho^2)(C_{a,1}^2 - 1) + \frac{\rho^2}{\sqrt{c}}(C_{s,1}^2 - 1).$$

It is simple to see that for the $G/G/1$ queue this reduces to

$$C_{d,1}^2 \approx (1-\rho^2)C_{a,1}^2 + \rho^2 C_{s,1}^2. \tag{7.1.1}$$

Next, we can use this formula for each station in the tandem network.

Combining Sakasegawa's formula with this expression provides us with a very useful insight for a line of queues. If we reduce $C_{s,1}^2$, i.e., the scv of service times at the first station, $\mathsf{E}[W_1]$ and $C_{d,1}^2$ become smaller. Since $C_{a,2}^2 = C_{d,1}^2$, $\mathsf{E}[W_2]$ also becomes lower, but also $C_{d,2}^2$, etc. In other words, the entire chain benefits from an improvement in service variability at the first station.[2]

[2] *This makes an early station a natural candidate for improvement, but the best choice also depends on the stations' loads and service variability.*

TF 7.1.1. Suppose that in a tandem network of $G/G/c$ queues we can reduce C_s^2 of just one station by a factor 2. To improve the average waiting time throughout the chain, it is best to reduce $C_{s,1}^2$.

TF 7.1.2. A production system consists of 2 stations in tandem. The first station has one machine, the second has two identical machines. Machines never fail and service times are deterministic. Jobs arrive at a rate of 1 per hour. The machine at the first station has a service time of 45 minutes per job, and a machine at the second station has a service time of 80 minutes. We claim that the second station is the bottleneck.

TF 7.1.3. Consider a network with n stations in tandem. At station i, the service times S_i for all machines at that station are the same and constant; station i contains N_i machines. The number of jobs required to keep all machines busy is $N = \sum_{i=1}^n N_i$, and the raw service time is $T_0 = \sum_{i=1}^n S_i$. Thus, if the number w of jobs allowed in the system is greater than N, the number of jobs waiting somewhere in queue is $w - N$.

TF 7.1.4. We have two $M/M/1$ stations in tandem. Claim: The average queueing time for the network is given by

$$\mathsf{E}[W] = \frac{\rho_1}{1-\rho_1} + \frac{\rho_2}{1-\rho_2}.$$

Ex 7.1.5. In a manufacturing setting, the Poisson process is not always a suitable model for the arrival process of jobs at a production station. Can you provide an example to show why this is the case?



Ex 7.1.6. Consider two $G/G/1$ stations in tandem. Suppose $\lambda = 2$ per hour, $C_{a,1}^2 = 2$, $C_{s,1}^2 = C_{s,2}^2 = 0.5$, and $\mathsf{E}[S_1] = 20$ minutes, and $\mathsf{E}[S_2] = 25$ minutes. Compute $\mathsf{E}[J] = \mathsf{E}[J_1] + \mathsf{E}[J_2]$.

7.2 DETERMINISTIC QUEUEING NETWORKS

Before we study queueing networks in stochastic settings, we should familiarize ourselves with the behavior of networks of queueing systems in the simplest setting possible. Arguably, this is a network in which all servers have constant service times, which is the topic of this section.

A QUEUEING NETWORK can be represented as a *graph*. *Stations* form the *nodes* in the graph, and *edges* between nodes represent the routing, i.e., the possibility that jobs can move from one station to another. A simple example is a network of n stations in tandem. Jobs arrive at station 1, and after service they move to station 2. Then they move to station 3, and so on, until they leave the network after being served at station n. A station can contain one or more servers. We assume that all servers have constant job processing times. However, service times need not be equal for all servers.

We need some more definitions. The *raw processing time* T_0 is the minimal amount of time needed to move from the start of the system to the end. More precisely, it is the sum of the long-run average processing times of each workstation in the routing. The *utilization* of station i is λ_i/μ_i, i.e., the rate λ_i at which work arrives divided by the processing rate of the server(s) at station i. The *bottleneck* station is the station with the highest utilization. We write r_b for the processing rate of the bottleneck station. The *critical work in progress (WIP)* W_0 is defined as $W_0 = r_b T_0$. This is the minimal amount of work required in the network to keep the bottleneck station fully loaded. To see how this relates to Little's law, observe that if jobs leave the network at rate r_b, they have to arrive at the network at rate r_b—we also assume that inter-arrival times are constant, so that no explosions occur. Note that this is the largest arrival rate that the network can handle without queues growing to infinity. But then, with Little's law, the amount of work in the system is equal to the arrival rate times the time in the system.

In the next set of exercises we illustrate the above concepts by means of a network with n single-server stations in tandem, where all stations have the same service time S.

Ex 7.2.1. Suppose we would release just one job at a time in the above tandem system, that is, only when a job has visited all stations, the next job is released. What is the throughput of the network?

Ex 7.2.2. Argue that if we are prepared to release w jobs, the throughput of the network becomes

$$\mu(w) = \min\{w, n\}\,\frac{1}{T_0}.$$

Conclude that we need a minimal amount w in the network to guarantee that the network's throughput $\mu(w)$ exceeds the arrival rate λ, that is, there is a minimal w such that $\mu(w) \geq \lambda$.

Ex 7.2.3. Show that only when $w \geq n$,

$$\mu(w) = \frac{1}{S},$$

in which case the maximal capacity of the network is achieved.

Thus, for given w the largest arrival rate the network can handle is $\lambda = \mu(w)$, and the maximal rate at which it can process jobs is $r_b = 1/S$.

Ex 7.2.4. Show that when $\lambda = \mu(w)$, the waiting time in the network is

$$\mathsf{E}[W] = \begin{cases} T_0 = nS, & \text{if } w \leq n, \\ \frac{w}{n} T_0 = wS, & \text{if } w \geq n. \end{cases}$$

Ex 7.2.5. What do you conclude from the above exercises?

In conclusion, to balance waiting times and throughput, we need to tune the amount of work in the system. The critical WIP W_0 is the minimal WIP we need to keep all bottleneck machines occupied without adding extra waiting time. Jobs can never pass through the network in less time than T_0, and the maximal output rate of the network is $r_b = 1/S$. Thus,

$$T_0 = nS, \qquad r_b = 1/S, \qquad W_0 = r_b T_0 = n. \tag{7.2.1}$$

THE PROBLEMS BELOW are generalizations to networks with multi-server stations, servers with different processing rates, and multiple routings. They are meant to help you become acquainted with network behavior, and also with Little's law, which we use time and again here.[1]

[1] *Warning, these problems are quite challenging, despite their apparent simplicity; I had much less trouble posing the problems than solving them. In fact, to ensure that I got the answer right, I often tried to find at least two different ways to solve the same problem.*

Ex 7.2.6. A production system consists of 2 separate stations. Jobs never go from one machine to another. The processing times are $(t_1, t_2) = (1, 2)$ hours. The arrival rates are $(\lambda_1, \lambda_2) = (1/3, 1/3)$ per hour. What is the fastest machine? Which machine is the bottleneck?

Ex 7.2.7. In the network of the previous exercise, what are T_0 and W_0?

Ex 7.2.8. A production network consists of 3 single-machine stations in tandem, i.e., in series. The processing times are $(2, 3, 2)$ hours respectively, i.e., constant and such that $t_1 = 2$ hours, $t_2 = 3$ hours and $t_3 = 2$ hours. What are the critical WIP W_0, bottleneck capacity r_b and raw processing time T_0?



Ex 7.2.9. A production network consists of 3 single-machine stations in tandem with processing times $(2, 3, 2)$ hours. Suppose we modify this network by adding one extra machine to station 2, with processing time $t = 3$ hours. What are W_0, r_b and T_0?

Ex 7.2.10. A production network consists of 3 single-machine stations in tandem with processing times $(2,3,2)$ hours. Suppose we modify this network by adding one extra machine to station 2 with processing time $t = 4$ hours. What are W_0, r_b and T_0?

Ex 7.2.11. Why does the critical WIP change in the different cases above?

Ex 7.2.12. A production network consists of 3 single-machine stations in tandem with processing times $(2,3,2)$ hours. Suppose we modify this network by adding one extra machine to station 2 with processing time $t = 1$ hour. What are W_0, r_b and T_0?

Ex 7.2.13. In the previous question, it might seem that one machine at the second station is superfluous. Which one, and why? Would you remove this superfluous machine in a real production network?

Ex 7.2.14. A production network consists of 3 single-machine stations in tandem with processing times $(2,3,2)$ hours. Suppose we modify this network by adding one extra machine to station 2 with processing time $t = 7$ hours. What are W_0, r_b and T_0?

Ex 7.2.15. A production network consists of 3 single-machine stations in tandem with processing times $(2,3,2)$ hours. Suppose we modify this network such that $2/3$ of the jobs *leave after the first* station, i.e., between station 1 and 2. What are W_0, r_b and T_0?

Ex 7.2.16. A production network consists of 3 single-machine stations in tandem with processing times $(2,3,2)$ hours. Suppose we modify this network such that $4/5$ of the jobs *bypass the second* station, i.e., $1/5$ of the jobs have the routing $[1,2,3]$, but $4/5$ have routing $[1,3]$. What are W_0, r_b and T_0?

Ex 7.2.17. Building upon the previous question, which stations would you include anyway in a CONWIP loop, and why?

Production situations can be pretty complicated. Jobs can enter the network at different places, not just at station 1 as we considered here. Jobs may also need rework at one or more stations. One consequence of these aspects is that the slowest machines or stations need not be the bottlenecks. If you like, you can try to find or develop a general algorithm to compute T_0, W_0 and r_b for any network of reasonable size. One way to approach this problem might be to place (imaginary) huge amounts of work in front of each station and just see how the network drains, another is to try to find an analogy with an electrical network. All in all, it's an elegant problem.

[1] *I.e., new jobs that have not visited any other station before.*

7.3 OPEN SINGLE-CLASS PRODUCT-FORM NETWORKS

Up to now our analysis focused on single-station queueing systems. However, jobs, or patients, may need several 'processing' steps at different stations. In such cases, jobs move from one queueing station to another. In this section we analyze such queueing networks in simple settings. We start with two stations in tandem, and then extend to general networks. Throughout we assume that external jobs[1] arrive as Poisson processes, and that service times at all stations are exponentially distributed. Under this condition we are able to obtain closed-form expressions for the stationary distribution of jobs at each station.[2]

[2] *Recall that Section 7.1 only considers the expected sojourn times in tandem networks of $G/G/c$ queues, not the distribution of the number of jobs at each station in a network of $M/M/1$ queues.*

WE START BY stating the remarkable and important result that the *inter-departure times* of jobs of an $M/M/1$ queue are exponentially distributed with rate λ, just as the inter-arrival times.[3] This property makes the analysis of a *tandem network* of stations, i.e., stations placed in sequence, particularly easy.

[3] *[7.3.2].*

When the first station is an $M/M/1$ queue, jobs arrive as a Poisson process, but its output process is also a Poisson process. Therefore the second station, i.e., the station immediately downstream of station 1, receives jobs as a Poisson process. If job service times at the second station are exponentially distributed, then this station is also an $M/M/1$ station. But then its departure process is a Poisson process in turn, and the third station receives jobs as a Poisson process.

With this insight, it follows from (5.1.2) that the expected total sojourn time in a tandem network of M stations is equal to

$$\mathsf{E}[J] = \sum_{i=1}^{M} \mathsf{E}[J_i] = \sum_{i=1}^{M} \frac{\mathsf{E}[S_i]}{1-\rho_i},$$

where $\mathsf{E}[J_i]$ is the sojourn time at station i with $\rho_i = \lambda \mathsf{E}[S_i]$.

IT IS NOT difficult to extend the above result to general networks of $M/M/1$ queues. For this, we first need to model such networks more formally. In particular, we assume that the probability that a job moves to station j after completing its service at station i is independent of anything else, and is given by the number $P_{ij} \in [0,1]$.[4] We assemble all these probabilities in a *routing matrix* P such that P_{ij} is the element of P on the ith row and jth column. Define $P_{i0} = 1 - \sum_{j>0} P_{ij}$ as the probability that a job leaves the network from station i. In Section 7.4 we discuss some properties of P such that the network is stable.

[4] *This is called Markov routing.*

Consider station i, say, and assume that jobs arrive as a Poisson process with rate λ_i. Since service times are exponentially distributed, the departure process is also Poisson with rate λ_i. Then, after departure, jobs are sent with probability P_{ij} to station j. It follows from [1.3.13] that jobs sent to station j from station i form a thinned Poisson process with rate $\lambda_i P_{ij}$.

Now take the perspective of station j. This station receives a merged 'stream' of thinned Poisson processes from all other stations. But by the results of Section 1.3, this merged process is also a Poisson process with rate $\sum_{i=1}^{M} \lambda_i P_{ij}$. Assuming that external jobs arrive at station j as a

Poisson process with rate γ_j, then by merging this process with the other Poisson stream, we obtain a Poisson process with rate $\gamma_j + \sum_{i=1}^{M} \lambda_i P_{ij}$.

By assumption, service times at station j are exponential, hence, the departure process of this station is Poisson. But then the thinned process resulting from routing its departing jobs to yet other stations is again Poisson, and so on.

It follows that this network behaves as a set of $M/M/1$ queues. Below we give a formal proof of this fact for two stations.

Note that we allow for external jobs arriving at any station. Therefore, this is an *open network*. This differs from so-called *closed networks*; in such networks jobs do not enter or leave the network.

IT IS EVIDENT that, when the network is stable, all jobs that enter a station must eventually leave that station. This insight leads us to the *traffic equations*, which state that for each station i the departure rate must equal the arrival rate, i.e.,

$$\lambda_i = \gamma_i + \sum_{j=1}^{M} \lambda_j P_{ji}, \quad i = 1, \ldots, M. \tag{7.3.1}$$

For the *network as a whole*, the total external arrival rate is given by $|\gamma| = \sum_{i=1}^{M} \gamma_i$. Hence, this is also the departure rate of the network.[5] In Section 7.4 we present a method to compute λ from γ and P.

[5] *Not for an individual station i, because $\delta_i = \lambda_i$.*

LET US FOR the moment assume that we can solve the traffic equations, in other words, for a given $\gamma = (\gamma_1, \ldots, \gamma_M)$ and routing matrix P we can find a set of numbers $\lambda = (\lambda_1, \ldots, \lambda_M)$ that satisfies (7.3.1). Then, we can define $\rho_i = \lambda_i \mathsf{E}[S_i]$ as the utilization of (the server of) station i. Clearly, we assume that $\rho_i < 1$ for all stations i.

Evidently, the average total number of jobs in this network of $M/M/1$ stations is equal to the sum of the average number of jobs at each station, hence, $\mathsf{E}[L] = \sum_{i=1}^{M} \mathsf{E}[L_i]$. With this we can also find an expression for the expected total sojourn time in the network $\mathsf{E}[J]$. Applying Little's law first to the network as a whole and then to each station individually, we see that

$$|\gamma| \mathsf{E}[J] = \mathsf{E}[L] = \sum_{i=1}^{M} \mathsf{E}[L_i] = \sum_{i=1}^{M} \lambda_i \mathsf{E}[J_i] = \sum_{i=1}^{M} \lambda_i \frac{\mathsf{E}[S_i]}{1-\rho_i}.$$

Dividing by $|\gamma|$ gives $\mathsf{E}[J]$ in terms of the *visit ratios* $\lambda_i/|\gamma|$,

$$\mathsf{E}[J] = \sum_{i=1}^{M} \frac{\lambda_i}{|\gamma|} \frac{\mathsf{E}[S_i]}{1-\rho_i}.$$

This is intuitive: the visit ratio $\lambda_i/|\gamma|$ tells how often station i is visited relative to the total number of arrivals. Thus, $\mathsf{E}[J_i]\,\lambda_i/|\gamma|$ is the amount of time an 'average' job spends at station i before leaving the network.

ABOVE WE DERIVED expressions for the average waiting time in a network of $M/M/1$ queues, but it is possible to obtain a much stronger result. In fact, the stationary probability $p(n)$ that the system contains $n = (n_1, n_2, \ldots, n_M)$ jobs at stations $1, \ldots, M$ can be written as the *product* of the probabilities of the individual stations, specifically,

$$p(n) = \mathsf{P}\{L_1 = n_1, \ldots, L_M = n_M\} = \prod_{i=1}^{M} p(n_i),$$

where $p(n_i) = (1-\rho_i)\rho_i^{n_i}$ is the stationary probability that station i contains n_i jobs, compare (5.1.1). In other words, the stationary probability $p(n_i)$ is *independent* of the state of the other stations. Hence, in stationarity, we can concentrate on each station individually. As a consequence, we can easily compute the excess probability $\mathsf{P}\{L_i > n_i\}$ for each station i.

As a matter of fact, any stable open network of $M/M/c$ stations[6] admits a *product-form solution*, a result known as *Jackson's theorem*.

[6] *Note, $M/M/c$ queues, not just $M/M/1$ queues.*

LET US PROVE this result for the case of two $M/M/1$ stations in tandem. Let $p(i,j) = \mathsf{P}\{L_1 = i, L_2 = j\}$ be the probability that station 1 contains i jobs and station 2 contains j jobs. Specifically, we have to show that

$$p(i,j) = (1-\rho_1)(1-\rho_2)\rho_1^i\rho_2^j \tag{7.3.2}$$

satisfies the balance equations for all $i, j \geq 0$, see Section 4.6.

Here we focus on the case $i, j \geq 1$, see Fig. 7.3.1. As the normalization factor appears at both sides of the balance equations, we write $p(i,j) = \rho_1^i\rho_2^j$ for ease. Clearly, the balance equations hold for $i, j \geq 1$.

$$\begin{aligned}
\text{rate out} &= (\lambda+\mu_1+\mu_2)p(i,j)\\
&= \lambda\rho_1^i\rho_2^j + \mu_1\rho_1^i\rho_2^j + \mu_2\rho_1^i\rho_2^j\\
&= \mu_2\rho_1^i\rho_2^{j+1} + \lambda\rho_1^{i-1}\rho_2^j + \mu_1\rho_1^{i+1}\rho_2^{j-1}\\
&= \mu_2 p(i,j+1) + \lambda p(i-1,j) + \mu_1 p(i+1,j-1)\\
&= \text{rate in.}
\end{aligned}$$

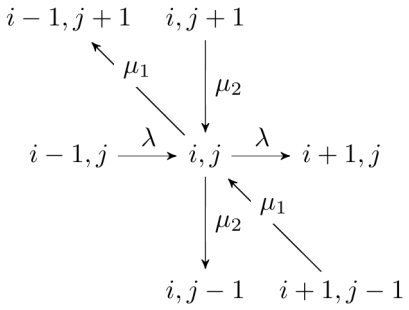


Fig. 7.3.1: Transitions in two stations in tandem.

It remains to check the boundary[7] to complete the claim.

[7] *[7.3.4]–[7.3.6].*

Ex 7.3.1. For the $M/M/1$ queue, show that if $L(D_{k-1}) = 0$ and $S_k \sim \text{Exp}(\mu)$, the density of the rv $D_k - D_{k-1}$ is

$$f(t) = \frac{\lambda\mu}{\mu-\lambda}(e^{-\lambda t} - e^{-\mu t}).$$

Ex 7.3.2. For the $M/M/1$ queue, show that the inter-departure times $D_k - D_{k-1} \sim \text{Exp}(\lambda)$.

Ex 7.3.3. We have a two-station single-server open network. Jobs enter the network at the first station with rate γ. A fraction α returns from station 1 to itself; the rest moves to station 2. At station 2 a fraction β_2 returns to station 2 again, a fraction β_1 goes to station 1.

Ex 7.3.4. Check that $p(0,0)$ satisfies (7.3.2).

Ex 7.3.5. Check that $p(i,0)$ satisfies (7.3.2) for $i > 0$.

Ex 7.3.6. Check that $p(0,j)$ satisfies (7.3.2) for $j > 0$.

7.4 ON $\lambda = \gamma + \lambda P$

In this final section, we concentrate on finding a general condition to ensure that $\lambda = \gamma + \lambda P$ has a solution.

TO START, WE define iteratively $\lambda^0 = 0$, $\lambda^n = \gamma + \lambda^{n-1} P$, for $n \geq 1$. Then, by substitution, we find

$$\lambda^n = \gamma + \lambda^{n-1} P = \gamma + (\gamma + \lambda^{n-2} P)P = \cdots = \gamma \sum_{m=0}^{n-1} P^m,$$

where P^0 is the identity matrix. Since P is a non-negative matrix, P^n is also non-negative for any n, hence, $\sum_{m=0}^{n} P^m$ is a monotonic increasing sequence (of matrices).[1] Thus, for $\lambda = \gamma + \lambda P$ to have a (unique) solution, this sum must remain finite as $n \to \infty$.

[1] *Formally, non-decreasing sequence.*

Now, observe the similarity with $\sum_{i=0}^{n} x^i$ with $|x| < 1$. We conclude that this converges to $1/(1-x)$ for $n \to \infty$. Similarly, if there is an $\epsilon > 0$ such that for all i, j, $0 \leq P_{ij}^m \leq (1-\epsilon)^m$, then $0 \leq \sum_{m=0}^{n} P_{ij}^m \leq \sum_{m=0}^{n} (1-\epsilon)^m \leq 1/\epsilon$.

It remains to find a condition to ensure that such ϵ exists. For this, we discuss two ways.

FROM THE PREVIOUS section, we know that $P_{i0} = 1 - \sum_{j=1}^{M} P_{ij}$ is the probability that a job leaves the network from station i. Next, consider a station k with $P_{ki} > 0$. Then the probability that a job starts at k and leaves the network after a visit to i is $P_{ki} P_{i0} > 0$. Consequently, $P_{k0}^2 = \sum_{j=0}^{M} P_{kj} P_{j0} \geq P_{ki} P_{i0} > 0$.

More generally, we say that an $M \times M$ matrix P is *transient* when it is possible to leave the network from any station in at most M steps. In other words, for any station j there is a sequence of intermediate stations $j_1, j_2, \ldots, j_{M-1}$ such that $P_{j0}^M \geq P_{jj_1} P_{j_1 j_2} \cdots P_{j_{M-1}0} > 0$. Using this, it is not hard to prove that $P^n \to 0$ exponentially fast when P is transient.[2]

[2] *[7.4.2].*

FOR THE SECOND method, let us make the simplifying assumption that P is a diagonalizable matrix with M different eigenvalues.[3] In this case, there exists an invertible matrix V with the (left) eigenvectors of P as its rows and a diagonal matrix Λ with the eigenvalues of P on its diagonal such that $VP = \Lambda V$. Hence, pre-multiplying with V^{-1}, $P = V^{-1} \Lambda V$. But then $P^2 = V^{-1} \Lambda V \cdot V^{-1} \Lambda V = V^{-1} \Lambda^2 V$, and in general $P^n = V^{-1} \Lambda^n V$. Clearly, if each eigenvalue has modulus less than 1, then $\Lambda^n \to 0$ exponentially fast, hence $P^n \to 0$ exponentially fast.

[3] *The argument below applies just as well to matrices reduced to Jordan normal form, but adds only notational clutter.*

So, let us prove that all eigenvalues of a finite, transient routing matrix P have modulus less than 1. For this we use *Gershgorin's disk theorem*. Define the Gershgorin disk of the ith row of the matrix P as the disk in the complex plane:

$$B_i = \left\{ z \in \mathbb{C}; |z - p_{ii}| \leq \sum_{j \neq i} |p_{ij}| \right\}.$$

In words, this is the set of complex numbers that lie within a distance $\sum_{j \neq i} |p_{i,j}|$ of the point $p_{i,i}$. Next, assume for notational simplicity that for each row i of P we have that $\sum_j p_{ij} < 1$.[4] Then this implies for all i that

[4] *Otherwise apply the argument to P^M.*

$$1 > \sum_{j=1}^{M} p_{ij} = p_{ii} + \sum_{j \neq i} p_{ij}.$$

Since all elements of P are non-negative, so that $|p_{ij}| = p_{ij}$, it follows that

$$-1 < p_{ii} - \sum_{j \neq i} p_{ij} \leq p_{ii} + \sum_{j \neq i} p_{ij} < 1.$$

With this and using that p_{ii} is a real number (so that it lies on the real number axis), it follows that the disk B_i lies strictly within the complex unit circle $\{z \in \mathbb{C}; |z| \leq 1\}$. As this applies to any row i, the union of the disks $\cup_i B_i$ also lies strictly within the complex unit circle. Now we invoke Gershgorin's theorem, which states that all eigenvalues of the matrix P must lie in $\cup_i B_i$, to conclude that all eigenvalues of P lie strictly in the unit circle, and hence all eigenvalues have modulus smaller than 1.

Ex 7.4.1. What is the routing matrix P for a tandem network with M stations? Show that $P^M = 0$.

Ex 7.4.2. Show that when P is transient, $P^n \to 0$ exponentially fast.

Ex 7.4.3. Why does the assumption of [7.4.2] not apply to an infinite transient matrix P?

7.5 MEAN VALUE ANALYSIS OF CLOSED QUEUEING NETWORKS (UNDER CONSTRUCTION)

WE NOW CONSIDER a closed network with n circulating jobs. There are m single-server, first-come-first-served stations. The service time S_i at station i is exponentially distributed, with mean $\mathsf{E}[S_i]$. Jobs follow fixed or Markov routing. We take station 1 as the reference station and let V_i be the mean number of visits to station i per visit to station 1, so $V_1 = 1$.

Let $L_i(n)$ be the stationary number of jobs at station i, including any job in service, and $J_i(n)$ the time a job spends there per visit. The throughput at station i is $\mathrm{TH}_i(n)$. We compute these measures for populations $n = 1, \ldots, N$.

THE ARRIVAL THEOREM says that when a job arrives at station i in a closed network with n jobs, the other jobs it sees have the same stationary distribution as the jobs in the network with only $n-1$ jobs. In particular, the mean number it sees at station i is $\mathsf{E}[L_i(n-1)]$. This result holds for the product-form closed network considered here; it is not an assumption that arrivals form a Poisson process. Recall from Section 7.3 that the inter-departure times of a stationary $M/M/1$ queue with Poisson arrivals are $\mathrm{Exp}(\lambda)$. The arrival theorem instead describes the queue lengths seen by a job moving through a closed network.

If the arriving job finds q jobs at station i, it must wait for those q services and then receive its own service. The exponential distribution is memoryless, so the remaining service time of a job already in service also has mean $\mathsf{E}[S_i]$. Hence the mean time spent at station i is

$$\mathsf{E}[J_i(n)] = \big(1 + \mathsf{E}[L_i(n-1)]\big) \cdot \mathsf{E}[S_i], \qquad n \geq 1. \tag{7.5.1}$$

THE MEAN CYCLE TIME for a visit to station 1 and return is $\mathsf{E}[J(n)] = \sum_{i=1}^{m} V_i \cdot \mathsf{E}[J_i(n)]$. Little's law for the whole closed network gives $n = \mathrm{TH}_1(n) \cdot \mathsf{E}[J(n)]$. The station throughputs follow from the visit ratios, and Little's law at each station then gives

$$\mathrm{TH}_1(n) = \frac{n}{\mathsf{E}[J(n)]}, \quad \mathrm{TH}_i(n) = V_i \cdot \mathrm{TH}_1(n), \quad \mathsf{E}[L_i(n)] = \mathrm{TH}_i(n) \cdot \mathsf{E}[J_i(n)]. \tag{7.5.2}$$

We start with $\mathsf{E}[L_i(0)] = 0$. Equation (7.5.1) supplies $\mathsf{E}[J_i(1)]$, which gives $\mathrm{TH}_i(1)$ and $\mathsf{E}[L_i(1)]$. Repeating this step for $n = 2, 3, \ldots$ yields the performance measures for any desired population. The equations are exact under the assumptions above.

FOR EXAMPLE , take two stations. Let $(\mathsf{E}[S_1], \mathsf{E}[S_2]) = (2, 1)$ and $(V_1, V_2) = (1, 1)$. The first three steps give

n	$\mathrm{TH}_1(n)$	$\mathsf{E}[L_1(n)]$	$\mathsf{E}[L_2(n)]$
0	0	0	0
1	1/3	2/3	1/3
2	3/7	10/7	4/7
3	7/15	34/15	11/15

At each step, $\mathsf{E}[L_1(n)] + \mathsf{E}[L_2(n)] = n$, as it must: all n jobs are at one of the two stations.

WITH FIVE STATIONS the buildup at a bottleneck is clearer. Take mean service times $2, 3/2, 1, 3/4, 1/2$ at stations 1 through 5, respectively. Each station is visited once per cycle. Fig. 7.5.1 shows that the mean populations at the four faster stations level off, while almost every additional job joins station 1.

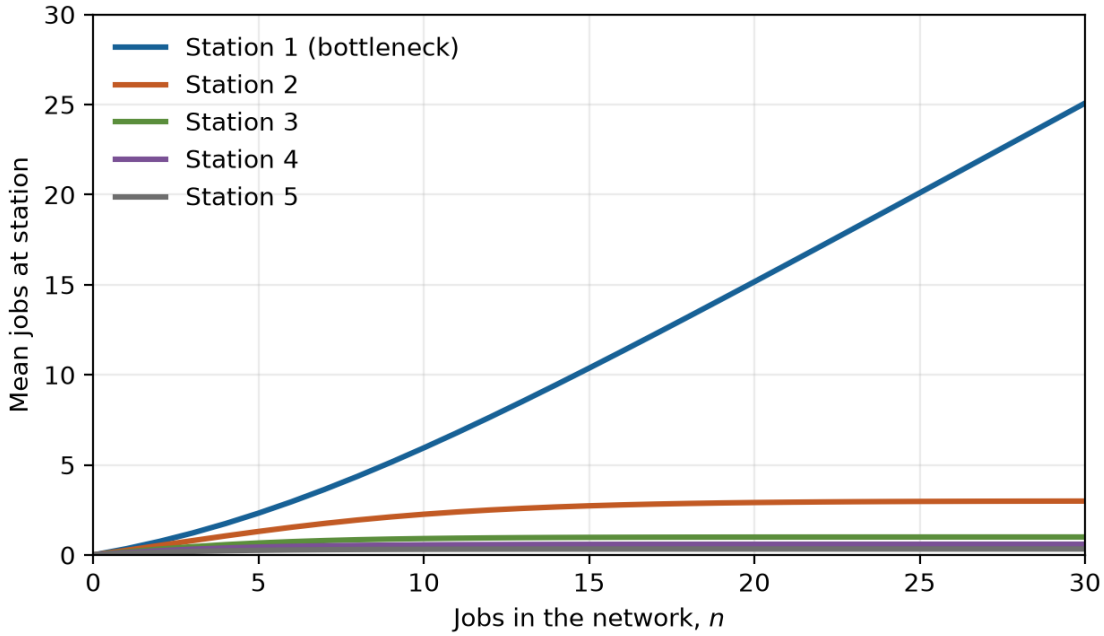


Fig. 7.5.1: Mean jobs at five stations; station 1 is the bottleneck.

IN A CONWIP LINE each job on the shop floor holds a card. When a job leaves, its card permits another job to enter. If new work is always waiting and the card is reused immediately, n cards keep exactly n jobs in the line. MVA then gives the throughput $\mathrm{TH}_1(n)$ and the mean cycle time $\mathsf{E}[J(n)] = n/\mathrm{TH}_1(n)$. Computing these measures for several card counts shows how much extra throughput we gain by allowing more work in process, and how much longer jobs spend in the line.

WE CAN COMPUTE populations in order. A cache is useful when we ask for several populations, perhaps in a different order. For example, after computing the result for $n = 2$, a request for $n = 3$ can reuse it; another request for $n = 2$ returns immediately.

The function below uses the cache and membership test from Alg. 1.2.13, and the default-zero map from Alg. 1.2.6. It returns a function MVA that retains *cache*. Each entry *cache*(n) stores the reference-station throughput and a map of mean station populations.

Algorithm 7.5.1 Compute mean value analysis results and cache them by population.

```
1:  Input: mean service times E[S_i] > 0, visit ratios V_i ≥ 0, 1 ≤ i ≤ m, with
    V_1 = 1.
2:  Output: a function MVA(n) returning TH_1(n) and E[L_i(n)], 1 ≤ i ≤ m.
3:  function MVA((E[S_i])_{i=1}^m, V)
4:      (cache, INCACHE?) ← CACHE()
5:      cache(0) ← (0, MAP())   ▷ At population zero, every station is empty.
6:      function MVA(n)
7:          if INCACHE?(n) then
8:              return cache(n)
9:          (TH_old, L_old) ← MVA(n − 1)
10:         J ← MAP(), L ← MAP(), T ← 0
11:         for 1 ≤ i ≤ m do
12:             J(i) ← (1 + L_old(i)) · E[S_i]          ▷ Arrival theorem.
13:             T +← V_i · J(i)
14:         TH_1 ← n/T                                  ▷ Little's law for the network.
15:         for 1 ≤ i ≤ m do
16:             L(i) ← V_i · TH_1 · J(i)                ▷ Little's law at station i.
17:         cache(n) ← (TH_1, L)
18:         return cache(n)
19:     return MVA
```

For the two-station example, let MVA ← MVA((2,1),(1,1)). The call MVA(2) fills *cache*(0), *cache*(1), *cache*(2). A subsequent call MVA(3) computes only the new entry *cache*(3). Repeating MVA(2) simply retrieves *cache*(2).

h.1.2.14. For (1) write $\sum_{k=0}^{\infty} G(k) = \sum_{k=0}^{\infty} \sum_{m=k+1}^{\infty} \mathsf{P}\{X = m\}$; reverse the summations. Then note that $\sum_{k=0}^{\infty} \mathbb{1}\{k < m\} = m$. For (2):

$$\sum_{i=0}^{\infty} iG(i) = \sum_{n=0}^{\infty} \mathsf{P}\{X = n\} \sum_{i=0}^{\infty} i\,\mathbb{1}\{n \geq i+1\},$$

and reverse the summations.

h.22. Recall that $\mathsf{E}[X] = M_X'(0)$, $\mathsf{E}\left[X^2\right] = M_X''(0)$, and $\mathsf{V}[X] = \mathsf{E}\left[X^2\right] - (\mathsf{E}[X])^2$.

h.1.3.7. Use that for independent rvs X, Y, $M_{X+Y}(t) = M_X(t)M_Y(t)$. Why is $M_{A_i}(t) = \mathsf{E}\left[e^{tA_i}\right] = \prod_{k=1}^{i} \mathsf{E}\left[e^{tX_k}\right]$?

h.1.3.8. LOTUS: $\mathsf{E}\left[\alpha^N\right] = \sum_{k=0}^{\infty} \alpha^k \mathsf{P}\{N = k\}$.

h.1.3.9. This result can be anticipated when you think about merging Poisson processes. Then, use $\{\min\{X, S\} > x\} = \{X > x\} \cap \{S > x\}$.

h.1.3.11. Think about splitting Poisson processes, and use that $\mathsf{P}\{S > X\} = \mathsf{E}[\mathsf{E}[\mathbb{1}\{S > X\} \mid X]]$.

h.1.3.13. Dropping the dependence of N on t for the moment for notational convenience, consider the rv

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

with $N \sim P(\lambda)$ and $Z_i \overset{\text{iid}}{\sim} B(p)$. Show that the moment-generating function of Y is equal to the moment-generating function of a Poisson rv with parameter λp.

h.2.1.3. Compute the number of jobs that depart from queue 1. Subtract the capacity used for these jobs from the total capacity to get the remaining capacity for queue 2.

h.2.1.7. Introduce a mode $M_k \in \{\mathsf{A}, \mathsf{AB}, \mathsf{B}, \mathsf{BA}\}$, where A and B mean that the nurse serves at that location, and AB and BA mean that the nurse is walking.

h.2.1.8. We need three queues: L_{1a} for the new cases arriving at the clerks, L_2 for the judge, and L_{1b} for the cases sent by the judge for the second step of the clerks. Next, convert the arrivals to hours of work arriving, and interpret the queue lengths L as work in hours to be served.

h.2.1.9. Modify d_k in (2.1.1) to incorporate the modified service behavior. Then, substitute d_k in the expression for L_k.

h.2.2.6. Introduce a variable $I_k \in \{0, 1\}$ to keep track of the state of the server. Then, $I_{k+1} = \mathbb{1}\{L_k \geq N\} + I_k \mathbb{1}\{0 < L_k < N\}$ implements the N policy.

h.2.3.9. When the machine makes n items, and each of these is faulty with probability p, then the number of faulty items is $\mathrm{Bin}(n,p)$.

h.2.3.10. Make a queue for new and repaired items and use [2.1.4].

h.2.3.11. In addition to the regular conditions, we need to impose $d^1_{k+1} \le M - L^2_k$.

h.2.4.1. Use recursion. Then apply, in order, $\max\{\max\{a,b\},c\} = \max\{a,b,c\}$; $L_0 = Z_0$; and $\max\{-a,-b\} = -\min\{a,b\}$.

h.2.4.2. Use the ideas of (1.3.7).

h.3.1.6. Use FIFO service: among the jobs that arrived before job k, job $k-1$ is the last to depart. Remember that $L(A_k-)$ is the number present immediately before A_k.

h.4.1.6. As a start, the function $\sin(t)$ does not have a limit as $t \to \infty$. However, the time-average $\sin(t)/t \to 0$. Now construct a function whose time-average does not converge; it must grow fast or oscillate increasingly wildly.

For the mathematically interested, we seek a function whose Ces\'aro limit does not exist.

h.4.2.6. Consider a queueing system with constant service times and constant inter-arrival times.

h.4.3.5. Observe that $R(t) = \int_0^t \mathbb{1}\{L(s) > 0\}\,\mathrm{d}s$ is the total time the server is busy during $[0,t]$, hence $R(t)/t$ is the fraction of time busy. Then take $T_k = D_k$ as the epochs at which to inspect $R(t)$, and realize that $R(D_k) = \sum_{i=1}^k S_i$.

h.4.3.6. For the direction of the inequalities (4.3.2), observe that t can lie halfway between a service interval, and $A(t) \ge D(t)$. Divide by t in (4.3.2). Then for the LHS, multiply by $A(t)/A(t)$ and take appropriate limits. Similar for the RHS, but multiply by $D(t)/D(t)$.

h.4.3.7. Take the process $R(\cdot)$ such that the system earns money when the server is busy, and sample at moments when busy times start.

h.4.4.6. Recall that $\lim_{t\to\infty} t^{-1}\int_0^t L_s(s)\,\mathrm{d}s$ is the time-average number of servers busy up to t.

h.4.4.7. What happens if the population increases?

h.4.5.7. Solve [4.5.6] first. Use that $\lambda \ge \delta$ always holds. Thus, when $\lambda \ne \delta$, necessarily $\lambda > \delta$. What are the consequences of this inequality; How does the queue length behave as a function of time?

h.4.6.8. Apply PASTA and (4.6.8).

h.5.1.8. Use (5.1.1), show that $M_L(s) = (1-\rho)\sum_n e^{sn}\rho^n$, then use the familiar expression of the sum over a geometric series. Similarly, $\mathsf{P}\{L \ge n\} = \sum_{k\ge n} p(k)$.

h.5.1.10. Fill in $c = 1$. Realize that this is a check on the formulas.

h.5.1.12. Use that $\sum_{i=0}^{n} x^i = (1-x^{n+1})/(1-x)$.

Is it necessary for this expression to be true that $|x| < 1$? What should you require for $|x|$ when you want to take the limit $n \to \infty$?

h.5.2.6. $\mathsf{E}[Q]$ follows right away from an application of Little's law. For the other quantities, we need to find $\mathsf{E}[S]$. Use the expression for $\mathsf{E}[W]$ for the queue $M/M/1$ to solve for ρ. Then, since λ is known, $\mathsf{E}[S]$ follows.

h.5.2.9. Let T be the number of patties waiting and G the number of persons waiting. There cannot be a patty and a person waiting at the same time, hence $T \cdot G = 0$. Take L as the number of jobs in an $M/M/1$ queue. If $L = 0$, then there are 3 patties on stock, when $L = 1$ there is one patty less $T = 2, G = 0$; when $L = 2$, we have two patties less, so $T = 1, G = 0$, when $L = 3$, there are no patties on stock, but also no people waiting, hence $T = 0, G = 0$. When $L = 4$, we ran out of patties and one person must be waiting, hence $T = 0, G = 1$.

With this reasoning we see a pattern: at all moments in time, $3 = T + L - G$. Now map this system to an $M/M/1$ queue.

h.5.2.10. How is the *invariant* $3 = T + L - G$ affected?

h.5.3.7. $\mathsf{P}\{B = k\} = q^{k-1}p$ with $q = 1 - p$. Look up the expectation and variance of a geometric rv.

h.5.3.11. Use [1.2.14], the recursion for $\pi(n)$, $\rho = \lambda_B/\mu \sum_{i=0}^{K_B-1} \mathsf{P}\{B > i\}$, and realize that $\mathsf{E}[B] \geq 1$ whatever the distribution of B.

h.5.3.12. Read the answer of [1.2.14].

h.5.3.13. Show first that

$$\mu\,\mathsf{E}[L] = \mu \sum_{n=0}^{\infty} n\pi(n) = \lambda_B \frac{\mathsf{E}\left[B^2\right]}{2} + \lambda_B\,\mathsf{E}[B]\,\mathsf{E}[L] + \lambda_B \frac{\mathsf{E}[B]}{2}.$$

h.5.4.7. Use $\mathsf{V}[S] = 0$ for the $M/D/1$ queue and $C_s^2 = 1$ for the $M/M/1$ queue.

h.5.4.9. Use the PASTA property. Note that when estimating $\mathsf{E}[S_r]$ along a sample path, $S_r = 0$ for jobs that arrive at an empty system.

h.5.5.2. Use conditional probability or [1.3.10] and [1.3.11].

h.5.5.3. Use [5.5.2].

h.5.5.4. Use the results of [5.5.2].

h.5.5.5. Solve [5.5.2] and [5.5.3] first. Use the following shorthands: $\alpha = \lambda/(\lambda+\mu) \implies \mu/(\lambda+\mu) = 1-\alpha \implies \alpha/(1-\alpha) = \lambda/\mu = \rho$.

h.5.6.7. Use the argumentation that leads to (4.3.2).

h.5.6.8. Use that $\sum_{n=0}^{N} \alpha^n = \frac{1-\alpha^{N+1}}{1-\alpha}$.

h.5.6.11.

$$\begin{aligned} aq^2 &= aq^2, \\ bq &= 2aq\,\mathsf{E}[Y] - 2aq + bq + hq\,\mathsf{E}[S], \\ 0 &= a\,\mathsf{E}\left[Y^2\right] - 2a\,\mathsf{E}[Y] + a + b\,\mathsf{E}[Y] - b + \frac{1}{2}\lambda h\,\mathsf{E}\left[S^2\right]. \end{aligned}$$

h.5.7.8. Use the following properties for convex functions.

1. f convex, $\alpha \geq 0 \implies \alpha f$ convex.
2. f, g convex $\implies f + g$ convex.
3. f convex, $R(x) = -x \implies f \circ R$ convex.

h.5.8.7. No backlogging and $l = 0 \implies r = -1$. Use [5.7.6].

h.5.9.6. Suppose we rename `lion` and `tiger` as follows:

```
square = exp_apply(2)
cube = exp_apply(3)
```

h.5.10.1. How do we handle states at which STOP?(x) is true earlier in the algorithm?

h.6.1.6. What is λ? What is C_a^2 in the $G/G/1$ setting; What is it in the $M/G/1$ setting?

h.6.1.7. Let S be the service (or service) time at the server, and S_T the service time of a type T job. Then,

$$S = \mathbb{1}\{T = 1\}\, S_1 + \mathbb{1}\{T = 2\}\, S_2.$$

h.6.2.7. An arbitrary job has to wait half the time it takes to form a batch.

h.6.3.6. Get the units right. Compute the load and then the rest.

h.6.3.7. Realize that we now deal with setups and batch processing.

h.6.4.6. Be sure to work in a consistent set of units, e.g., hours. It is easy to make mistakes.

h.6.4.7. Observe that $m_f = 1/\lambda_f$ and $m_r = \mathsf{E}[R]$.

h.6.4.8. Condition on S_0 and N.

h.6.4.9. Just realize that $\mathsf{E}[S] = \mathsf{E}[S_0]\,/\,A$, and use the above.

h.6.4.11. Define suitable moments $T_1 < T_2 < \ldots$ to inspect the system, and try to find a suitable function $U(t)$ that captures what we want to measure.

h.7.1.5. Think about the inter-departure time of a machine that produces jobs every 10 minutes. What inter-arrival times will a downstream machine see?

h.7.2.4. Use Little's law.

h.7.2.5. If $w < n$, then what happens? If $w > n$, what are the $w - n$ jobs doing?

h.7.2.9. Is the new machine placed in parallel or in series?

h.7.2.10. Check your reasoning very carefully here. It is easy to do it wrong.

h.7.3.1. Do [5.5.1] first.

h.7.3.2. Conditioning on the server being idle or busy at a departure leads to $f_D(t) = f_{X+S}(t)\,\mathsf{P}\{\text{server is idle}\} + f_S(t)\,\mathsf{P}\left\{\text{server is busy}\right\}$. Next, use [7.3.1].

h.7.3.3. First, compute λ, then analyze what happens if $\alpha \to 1$ or $\beta_1 \to 0$.

h.7.3.4. Realize that an arrival is required to leave state $(0,0)$, and a departure at the second queue is necessary to enter state $(0,0)$.

h.7.4.1. After visiting M stations, any job must have left the tandem network.

h.7.4.2. Define $Q = P^M$. As P is transient, the row sums of Q are strictly smaller than 1. Then show that $Q^n \to 0$.

s.1.2.1. False. The function G retains $b = 3$, so $\mathrm{G}(5) = 5 - 3 = 2$.

s.1.2.2. False. The call $\mathrm{G}(2)$ returns the function $\lambda\colon\ x \mapsto x \cdot \mathrm{F}(x)$, not a number. The inner x is a new argument, so the 2 passed to G is unused. Calling the returned function at 3 gives $\mathrm{G}(2)(3) = 3 \cdot \mathrm{F}(3) = 3 \cdot 6 = 18$.

s.1.2.3. False. The function LATE retains the threshold $b = 5$. It returns $0, 0, 1$ for these waiting times, so the sum is 1.

s.1.2.4. True. When a key is encountered for the first time, the function stores the default value d under that key.

s.1.2.5. True.

s.1.2.6. False. The successive values are

w	4	9	2	11	4
T	4	13	15	26	30
C	0	1	1	2	2

.

Thus, $\bar{w} = 30/5 = 6$ and $q = 2/5$. However, the algorithm returns only q.

s.1.2.7. False. The loop should start at $k = 1$. If it did, then after the loop, $N = n$ and $T = \sum_{k=1}^{n} x_k$, so T/N is the average.

s.1.2.8. True. After iteration k, $M = \max\{w_1, \ldots, w_k\}$. Therefore, after the final iteration, M is the largest observation.

s.1.2.9. False. The algorithm computes the same inter-arrival times, but sorts them before returning them. Consequently, their chronological order is lost. The two outputs are equal only when $X_2, \ldots, X_{NT}$ are already in increasing order.

s.1.2.10. False. It should have $[x \in \mathcal{S} : x < p]$. Otherwise, every copy of the pivot also occurs in $\mathcal{L}$. In particular, if all elements are equal, then $\mathcal{L} = \mathcal{S}$, so the recursive call does not receive a shorter list.

s.1.2.11. True. The recursive calls sort the elements below and above the pivot. Every element of $\mathcal{L}$ is smaller than every element of $\mathcal{M}$, and every element of $\mathcal{M}$ is smaller than every element of $\mathcal{R}$, so their concatenation is sorted increasingly.

s.1.2.12. True. The arguments $0, 1, 2$ are stored as keys in that order, and the numbers assigned to $p(0), p(1), p(2)$ are returned as their corresponding values.

s.1.2.13. False. We have $\textsc{Keys}(M) = [2, 5]$ and $\textsc{Values}(M) = [6, 2]$. The value 6 stored under key 2 does not itself become a key.

s.1.2.14. For (1): observe first that $\sum_{k=0}^{\infty} \mathbb{1}\{m > k\} = m$, since $\mathbb{1}\{m > k\} = 1$ if $k < m$ and $\mathbb{1}\{m > k\} = 0$ if $k \geq m$. With this,

$$\begin{aligned}\sum_{k=0}^{\infty} G(k) &= \sum_{k=0}^{\infty} \mathsf{P}\{X > k\} = \sum_{k=0}^{\infty} \sum_{m=k+1}^{\infty} \mathsf{P}\{X = m\} \\ &= \sum_{k=0}^{\infty} \sum_{m=0}^{\infty} \mathbb{1}\{m > k\}\, \mathsf{P}\{X = m\} = \sum_{m=0}^{\infty} \sum_{k=0}^{\infty} \mathbb{1}\{m > k\}\, \mathsf{P}\{X = m\} \\ &= \sum_{m=0}^{\infty} m\, \mathsf{P}\{X = m\} = \mathsf{E}[X].\end{aligned}$$

Since all terms are non-negative, interchanging the summations is valid.

For (2):

$$\begin{aligned}\sum_{i=0}^{\infty} iG(i) &= \sum_{i=0}^{\infty} i \sum_{n=i+1}^{\infty} \mathsf{P}\{X = n\} = \sum_{n=0}^{\infty} \mathsf{P}\{X = n\} \sum_{i=0}^{\infty} i\, \mathbb{1}\{n \geq i+1\} \\ &= \sum_{n=0}^{\infty} \mathsf{P}\{X = n\} \sum_{i=0}^{n-1} i = \sum_{n=0}^{\infty} \mathsf{P}\{X = n\} \frac{(n-1)n}{2} \\ &= \sum_{n=0}^{\infty} \frac{n^2}{2} \mathsf{P}\{X = n\} - \frac{\mathsf{E}[X]}{2} = \frac{\mathsf{E}[X^2]}{2} - \frac{\mathsf{E}[X]}{2}.\end{aligned}$$

s.1.3.1. False. Since $N(0,s] + N(s,t] = N(0,t]$, the LHS just says that there was one arrival during $(0,t]$. The RHS says something different, in particular because $s < t$.

s.1.3.2. True.

s.1.3.3. True. Each value $-\lambda^{-1}\log U$ is independently distributed as $\mathrm{Exp}(\lambda)$. The sum of k such values has distribution $\mathrm{Gamma}(\lambda, k)$.

s.1.3.4. True. The solution follows directly from standard probability and Section 1.3. Take $U \sim \mathrm{Unif}((0,1))$, and $f(x) = -\frac{1}{\lambda}\log x$. Then,

$$\mathsf{P}\{f(U) \leq a\} \overset{1}{=} \mathsf{P}\{U \geq f^{-1}(a)\} \overset{2}{=} 1 - f^{-1}(a) \overset{3}{=} 1 - e^{-\lambda a},$$

where 1 follows from the fact that f is strictly decreasing, 2 from the general property of the uniform distribution $\mathsf{P}\{U \geq b\} = 1 - b$, and 3 because $f^{-1}(y) = e^{-\lambda y}$. Thus, if we define $X := f(U)$, then $X = -\frac{1}{\lambda}\log U \sim \mathrm{Exp}(\lambda)$. In other words, the rv $X = -\frac{1}{\lambda}\log U$ is exponentially distributed.

Next, define

$$N(t) = \inf\{j : A_j \leq t < A_{j+1}\},$$

in words, $N(t)$ is the index of the last arrival before or equal to t. Moreover, when $\{U_i\}$ is a sequence of iid uniform rvs on $(0,1)$ then the $X_i = -\frac{1}{\lambda}\log U_i$ satisfy $X_i \overset{\text{iid}}{\sim} \mathrm{Exp}(\lambda)$. Since $A_k = A_{k-1} + X_k$, the event

$$\begin{aligned}\{N(t) = k\} &= \{A_k \leq t < A_{k+1}\} \\ &= \left\{\sum_{i=1}^{k} X_i \leq t < \sum_{i=1}^{k+1} X_i\right\} \\ &= \left\{-\frac{1}{\lambda}\sum_{i=1}^{k} \log U_i \leq t < -\frac{1}{\lambda}\sum_{i=1}^{k+1} \log U_i\right\}.\end{aligned}$$

Now taking $t = 1$, we arrive at the equality:

$$\{N(1) = k\} = \left\{ -\frac{1}{\lambda} \sum_{i=1}^{k} \log U_i \le 1 < -\frac{1}{\lambda} \sum_{i=1}^{k+1} \log U_i \right\}.$$

Since $-\frac{1}{\lambda} \log U \sim \text{Exp}(\lambda)$, the sum of k such rvs is Gamma (λ, k), and, by (1.3.4) and (1.3.5),

$$\mathsf{P}\{N(1) = k\} = \mathsf{P}\left\{ -\frac{1}{\lambda} \sum_{i=1}^{k} \log U_i \le 1 < -\frac{1}{\lambda} \sum_{i=1}^{k+1} \log U_i \right\} = \frac{\lambda^k}{k!} e^{-\lambda}.$$

One could divide every unit-rate exponential draw $-\log U$ by λ and use threshold 1. [1.3.4] avoids these divisions by multiplying the threshold by λ instead.

Note that each uniform rv drawn in the while loop must be a new realization. Otherwise, the algorithm repeatedly adds the same number, which is incorrect.

s.1.3.5. True. Observe that

$$\mathsf{P}\{A_k \le t\} - \mathsf{P}\{A_{k+1} \le t\} = \frac{(\lambda t)^k}{k!} e^{-\lambda t}.$$

s.22. For $t < \lambda$, and $f(t) = \lambda e^{-\lambda t} \mathbb{1}\{t \ge 0\}$,

$$M_X(t) = \mathsf{E}\left[\exp(tX)\right] = \int_0^\infty e^{tx} f(x)\, \mathrm{d}x = \int_0^\infty e^{tx} \lambda e^{-\lambda x}\, \mathrm{d}x = \frac{\lambda}{\lambda - t}.$$

This last integral only converges when $\lambda > t$. Next, $M_X'(t) = \lambda/(\lambda - t)^2 \implies M_X'(0) = 1/\lambda$, $M_X''(t) = 2\lambda/(\lambda - t)^3 \implies \mathsf{E}\left[X^2\right] = 2\lambda^{-2}$. Thus, $\mathsf{V}[X] = \mathsf{E}\left[X^2\right] - (\mathsf{E}[X])^2 = \frac{2}{\lambda^2} - (\frac{1}{\lambda})^2 = \lambda^{-2}$.

s.1.3.7. Using the iid property of the $\{X_i\}$,

$$M_{A_i}(t) = \mathsf{E}\left[e^{tA_i}\right] = \mathsf{E}\left[\exp\left(t \sum_{k=1}^{i} X_k\right)\right] = \prod_{k=1}^{i} \mathsf{E}\left[e^{tX_k}\right] = \left(\frac{\lambda}{\lambda - t}\right)^i.$$

From a table of moment-generating functions, it follows immediately that $A_i \sim \text{Gamma}(\lambda, i)$, i.e., A_i is Gamma distributed.

s.1.3.8.

$$\sum_{k=0}^{\infty} \alpha^k \mathsf{P}\{N = k\} = \sum_{k=0}^{\infty} \alpha^k \frac{(\lambda)^k}{k!} e^{-\lambda} = e^{-\lambda} \sum_{k=0}^{\infty} \frac{(\alpha\lambda)^k}{k!} = \exp(\lambda(\alpha - 1)).$$

Taking $\alpha = e^s$ in this equation and differentiating gives

$$M_{N(t)}'(s) = \lambda t e^s \exp(\lambda t(e^s - 1)).$$

Hence $\mathsf{E}[N(t)] = M_{N(t)}'(0) = \lambda t$. Next, $M_{N(t)}''(s) = (\lambda t e^s + (\lambda t e^s)^2) \exp(\lambda t(e^s - 1))$, hence $\mathsf{E}\left[(N(t))^2\right] = M_{N(t)}''(0) = \lambda t + (\lambda t)^2$, and thus, $\mathsf{V}[N(t)] = \mathsf{E}\left[(N(t))^2\right] - (\mathsf{E}[N(t)])^2 = \lambda t + (\lambda t)^2 - (\lambda t)^2 = \lambda t$.

s.1.3.9. $\mathsf{P}\{Z > x\} = \mathsf{P}\left\{\min\{X, S\} > x\right\} = \mathsf{P}\{X > x, S > x\} = \mathsf{P}\{X > x\}\, \mathsf{P}\{S > x\} = e^{-\lambda x} e^{-\mu x}$, since X and S are independent.

s.1.3.10. From (1.3.5) and (1.3.7) and

$$\begin{aligned}\mathsf{P}\left\{N_\lambda(t)=1\,\middle|\,N_\lambda(t)+N_\mu(t)=1\right\} &= \frac{\mathsf{P}\{N_\lambda(t)=1\}\,\mathsf{P}\left\{N_\mu(t)=0\right\}}{\mathsf{P}\left\{N_{\lambda+\mu}(t)=1\right\}}\\ &= \frac{\lambda t\exp(-\lambda t)\exp(-\mu t)}{((\lambda+\mu)t)\exp\left(-(\lambda+\mu)t\right)} = \frac{\lambda t\exp\left(-(\lambda+\mu)t\right)}{((\lambda+\mu)t)\exp\left(-(\lambda+\mu)t\right)} = \frac{\lambda}{\lambda+\mu}.\end{aligned}$$

s.1.3.11. With conditioning, $\mathsf{P}\{S>X\,|\,X=t\}=\mathsf{P}\{S>t\}=e^{-\mu t}$. With the fundamental bridge and conditional expectation, $\mathsf{E}[\mathbb{1}\{S>X\}\,|\,X]=e^{-\mu X}$, hence $\mathsf{E}[\mathbb{1}\{S>X\}]=\mathsf{E}\left[e^{-\mu X}\right]$. But this last formula is equal to $M_X(-\mu)$, so taking $t=-\mu$ in the mgf of X we obtain $\lambda/(\lambda+\mu)$.

s.1.3.12.

$$\begin{aligned}M_{N_\lambda(t)+N_\mu(t)}(s) &= M_{N_\lambda(t)}(s)\cdot M_{N_\mu(t)}(s)=\exp(\lambda t(e^s-1))\cdot\exp(\mu t(e^s-1))\\ &= \exp((\lambda+\mu)t(e^s-1)).\end{aligned}$$

Thus, the number of arrivals in an interval of length t is Poisson distributed with parameter $(\lambda+\mu)t$. The merged process also has stationary and independent increments, since its increments are sums of the corresponding independent increments of the two input processes.

s.1.3.13.

$$\mathsf{E}\left[e^{sZ}\right]=e^0\,\mathsf{P}\{Z=0\}+e^s\,\mathsf{P}\{Z=1\}=(1-p)+e^s p,$$

from which

$$\mathsf{E}\left[e^{s\Sigma_{i=1}^n Z_i}\right]=\left(\mathsf{E}\left[e^{sZ}\right]\right)^n=\left(1+p(e^s-1)\right)^n,$$

where we use that the Z_i are iid. Thus, (see the hint) $\mathsf{E}\left[e^{sY}|N=n\right]=\mathsf{E}\left[e^{s\Sigma_{i=1}^n Z_i}\right]=\alpha^n$ where $\alpha=1+p(e^s-1)$. With Adam's law and [1.3.8], we get

$$\mathsf{E}\left[e^{sY}\right]=\mathsf{E}\left[\alpha^N\right]=e^{\lambda(\alpha-1)}=\exp(\lambda p(e^s-1)).$$

This is the mgf of a Poisson rv with parameter λp. For an interval of length t, the same computation with λt in place of λ gives parameter λpt. The thinned process has stationary and independent increments because the original process does and the customers' decisions are independent.

s.1.5.1. True. The original relative weights are 0.1, 0.2, and 0.7, so the key a is removed. The retained weights 2 and 7 are then normalized by their sum 9.

s.1.5.2. True. The successive sums are 3, $3-1=2$, $2+4=6$, and $6-2=4$.

s.1.5.3. True. Independence gives probability $p_X(x)p_Y(y)$ to the pair (x,y), and all pairs with the same value $z=f(x,y)$ contribute to the same key.

s.1.5.4. True. When $a_m\le x$, the first m entries are no longer in doubt, so the recursive call searches the part starting at $m+1$.

s.1.5.5. False when F has jumps.

s.1.5.6. False. Check the index of the second summation.

s.1.5.7. False. It first uses denominator 100 and numerator round(100/3) = 33. Since 33 and 100 have greatest common divisor one, the function returns 33/100.

s.2.1.1. True.

s.2.1.2. True.

s.2.1.3. True.

s.2.1.4. True. Queue 2 minimally needs $c_k^2 = \min\{L_{k-1}^2, r^2\}$, and with this,

$$d_k^1 = \min\{L_{k-1}^1, c_k - c_k^2\}, \qquad d_k^2 = \min\{L_{k-1}^2, c_k - d_k^1\}.$$

s.2.1.5. False. Let

$$H_n = \sum_{k=1}^{n} \mathbb{1}\{L_{k-1} \geq 48\}, \qquad D_n = \sum_{k=1}^{n} \mathbb{1}\{L_{k-1} \leq 24\}.$$

Then

$$\frac{1}{n}\sum_{k=1}^{n} c_k = 12 + e\frac{H_n - D_n}{n}.$$

Thus, the average capacity equals 12 only when $H_n = D_n$. Symmetric adjustments need not occur equally often.

s.2.1.6. With n_k servers, $c_k = n_k / \mathsf{E}[S]$. The fast lane has βn_k servers, each with service rate $1/(\mathsf{E}[S] - 0.5)$, so its capacity is $\beta n_k/(\mathsf{E}[S] - 0.5) = \beta c_k \mathsf{E}[S]/(\mathsf{E}[S] - 0.5) = 4\beta c_k/3$. The regular lane has capacity $(1-\beta)c_k$. Hence,

$$d_k^f = \min\left\{L_{k-1}^f, \beta c_k \mathsf{E}[S]/(\mathsf{E}[S] - 0.5)\right\}, \qquad L_k^f = L_{k-1}^f + \alpha a_k - d_k^f,$$
$$d_k^r = \min\{L_{k-1}^r, (1-\beta)c_k\}, \qquad L_k^r = L_{k-1}^r + (1-\alpha)a_k - d_k^r.$$

s.2.1.7. The state of the nurse evolves according to

$$M_k = \begin{cases} \mathsf{AB}, & M_{k-1} = \mathsf{A} \text{ and } L_{k-1}^A \leq h_A, \\ \mathsf{B}, & M_{k-1} = \mathsf{AB}, \\ \mathsf{BA}, & M_{k-1} = \mathsf{B} \text{ and } L_{k-1}^B \leq h_B, \\ \mathsf{A}, & M_{k-1} = \mathsf{BA}, \\ M_{k-1}, & \text{otherwise.} \end{cases}$$

The departure and queue-length recursions are

$$d_k^A = \min\{L_{k-1}^A + a_k^A,\ c_k^A \mathbb{1}\{M_k = \mathsf{A}\}\}, \qquad L_k^A = L_{k-1}^A + a_k^A - d_k^A,$$
$$d_k^B = \min\{L_{k-1}^B + a_k^B,\ c_k^B \mathbb{1}\{M_k = \mathsf{B}\}\}, \qquad L_k^B = L_{k-1}^B + a_k^B - d_k^B.$$

When the nurse is walking, both indicators vanish, so no patients depart from either queue.

s.2.1.8. The clerks first use their capacity for the initial work on new cases. Any remaining capacity is used for the final administrative work. Thus,

$$\begin{aligned} d_k^{1a} &= \min\left\{L_{k-1}^{1a}, 80\right\}, & L_k^{1a} &= L_{k-1}^{1a} - d_k^{1a} + 5a_k, \\ d_k^{1b} &= \min\left\{L_{k-1}^{1b}, 80 - d_k^{1a}\right\}, & L_k^{1b} &= L_{k-1}^{1b} - d_k^{1b} + \frac{2}{3}d_k^2, \\ d_k^2 &= \min\left\{L_{k-1}^2, 34\right\}, & L_k^2 &= L_{k-1}^2 - d_k^2 + \frac{3}{5}d_k^{1a}. \end{aligned}$$

The factor $3/5$ converts completed initial clerk work into newly arriving judge work: each case requires 5 clerk hours and then 3 judge hours. Likewise, $2/3$ converts completed judge work into final clerk work. Since each departure depends only on the backlog at the end of week $k-1$, work passed downstream during week k cannot be processed there until the next week.

s.2.1.9.

$$\begin{aligned} d_k &= \min\left\{L_{k-1} + a_k, c_k\right\}, \\ L_k &= L_{k-1} + a_k - d_k = L_{k-1} + a_k - \min\left\{L_{k-1} + a_k, c_k\right\} \\ &\overset{1}{=} -\min\left\{c_k - L_{k-1} - a_k, 0\right\} \overset{2}{=} \max\left\{L_{k-1} + a_k - c_k, 0\right\}, \end{aligned}$$

where 1 uses that $x - \min\{x,y\} = -\min\{0, y-x\}$ and 2 that $-\min\{x,0\} = \max\{-x,0\}$.

s.2.2.1. False. If $d_k^1 < r^1$, the expression $c_k - d_k^1$ incorrectly assigns unused capacity reserved for queue 1 to queue 2. The correct recursion is

$$c_k^2 = c_k - \max\left\{r^1, d_k^1\right\}.$$

The other recursions remain unchanged.

An example is the operation room of a hospital. There is a weekly capacity, part of which 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. As a result, unused capacity is lost. In practice, it may not be as extreme as in the model, but part of the unused capacity is still lost. 'Use it, or lose it' often applies to service capacity.

s.2.2.2. False. In a stable system, flow balance implies that the long-run departure rate is λ, so $\rho = \lambda \mathsf{E}[S]$, independently of N. The threshold changes the lengths of the busy and idle periods, but not the long-run fraction of time spent serving jobs.

s.2.2.3. False. Jobs that arrive while the server is off must wait until the queue reaches N. The policy trades longer waiting times for a lower long-run average switching cost.

s.2.2.4. True. When $L_{k-1} = 0$, the first term is 0 (since $0 < N$) and the second term is $I_{k-1} \cdot 0 = 0$ (since $0 < L_{k-1}$ is false). Hence $I_k = 0$.

s.2.2.5. True. Since $I_k \in \{0,1\}$, the inequality $I_k > I_{k-1}$ holds exactly when the state changes from off to on.

s.2.2.6. The required recursions are

$$\begin{aligned} I_k &= \mathbb{1}\{L_{k-1} \geq N\} + I_{k-1}\mathbb{1}\{0 < L_{k-1} < N\}, \\ d_k &= \min\left\{L_{k-1} + a_k, I_k c_k\right\}, \\ L_k &= L_{k-1} + a_k - d_k. \end{aligned}$$

The first recursion implements the switching rule, while the factor I_k makes the available capacity zero whenever the server is off. Note that when arrivals cannot be served in the period in which they arrive, it might be that the system is never empty. For instance, if $a_k \sim \text{Unif}((4,10))$, then $L_k \geq a_k \geq 4$ for all k.

s.2.2.7. The total cost is

$$C_T = \sum_{k=1}^{T} \left(\beta I_k + hL_k + K\mathbb{1}\{I_k > I_{k-1}\}\right).$$

The three terms are, respectively, the operating cost, the holding cost, and the switch-on cost.

s.4. Let a_k^1 be the external arrivals at station 1, and the departures of station 1 are the arrivals at station 2: $a_k^2 = d_k^1$. Thus,

$$d_k^i = \min\{L_{k-1}^i, c_k^i\}, \qquad L_k^i = L_{k-1}^i - d_k^i + a_k^i.$$

s.2.3.1. False. The condition allows $\lambda = \mu$, which does not guarantee stability.

s.2.3.2. False. We can also block jobs, or reduce the variability of the arrival or service process.

s.2.3.3. False. Since, on average, two jobs arrive per period and one leaves, the queue length increases on average by one job per period. After 1000 periods, the system must contain about 1000 jobs. As the Poisson distribution has a small relative variability (if $X \sim \text{Pois}(\lambda)$, then $\mathsf{V}[X]/(\mathsf{E}[X])^2 = 1/\lambda$), the probability that $L_{1000} < 100$ is exceedingly small.

s.2.3.4. True.

s.2.3.5. False. Variability in payment time also affects the queue length.

s.2.3.6. False.

s.2.3.7. True.

s.2.3.8. Jobs that would make the queue length exceed K are blocked. First, d_k jobs depart. Let $\widetilde{L}_k$ be the queue length that would result after adding the new arrivals without blocking. The actual queue length is capped at K, and the difference is the number of jobs lost:

$$\begin{aligned} d_k &= \min\left\{L_{k-1}, c_k\right\}, & \widetilde{L}_k &= L_{k-1} + a_k - d_k, \\ L_k &= \min\left\{\widetilde{L}_k, K\right\}, & \ell_k &= \widetilde{L}_k - L_k. \end{aligned}$$

Thus, the average number of jobs lost per period and the fraction of arrivals lost over T periods are, respectively,

$$\overline{\ell}_T = \frac{1}{T}\sum_{k=1}^{T} \ell_k, \qquad q_T = \frac{\sum_{k=1}^{T} \ell_k}{\sum_{k=1}^{T} a_k}.$$

s.2.3.9. First, s_k items are processed. Of these, r_k are defective and return to the head of the queue, so only $d_k = s_k - r_k$ items leave the system permanently. The queue update therefore subtracts only d_k, which is how the defective items remain in the system. This gives the recursions

$$s_k = \min\{L_{k-1}, c_k\}, \qquad r_k \mid s_k \sim \text{Bin}(s_k, p),$$
$$d_k = s_k - r_k, \qquad L_k = L_{k-1} + a_k - d_k.$$

An interesting challenge: Can you use these recursions to *prove* that, if $L_n/n \to 0$, the long-run average service capacity $\overline{c} = \lim_{n\to\infty} n^{-1}\sum_{k=1}^{n} c_k$ must satisfy $\overline{c} \geq \gamma/(1-p)$, where $\gamma = \lim_{n\to\infty} n^{-1}\sum_{k=1}^{n} a_k$ is the arrival rate of new jobs?

s.2.3.10. Let L_k^R and L_k^N be the numbers of repair and new jobs, respectively. Repair jobs have priority. Because a repair needs half the service time of a new job, one unit of capacity can process two repairs. We first process as many waiting repairs as the capacity allows, including a single repair when the repair queue is odd. Only whole new jobs can be completed. After the repairs, c_k^N is the capacity left for new jobs. Model the quality outcomes of new jobs as mutually independent, each requiring repair with probability p. Of the d_k^N new jobs processed, a_k^R require repair and join the repair queue for the next period. These modeling choices give the recursions

$$d_k^R = \min\{L_{k-1}^R, 2c_k\}, \qquad c_k^N = c_k - \frac{d_k^R}{2},$$
$$d_k^N = \min\{L_{k-1}^N, \lfloor c_k^N \rfloor\}, \qquad a_k^R \mid d_k^N \sim \text{Bin}(d_k^N, p),$$
$$L_k^R = L_{k-1}^R + a_k^R - d_k^R, \qquad L_k^N = L_{k-1}^N + a_k - d_k^N.$$

As a general observation: there are many ways to organize the repairs. Do we serve them with priority or not, do we bin them and make new items instead, do we do the repairs on another, separate, station? Such differences should be reflected in a simulation model.

s.2.3.11. A first attempt would limit departures from station 1 by $M - L_{k-1}^2$, the free space at station 2. This quantity can be negative, for example when the initial queue at station 2 exceeds M, so we use its positive part. Moreover, because transfers from station 1 must fit before station 2 serves any jobs, the available space is based on L_{k-1}^2, not on the queue after d_k^2 departures. The complete recursions are

$$d_k^1 = \min\{L_{k-1}^1, c_k^1, [M - L_{k-1}^2]^+\}, \qquad d_k^2 = \min\{L_{k-1}^2, c_k^2\},$$
$$L_k^1 = L_{k-1}^1 + a_k^1 - d_k^1, \qquad L_k^2 = L_{k-1}^2 + d_k^1 - d_k^2.$$

s.2.3.12. Because departures occur before the batch arrives, first determine d_k, then add a_k to obtain the new queue length. Thus,

$$d_k = \min\{L_{k-1}, c_k\}, \qquad L_k = L_{k-1} + a_k - d_k.$$

The jth job in the batch arriving in period k sees $L_{k-1} - d_k + j - 1$ jobs. Let $N_T = \sum_{k=1}^{T} a_k$. Counting the arriving jobs that see at most m jobs in the system and dividing by the total number of arrivals gives an arrival-weighted empirical distribution, rather than a time average of the queue length:

$$\widehat{F}_T(m) = \frac{1}{N_T}\sum_{k=1}^{T}\sum_{j=1}^{a_k} \mathbb{1}\{L_{k-1} - d_k + j - 1 \leq m\}.$$

s.2.4.1.

$$\begin{aligned} L_k - Z_k &= \max\left\{\max\left\{L_{k-2} - Z_{k-2}, -Z_{k-1}\right\}, -Z_k\right\} \\ &= \max\left\{L_{k-2} - Z_{k-2}, -Z_{k-1}, -Z_k\right\} \\ &= \max\left\{L_0 - Z_0, -Z_1, \ldots, -Z_k\right\} \\ &= \max\left\{0, -Z_1, \ldots, -Z_k\right\} \\ &= -\min\left\{0, Z_1, \ldots, Z_k\right\}. \end{aligned}$$

s.2.4.2.

$$\begin{aligned} \mathsf{P}_m\{Z_k = n\} &= \mathsf{P}\left\{m + N_{\lambda,k} - N_{\mu,k} = n\right\} = \mathsf{P}\left\{N_{\lambda,k} = n - m + N_{\mu,k}\right\} \\ &= \sum_{j=0}^{\infty} \mathsf{P}\left\{N_{\lambda,k} = n - m + j, N_{\mu,k} = j\right\} = \sum_{j=\max\left\{0,m-n\right\}}^{\infty} e^{-\lambda k} \frac{(\lambda k)^{n-m+j}}{(n-m+j)!} e^{-\mu k} \frac{(\mu k)^j}{j!} \\ &= e^{-(\lambda+\mu)k} (\lambda k)^{n-m} \sum_{j=\max\left\{0,m-n\right\}}^{\infty} \frac{(\lambda\mu k^2)^j}{j!(n-m+j)!}. \end{aligned}$$

s.2.5.1. False. It is the inventory position.

s.2.5.2. True.

s.2.5.3. True.

s.2.5.4. True. The base-stock policy is also an (s, S) policy with $s = S - 1$, so the formula covers both models with a maximum order size Q. Variation. Consider the (T, S) model. Then $Q_k = \min\left\{Q, S - P'_{k-1}\right\}$ only if k is a multiple of T.

s.2.5.5. False. An order is placed only when the inventory position drops to or below the reorder level s. If the position is above s, no order is placed.

s.2.5.6. In a queueing system, an item is not produced before the demand for it arrives. This is a cost-effective way to produce when items are very expensive, so holding cost is high, or customer specific. In addition, services (which are not products) cannot be put on stock. For instance, keeping an inventory of haircuts seems impossible.

Note that in a queueing system, the server has a finite capacity. In the inventory models above, the production capacity is assumed infinite.

A base-stock model with $s = -1$ is appropriate. When a customer arrives, the server switches on, and keeps on producing until there are no further jobs in queue. Note also that a job in queue corresponds to a customer in backlog.

s.2.5.7. We would prefer all service levels to be zero. If $h \gg b$ for some inventory system, the cost structure expresses that we dislike stock on hand. In other words, $h \gg b$ expresses that we want to produce according to MTO, that is, as a queueing system with zero on-hand inventory.

s.2.5.8. Note the differences between these schemes. In the (s, S) policy, the order level is fixed, but in the case of stochastic demand, the specific period in which an order is triggered is

not fixed. The order quantity (which is not the same as the order-up-to level) may vary from period to period. In the (Q,r) policy, the quantity of orders is always a multiple of Q, but the specific period in which a replenishment order is triggered varies in general. The (T,S) policy fixes the period but leaves the order quantity free. The (s,S) policy can achieve the lowest average cost, because it is the least constrained of the policies.

s.2.5.9. Since only the on-hand inventory I_k is available to meet demand, the rules for a loss system become

$$\begin{aligned} D'_k &= \min\{I_k, D_k\} && \text{accepted demand} \\ L_k &= D_k - D'_k && \text{lost demand} \\ P'_k &= P_k - D'_k, && \text{subtract the accepted demand} \\ I'_k &= I_k - D'_k, && \text{subtract the accepted demand.} \end{aligned}$$

Then use the standard rules to compute Q_{k+1}, and update P_{k+1} and I_{k+1} accordingly. The (s,S) and (Q,r) policy do not control demand, so these rules for $D'_k, \ldots$ apply then too.

For the cost, it is evident that there is no backlog. However, we should include a term for the average lost demand per period, for instance, proportional to $n^{-1}\sum_{k=1}^{n} L_k$.

s.2.5.10. The variable O_k equals 1 if an order is placed at the start of period k, and equals 0 otherwise. It is given by

$$O_k = 1 - \prod_{i=1}^{N} \mathbb{1}\left\{P'_{i,k-1} > s_i\right\}.$$

The order quantities and inventory positions then satisfy

$$Q_{i,k} = (S_i - P'_{i,k-1})O_k\,\mathbb{1}\left\{P'_{i,k-1} \le c_i\right\}, \qquad P_{i,k} = P'_{i,k-1} + Q_{i,k}.$$

With holding cost h_i and backlog cost b_i per unit of item i per period, the cost up to period n is

$$C_n = \sum_{k=1}^{n}\left(KO_k + \sum_{i=1}^{N}\left(K'\,\mathbb{1}\{Q_{i,k} > 0\} + h_i I^+_{i,k} + b_i I^-_{i,k}\right)\right).$$

Suppose we have set $S_i - s_i = 10$ for all items. Then c_i can take 10 values, hence the total number of can-order levels is 10^N. A supermarket has $N \gg 100$ item types (SKUs).

s.3.1.1. True. A simple variation: Claim: with our notation, the following mapping is correct:

$$A(t) : \mathbb{R} \to \mathbb{N}, \quad \text{time (real number) to number of jobs (integer).}$$

s.3.1.2. False. Suppose $A_3 = 10$ and $A_4 = 20$. Take $t = 15$. Then $\min\{k : A_k \ge 15\} = 4$, since $A_3 < t = 15 < A_4$. However, $\max\{k : A_k \le t\} = 3$. In fact, at time $t = 15$, 3 jobs arrived, not 4. Here is a simple variation: Claim: the number of arrivals $A(t)$ during $[0,t]$ can be defined as $\min\{k : A_k > t\}$?

s.3.1.3. False.

s.3.1.4. False. The first part is false, the second is true. $A(t)$ is the number of arrivals during $[0, t]$. Suppose that $A(t) = n$. This nth 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 nth job. Thus, the number of arrivals up to time A_n, i.e., $A(A_n)$, must be n.

s.3.1.5. False. $J(A_{A(t)})$ is nonsense. The correct expression is $[J_{A(t)} - (t - A_{A(t)})]^+ = [D_{A(t)} - t]^+$ because $A_{A(t)} + J_{A(t)} = D_{A(t)}$.

s.3.1.6. For $k \geq 2$, the only jobs that can be present immediately before A_k are jobs $1, \ldots, k-1$. Under FIFO service, job $k-1$ leaves no earlier than any of these jobs. Thus, the system is nonempty immediately before A_k if and only if job $k-1$ has not departed before A_k, that is, if and only if $D_{k-1} \geq A_k$. Equality is included: if $D_{k-1} = A_k$, job $k-1$ is still present in the left limit, although job k need not wait. For $k = 1$, the system is initially empty and $D_0 = 0 < A_1$, so both sides are false.

s.3.1.7. Job k arrives at time $t = A_k$. Its sojourn time is J_k. If no further jobs arrive, the system will be empty at time $D_k = A_k + J_k$. Therefore, the work in the system at the arrival time $t = A_k$ is $J_k = D_k - t = D_k - A_k$.

s.3.1.8. $s_k = \operatorname{argmin}_{i \in \{1,\ldots,c\}} \{w_k(i) + S_k/\mu(i)\}$.

s.3.2.1. False. All three statements are true, so it is not the case that exactly one is false. The first two statements are evident. For the third, K is the capacity of the system, not the queue capacity. Hence, the queue cannot contain more than $K - c$ jobs.

s.3.2.2. False. Service times are iid and exponential.

s.3.2.3. False. The service for business customers follows a priority queue. Therefore, one of the $c + 1$ servers, as perceived by the economy customers, works differently.

s.3.2.4. True. In Kendall's notation, M stands for memoryless (Poisson/exponential), G for general, and 1 for a single server.

s.3.2.5. False. The c denotes the number of servers. The K denotes the system capacity, i.e., the maximum number of jobs allowed in the system.

s.3.3.1. False. The initialization D_0 should not be included. The mean departure time is

$$\frac{\text{SUM}([D_1, \ldots, D_n])}{n}.$$

s.3.3.2. True.

s.3.3.3. False. The call to MAXIMUMPMF incorrectly treats A_k and D_{k-1} as independent.

s.3.3.4. False. It selects the server with the smallest waiting time. When service rates differ, the smallest sojourn time is obtained by selecting

$$\operatorname{argmin}_{j \in \{1,\ldots,c\}} \{w(j) + s_k/\mu(j)\}.$$

s.3.3.5. False. This recursion is only valid for the $G/G/1$ queue with a single server. For $c > 1$ servers, we must track the workload at each server and assign the job to the server that becomes free earliest.

s.4.1.1. False. In any reasonable queueing system this limit is ∞.

s.4.1.2. False.

s.4.1.3. False. δ differs from μ.

s.4.1.4. False. Here, λ counts all arriving jobs, including rejected jobs, whereas δ counts departures after service. Since $\lambda > \mu$ and $\delta \le \mu$, we have $\lambda > \delta$, so the system is not rate-stable according to our definition. Rejection can keep the number of jobs in the system bounded, but that does not make the offered arrival rate equal to the departure rate. If we instead count only admitted arrivals, their rate can equal δ; that uses a different arrival process.

s.4.1.5. False. The load can exceed 1; it is the utilization that is at most 1. When $\rho > 1$ the system is not rate-stable.

s.4.1.6. If $A(t) = \lfloor 3t^2 \rfloor$, then $A(t)/t \sim 3t$. This does not converge to a finite limit.

Another example, let the arrival rate $\lambda(t)$ be given as follows:

$$\lambda(t) = \begin{cases} 1 & \text{if } 2^{2k} \le t < 2^{2k+1} \\ 0 & \text{if } 2^{2k+1} \le t < 2^{2(k+1)}, \end{cases}$$

for $k = 0, 1, 2, \ldots$, and set $\lambda(t) = 0$ for $0 \le t < 1$. Let $A(t) = \left\lfloor \int_0^t \lambda(s)\,\mathrm{d}s \right\rfloor$. Then $A(t)$ is a counting process, but $A(t)/t$ tends to $2/3$ along the ends of the intervals with $\lambda(t) = 1$, and to $1/3$ along the ends of the intervals with $\lambda(t) = 0$. Hence, $A(t)/t$ fails to converge. Of course, these examples are quite pathological, and are not representative of 'real life cases'. (Though this is somewhat vague: What is actually a real-life case?)

s.4.2.1. False.

$$\frac{1}{t}\sum_{k=1}^{A(t)} \mathbb{1}\{W_k \le x\} = \frac{A(t)}{t}\frac{1}{A(t)}\sum_{k=1}^{A(t)} \mathbb{1}\{W_k \le x\} \to \lambda\,\mathsf{P}\{W \le x\}.$$

Moreover, the final w in the claim should be x.

s.4.2.2. True.

s.4.2.3. False. These are equal only under special conditions, such as when arrivals form a Poisson process (PASTA). As a counterexample, with constant inter-arrival time $X = 10$ and service time $S = 9$, arrivals always see an empty system, but the time-average number in the system is $9/10$.

s.4.2.4. True. This is the time-average definition of $\mathsf{P}\{L \ge n\}$.

s.4.2.5. True. This is a sample-average over arrivals, so it reflects the customer perspective.

s.4.2.6. Take $L(0) = 0$, $X_k = 10$ and $S_k = 10 - \epsilon$ for some tiny $\epsilon > 0$. Then $L(t) = 1$ almost always. Indeed, $\lim_{t\to\infty} t^{-1}\int_0^t L(s)\,\mathrm{d}s = 1 - \epsilon/10$. However, $L(A_k-) = 0$ for all k.

s.4.2.7.

$$\mathsf{E}[L] = \lim_{t\to\infty} \frac{1}{t}\int_0^t L(s)\,\mathrm{d}s \neq \lim_{t\to\infty} \frac{L(t)}{t}.$$

If $L(t) = 1$ for all t, $\mathsf{E}[L] = 1$, yet $L(t)/t \to 0$.

s.4.3.1. True. The cost increase per failure is c. Thus, this c takes the role of Z in the renewal reward theorem. The failure rate is $\alpha = 1/\mathsf{E}[X]$, where X is the time between two failures. Therefore, $R = c\alpha = c/\mathsf{E}[X]$.

s.4.3.2. True. See [4.3.7].

s.4.3.3. False. In an (s, S) policy, s denotes the reorder point and S the order-up-to level.

s.4.3.4. False. The quantity α is the rate at which the epochs $\{T_k\}$ occur, which can be any renewal process, not necessarily the arrival process.

s.4.3.5. As $k/T_k \to \delta$, it follows from the renewal reward theorem that $R(t)/t \to \delta\,\mathsf{E}[S]$. By rate stability, $\delta = \lambda$, so that $\rho = \lambda\,\mathsf{E}[S] = \delta\,\mathsf{E}[S] = R$, from which the claim follows.

s.4.3.6. For the LHS, as $A(t) \to \infty$ as $t \to \infty$,

$$\lim_{t\to\infty} \frac{1}{t}\sum_{k=1}^{A(t)} S_k = \lim_{t\to\infty} \frac{A(t)}{t}\frac{1}{A(t)}\sum_{k=1}^{A(t)} S_k = \lim_{t\to\infty} \frac{A(t)}{t}\cdot \lim_{t\to\infty} \frac{1}{A(t)}\sum_{k=1}^{A(t)} S_k = \lambda\,\mathsf{E}[S].$$

Apply similar limits to the RHS. The limit of the middle term gives the fraction of time the server is busy.

s.4.3.7. Take $R(t) = \int_0^t \mathbb{1}\{L_s(s) = 1\}\,\mathrm{d}s$, so that $R(t)/t \to \rho$, i.e., the fraction of time there is someone on the server. (We could also take $R(t) = \int_0^t \mathbb{1}\{L(s) > 0\}\,\mathrm{d}s$, because $L_s(t) = 1 \iff L(t) > 0$.) Writing U_k for the kth busy time and I_k for the kth idle time, we sample $R(\cdot)$ at moments $T_k = \sum_{j=1}^k (I_j + U_j)$, that is, we check $R(\cdot)$ at the end of each busy period. Then, by the definition of $R(\cdot)$, its increase between two sample moments is $Z_k := R(T_k) - R(T_{k-1}) = U_k$. The rate at which we sample the system is $\alpha \approx N(t)/t \approx k/T_k \to 1/(\mathsf{E}[I] + \mathsf{E}[U])$ as $k \to \infty$ (by the strong law). And thus, with the renewal reward theorem $\rho = R = \alpha Z = \mathsf{E}[U]/(\mathsf{E}[I] + \mathsf{E}[U])$.

s.4.3.8. Suppose $0 \le s < S$. The average holding cost for the rectangle with base T and height s is hTs, i.e., h times the area of the rectangle. The cost of the triangular area with base T and height $S - s$ is $hT(S-s)/2$. Thus, the total cost of a cycle is $K + hT(S+s)/2$. Dividing by T gives the time-average cost.

The other case follows by the same reasoning.

s.4.4.1. False.

s.4.4.2. False. First, W_k should be J_k. Second, even after replacing W_k by J_k, it is only valid at times T when the system is empty.

s.4.4.3. True, [4.4.6].

s.4.4.4. False.

s.4.4.5. False. Little's law alone does not imply that $\lambda \mathsf{E}[J]$ decreases when λ decreases, because it supplies no bound on the accompanying change in $\mathsf{E}[J]$. For example, with unit service times, arrivals every two time units give $\lambda = 1/2$ and $\mathsf{E}[L] = 1/2$, whereas pairs arriving at offsets 0 and 0.1 in each five-unit cycle give $\lambda = 2/5$ and $\mathsf{E}[L] = 0.58$.

s.4.4.6. Since the queue is stable, the jobs leave at rate λ, thus each job that enters the box leaves it. The average time a job spends in the box is $\mathsf{E}[S]$. By Little's law, the average number in the box is $\mathsf{E}[L_s] = \lambda \mathsf{E}[S]$.

s.4.4.7. Let us make a very simple model to clarify the problem. We start with an initial population of A people and, for this heuristic, assume that everyone initially present or subsequently added remains in the population for exactly 80 more years. In year $k = 1, 2, \ldots$, we add $A(1.01)^k$ people, so the annual increase through these additions grows by 1% per year. After 79 years, the population size is $A\sum_{k=0}^{79}(1.01)^k$. If we assume (as we do when we apply Little's law) that each year contains the same amount of people, then the number of people per year equals

$$\frac{A}{80}\sum_{k=0}^{79}(1.01)^k = \frac{A}{80}\frac{(1.01)^{80}-1}{1.01-1} \approx 1.52A.$$

However, after 80 years, only A people die, not $1.52A$. Here, $4000/1.52 \approx 2630$. This illustrates how applying the constant-population calculation to a growing population can overestimate the current death rate.

In conclusion, Little's law must be applied with caution; during periods when the population increases or decreases on average, it does not hold.

s.4.5.1. False. In the $G/M/1$ queue, jobs do not arrive as a Poisson process.

s.4.5.2. True.

s.4.5.3. True.

s.4.5.4. False. First we should use $L_S(A_k-)$ instead of $L_S(A_k)$. Second, by PASTA α must be the load, which is ρ for the $M/G/1$ queue; $1-\rho$ is the fraction of time the server is idle.

s.4.5.5. False. PASTA requires that arrivals form a Poisson process. For a general $G/G/1$ queue, the arrival process need not be Poisson, so sample and time averages can differ.

s.4.5.6. If $\lambda > \delta$, then $L(t) \to \infty$. But then there must be a last time s, say, that $L(s) = n+1$, and $L(t) > n+1$ for all $t > s$. Hence, after time s there will never be a job that will see the system with n jobs. Thus $A(n,t) = A(n,s)$ for all $t > s$ (recall that $A(n,t)$ can only change at time t, for example when a job arriving at t sees n in the system. But since $L(t) > n+1$ for $t > s$, this will never happen after s.) As $A(n,t)$ remains finite, $\lim_{t\to\infty} A(n,t)/t = 0$.

s.4.5.7. The assumptions lead us to conclude that $\lambda > \delta$. As a consequence, the queue length must increase in the long run (jobs come in faster than they leave). Hence, $A(n,t)/t \to 0$ for all n, and $D(n,t)/t \to 0$. Consequently, $\pi(n) = \delta(n) = 0$, which is the only sensible reconciliation with (4.6.7).

s.4.5.8. By the memoryless property of the exponentially distributed service times of the $M/M/1$ queue, the remaining service time of a job in service, if any, is $\text{Exp}(\mu)$. Therefore, at an arrival moment, all jobs in the system (whether in service or not) have the same expected duration. Hence, the expected time to spend in queue equals the expected number of jobs in the system times the expected service time of each job, i.e., $\mathsf{E}[W] = \mathsf{E}[L]\,\mathsf{E}[S]$. Note that we use PASTA to see that the expected number of jobs in the system at an arrival is $\mathsf{E}[L]$. For the $M/G/1$ queue, the remaining service time of a job in service (if any) does not have the same distribution as the jobs in the queue. Hence, for the $M/G/1$ queue, the *expected time in queue is not* $\mathsf{E}[L]\,\mathsf{E}[S]$.

s.4.6.1. False: rightward transitions are caused by arriving rider parties and must have rate $\mu = 12$, while leftward transitions are caused by taxis and must have rate $\lambda = 15$.

s.4.6.2. True.

s.4.6.3. False. The figure sketches the $M/M^2/1/3$ queue.

s.4.6.4. False. Since service times are constant in the $M/D/1$ queue and thus not memoryless, $\mu \neq \mu(n+1)$ for the $M/D/1$ queue.

s.4.6.5. True. When jobs arrive and depart as single units, the number of up-crossings and down-crossings of any level n can differ by at most one, leading to this equation in the long run.

s.4.6.6. 1. $A_k = 2(k-1)$, $k = 1,2,\ldots$, as jobs arrive at $t = 0,2,4,\ldots$ Thus, $A(t) = 1 + \lfloor t/2 \rfloor$ for $t \geq 0$.

(BTW, $A(t)/t \to 1/2$ as $t \to \infty$.) We also know that $L(s) = 1$ if $s \in [2k, 2k+1)$ and $L(s) = 0$ for $s \in [2k-1, 2k)$ for $k = 0,1,2,\ldots$. Thus, $L(A_k-) = L(2k-) = 0$. Therefore, $A(0,t) = A(t)$ for $t \geq 0$, and $A(n,t) = 0$ for $n \geq 1$.

1. All arrivals see an empty system. Hence, $A(0,t)/A(t) \approx (t/2)/(t/2) = 1$, and $A(n,t) = 0$ for $n > 0$. Thus, $\pi(0) = \lim_{t\to\infty} A(0,t)/A(t) = 1$ and $\pi(n) = 0$ for $n > 0$. Recall from the other exercises that $p(0) = 1/2$. Hence, statistics as obtained via time-averages are not necessarily the same as statistics obtained at arrival moments (or any other point process).

2. $\lambda = \lim_{t\to\infty} A(t)/t = 1/2$. $\lambda(0) = 1$, $p(0) = 1/2$, and $\pi(0) = 1$. Hence,

$$\lambda\pi(0) = \lambda(0)p(0) \implies \frac{1}{2} \times 1 = 1 \times \frac{1}{2}.$$

For $n > 0$, the argument is straightforward: everything is 0.

1. Observe that the system never contains more than 1 job. Hence, $Y(n,t) = 0$ for all $n \geq 2$. Then we see that $Y(1,t) = \int_0^t \mathbb{1}\{L(s) = 1\}\,\mathrm{d}s$. Now observe that for our queueing system $L(s) = 1$ for $s \in [0,1)$, $L(s) = 0$ for $s \in [1,2)$, $L(s) = 1$ for $s \in [2,3)$, and so on. Thus, when $t < 1$, $Y(1,t) = \int_0^t \mathbb{1}\{L(s) = 1\}\,\mathrm{d}s = \int_0^t 1\,\mathrm{d}s = t$. When $t \in [1,2)$,

$$L(t) = 0 \implies \mathbb{1}\{L(t) = 0\} \implies Y(1,t) \text{ does not change.}$$

Continuing to $[2,3)$ and so on gives

$$Y(1,t) = \begin{cases} t & t \in [0,1), \\ 1 & t \in [1,2), \\ 1+(t-2) & t \in [2,3), \\ 2+(t-4) & t \in [4,5), \end{cases}$$

and so on. With this, we see that the general formula must be

$$Y(1,t) = \begin{cases} \lfloor t/2 \rfloor + t - \lfloor t \rfloor, & \text{if } \lfloor t \rfloor \text{ is even}, \\ 1 + \lfloor t/2 \rfloor, & \text{if } \lfloor t \rfloor \text{ is odd}. \end{cases}$$

Since $Y(n,t) = 0$ for all $n \geq 2$, $L(s) = 1$ or $L(s) = 0$ for all s, therefore,

$$Y(0,t) = t - Y(1,t).$$

5.

$$\begin{aligned} \lambda(0) &\approx \frac{A(0,t)}{Y(0,t)} \approx \frac{t/2}{t/2} = 1, \\ \lambda(1) &\approx \frac{A(1,t)}{Y(1,t)} \approx \frac{0}{t/2} = 0, \\ p(0) &\approx \frac{Y(0,t)}{t} \approx \frac{t/2}{t} = \frac{1}{2}, \\ p(1) &\approx \frac{Y(1,t)}{t} \approx \frac{t/2}{t} = \frac{1}{2}. \end{aligned}$$

For $n \geq 2$, $p(n) = 0$; the ratios defining rates in these unvisited states are $0/0$, so we assign those unused rates the value zero by convention.

6. $D(0,t) = \sum_{k=1}^{\infty} \mathbb{1}\{D_k \leq t, L(D_k) = 0\}$.

From the graph of $\{L(s)\}$ we see that all jobs leave an empty system behind. Thus, $D(0,t) \approx t/2$, and $D(n,t) = 0$ for $n \geq 1$. With this, $D(0,t)/Y(1,t) \sim (t/2)/(t/2) = 1$, and so,

$$\mu(1) = \lim_{t\to\infty} \frac{D(0,t)}{Y(1,t)} = 1,$$

and $\mu(n) = 0$ for $n \geq 2$.

7. $\lambda(0)p(0) = 1 \cdot 1/2 = 1/2$, $\lambda(n)p(n) = 0$ for $n > 1$, as $\lambda(n) = 0$ for $n > 0$.

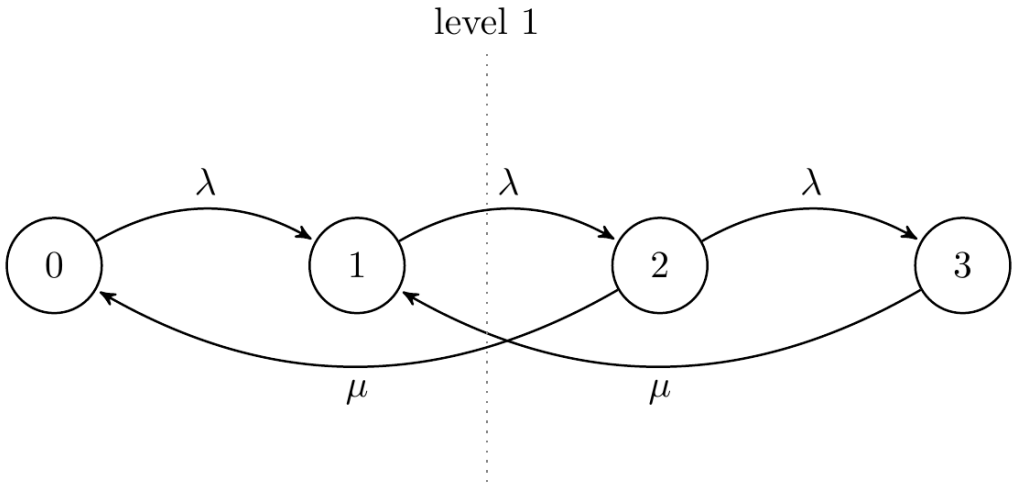
Next, $\mu(1) = 1$, hence $\mu(1)p(1) = 1 \cdot 1/2 = 1/2$. Moreover, $\mu(n) = 0$ for $n \geq 2$. Clearly, for all n we have $\lambda(n)p(n) = \mu(n+1)p(n+1)$.

s.4.6.7. Noting that $L(s) = \sum_{n=0}^{\infty} n\, \mathbb{1}\{L(s) = n\}$, we see that

$$\frac{1}{t}\int_0^t L(s)\,\mathrm{d}s = \frac{1}{t}\int_0^t \left(\sum_{n=0}^{\infty} n\, \mathbb{1}\{L(s) = n\}\right) \mathrm{d}s = \sum_{n=0}^{\infty} \frac{n}{t}\int_0^t \mathbb{1}\{L(s) = n\}\,\mathrm{d}s = \sum_{n=0}^{\infty} n\frac{Y(n,t)}{t}$$

Taking the limit $t \to \infty$ and applying (4.5.1a) (and reversing the limit and the summation, which we assume is okay here) we obtain that $\mathsf{E}[L] = \sum_n np(n)$.

s.4.6.8. Use the references of the hint to see that $\rho = \sum_{i=1}^{\infty} p(i) = \sum_{i=1}^{\infty} \pi(i) = \sum_{i=1}^{\infty} \delta(i)$.



s.4.6.9.

With level-crossing:

$$\begin{aligned}
&\lambda p(0) = \mu p(2), \quad \text{the level between 0 and 1,}\\
&\lambda p(1) = \mu p(2) + \mu p(3), \quad \text{see level 1,}\\
&\lambda p(2) = \mu p(3), \quad \text{the level between 2 and 3.}
\end{aligned}$$

Solving in terms of $p(0)$ gives $p(2) = p(0)\lambda/\mu$ and $p(3) = p(0)\lambda^2/\mu^2$. Substituting these into the middle level-crossing equation gives $p(1) = p(0)(\mu+\lambda)/\mu$. For the final answer, use the normalization constraint $p(0) + p(1) + p(2) + p(3) = 1$. (As this is simple, we skip it.)

The figure shows the allowed transitions and their individual rates; the inverse mean time spent in a state is the sum of its outgoing rates. It does not imply that, once a state is entered, the state will be left immediately.

s.4.6.10.

$$\lim_{t\to\infty} \frac{I(n,t)}{t} = \lim_{t\to\infty} \frac{A(n-1,t)}{t} + \lim_{t\to\infty} \frac{D(n,t)}{t} = \lambda(n-1)p(n-1) + \mu(n+1)p(n+1)$$

and

$$\lim_{t\to\infty} \frac{O(n,t)}{t} = \lim_{t\to\infty} \frac{A(n,t)}{t} + \lim_{t\to\infty} \frac{D(n-1,t)}{t} = \lambda(n)p(n) + \mu(n)p(n).$$

s.5.1.1. False. In this queueing model, the arrival rate $\lambda(n)$ can depend on the number of jobs in the queue. If this is the case, the inter-arrival times can still be independent but not identically distributed.

s.5.1.2. True. See [5.1.8].

s.5.1.3. False. For the $M/M/c/c$ queue, $\lambda^+(n) = \lambda\,\mathbb{1}\{n < c\}$, so $\lambda^+(n) = 0$ when $n \geq c$.

s.5.1.4. True.

s.5.1.5. False. This formula holds only for $n \leq c$. For $n > c$, all c servers are busy and the formula changes to $p(n) = p(0)\frac{\lambda^n}{c!\,c^{n-c}\mu^n}$.

s.5.1.6. The fraction of time the system contains n jobs is $\pi(n)$, by PASTA. When the system contains $n>0$ jobs, the number in the queue is one less, namely, $n-1$.

$$\mathsf{E}[Q] = \sum_{n=1}^{\infty}(n-1)\pi(n) = \sum_{n=1}^{\infty} n\pi(n) - \sum_{n=1}^{\infty}\pi(n) = \mathsf{E}[L] - (1-\pi(0)) = \mathsf{E}[L] - \rho.$$

s.5.1.7. Take $\lambda(n) = (N-n)\lambda$ and $\mu(n) = n\mu$. Then

$$\begin{aligned} p(n+1) &= \frac{\lambda(n)}{\mu(n+1)}p(n) = \frac{(N-n)\lambda}{(n+1)\mu}p(n) = \frac{(N-n)(N-(n-1))}{(n+1)n}\frac{\lambda^2}{\mu^2}p(n-1) \\ &= \frac{N!}{(N-(n+1))!}\frac{1}{(n+1)!}\rho^{n+1}p(0) = \binom{N}{n+1}\rho^{n+1}p(0). \end{aligned}$$

Therefore, $G = \sum_{k=0}^{N}\rho^k\binom{N}{k}$. Now, using that $(a+b)^m = \sum_{k=0}^{m}\binom{m}{k}a^k b^{m-k}$, we see that $(1+\rho)^N = \sum_{k=0}^{N}\binom{N}{k}\rho^k 1^{N-k} = G$. So,

$$p(n+1) = \binom{N}{n+1}\frac{\rho^{n+1}}{(1+\rho)^N}.$$

When $\rho = 0$, $p(0) = 1$ and $p(n) = 0$ for all $n \geq 1$. When $\rho > 0$, then, for fixed n, $\binom{N}{n} \sim N^n/n!$, while $(1+\rho)^N$ grows exponentially. Hence, $p(n) = \binom{N}{n}\rho^n/(1+\rho)^N \to 0$ as $N \to \infty$ for every fixed n.

s.5.1.8.

$$M_L(s) = \mathsf{E}\left[e^{sL}\right] = \sum_{n=0}^{\infty} e^{sn}p(n) = (1-\rho)\sum_{n} e^{sn}\rho^n = \frac{1-\rho}{1-e^s\rho},$$

where we assume that s is such that $e^s\rho < 1$. Then,

$$\begin{aligned} M_L'(s) = (1-\rho)\frac{1}{(1-e^s\rho)^2}e^s\rho \implies \mathsf{E}[L] = M_L'(0) &= \frac{\rho}{1-\rho}, \\ \mathsf{E}\left[L^2\right] = M''(0) &= \frac{2\rho^2}{(1-\rho)^2} + \frac{\rho}{1-\rho} = \rho\frac{1+\rho}{(1-\rho)^2}, \\ \mathsf{V}[L] = \mathsf{E}\left[L^2\right] - (\mathsf{E}[L])^2 &= \frac{\rho(1+\rho)}{(1-\rho)^2} - \frac{\rho^2}{(1-\rho)^2} = \frac{\rho}{(1-\rho)^2}, \\ \mathsf{P}\{L \geq n\} = \sum_{k=n}^{\infty} p(k) = \sum_{k=n}^{\infty} p(0)\rho^k &= (1-\rho)\sum_{k=n}^{\infty}\rho^k = (1-\rho)\rho^n\sum_{k=0}^{\infty}\rho^k = (1-\rho)\rho^n\frac{1}{1-\rho} = \rho^n. \end{aligned}$$

s.5.1.9. For $n \geq 1$,

$$\begin{aligned} p(n) &= \frac{\lambda(n-1)}{\mu(n)}p(n-1) = \frac{\lambda}{\min\{c,n\}\mu}p(n-1) = \frac{1}{\min\{c,n\}}(c\rho)p(n-1) \\ &= \frac{1}{\min\{c,n\}\min\{c,n-1\}}(c\rho)^2 p(n-2) \\ &= \frac{1}{\prod_{k=1}^{n}\min\{c,k\}}(c\rho)^n p(0). \end{aligned}$$

To obtain the normalization constant $G = 1/p(0)$,

$$\begin{aligned}
1 &= \sum_{n=0}^{\infty} p(n) = p(0) + \sum_{n=1}^{c-1} p(n) + \sum_{n=c}^{\infty} p(n) \\
&= p(0) + p(0) \sum_{n=1}^{c-1} \frac{(c\rho)^n}{n!} + p(0) \sum_{n=c}^{\infty} \frac{c^c}{c!} \rho^n \\
&= p(0) \sum_{n=0}^{c-1} \frac{(c\rho)^n}{n!} + p(0) \sum_{n=c}^{\infty} \frac{(c\rho)^c}{c!} \rho^{n-c} \\
&= p(0) \sum_{n=0}^{c-1} \frac{(c\rho)^n}{n!} + p(0) \frac{(c\rho)^c}{c!} \sum_{n=0}^{\infty} \rho^n \\
&= p(0) \sum_{n=0}^{c-1} \frac{(c\rho)^n}{n!} + p(0) \frac{(c\rho)^c}{c!(1-\rho)}.
\end{aligned}$$

For $n \geq c$, observe that $p(n) = (c\rho)^n/(c^{n-c}c!) = c^c\rho^n/c!$, so that

$$\begin{aligned}
\mathsf{E}[Q] &= \sum_{n=c}^{\infty} (n-c)p(n) = \sum_{n=c}^{\infty} (n-c) \frac{c^c}{c!} \rho^n p(0) \\
&= \frac{c^c\rho^c}{Gc!} \sum_{n=c}^{\infty} (n-c)\rho^{n-c} = \frac{c^c\rho^c}{Gc!} \sum_{n=0}^{\infty} n\rho^n \\
&= \frac{c^c\rho^c}{Gc!} \frac{\rho}{(1-\rho)^2}.
\end{aligned}$$

The derivation of the expected number of jobs in service becomes easier if we pre-multiply the normalization constant G:

$$\begin{aligned}
G\mathsf{E}[L_s] &= G\left(\sum_{n=0}^{c} np(n) + \sum_{n=c+1}^{\infty} cp(n) \right) \\
&= \sum_{n=1}^{c} n \frac{(c\rho)^n}{n!} + \sum_{n=c+1}^{\infty} c \frac{c^c\rho^n}{c!} = \sum_{n=1}^{c} \frac{(c\rho)^n}{(n-1)!} + \frac{c^{c+1}}{c!} \sum_{n=c+1}^{\infty} \rho^n \\
&= \sum_{n=0}^{c-1} \frac{(c\rho)^{n+1}}{n!} + \frac{(c\rho)^{c+1}}{c!} \sum_{n=0}^{\infty} \rho^n = c\rho \left(\sum_{n=0}^{c-1} \frac{(c\rho)^n}{n!} + \frac{(c\rho)^c}{c!(1-\rho)} \right).
\end{aligned}$$

Observe that the RHS is precisely equal to ρcG, so that $\mathsf{E}[L_s] = c\rho$.

s.5.1.10. Take $c = 1$

$$\begin{aligned}
p(0) &= \frac{1}{G} \frac{(c\rho)^0}{0!} = \frac{1}{G}, \\
p(n) &= \frac{1}{G} \frac{c^c\rho^n}{c!} = \frac{1}{G} \frac{1^1\rho^n}{1!} = \frac{\rho^n}{G}, \quad n = 1, 2, \ldots \\
G &= \sum_{n=0}^{c-1} \frac{(c\rho)^n}{n!} + \frac{(c\rho)^c}{(1-\rho)c!} = \sum_{n=0}^{0} \frac{\rho^0}{0!} + \frac{\rho}{(1-\rho)} = 1 + \frac{\rho}{1-\rho} = \frac{1}{1-\rho}, \\
\mathsf{E}[L] &= \frac{(c\rho)^c}{c!G} \frac{\rho}{(1-\rho)^2} = \frac{\rho}{1/(1-\rho)} \frac{\rho}{(1-\rho)^2} = \frac{\rho^2}{1-\rho}, \\
\mathsf{E}[L] &= \sum_{n=0}^{c} np(n) + \sum_{n=c+1}^{\infty} cp(n) = p(1) + 1 \sum_{n=2}^{\infty} p(n) = 1 - p(0) = \rho.
\end{aligned}$$

Everything agrees with the formulas we derived earlier for the $M/M/1$ queue.

s.5.1.11. With the other results for the $M/M/1/K$ queue,

$$1-p(0) = 1 - \frac{1-\rho}{1-\rho^{K+1}} = \frac{1-\rho^{K+1}-1+\rho}{1-\rho^{K+1}} = \rho\frac{1-\rho^{K}}{1-\rho^{K+1}},$$
$$\rho(1-p(K)) = \rho\frac{1-\rho^{K+1}-\rho^{K}+\rho^{K+1}}{1-\rho^{K+1}} = \rho\frac{1-\rho^{K}}{1-\rho^{K+1}}.$$

s.5.1.12. To take the limit $K\to\infty$—not the limit $n\to\infty$—, write

$$G = \frac{1-\rho^{K+1}}{1-\rho} = \frac{1}{1-\rho} - \frac{\rho^{K+1}}{1-\rho}.$$

Since $\rho^{K+1}\to 0$ as $K\to\infty$ (recall, $\rho<1$), we get

$$G \to \frac{1}{1-\rho},$$

as $K\to\infty$. Hence, $p(n) = \rho^n/G \to \rho^n(1-\rho)$, and the latter are the steady-state probabilities of the $M/M/1$ queue. Finally, if the steady-state probabilities coincide, the performance measures derived from $p(n)$ coincide as well.

s.5.2.1. True.

s.5.2.2. False. This depends on the objective. The service time at a slow server is longer. However, when multiple servers are present, if one server breaks down, service can still continue with the other servers, while when a single fast server fails, everything stops.

s.5.2.3. True.

s.5.2.4. False. A balking arrival leaves the system at K, but a service completion moves it below K. The finite-state queue has a limiting probability $p(K)<1$, so $\mathsf{P}\{L(t)=K\}$ does not tend to 1. It is not rate stable in the sense of Section 4.1 because rejected arrivals are counted in the offered arrival rate but not in the departure rate.

s.5.2.5. False. Adding a second server reduces the waiting time, but not by exactly half. The reduction depends nonlinearly on ρ and is typically much more than a factor 2.

s.5.2.6. As people arrive without making appointments, it is reasonable to say that the inter-arrival times are memoryless. As for the services, the employee deals with many different questions, some are short, some are long. So, we also model service times as exponential.

With Little's law:

$$\mathsf{E}[Q] = \lambda\,\mathsf{E}[W] = 5\cdot 6 = 30.$$

Moreover, $\mathsf{E}[S] = \rho/\lambda$, so the waiting-time formula gives

$$6 = \frac{\rho}{1-\rho}\frac{\rho}{5}.$$

Solving this quadratic equation yields

$$\rho \approx 0.9687, \qquad \mathsf{E}[S] \approx 0.1937 \text{ minutes}.$$

Therefore,

$$\mu = 1/\mathsf{E}[S] \approx 5.161 \text{ per minute}, \qquad \mathsf{E}[J] = \mathsf{E}[W] + \mathsf{E}[S] \approx 6.1937 \text{ minutes}.$$

Thus, $\mathsf{E}[L] = \lambda \mathsf{E}[J] \approx 30.9687$.

With s seats, all waiting customers can sit precisely when $L \le s+1$: one customer can be at the employee's desk. Since $\mathsf{P}\{L \le k\} = 1 - \rho^{k+1}$, we need

$$1 - \rho^{s+2} \ge 0.9.$$

The smallest solution is $s = 71$. This very large number confirms that the municipality should hire another employee rather than provide so many seats.

The average queue is huge: hire an extra employee.

s.5.2.7. This is an $M/M/1/3$ queue: there is room for 1 customer in service and two in queue.

Write K for the total system capacity and $\rho = \lambda/\mu = 6/5$. The level-crossing equations give

$$p(n) = \frac{\rho^n}{\sum_{j=0}^{K} \rho^j}, \qquad 0 \le n \le K.$$

With two waiting chairs, $K = 3$. Therefore,

$$(p(0), \dots, p(3)) \approx (0.186, 0.224, 0.268, 0.322),$$

and

$$\mathsf{E}[Q] = \sum_{n=0}^{3} (n-1)^+ p(n) \approx 0.912.$$

Jobs are accepted, and hence served, at rate

$$\bar{\lambda} = \lambda(1 - p(3)) \approx 4.069 \text{ per hour}.$$

With three extra waiting chairs, $K = 6$. The same calculation gives

$$\mathsf{E}[Q] \approx 2.787, \qquad \bar{\lambda} = \lambda(1 - p(6)) \approx 4.613 \text{ per hour}.$$

Thus, the extra chairs increase the throughput only modestly and create a longer queue. Since the barber is overloaded, an extra server is more useful.

s.5.2.8. The state-dependent service rates are

$$\mu(0) = 0, \qquad \mu(1) = 30, \qquad \mu(n) = 60 \quad \text{for } n \ge 2.$$

Starting with $q(0) = 1$, the level-crossing recursion

$$q(n+1) = q(n) \frac{\lambda(n)}{\mu(n+1)}$$

and normalization give

$$(p(0), \dots, p(5)) \approx (0.7072, 0.2357, 0.0393, 0.0131, 0.0044, 0.0004).$$

Consequently,

$$\bar{\lambda} = \sum_{n=0}^{4} \lambda(n)p(n) \approx 10.4984,$$

$$\mathsf{E}[L] = \sum_{n=0}^{5} np(n) \approx 0.3729, \qquad \mathsf{E}[Q] = \sum_{n=0}^{5} (n-2)^+ p(n) \approx 0.0229.$$

Little's law then yields

$$\mathsf{E}[J] = \frac{\mathsf{E}[L]}{\bar{\lambda}} \approx 0.0355 \text{ hours} \approx 2.13 \text{ minutes},$$

and

$$\mathsf{E}[W] = \frac{\mathsf{E}[Q]}{\bar{\lambda}} \approx 0.00218 \text{ hours} \approx 0.13 \text{ minutes}.$$

s.5.2.9. The rv L behaves like the number of jobs in an $M/M/1$ queue. If we have L, then with the two properties $TG = 0$ and $3 = T + L - G$, we can retrieve (uniquely at all moments in time) the number of patties T on stock and the number of people waiting G. In inventory terms, G represents the number of *backlogged* patties.

With this mapping, the expected number of patties on stock is $\mathsf{E}[T] = \sum_{l=0}^{3}(3-l)p(l)$. Here $\rho = \lambda/\mu = 0.8$ and $p(l) = (1-\rho)\rho^l$, so

$$\mathsf{E}[T] = \sum_{l=0}^{2}(3-l)(1-\rho)\rho^l = 1.048.$$

For the expected number of people waiting, use that $3 = T + L - G$ always. Taking expectations gives

$$\mathsf{E}[G] = \mathsf{E}[T] + \mathsf{E}[L] - 3 = 1.048 + \frac{0.8}{1-0.8} - 3 = 2.048.$$

In steady state, patties are produced and customers arrive at the same average rate λ. Little's law therefore gives

$$\frac{\mathsf{E}[T]}{\lambda} = \frac{1.048}{12} \text{ hour} \approx 5.24 \text{ minutes}$$

for the average time a patty waits before it ends up in a bun, and

$$\frac{\mathsf{E}[G]}{\lambda} = \frac{2.048}{12} \text{ hour} \approx 10.24 \text{ minutes}$$

for the average time a customer waits for a hamburger.

What would be the impact of allowing 4 patties maximally?

s.5.2.10. The *invariant* $3 = T + L - G$ becomes $4 = T + L - G$.

The queue itself does not change, so $\rho = 0.8$ and $p(l) = (1-\rho)\rho^l$ still apply. Hence,

$$\mathsf{E}[T] = \sum_{l=0}^{3}(4-l)(1-\rho)\rho^l = 1.6384, \qquad \mathsf{E}[G] = \mathsf{E}[T] + \mathsf{E}[L] - 4 = 1.6384.$$

Compared with a maximum of three patties, more patties wait in stock, while fewer customers wait for a hamburger. In particular, the average customer waiting time decreases to

$$\frac{\mathsf{E}[G]}{\lambda} = \frac{1.6384}{12} \text{ hour} \approx 8.19 \text{ minutes}.$$

s.5.2.11. The *invariant* $3 = T + L - G$ becomes $0 = T + L - G$. This implies that $T = 0$ and $L = G$. Hence $\mathsf{E}[G] = \mathsf{E}[L] = \rho/(1-\rho) = 4$, and the expected customer waiting time is

$$\frac{\mathsf{E}[G]}{\lambda} = \frac{4}{12} \text{ hour} = 20 \text{ minutes}.$$

s.5.3.1. True. We have for the batches that $\lambda_B \mathsf{E}\left[W^B\right] = \mathsf{E}\left[Q^B\right]$. We also find that $\lambda_B = \lambda/\mathsf{E}[B]$, and $\mathsf{E}\left[W^B\right] = \mathsf{E}[W]$.

s.5.3.2. True. $\mathsf{V}[B] = 0 \implies C_B^2 = 0 \implies$ the first term on the RHS does not change when ρ remains constant. The second term becomes smaller.

s.5.3.3. False. Jobs come in batches; hence, multiple jobs can arrive at the same time, implying that the inter-arrival distribution is no longer exponential.

s.5.3.4. False. The values of m should include n; the sum should run from $m = 0$ through $m = n$.

s.5.3.5. False. When B is deterministic, $C_B^2 = 0$, but the waiting time formula still contains the factor $\mathsf{E}[B]$ from the batch structure. Only when $B \equiv 1$ does the formula reduce to the $M/M/1$ result.

s.5.3.6. Start with the simple case. $B \equiv 2 \implies \mathsf{V}[B] = 0 \implies C_B^2 = 0$, $\rho = \lambda_B \mathsf{E}[B]\,\mathsf{E}[S] = 1 \cdot 2 \cdot 25/60 = 5/6$. Hence,

$$\mathsf{E}[L] = \frac{1}{2}\frac{5/6}{1/6}2 + \frac{1}{2}\frac{5/6}{1/6} = 5 + \frac{5}{2}.$$

Now we have the other case. $\mathsf{E}\left[B^2\right] = (1+4+9)/3 = 14/3$. Hence, $\mathsf{V}[B] = 14/3 - 4 = 2/3$. Hence, $C_B^2 = \frac{1}{6}$. And thus,

$$\mathsf{E}[L] = \frac{1+1/6}{2}\frac{5/6}{1/6}2 + \frac{1}{2}\frac{5/6}{1/6} = \frac{7}{6}5 + \frac{5}{2}.$$

The ratio between $\mathsf{E}[L]$ is 10/9: the average waiting time can be reduced by about 10\

s.5.3.7. For a geometric rv,

$$\mathsf{E}[B] = \frac{1}{p}, \qquad \mathsf{V}[B] = \frac{1}{p^2} - \frac{1}{p}, \qquad C_B^2 = \frac{\mathsf{V}[B]}{(\mathsf{E}[B])^2} = p^2\left(\frac{1}{p^2} - \frac{1}{p}\right) = 1 - p,$$
$$(1 + C_B^2)/2 = 1 - p/2,$$

With this, fill in the formula for $\mathsf{E}[W]$ to get

$$\mathsf{E}[L] = \left(1 - \frac{p}{2}\right)\frac{\rho}{1-\rho}\frac{1}{p} + \frac{1}{2}\frac{\rho}{1-\rho} = \frac{\rho}{1-\rho}\frac{1}{p}.$$

For the $M/M/1$ queue, $\mathsf{E}[B] = 1 \implies p = 1 \implies \mathsf{E}[L] = \rho/(1-\rho)$.

If you cannot find the expectation and variance of a geometric rv, then here is the derivation.

$$\begin{aligned}
M_B(s) &= \mathsf{E}\left[e^{sB}\right] = \sum_{k=0}^{\infty} e^{sk}\,\mathsf{P}\{B=k\} \\
&= \sum_{k=0}^{\infty} e^{sk} p q^{k-1} = \frac{p}{q}\sum_{k=0}^{\infty}(qe^s)^k = \frac{p}{q}\frac{1}{1-qe^s}, \\
\mathsf{E}[B] &= M_B'(0) = \frac{p}{q}\frac{q}{(1-qe^s)^2}\bigg|_{s=0} = \frac{p}{(1-q)^2} = \frac{1}{p}, \\
\mathsf{E}\left[B^2\right] &= M_B''(0) = \frac{2}{p^2} - \frac{1}{p}, \\
\mathsf{V}[B] &= \mathsf{E}\left[B^2\right] - (\mathsf{E}[B])^2 = \frac{2}{p^2} - \frac{1}{p} - \frac{1}{p^2} = \frac{1}{p^2} - \frac{1}{p},
\end{aligned}$$

s.5.3.8. The complete acceptance policy is quite simple. Any batch is accepted when the system contains fewer than K jobs. When the system contains K or more jobs, arriving batches are rejected.

$$\mu\pi(n+1) = \begin{cases} \lambda_B \sum_{m=0}^{n} \pi(m) G(n-m), & \text{for } n < K, \\ \lambda_B \sum_{m=0}^{K-1} \pi(m) G(n-m), & \text{for } n \geq K. \end{cases}$$

The function implements precisely these two cases. Since batches contain at most K_B jobs, the largest possible state is $K-1+K_B$.

```
1: function CompleteAcceptance(K, λ_B, μ, G, K_B)
2:     q ← Map(), q(0) ← 1
3:     for n = 0, ..., K + K_B − 2 do
4:         M ← min{n, K − 1}
5:         q(n+1) ← (λ_B/μ) · Sum([q(m) · G(n − m) : m = 0, ..., M])
6:
7:     return NormalizeMap(q)
```

s.5.3.9. For the partial acceptance case, any job is accepted, but the system accepts only what fits. As level $n \in 0, 1, \ldots, K-1$ is still up-crossed by any batch of size at least $n-m$ when the system is in state m, the formula for the up-crossing rate is identical to the case without this acceptance policy. Hence, $\mu\pi(n+1) = \lambda_B \sum_{m=0}^{n} \pi(m) G(n-m)$, for $n = 0, 1, \ldots, K-1$.

The following function applies this recursion for all levels below K and then normalizes the resulting weights.

```
1: function PartialAcceptance(K, λ_B, μ, G)
2:     q ← Map(), q(0) ← 1
3:     for n = 0, ..., K − 1 do
4:         q(n+1) ← (λ_B/μ) · Sum([q(m) · G(n − m) : m = 0, ..., n])
5:
6:     return NormalizeMap(q)
```

s.5.3.10. Suppose a batch of size k arrives when the system contains m jobs. When $m+k \le K$, the batch can be accepted since the entire batch will fit into the queue; otherwise, it will be rejected. Further, level $n, 0 \le n < K$, can only be crossed when $m+k > n$. Thus, for $n = 0, \ldots, K-1$,

$$\begin{aligned}\mu\pi(n+1) &= \lambda_B \sum_{m=0}^{n} \pi(m)\,\mathsf{P}\{n-m < B \le K-m\} \\ &= \lambda_B \sum_{m=0}^{n} \pi(m)[G(n-m) - G(K-m)],\end{aligned}$$

Let us check this for some simple cases. First, when $K \to \infty$, then $G(K-m) \to 0$, so we get our earlier result. Second, take $n = K$. Then $G(n-m) - G(K-m) = 0$ for all m, so that the RHS is 0, as it should. Third, take $n = 0$ and $K = 1$, then $\mu\pi(1) = \lambda_B \pi(0)\,\mathsf{P}\{B = 1\}$. This also makes sense.

The subtraction in the recursion removes batches that cross level n but do not fit below capacity K.

```
1: function CompleteRejection(K, λ_B, μ, G)
2:     q ← Map(), q(0) ← 1
3:     for n = 0, ..., K − 1 do
4:         q(n+1) ← (λ_B/μ) · Sum([q(m) · [G(n−m) − G(K−m)] : m = 0, ..., n])
5:
6:     return NormalizeMap(q)
```

s.5.3.11. The recursion for $\pi(n+1)$ states that

$$\pi(n+1) = \lambda_B/\mu \sum_{i=0}^{K_B-1} \pi(n-i)\,\mathsf{P}\{B > i\}.$$

Start with taking $\pi(0) = 1$, so we consider the unnormalized probabilities. Then, by the recursion, $\pi(1) = \lambda_B/\mu \cdot \pi(0) \le \rho$, where we use the hint. Since $\pi(1) < \pi(0)$, $\pi(2) \le \lambda_B/\mu \sum_{i=0}^{1} \mathsf{P}\{B > i\} \le \rho$. By similar reasoning, $\pi(n) \le \rho$ for all $n = 1, \ldots K_B - 1$. This in turn implies that

$$\pi(K_B) = \lambda_B/\mu \sum_{i=0}^{K_B-1} \pi(n-i)\,\mathsf{P}\{B > i\} \le \lambda_B/\mu \cdot \rho \sum_{i=0}^{K_B-1} \mathsf{P}\{B > i\} = \rho^2.$$

Continuing like this we see that $\pi(n) \le \rho^2$ for all $n = K_B, \ldots 2K_B - 1$. Clearly, this in turn implies that $\pi(n) \le \rho^3$ for all $n = 2K_B, \ldots 3K_B - 1$, and so on.

s.5.3.12.

$$\sum_{i=1}^{\infty} iG(i-1) = \sum_{i=0}^{\infty} (i+1)G(i) = \sum_{i=0}^{\infty} iG(i) + \sum_{i=0}^{\infty} G(i) = (\mathsf{E}\left[B^2\right] - \mathsf{E}[B])/2 + \mathsf{E}[B].$$

s.5.3.13. We use that $\mu\pi(n) = \lambda_B \sum_{i=0}^{n-1} \pi(i)G(n-1-i)$ and the results of the exercises of Section 5.3 to see that

$$\begin{aligned}
\mu \mathsf{E}[L] &= \sum_{n=0}^{\infty} n\mu\pi(n), \quad \text{now substitute for } \mu\pi(n) \text{ the recursion (5.3.5)},\\
&= \lambda_B \sum_{n=0}^{\infty} n \sum_{i=0}^{n-1} \pi(i)G(n-1-i) = \lambda_B \sum_{n=0}^{\infty} n \sum_{i=0}^{\infty} \mathbb{1}\{i<n\}\pi(i)G(n-1-i)\\
&= \lambda_B \sum_{i=0}^{\infty} \pi(i) \sum_{n=0}^{\infty} \mathbb{1}\{i<n\} nG(n-1-i) = \lambda_B \sum_{i=0}^{\infty} \pi(i) \sum_{n=i+1}^{\infty} nG(n-1-i)\\
&= \lambda_B \sum_{i=0}^{\infty} \pi(i) \sum_{n=0}^{\infty} (n+i+1)G(n)\\
&= \lambda_B \sum_{i=0}^{\infty} \pi(i) \left[\sum_{n=0}^{\infty} nG(n) + (i+1) \sum_{n=0}^{\infty} G(n) \right]\\
&= \lambda_B \sum_{i=0}^{\infty} \pi(i) \sum_{n=0}^{\infty} nG(n) + \lambda_B \mathsf{E}[B] \sum_{i=0}^{\infty} \pi(i)(i+1)\\
&= \lambda_B \sum_{i=0}^{\infty} \pi(i) \frac{\mathsf{E}\left[B^2\right] - \mathsf{E}[B]}{2} + \lambda_B \mathsf{E}[B] (\mathsf{E}[L]+1)\\
&= \lambda_B \frac{\mathsf{E}\left[B^2\right] - \mathsf{E}[B]}{2} + \lambda_B \mathsf{E}[B] \mathsf{E}[L] + \lambda_B \mathsf{E}[B]\\
&= \lambda_B \frac{\mathsf{E}\left[B^2\right]}{2} + \lambda_B \mathsf{E}[B] \mathsf{E}[L] + \lambda_B \frac{\mathsf{E}[B]}{2}.
\end{aligned}$$

Dividing both sides by μ and using that $\lambda_B \mathsf{E}[B]/\mu = \rho$, we obtain

$$\mathsf{E}[L] = \lambda_B \frac{\mathsf{E}\left[B^2\right]}{2} \mathsf{E}[S] + \rho \mathsf{E}[L] + \frac{\rho}{2}.$$

By bringing $\rho \mathsf{E}[L]$ to the LHS, the RHS becomes equal to (5.3.3).

s.5.4.1. True.

s.5.4.2. True.

s.5.4.3. False. The factor $\rho/(1-\rho)$ makes $\mathsf{E}[W]$ a nonlinear, convex function of ρ that diverges as $\rho \uparrow 1$.

s.5.4.4. False. $\mathsf{E}[S_r] = \lambda \mathsf{E}\left[S^2\right]/2$, which depends on the second moment of the service time, not just the mean.

s.5.4.5. True. PASTA is used to conclude that $\mathsf{E}[W] = \mathsf{E}[S_r] + \mathsf{E}[Q]\,\mathsf{E}[S]$, and Little's law to replace $\mathsf{E}[Q]$ by $\lambda \mathsf{E}[W]$.

s.5.4.6. Expressing the arrival rate per hour gives

$$\lambda = \frac{3}{8}, \qquad \rho = \lambda \mathsf{E}[S] = \frac{3}{8} 2 = \frac{3}{4}.$$

Since the arrivals are Poisson, $C_a^2 = 1$. Substitution in the waiting-time formula gives

$$\mathsf{E}[W] = \frac{C_a^2 + C_s^2}{2} \frac{\rho}{1-\rho} \mathsf{E}[S] = \frac{1+1/2}{2} \frac{3/4}{1/4} 2 = 4.5 \text{ hours}.$$

Eliminating all service-time variability is not sufficient. Indeed, when $C_s^2 = 0$,

$$\mathsf{E}[W] = \frac{1}{2}\frac{3/4}{1/4}2 = 3 \text{ hours.}$$

Thus, capacity must also be increased, for instance by adding another machine or reducing the average service time. As an illustration, if $\mathsf{E}[S] = 1$ hour and $C_s^2 = 1/3$, then $\rho = 3/8$ and

$$\mathsf{E}[W] = \frac{1+1/3}{2}\frac{3/8}{5/8}1 = 0.4 \text{ hours.}$$

s.5.4.7. $\mathsf{V}[S] = 0 \implies C_s^2 = 0 \implies \mathsf{E}[W_{M/D/1}] = \mathsf{E}[W_{M/M/1}]/2$.

s.5.4.8.

$$\mathsf{E}[S] = \alpha/2, \qquad \mathsf{E}\left[S^2\right] = \frac{1}{\alpha}\int_0^\alpha x^2\,\mathrm{d}x = \alpha^2/3,$$
$$\mathsf{V}[S] = \alpha^2/3 - \alpha^2/4 = \alpha^2/12, \qquad C_s^2 = (\alpha^2/12)/(\alpha^2/4) = 1/3,$$
$$\rho = \lambda\alpha/2,$$
$$\mathsf{E}[W] = \frac{1+C_s^2}{2}\frac{\lambda\alpha/2}{1-\lambda\alpha/2}\frac{\alpha}{2}, \qquad \mathsf{E}[J] = \mathsf{E}[W] + \frac{\alpha}{2}.$$

s.5.4.9. The probability of finding the server busy upon arrival is ρ, and only arrivals that find the server busy observe $S_r > 0$.

$$\mathsf{E}[S_r] = \rho\,\mathsf{E}[S_r \mid S_r > 0] + (1-\rho)\,\mathsf{E}[S_r \mid S_r = 0] = \rho\,\mathsf{E}[S_r \mid S_r > 0].$$

For the $M/M/1$, service times are memoryless, so $\mathsf{E}[S_r \mid S_r > 0] = \mathsf{E}[S]$, which implies $\mathsf{E}[S_r] = \rho\,\mathsf{E}[S]$ for the $M/M/1$ queue.

s.5.5.1. After job $k-1$ left, job k first has to arrive. Hence, $\mathsf{E}[D_k - D_{k-1}] = \mathsf{E}[X_k + S_k] = 1/\lambda + \mathsf{E}[S]$, where we use that X_k is memoryless.

s.5.5.2. Use conditional probability to see that

$$\mathsf{P}\{Y_n = j\} = \int_0^\infty e^{-\lambda x}\frac{(\lambda x)^j}{j!}\,\mathrm{d}F(x) = \int_0^\infty e^{-\lambda x}\frac{(\lambda x)^j}{j!}\mu e^{-\mu x}\,\mathrm{d}x = \frac{\mu}{j!}\lambda^j\int_0^\infty e^{-(\lambda+\mu)x}x^j\,\mathrm{d}x$$
$$= \frac{\mu}{j!}\left(\frac{\lambda}{\lambda+\mu}\right)^j\int_0^\infty e^{-(\lambda+\mu)x}((\lambda+\mu)x)^j\,\mathrm{d}x = \frac{\mu}{j!}\left(\frac{\lambda}{\lambda+\mu}\right)^j\frac{j!}{\lambda+\mu}.$$

Method 2. Consider the Poisson process with rate $\lambda+\mu$, and thin with probability $\mu/(\lambda+\mu)$. Then the probability that j 'failures' occur before a 'success' is precisely $\mathsf{P}\{Y = j\}$.

s.5.5.3. Take $\alpha = \lambda/(\lambda+\mu)$ so that $f(j) = (1-\alpha)\alpha^j$.

$$G(j) = \sum_{k=j+1}^{\infty} f(k) = (1-\alpha)\sum_{k=j+1}^{\infty}\alpha^k = (1-\alpha)\alpha^{j+1}\sum_{k=0}^{\infty}\alpha^k = \alpha^{j+1}.$$

s.5.5.4. Here

$$\alpha = \frac{\lambda}{\lambda+\mu} = \frac{1}{3}, \qquad f(j) = (1-\alpha)\alpha^j, \qquad G(j) = \alpha^{j+1}.$$

Start with the unnormalized weight $q(0) = 1$. For $n = 0$, recursion (5.5.1) gives

$$q(1)f(0) = q(0)G(0), \qquad q(1) = \frac{1/3}{2/3} = \frac{1}{2}.$$

For the next two values, the same recursion gives

$$q(2) = \frac{1}{4}, \qquad q(3) = \frac{1}{8}.$$

Continuing in this way produces $q(n) = (1/2)^n$. Since $\sum_{n=0}^{\infty} q(n) = 2$, normalization yields

$$\pi(n) = \frac{q(n)}{2} = \frac{1}{2}\left(\frac{1}{2}\right)^n.$$

Therefore,

$$\pi(0) = 0.5, \qquad \pi(3) = \frac{1}{16} = 0.0625.$$

These are precisely $(1-\rho)\rho^n$ with $\rho = \lambda/\mu = 1/2$.

s.5.5.5. Observe that $f(j) = (1-\alpha)\alpha^j$, and $G(j) = \alpha^{j+1}$. As the normalization factor cancels on both sides, we drop the normalization and just write $\pi(n) = \rho^n$ to simplify the algebra.

For $n = 0$: $f(0)\pi(1) = \pi(0)G(0) \iff (1-\alpha)\rho = 1\,\alpha$, and this checks with the hint. For $n \geq 1$:

$$\begin{aligned}
(1-\alpha)\rho^{n+1} &= \pi(0)G(n) + \sum_{m=1}^{n} \pi(m)G(n+1-m) = \alpha^{n+1} + \sum_{m=1}^{n} \rho^m \alpha^{n-m+2} \\
&= \alpha^{n+1} + \alpha^{n+2} \sum_{m=1}^{n} (\rho/\alpha)^m = \alpha^{n+1} + \alpha^{n+1}\rho \sum_{m=0}^{n-1} (\rho/\alpha)^m \\
&= \alpha^{n+1} + \alpha^{n+1}\rho \frac{1-(\rho/\alpha)^n}{1-\rho/\alpha} \\
&= \alpha^{n+1} - \alpha^{n+1}(1-(\rho/\alpha)^n), \quad \text{as } 1-\rho/\alpha = -\rho, \\
&= \alpha^{n+1}(\rho/\alpha)^n = \alpha\rho^n.
\end{aligned}$$

Since $\rho = \alpha/(1-\alpha)$ we see that the LHS and RHS are the same.

s.5.6.1. False. It should be this: $d_k = I_k \min\{L_{k-1}, c_k\}$, $L_k = L_{k-1} - d_k + a_k$.

s.5.6.2. True.

s.5.6.3. True. This follows from the fact that the net drain rate is $\mu - \lambda$, and can be verified by substituting $T(q) = aq$ into the recursion (5.6.2).

s.5.6.4. False. The optimal N^* depends on λ, ρ, K, and h, but $\rho = \lambda\,\mathsf{E}[S]$ itself depends on the mean service time. However, N^* does not depend on higher moments of the service time such as C_s^2.

s.5.6.5. True. With $N = 1$ the server never switches off, and with $K = 0$ there is no setup cost, so the N-policy reduces to the standard $M/G/1$ queue.

s.5.6.6. When we switch on the server, the queue drains at a rate $\mu - \lambda > 0$, with $\mu = 1/\mathsf{E}[S]$. Consequently, no matter how large N, $T(N) < \infty$. Whenever the system is empty, the stochastic process restarts. As such cycles start over and over again, the queue length cannot 'escape to infinity'.

s.5.6.7. The total number $A(t)$ of jobs that arrive during $[0, t]$ does not depend on N. Thus, in (4.3.2), $\sum_{k=1}^{A(t)} S_k$ does not depend on N. Now use rate-stability.

s.5.6.8. The cost up to the qth job is the cost $W(q-1)$ up to the arrival of job $q-1$ plus the cost while there are $q-1$ jobs in the system. The time between the arrival of job $q-1$ and q is $1/\lambda$. When there are $q-1$ jobs in the system, $h(q-1)/\lambda$ is the expected cost between the arrivals of job $q-1$ and job q. And thus, $W(q) = h\lambda^{-1}\sum_{i=1}^{q}(i-1)$.

s.5.6.9. Consider an arbitrary moment in time at which $q > 0$ and the server is busy. Now, either of two events happens first: a new job enters the system, or the job in service leaves. The probability that an arrival occurs first is $\alpha = \lambda/(\lambda+\mu)$,[1] the probability of a departure to occur first is $\beta = 1 - \alpha = \mu/(\lambda+\mu)$. In addition, the expected time for an arrival or a departure, whichever is first, is $1/(\mu+\lambda)$.[2] In other words, the system stays in state q for an expected time $1/(\lambda+\mu)$ until an arrival or departure occurs. Then, it moves to state $q+1$ or $q-1$, and from there it takes $T(q+1)$ or $T(q-1)$ until the system is empty. This reasoning depends crucially on the memoryless property.

[1] *See [1.3.11].*

[2] *See [1.3.9].*

s.5.6.10. Note that the queueing cost is hq per unit time when there are q jobs in the system; it costs $hq/(\lambda+\mu)$ until an arrival or departure occurs. For the rest, we follow the reasoning by which we derive the recursion for $T(q)$ for the $M/M/1$ queue.

s.5.6.11. In the hint, the first equation is superfluous. In the second, bq cancels on both sides, by which we find a. The third now follows.

s.5.6.12. For b, using the expressions for $\mathsf{E}[Y]$ and $\mathsf{E}\left[Y^2\right]$,

$$
\begin{aligned}
b(1-\mathsf{E}[Y]) &= a(\mathsf{E}\left[Y^2\right] - 2\mathsf{E}[Y] + 1) + \frac{1}{2}h\lambda\,\mathsf{E}\left[S^2\right] \\
&= \frac{h\,\mathsf{E}[S]}{2(1-\mathsf{E}[Y])}(\mathsf{E}\left[Y^2\right] - 2\mathsf{E}[Y] + 1) + \frac{1}{2}h\lambda\,\mathsf{E}\left[S^2\right] \\
&= \frac{h\,\mathsf{E}[S]}{2(1-\lambda\,\mathsf{E}[S])}\left(\lambda^2\,\mathsf{E}\left[S^2\right] + \lambda\,\mathsf{E}[S] - 2\lambda\,\mathsf{E}[S] + 1 + \lambda\,\mathsf{E}\left[S^2\right]\frac{1-\lambda\,\mathsf{E}[S]}{\mathsf{E}[S]}\right) \\
&= \frac{h\,\mathsf{E}[S]}{2(1-\lambda\,\mathsf{E}[S])}\left(\lambda^2\,\mathsf{E}\left[S^2\right] - \lambda\,\mathsf{E}[S] + 1 + \frac{\lambda\,\mathsf{E}\left[S^2\right]}{\mathsf{E}[S]} - \lambda^2\,\mathsf{E}\left[S^2\right]\right) \\
&= \frac{h\,\mathsf{E}[S]}{2(1-\lambda\,\mathsf{E}[S])}\left(1 + \frac{\lambda\,\mathsf{E}\left[S^2\right]}{\mathsf{E}[S]} - \lambda\,\mathsf{E}[S]\right) \\
&= \frac{h\,\mathsf{E}[S]}{2(1-\lambda\,\mathsf{E}[S])}\left(1 + \frac{\lambda\,\mathsf{E}\left[S^2\right]}{\mathsf{E}[S]} - \lambda\frac{(\mathsf{E}[S])^2}{\mathsf{E}[S]}\right) \\
&= \frac{h\,\mathsf{E}[S]}{2(1-\lambda\,\mathsf{E}[S])}\left(1 + \lambda\frac{\mathsf{V}[S]}{\mathsf{E}[S]}\right) \\
&= \frac{h\,\mathsf{E}[S]}{2(1-\lambda\,\mathsf{E}[S])}\left(1 + \lambda\frac{\mathsf{V}[S]}{(\mathsf{E}[S])^2}\,\mathsf{E}[S]\right) \\
&= \frac{h\,\mathsf{E}[S]}{2(1-\lambda\,\mathsf{E}[S])}\left(1 + \rho C_s^2\right).
\end{aligned}
$$

Divide now both sides by $1-\mathsf{E}[Y]$.

s.5.6.13. Note first that $C(N) = N(1/\lambda + \mathsf{E}[S]/(1-\rho)) = N/(\lambda(1-\rho))$. Then,

$$\begin{aligned}
\frac{V(N)+K+W(N)}{C(N)} &= \left(aN^2+bN+K+hN(N-1)/2\lambda\right)\frac{\lambda(1-\rho)}{N} \\
&= \frac{h}{2}\rho N + \frac{h}{2}\frac{\rho}{1-\rho}(1+\rho C_s^2) + \frac{h}{2}(N-1)(1-\rho) + K\frac{\lambda(1-\rho)}{N} \\
&= \frac{h}{2}\frac{\rho}{1-\rho}(1+\rho C_s^2) + \frac{h}{2}(N-1+\rho) + K\frac{\lambda(1-\rho)}{N} \\
&= \frac{h}{2}\frac{\rho}{1-\rho}(\rho+\rho C_s^2+1-\rho) + \frac{h}{2}(N-1+\rho) + K\frac{\lambda(1-\rho)}{N} \\
&= \frac{h}{2}\frac{\rho^2}{1-\rho}(1+C_s^2) + \frac{h}{2}\rho + \frac{h}{2}(N-1+\rho) + K\frac{\lambda(1-\rho)}{N} \\
&= \frac{h}{2}\frac{\rho^2}{1-\rho}(1+C_s^2) + h\rho + h\frac{N-1}{2} + K\frac{\lambda(1-\rho)}{N}.
\end{aligned}$$

s.5.7.1. True. Since $I = S - X$, the event $I - D \geq 0$ is the same as $X + D \leq S$.

s.5.7.2. False. The backlog is $\mathsf{E}\left[[X-S]^+\right]$, not $\mathsf{E}\left[[S-X]^+\right]$. The expression $[S-X]^+$ is the on-hand inventory, see (5.7.4).

s.5.7.3. False. Increasing S reduces backlog cost $b\mathsf{E}[I^-]$ but increases holding cost $h\mathsf{E}\left[I^+\right]$. Since $c(S)$ is convex, there is an optimal S^* beyond which the cost increases.

s.5.7.4. True. If S is small relative to the lead-time demand, then $\mathsf{E}[I] < 0$, meaning that on average there is a backlog.

s.5.7.5. True. By definition, $I^+ = \max\{I, 0\}$ and $I^- = \max\{-I, 0\}$, so $I^+ - I^- = I$.

s.5.7.6. If $l = 0$, then $X = 0$, hence $f_0 = \mathsf{P}\{X = 0\} = 1$. Consequently, $\mathsf{E}[I] = S$, $\mathsf{E}[I^-] = [-S]^+$ and $\mathsf{E}\left[I^+\right] = S^+$. For $S \geq 0$, these reduce to zero backlog and on-hand inventory S.

s.5.7.7. D_k and I_k are independent, but D_k and I'_k not.

s.5.7.8. Fix i. The function $S \to [S-i]^+$ is convex. Since a weighted sum of convex functions is still convex, $\sum_{i=0}^{\infty} f(i)[S-i]^+$ is also convex. But $\sum_{i=0}^{\infty} f(i)[S-i]^+ = \mathsf{E}\left[[S-X]^+\right] = \mathsf{E}\left[I^+\right]$, and therefore $h\mathsf{E}\left[I^+\right]$ is a convex function of S. Similarly, $S \to [i-S]^+$ is convex, and therefore $\mathsf{E}[I^-]$ is convex. Consequently, $c(S)$, being the sum of two convex functions, is convex. As $\mathsf{E}\left[I^+\right] \to \infty$ when $S \to \infty$, and $\mathsf{E}[I^-] \to \infty$ when $S \to -\infty$, $c(\cdot)$ attains a minimum.

s.5.8.1. True.

s.5.8.2. False.

s.5.8.3. True. Demand is two in each of the three lead-time periods.

s.5.8.4. True.

s.5.8.5. False.

s.5.8.6. False. The order frequency is $\mathsf{E}[D]/Q$, so the order-cost term should be $K\mathsf{E}[D]/Q$.

s.5.8.7. When $l = 0$, we have $\mathsf{E}[X] = 0$ so that for the base-stock model, $\mathsf{E}\left[I^+(i)\right] = i$. Hence, for the (Q, r) model, $\mathsf{E}\left[I^+\right] = Q(Q-1)/2Q = (Q-1)/2$. Since demand arrives at a constant rate d, the average cost becomes $Kd/Q + hQ/2 - h/2$. Treating Q as continuous, differentiating and setting the derivative to zero yields the result.

s.5.9.1. False. For the (s, S) policy the recursion is circular in x and cannot be solved by backward induction in time, and therefore needs the extra factor $1 - f_0$; this is explicitly contrasted with the (T, S) policy.

s.5.9.2. False. It is the time to hit the set $\{i : i \le s\}$, not the time to hit s.

s.5.9.3. True. In both calls the running cost is L and the stopping cost is K. For the (T, S) policy, taking $s = -\infty$ ensures that ordering occurs when the time index reaches zero.

s.5.9.4. False. The term for $d = 0$ was moved to the left-hand side of the recursion, so the return value must be

$$\frac{L(x) + \textsc{Sum}\left(\left[p_D(d)\mathrm{V}(s, x-d) : d \in \textsc{Keys}\left(p_D\right),\ d > 0\right]\right)}{1 - p_D(0)}.$$

s.5.9.5. False. It should be $L(x)$.

s.5.9.6. Observe that the function `exp_apply` returns an inner function `apply`. This inner function **captures** the variable `exponent` from its surrounding scope. As a result, the returned function remembers the value of `exponent`. For example, in `square = exp_apply(2)`, the function `exp_apply(2)` does not perform any computation yet. Instead, it returns a function that gets assigned to `square` for which the exponent is fixed to 2. Calling the returned function on a number completes the computation:

```
print(exp_apply(1)(9))  # 9
print(square(5))  # 25 = 5**2
print(cube(4))  # 64 = 4**3
```

In general terms: the outer function `exp_apply` specifies the general structure of a computation: it describes the pattern "take a number and raise it to a fixed exponent," without yet committing to a particular exponent. By calling the outer function and assigning its output to a new name, we specialize this general function. The returned function has the same structure, but with the free parameter `exponent` bound to a concrete value. In this way, we obtain a new, more specific function from a general computational template.

Look how elegant the sqrt function can be defined.

```
sqrt = exp_apply(0.5)
```

A compact version of `exp_apply` would be like this.

```
def exp_apply(exponent):
    return lambda base: base**exponent
```

Hopefully you find such ideas interesting: the concepts illustrated in this exercise are called *functional programming*, *currying*, and *closures*.

s.5.10.1. When a state x is in the stopping set, it has been visited once. If $\mu(x) \le \varepsilon$, the proposed rule discards its mass before this visit is counted. We need to count this visit because there can be a cost associated with stopping at this x.

s.6.1.1. True.

s.6.1.2. True. Suppose for ease that the scvs do not change when we reduce the load ρ by 5%, and that we achieve this load by reducing λ by 5% so that $\mathsf{E}[S]$ remains the same as well. Then the only factor that changes in Sakasegawa's formula is $\rho/(1-\rho)$. Initially, this factor is $0.95/(1-0.95) = 19$. After the reduction, $\rho = 0.95 \cdot 0.95 = 0.9025$, so the factor becomes $0.9025/(1-0.9025) \approx 9.26$. Since $9.26/19 \approx 0.49$, the waiting time is reduced by about half.

s.6.1.3. True. If the queue is not stable, $\mathsf{E}[W] = \infty$.

s.6.1.4. $\rho = \lambda \mathsf{E}[S] = 1/60 \cdot 50 = 5/6$. Since job arrivals do not overlap any job service, the number of jobs in the system is 1 for 50 seconds, then the server is idle for 10 seconds, and so on. Thus, $\mathsf{E}[L] = 1 \cdot 5/6 = 5/6$. There is no variance in the inter-arrival times, and also not in the service times, thus $C_a^2 = C_s^2 = 0$. Also $\mathsf{E}[W] = 0$ since $\mathsf{E}[Q] = 0$.

s.6.1.5. Again, $\mathsf{E}[S]$ is 50 seconds, so that $\rho = 5/6$. Also, $C_a^2 = 0$. For C_s^2, we have to do some work.

$$\begin{aligned}
\mathsf{E}[S] &= \frac{20}{2} + \frac{80}{2} = 50 \\
\mathsf{E}\left[S^2\right] &= \frac{400}{2} + \frac{6400}{2} = 3400 \\
\mathsf{V}[S] &= \mathsf{E}\left[S^2\right] - (\mathsf{E}[S])^2 = 3400 - 2500 = 900 \\
C_s^2 &= \frac{\mathsf{V}[S]}{(\mathsf{E}[S])^2} = \frac{900}{2500} = \frac{9}{25}.
\end{aligned}$$

s.6.1.6. For $X \sim \text{Unif}([3,9])$,

$$\mathsf{E}[X] = \frac{3+9}{2} = 6, \qquad \mathsf{V}[X] = \frac{(9-3)^2}{12} = 3.$$

Hence,

$$\lambda = \frac{1}{\mathsf{E}[X]} = \frac{1}{6}, \qquad C_a^2 = \frac{\mathsf{V}[X]}{(\mathsf{E}[X])^2} = \frac{3}{36} = \frac{1}{12}.$$

For the inspection time,

$$C_s^2 = \frac{\mathsf{V}[S]}{(\mathsf{E}[S])^2} = \frac{4}{25}, \qquad \rho = \lambda \mathsf{E}[S] = \frac{5}{6}.$$

Sakasegawa's formula therefore gives

$$\mathsf{E}[W] = \frac{C_a^2 + C_s^2}{2} \frac{\rho}{1-\rho} \mathsf{E}[S] = \frac{1/12 + 4/25}{2} \cdot 5 \cdot 5 = \frac{73}{24} \approx 3.04 \text{ minutes}.$$

For Poisson arrivals, $C_a^2 = 1$, while the load and the service-time variability remain unchanged. Thus,

$$\mathsf{E}[W] = \frac{1+4/25}{2} \cdot 5 \cdot 5 = \frac{29}{2} = 14.5 \text{ minutes.}$$

The uniform inter-arrival times are much more regular than Poisson inter-arrival times: consecutive arrivals are always between 3 and 9 minutes apart. Their smaller variability therefore results in a much shorter expected waiting time.

s.6.1.7.

$$\begin{aligned}
\mathsf{E}[S] &= \mathsf{E}[\mathbb{1}\{T=1\}S_1] + \mathsf{E}[\mathbb{1}\{T=2\}S_2] \\
&= \mathsf{E}[\mathbb{1}\{T=1\}]\,\mathsf{E}[S_1] + \mathsf{E}[\mathbb{1}\{T=2\}]\,\mathsf{E}[S_2], \quad \text{by the independence of } T, \\
&= \mathsf{P}\{T=1\}\,\mathsf{E}[S_1] + \mathsf{P}\{T=2\}\,\mathsf{E}[S_2] \\
&= p\,\mathsf{E}[S_1] + q\,\mathsf{E}[S_2].
\end{aligned}$$

For the variance, we need some algebra. Since,

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

we get

$$\begin{aligned}
\mathsf{V}[S] &= \mathsf{E}\left[S^2\right] - (\mathsf{E}[S])^2 \\
&= \mathsf{E}\left[(\mathbb{1}\{T=1\}S_1 + \mathbb{1}\{T=2\}S_2)^2\right] - (\mathsf{E}[S])^2 \\
&= \mathsf{E}\left[\mathbb{1}\{T=1\}S_1^2 + \mathbb{1}\{T=2\}S_2^2\right] - (\mathsf{E}[S])^2 \\
&= p\,\mathsf{E}\left[S_1^2\right] + q\,\mathsf{E}\left[S_2^2\right] - (\mathsf{E}[S])^2 \\
&= p\,\mathsf{V}[S_1] + p(\mathsf{E}[S_1])^2 + q\,\mathsf{V}[S_2] + q(\mathsf{E}[S_2])^2 - (\mathsf{E}[S])^2 \\
&= p\,\mathsf{V}[S_1] + p(\mathsf{E}[S_1])^2 + q\,\mathsf{V}[S_2] + q(\mathsf{E}[S_2])^2 - p^2(\mathsf{E}[S_1])^2 - q^2(\mathsf{E}[S_2])^2 - 2pq\,\mathsf{E}[S_1]\,\mathsf{E}[S_2] \\
&= p\,\mathsf{V}[S_1] + q\,\mathsf{V}[S_2] + pq(\mathsf{E}[S_1])^2 + pq(\mathsf{E}[S_2])^2 - 2pq\,\mathsf{E}[S_1]\,\mathsf{E}[S_2], \quad \text{as } p = 1-q \\
&= p\,\mathsf{V}[S_1] + q\,\mathsf{V}[S_2] + pq(\mathsf{E}[S_1] - \mathsf{E}[S_2])^2.
\end{aligned}$$

s.6.2.1. True, [6.2.7]

s.6.2.2. False.

s.6.2.3. True. A larger batch means that the setup cost is amortized over more jobs, so the effective service time per job decreases, and with it the load.

s.6.2.4. True. Aggregating B inter-arrival times into one batch inter-arrival time reduces the scv by a factor B.

s.6.2.5. False. Larger batches reduce the time in queue (lower load, lower variability), but increase the time a job waits for its batch to be formed, which is $(B-1)/(2\lambda_i)$. There is a trade-off.

s.6.2.6. Since the service times are exponential,

$$\mathsf{E}[S_0] = \frac{15}{60} = \frac{1}{4} \text{ hour}, \qquad \mathsf{V}[S_0] = (\mathsf{E}[S_0])^2 = \frac{1}{16} \text{ hour}^2.$$

Moreover, $\mathsf{E}[R] = 2$ hours and $\mathsf{V}[R] = 1$ hour squared. For stability, the batch size must satisfy

$$B > \frac{\lambda \mathsf{E}[R]}{1 - \lambda \mathsf{E}[S_0]} = \frac{3 \cdot 2}{1 - 3/4} = 24.$$

Thus, the smallest allowable integer batch size is $B = 25$.

Now take $B = 30$. The batch arrival rate, expected batch service time, and load are

$$\lambda_B = \frac{\lambda}{B} = \frac{1}{10},$$
$$\mathsf{E}[S_B] = \mathsf{E}[R] + B\mathsf{E}[S_0] = 2 + 30 \cdot \frac{1}{4} = \frac{19}{2},$$
$$\rho = \lambda_B \mathsf{E}[S_B] = \frac{19}{20}.$$

The scv of the batch inter-arrival times is

$$C_{a,B}^2 = \frac{C_a^2}{B} = \frac{1}{30}.$$

For the batch service times,

$$\mathsf{V}[S_B] = \mathsf{V}[R] + B\mathsf{V}[S_0] = 1 + \frac{30}{16} = \frac{23}{8},$$

and hence

$$C_{s,B}^2 = \frac{\mathsf{V}[S_B]}{(\mathsf{E}[S_B])^2} = \frac{23}{722}.$$

Sakasegawa's formula gives the expected time a batch spends in queue:

$$\mathsf{E}[W_B] = \frac{C_{a,B}^2 + C_{s,B}^2}{2} \frac{\rho}{1-\rho} \mathsf{E}[S_B] = \frac{353}{60} \approx 5.88 \text{ hours}.$$

A red job waits on average

$$\frac{B-1}{2\lambda_r} = \frac{29}{2 \cdot 0.5} = 29 \text{ hours}$$

for its batch to be formed. Once its batch reaches the server, its expected time there is

$$\mathsf{E}[R] + \frac{B+1}{2} \mathsf{E}[S_0] = 2 + \frac{31}{2} \cdot \frac{1}{4} = \frac{47}{8} = 5.875 \text{ hours}.$$

Therefore, the average time a red job spends in the system is

$$\mathsf{E}[J_r] = 29 + \frac{353}{60} + \frac{47}{8} = \frac{4891}{120} \approx 40.76 \text{ hours}.$$

s.6.2.7. Suppose that a batch is just finished. The first job of a new batch needs to wait, on average, $B-1$ inter-arrival times until the batch is complete, the second $B-2$ inter-arrival times, and so on. The last job does not have to wait at all. Thus, the total time to form a batch is $(B-1)/\lambda_r$. An arbitrary job can be anywhere in the batch, hence its expected time is half the total time.

s.6.2.8. The variance of the inter-arrival time of batches is B times the variance in job inter-arrival times. The inter-arrival times of batches is also B times the inter-arrival times of jobs. Thus,

$$C_{a,B}^2 = \frac{B\mathsf{V}[X]}{(B\mathsf{E}[X])^2} = \frac{\mathsf{V}[X]}{(\mathsf{E}[X])^2} \frac{1}{B} = \frac{C_a^2}{B}.$$

s.6.2.9. The variance of a batch is $\mathsf{V}\left[R+\sum_{i=1}^{B} S_{0,i}\right] = \mathsf{V}[R] + B\mathsf{V}[S_0]$, since the normal service times $S_{0,i}$, $i = 1,\ldots,B$, of the jobs are independent, and also independent of the setup time R of the batch.

s.6.2.10. First, wait until the setup is finished, then wait (on average) for half of the batch (minus the job itself) to be served, and then the job has to be served itself, that is, $\mathsf{E}[R] + \frac{B-1}{2}\mathsf{E}[S_0] + \mathsf{E}[S_0]$.

s.6.3.1. False.

s.6.3.2. False. Having fewer but larger breakdowns increases the variance more. This is bad from a queueing perspective.

s.6.3.3. False. Random adjustments add an extra term $p(1-p)(\mathsf{E}[R])^2$ to the variance compared to the planned setup case; see (6.3.2) versus (6.2.4). Unplanned interruptions increase variability.

s.6.3.4. False. Fewer but longer adjustments increase the variance of the effective service time, which increases the waiting time through Sakasegawa's formula.

s.6.3.5. True. Each job independently triggers an adjustment with probability p, so the number of jobs between adjustments is geometric with parameter p.

s.6.3.6. The machine is available for 16 hours per working day, so the hourly arrival rate is

$$\lambda = \frac{9}{16}.$$

An outage occurs on average once every 30 jobs; hence $p = 1/30$. Furthermore,

$$\mathsf{E}[S_0] = \frac{3}{2}, \qquad \mathsf{V}[S_0] = \frac{1}{4}, \qquad \mathsf{E}[R] = 5, \qquad \mathsf{V}[R] = 4.$$

The mean effective service time and the load are therefore

$$\mathsf{E}[S] = \mathsf{E}[S_0] + p\,\mathsf{E}[R] = \frac{3}{2} + \frac{1}{30}\cdot 5 = \frac{5}{3},$$
$$\rho = \lambda\,\mathsf{E}[S] = \frac{9}{16}\cdot\frac{5}{3} = \frac{15}{16}.$$

Thus, the system is stable, although its load is high.

The variance of the effective service time is

$$\mathsf{V}[S] = \mathsf{V}[S_0] + p\mathsf{V}[R] + p(1-p)(\mathsf{E}[R])^2$$
$$= \frac{1}{4} + \frac{1}{30}\cdot 4 + \frac{1}{30}\frac{29}{30}\cdot 25 = \frac{107}{90}.$$

Consequently,

$$C_s^2 = \frac{\mathsf{V}[S]}{(\mathsf{E}[S])^2} = \frac{107}{250}.$$

Since arrivals are Poisson, $C_a^2 = 1$. Sakasegawa's formula now gives

$$\mathsf{E}[W] = \frac{1+107/250}{2}\,\frac{15/16}{1-15/16}\,\frac{5}{3} = \frac{357}{20} = 17.85 \text{ hours}.$$

Hence, the expected sojourn time is

$$\mathsf{E}[J] = \mathsf{E}[W] + \mathsf{E}[S] = \frac{357}{20} + \frac{5}{3} = \frac{1171}{60} \approx 19.52 \text{ hours}.$$

s.6.3.7. We use the effective-service-time model for planned setups from Section 6.2. Here $B = 20$, $\mathsf{E}[R] = 4.5 = 9/2$, and $\mathsf{V}[R] = 0$. Thus,

$$\mathsf{E}[S] = \mathsf{E}[S_0] + \frac{\mathsf{E}[R]}{B} = \frac{3}{2} + \frac{9/2}{20} = \frac{69}{40},$$
$$\rho = \lambda\,\mathsf{E}[S] = \frac{9}{16}\frac{69}{40} = \frac{621}{640} \approx 0.9703.$$

The variance and scv of the effective service time are

$$\mathsf{V}[S] = \mathsf{V}[S_0] + \frac{\mathsf{V}[R]}{B} = \frac{1}{4},$$
$$C_s^2 = \frac{\mathsf{V}[S]}{(\mathsf{E}[S])^2} = \frac{400}{4761} \approx 0.0840.$$

Since $C_a^2 = 1$, Sakasegawa's formula gives

$$\mathsf{E}[W] = \frac{1 + 400/4761}{2}\,\frac{621/640}{1 - 621/640}\,\frac{69}{40} = \frac{46449}{1520} \approx 30.56 \text{ hours}.$$

Therefore,

$$\mathsf{E}[J] = \mathsf{E}[W] + \mathsf{E}[S] = \frac{49071}{1520} \approx 32.28 \text{ hours}.$$

Compared with [6.3.6], the planned adjustments reduce service-time variability, but they increase the load from 15/16 to 621/640, which is very close to one. The increase in load dominates, so the expected sojourn time becomes much larger.

s.6.3.8. Applying Adam's and Eve's rule is straightforward, but the details require attention. Using independence where necessary (and allowed by our assumptions),

$$\begin{aligned}
\mathsf{E}[S|F] &= \mathsf{E}[S_0 + RF|F] = \mathsf{E}[S_0] + F\,\mathsf{E}[R],\\
\mathsf{E}[S] &= \mathsf{E}[\mathsf{E}[S|F]] = \mathsf{E}[S_0] + \mathsf{E}[F]\,\mathsf{E}[R] = \mathsf{E}[S_0] + p\,\mathsf{E}[R],\\
\mathsf{V}[\mathsf{E}[S|F]] &= \mathsf{V}[\mathsf{E}[S_0] + F\,\mathsf{E}[R]] = (\mathsf{E}[R])^2\,\mathsf{V}[F] = (\mathsf{E}[R])^2 p(1-p),\\
\mathsf{V}[S|F] &= \mathsf{V}[S_0 + RF|F] = \mathsf{V}[S_0] + F\,\mathsf{V}[R],\\
\mathsf{E}[\mathsf{V}[S|F]] &= \mathsf{V}[S_0] + p\,\mathsf{V}[R],\\
\mathsf{V}[S] &= \mathsf{E}[\mathsf{V}[S|F]] + \mathsf{V}[\mathsf{E}[S|F]] = \mathsf{V}[S_0] + p\,\mathsf{V}[R] + p(1-p)(\mathsf{E}[R])^2.
\end{aligned}$$

s.6.4.1. False. The two expectations in the middle cannot be split like this.

s.6.4.2. True. The service time is less variable.

s.6.4.3. False. Since $\mathsf{E}[S] = \mathsf{E}[S_0]/A$, a lower availability increases the effective service time, and consequently the load $\rho = \lambda\,\mathsf{E}[S]$ increases too.

s.6.4.4. False. Even with deterministic S_0, the number of failures N and the repair times R_i are random, so S is random.

s.6.4.5. True. From (6.4.5), $C_s^2 = C_0^2 + 2A(1-A)\,\mathsf{E}[R]/\mathsf{E}[S_0]$, and the second term is non-negative.

s.6.4.6. We express all times in hours. The failure rate, net service time, and expected repair time are

$$\lambda_f = \frac{1}{3}, \qquad \mathsf{E}[S_0] = \frac{10}{60} = \frac{1}{6}, \qquad \mathsf{E}[R] = \frac{30}{60} = \frac{1}{2}.$$

The availability is therefore

$$A = \frac{1}{1+\lambda_f \mathsf{E}[R]} = \frac{1}{1+(1/3)(1/2)} = \frac{6}{7}.$$

It follows that the mean effective service time and load are

$$\mathsf{E}[S] = \frac{\mathsf{E}[S_0]}{A} = \frac{1/6}{6/7} = \frac{7}{36},$$
$$\rho = \lambda \mathsf{E}[S] = 4 \cdot \frac{7}{36} = \frac{7}{9}.$$

Thus, the system is stable.

Since the net service time is deterministic, $C_0^2 = 0$. The repair times are exponential, so (6.4.5) gives

$$C_s^2 = 2A(1-A)\frac{\mathsf{E}[R]}{\mathsf{E}[S_0]} = 2\frac{6}{7}\frac{1}{7}\frac{1/2}{1/6} = \frac{36}{49}.$$

Arrivals are Poisson, so $C_a^2 = 1$. Sakasegawa's formula now yields

$$\mathsf{E}[W] = \frac{1+36/49}{2}\frac{7/9}{1-7/9}\frac{7}{36} = \frac{85}{144} \approx 0.5903 \text{ hours}.$$

Consequently,

$$\mathsf{E}[J] = \mathsf{E}[W] + \mathsf{E}[S] = \frac{85}{144} + \frac{7}{36} = \frac{113}{144} \approx 0.7847 \text{ hours}.$$

s.6.4.7. The time to fail is the time between two interruptions. By assumption, the failure times are $\mathrm{Exp}(\lambda_f)$, hence $m_f = 1/\lambda_f$. The expected duration of an interruption is $\mathsf{E}[R]$. With this

$$A = \frac{m_f}{m_f + \mathsf{E}[R]} = \frac{1/\lambda_f}{1/\lambda_f + \mathsf{E}[R]} = (1+\lambda_f \mathsf{E}[R])^{-1}.$$

s.6.4.8. When doing the calculations by hand, I worked from bottom to top. However, for the solutions, the current sequence is perhaps easier to understand.

$$\mathsf{E}[S|S_0,N] = \mathsf{E}\left[S_0 + \sum_{i=1}^{N} R \middle| S_0, N\right] = S_0 + N\mathsf{E}[R], \text{ since } \{R_i\} \text{ are iid,}$$
$$\mathsf{E}[S|S_0] = \mathsf{E}[\mathsf{E}[S|S_0,N]|S_0] = \mathsf{E}[S_0 + N\mathsf{E}[R]|S_0] = S_0 + \lambda_f S_0 \mathsf{E}[R] = S_0/A$$
$$\mathsf{E}[S] = \mathsf{E}[\mathsf{E}[S|S_0]] = \mathsf{E}[S_0/A] = \mathsf{E}[S_0]/A.$$
$$\mathsf{V}[S|S_0,N] = \mathsf{V}\left[S_0 + \sum_{i}^{N} R_i|S_0,N\right] = N\mathsf{V}[R], \text{ as } \mathsf{V}[S_0|S_0] = 0,$$
$$\mathsf{V}[S|S_0] = \mathsf{E}[\mathsf{V}[S|S_0,N]|S_0] + \mathsf{V}[\mathsf{E}[S|S_0N]|S_0] = \lambda_f S_0 \mathsf{V}[R] + \mathsf{V}[S_0 + N\mathsf{E}[R]|S_0]$$
$$= \lambda_f S_0 \mathsf{V}[R] + (\mathsf{E}[R])^2 \mathsf{V}[N|S_0] = \lambda_f S_0 \mathsf{V}[R] + (\mathsf{E}[R])^2 \lambda_f S_0 = \lambda_f \mathsf{E}\left[R^2\right] S_0$$
$$\mathsf{V}[S] = \mathsf{E}[\mathsf{V}[S|S_0]] + \mathsf{V}[\mathsf{E}[S|S_0]] = \lambda_f \mathsf{E}\left[R^2\right] \mathsf{E}[S_0] + \mathsf{V}[S_0]/A^2.$$

s.6.4.9.

$$C_s^2 = \frac{\mathsf{V}[S]}{(\mathsf{E}[S])^2} = \frac{V(S)A^2}{(\mathsf{E}[S_0])^2} = \frac{\mathsf{E}\left[S_0^2\right] + \lambda_f \mathsf{E}\left[R^2\right] \mathsf{E}[S_0]\, A^2 - (\mathsf{E}[S_0])^2}{(\mathsf{E}[S_0])^2}$$
$$= \frac{\mathsf{E}\left[S_0^2\right] - (\mathsf{E}[S_0])^2}{(\mathsf{E}[S_0])^2} + \frac{\lambda_f \mathsf{E}\left[R^2\right] \mathsf{E}[S_0]\, A^2}{(\mathsf{E}[S_0])^2} = C_0^2 + \frac{\lambda_f \mathsf{E}\left[R^2\right] A^2}{\mathsf{E}[S_0]}.$$

s.6.4.10. When repair times are exponentially distributed with mean $\mathsf{E}[R]$, we have $\mathsf{E}\left[R^2\right] = 2(\mathsf{E}[R])^2$. Since $A = 1/(1+\lambda_f \mathsf{E}[R])$,

$$\lambda_f \mathsf{E}\left[R^2\right] A^2 = 2\lambda_f (\mathsf{E}[R])^2 A^2 = 2\frac{\lambda_f \mathsf{E}[R]}{1+\lambda_f \mathsf{E}[R]} A \mathsf{E}[R]$$
$$= 2\left(1 - \frac{1}{1+\lambda_f \mathsf{E}[R]}\right) A \mathsf{E}[R] = 2(1-A)A\mathsf{E}[R].$$

s.6.4.11. Take T_k as the end of the kth repair, and $U(t) = \int_0^t u(s)\,\mathrm{d}s$, where $u(s) = 1$ if the server is up (operational) and 0 if the server is down (broken). Thus, $U(t)$ is the total amount of time the system has been up during $[0, t]$, and $U(t)/t$ becomes the fraction of uptime (which is the availability) in the limit. For Z_k, this is $Z_k = U(T_k) - U(T_{k-1})$, which is the uptime in $[T_{k-1}, T_k]$. Therefore, in the limit, Z is the average uptime between two failures, hence $Z \sim m_f$. Finally, the time between two inspections consists of an uptime and a breakdown. Hence, the rate at which such inspection epochs occur is $1/(m_f + \mathsf{E}[R])$.

s.7.1.1. False. Improving the first station helps downstream stations too, but need not give the largest reduction in total waiting time. For the parameters of [7.1.6], halving $C_{s,1}^2$ reduces the approximate total waiting time by 0.199 hour, whereas halving $C_{s,2}^2$ reduces it by 0.260 hour.

s.7.1.2. False. The second station has a utilization of $80/(2*60) = 8/12 = 2/3$, while the first has a utilization of $45/60 = 3/4$, which is higher.

s.7.1.3. False. In general, the number of jobs in queue is much higher than $w - N$. Consider the example $S_1 = 10$ and $N_1 = 10$, and $S_2 = 1, N_2 = 20$, and $n = 2$. Even if station 1 is fully occupied, its output rate is only one job per unit time, so the mean number of busy machines at station 2 is only 1.

s.7.1.4. False. This is incorrect: the units on the LHS and RHS do not match.

s.7.1.5. When a station is kept busy and its service times are nearly constant, the jobs sent downstream have nearly equal inter-arrival times. But then the inter-arrival times are not well approximated by the exponential distribution, consequently, the arrival process is not well described by a Poisson process.

s.7.1.6. At station 1, $\mathsf{E}[S_1] = 1/3$ hour and $\rho_1 = \lambda \mathsf{E}[S_1] = 2/3$. Sakasegawa's formula gives

$$\mathsf{E}[W_1] = \frac{C_{a,1}^2 + C_{s,1}^2}{2} \frac{\rho_1}{1-\rho_1} \mathsf{E}[S_1] = \frac{2+1/2}{2} \cdot 2 \cdot \frac{1}{3} = \frac{5}{6} \text{ hour},$$
$$\mathsf{E}[J_1] = \mathsf{E}[W_1] + \mathsf{E}[S_1] = \frac{7}{6} \text{ hours}.$$

The departures from station 1 are the arrivals at station 2. Therefore,

$$C_{a,2}^2 = C_{d,1}^2 \approx (1-\rho_1^2)C_{a,1}^2 + \rho_1^2 C_{s,1}^2 = \left(1-\frac{4}{9}\right)2 + \frac{4}{9}\cdot\frac{1}{2} = \frac{4}{3}.$$

At station 2, $\mathsf{E}[S_2] = 5/12$ hour and $\rho_2 = \lambda\,\mathsf{E}[S_2] = 5/6$. Thus,

$$\mathsf{E}[W_2] \approx \frac{C_{a,2}^2 + C_{s,2}^2}{2}\frac{\rho_2}{1-\rho_2}\mathsf{E}[S_2] = \frac{4/3+1/2}{2}\cdot 5\cdot\frac{5}{12} = \frac{275}{144}\text{ hours},$$
$$\mathsf{E}[J_2] \approx \mathsf{E}[W_2] + \mathsf{E}[S_2] = \frac{335}{144}\text{ hours}.$$

Hence, the expected sojourn time in the two-station line is

$$\mathsf{E}[J] = \mathsf{E}[J_1] + \mathsf{E}[J_2] \approx \frac{503}{144}\text{ hours} \approx 3.4931\text{ hours}.$$

s.7.2.1. In this case, the throughput μ of the network is $1/T_0$: only one server at a time is serving a job, the rest of the servers are forced to be idle as we only allow one job at a time in the network.

s.7.2.2. If there is one job in the system, $\mu(1) = 1/T_0$. If there are two, both jobs can be simultaneously worked on, each at a different station, hence $\mu(2) = 2/T_0$. And so on, until all stations are filled, at the same time, by a job, in which case $\mu(n) = n/T_0$. When there are yet more jobs in the network, only n can be simultaneously in service, the rest must wait at some station.

s.7.2.3.

$$\mu(w) = \frac{n}{T_0} = \frac{n}{nS} = \frac{1}{S}.$$

s.7.2.4.

$$\mathsf{E}[W] = \frac{\mathsf{E}[L]}{\lambda} = \frac{w}{\mu(w)} = \frac{w}{\min\{w,n\}}T_0 = \begin{cases} T_0 = nS, & \text{if } w \le n,\\ \frac{w}{n}T_0 = wS, & \text{if } w \ge n.\end{cases}$$

s.7.2.5. When $w > n$, there must be $w-n$ jobs in queue and the other n jobs are in service. So, in conclusion, in deterministic tandem networks adding more than n jobs does not increase the network's throughput, but only increases the waiting times. On the other hand, setting $w < n$ minimizes the waiting times, but also negatively affects the throughput of the network.

s.7.2.6. The processing rates are $\mu_1 = 1/1$ and $\mu_2 = 1/2$ per hour, respectively. The utilizations $\rho_i = \lambda_i/\mu_i$ are $1/3$ and $2/3$, respectively. Thus, machine 2 is the bottleneck: it has the highest utilization.

s.7.2.7. Since these machines are not related by a routing (jobs going from one machine to another), computing T_0 and so on does not make much sense for the network. We can just consider each machine on its own. As we need 2 jobs to keep each machine busy, it follows that $W_0 = 2$.

We can use Little's law to find T_0. The rate at which jobs can be served is $\mu = 1 + 1/2 = 3/2$. Thus,

$$T_0 = W_0/\mu = 2/(3/2) = 4/3.$$

Clearly, the average time a job spends in this system is 4/3 time units if jobs arrive at the rate they are served. (Recall, everything is deterministic.)

Can we see this in another way? Yes, because 2 out of 3 jobs have service time of 1, and 1 out of 3 has a service time of 2,

$$\frac{2}{3}1+\frac{1}{3}2=\frac{4}{3}.$$

s.7.2.8. The raw processing time is just the sum of the processing times at each station. All stations have the same arrival rate, hence the second machine, being the slowest, is the bottleneck. Hence,

$$\begin{aligned} T_0 &= 2+3+2=7 \\ r_b &= \mu_2 = \frac{1}{3} \\ W_0 &= r_b T_0 = \frac{7}{3}. \end{aligned}$$

s.7.2.9. Realize that machines at a station are in parallel, not in series. Thus, when we add capacity to station 2, we don't add an extra processing step to station 2, we increase capacity.

Now,

$$r_2=\frac{1}{3}+\frac{1}{3}=\frac{2}{3},$$

so that station 2 is no longer the bottleneck. Station 1 and station 3 are the bottlenecks.

$$\begin{aligned} r_b &= \frac{1}{2} \\ T_0 &= 2+3+2=7 \\ W_0 &= r_b T_0 = \frac{7}{2}. \end{aligned}$$

Often, students think that the processing rate is the inverse of the processing time, i.e., $t=r^{-1}$. This is not the case for multi-server stations. Thus, realize that the raw processing time at station 2 is still 3 hours, not 3/2.

s.7.2.10. Now,

$$r_2=\frac{1}{3}+\frac{1}{4}=\frac{7}{12}>\frac{1}{2},$$

hence, stations 1 and 3 are bottlenecks.

However, t_2 cannot be 3 hours anymore: some jobs will spend 4 hours at station 2 rather than 3 hours. In fact, the computation of the raw processing time at station 2, t_2 say, is a bit tricky. Some students reason that both machines serve half of the jobs, so that $t_2=(3+4)/2$ hours. This, however, is not quite realistic. This way, you would load the slow machine all of the time, and the faster machine would be idle quite a bit of the time.

Rather than spreading the jobs evenly over the two machines, we can also assume that each machine takes in work in proportion to its processing rate. Then we reason as follows. Consider 12 hours. In those 12 hours, the slow machine processes 3 jobs with duration 4 and the fast machine processes 4 jobs with duration 3. Therefore:

$$t_2=\frac{3}{7}4+\frac{4}{7}3=\frac{24}{7}.$$

Another way to get this is like this. We can 'stuff' station 2 with jobs. The most jobs that fit in simultaneously is 2, one job per machine. Then, with Little's law,

$$t_2 = \frac{W}{r_2} = \frac{2}{7/12} = \frac{24}{7},$$

i.e., the same as the other answer.

Now, finally,

$$\begin{aligned} r_b &= \frac{1}{2} \\ T_0 &= 2 + \frac{24}{7} + 2 = 4 + 3 + \frac{3}{7} = 7\frac{3}{7} \\ W_0 &= r_b T_0. \end{aligned}$$

Finally, the most realistic assumption is that the fast machine is always working, and the slow machine covers the rest. In that case, suppose that the fast machine processes jobs at rate r_1 with processing time $s_1 = 3$ and the slow at rate r_2 with processing time $s_2 = 4$. Then, since jobs arrive with rate r_a, the fast machine processes a fraction of r_1/r_a of the jobs, and the slow machine processes the leftovers, i.e., a fraction of $(r_a - r_1)/r_a$ of the jobs. The raw processing time then becomes

$$\begin{aligned} t_2 &= \frac{r_1}{r_a} s_1 + \frac{r_a - r_1}{r_a} s_2 \\ &= \frac{1/3}{1/2} 3 + \frac{1/2 - 1/3}{1/2} 4 \\ &= 2 + \frac{1/6}{1/2} 4 \\ &= 2 + \frac{4}{3} = 3\frac{1}{3}, \end{aligned}$$

and then,

$$T_0 = 2 + t_2 + 2 = 7\frac{1}{3},$$

and this is a tiny bit less than the previous way of organizing things.

So, why are the above answers different? One reason is that the objective is not explicitly stated. It might be that, to minimize time in a multi-server station, it is optimal to let a fast machine idle and wait until the next job comes in. If we were to assign this next job to a slow machine, the time in the system may be longer. In general, it might be that schedules that minimize T_0 (on average) are not work-conserving.

s.7.2.11. Because T_0 increases and/or the bottleneck rate increases. In both cases, we need more W_0 to fill the network and achieve that the throughput can be equal to r_b in the best case.

s.7.2.12. Now the slow machine at station 2 becomes superfluous. The fast machine at station 2 can cope with all jobs that arrive from station 1.

$$\begin{aligned} r_b &= \frac{1}{2} \\ T_0 &= 2 + 1 + 2 = 5 \\ W_0 &= r_b T_0. \end{aligned}$$

Thus, why would you assign part of the jobs to the slow machine at station 2? There is no queue, the fast machine can cope with all the jobs. Moreover, assigning any job to the slow machine just adds to the cycle time.

Taking $t_2 = (1+3)/2 = 2$ is plainly silly.

s.7.2.13. See above. I would not remove the slow machine. In real life there is variability: the fast machine might fail for instance. Spare capacity is useful. However, if it is very costly to keep the slow machine operational, I would scrap it.

s.7.2.14.

$$r_2 = \frac{1}{3} + \frac{1}{7} = \frac{10}{21} < \frac{1}{2},$$

hence station 2 is still the bottleneck

$$\begin{aligned} t_2 &= \frac{3}{10}7 + \frac{7}{10}3 = \frac{42}{10} = \frac{21}{5}\text{hour} \\ T_0 &= 2 + \frac{21}{5} + 2 \\ r_b &= \frac{10}{21} \\ W_0 &= r_b T_0. \end{aligned}$$

Note that we add capacity, so that we can process more jobs per unit time, but the raw processing time increases!

s.7.2.15. Now there are two loops. One-third of the jobs pass all three stations. Station 1 can process these jobs at rate

$$\frac{1}{3}r_1 = \frac{1}{3}\frac{1}{2} = \frac{1}{6}$$

per hour. Since this is smaller than r_2 and r_3, station 1 is the bottleneck rate for this loop. Thus, W_0 in this loop is $r_b T_0 = 1/6 \cdot 7 = 7/6$.

The other loop only contains station 1. The bottleneck rate in this loop is

$$r_1\frac{2}{3} = \frac{1}{2}\frac{2}{3} = \frac{1}{3}.$$

Thus, $W_0 = r_b T_0 = 1/3 \cdot 2 = 2/3$ for this loop.

Hence, the total WIP is $7/6 + 2/3 = 11/6$ and

$$T_0 = \frac{1}{3}7 + \frac{2}{3}2 = \frac{11}{3},$$

since 1/3 take the long loop with $T = 7$ and 2/3 the short loop with $T = 2$. Thus, T_0 is the weighted average raw processing time.

Another way to analyze this situation is like this. Station 1 is the bottleneck, because its utilization is 1, while

$$\begin{aligned} \rho_2 &= \lambda_2 t_2 = \frac{1}{2}\frac{1}{3}3 = \frac{1}{2} \\ \rho_3 &= \lambda_3 t_3 = \frac{1}{2}\frac{1}{3}2 = \frac{1}{3}. \end{aligned}$$

Since station 1 is the bottleneck in this network, $r_b = 1/2$. Since $T_0 = 11/3$, which we obtain as the weighted average computation above, we must have that

$$W_0 = r_b T_0 = \frac{1}{2}\frac{11}{3} = \frac{11}{6},$$

which agrees with our earlier result.

s.7.2.16. Now stations 1 and 3 are the bottlenecks with $r_b = 1/2$. For T_0 we have

$$T_0 = \frac{1}{5}7 + \frac{4}{5}4 = \frac{23}{5}.$$

Thus,

$$W_0 = r_b T_0 = \frac{1}{2}\frac{23}{5} = \frac{23}{10}.$$

s.7.2.17. Now stations 1 and 3 are the bottlenecks. Thus, both stations need to be part of the conwip loop. For that matter, it is just as easy to also include station 2, and just include all work on the floor in the WIP measurements.

s.7.3.1. Since $X \sim \text{Exp}(\lambda)$ and $S \sim \text{Exp}(\mu)$, and X and S are independent, their joint density is $f_{X,S}(x,y) = \lambda\mu e^{-\lambda x - \mu y}$. With this,

$$\begin{aligned}
\mathsf{P}\{X+S \le t\} &= \lambda\mu \int_0^\infty \int_0^\infty e^{-\lambda x - \mu y} \mathbb{1}\{x+y \le t\}\, \mathrm{d}x\, \mathrm{d}y \\
&= \lambda\mu \int_0^t \int_0^{t-x} e^{-\lambda x - \mu y}\, \mathrm{d}y\, \mathrm{d}x \\
&= \lambda\mu \int_0^t e^{-\lambda x} \int_0^{t-x} e^{-\mu y}\, \mathrm{d}y\, \mathrm{d}x \\
&= \lambda \int_0^t e^{-\lambda x}(1 - e^{-\mu(t-x)})\, \mathrm{d}x \\
&= \lambda \int_0^t e^{-\lambda x}\, \mathrm{d}x - \lambda e^{-\mu t} \int_0^t e^{(\mu-\lambda)x}\, \mathrm{d}x \\
&= 1 - e^{-\lambda t} - \frac{\lambda}{\mu-\lambda} e^{-\mu t}(e^{(\mu-\lambda)t} - 1) \\
&= 1 - e^{-\lambda t} - \frac{\lambda}{\mu-\lambda} e^{-\lambda t} + \frac{\lambda}{\mu-\lambda} e^{-\mu t} \\
&= 1 - \frac{\mu}{\mu-\lambda} e^{-\lambda t} + \frac{\lambda}{\mu-\lambda} e^{-\mu t}.
\end{aligned}$$

The density $f_{X+S}(t)$ is the derivative of this expression with respect to t, hence,

$$\begin{aligned}
f_{X+S}(t) &= \frac{\lambda\mu}{\mu-\lambda} e^{-\lambda t} - \frac{\mu\lambda}{\mu-\lambda} e^{-\mu t} \\
&= \frac{\lambda\mu}{\lambda-\mu}(e^{-\mu t} - e^{-\lambda t}).
\end{aligned}$$

Conditioning is much faster:

$$\begin{aligned} f_{X+S}(t) &= \mathsf{P}\{X+S \in \mathrm{d}t\} = \int_0^t \mathsf{P}\{S+x \in \mathrm{d}t\}\,\mathsf{P}\{X \in \mathrm{d}x\} \\ &= \int_0^t f_S(t-x) f_X(x)\,\mathrm{d}x = \int_0^t \mu e^{-\mu(t-x)} \lambda e^{-\lambda x}\,\mathrm{d}x \\ &= \lambda\mu e^{-\mu t} \int_0^t e^{x(\mu-\lambda)}\,\mathrm{d}x = \frac{\lambda\mu}{\mu-\lambda} e^{-\mu t}\left(e^{(\mu-\lambda)t}-1\right). \end{aligned}$$

s.7.3.2.

$$\begin{aligned} f_D(t) &= (1-\rho) f_{X+S}(t) + \rho\mu e^{-\mu t} = (1-\rho)\frac{\mu\lambda}{\lambda-\mu}\left(e^{-\mu t}-e^{-\lambda t}\right) + \rho\mu e^{-\mu t} \\ &= \left(1-\frac{\lambda}{\mu}\right)\frac{\mu\lambda}{\lambda-\mu}\left(e^{-\mu t}-e^{-\lambda t}\right) + \rho\mu e^{-\mu t} \\ &= \frac{\mu-\lambda}{\mu}\frac{\mu\lambda}{\lambda-\mu}\left(e^{-\mu t}-e^{-\lambda t}\right) + \frac{\lambda}{\mu}\mu e^{-\mu t} = -\lambda\left(e^{-\mu t}-e^{-\lambda t}\right) + \lambda e^{-\mu t} \end{aligned}$$

s.7.3.3.

$$P = \begin{pmatrix} \alpha & 1-\alpha \\ \beta_1 & \beta_2 \end{pmatrix}, \qquad (\lambda_1,\lambda_2) = (\gamma,0) + (\lambda_1,\lambda_2)P.$$

Solving first for λ_2 leads to $\lambda_2 = (1-\alpha)\lambda_1 + \beta_2\lambda_2$, so that

$$\lambda_2 = \frac{1-\alpha}{1-\beta_2}\lambda_1.$$

Next, using this and that $\lambda_1 = \alpha\lambda_1 + \beta_1\lambda_2 + \gamma$ gives

$$\begin{aligned} \gamma &= \lambda_1(1-\alpha) - \beta_1\lambda_2 = \lambda_1\left(1-\alpha-\beta_1\frac{1-\alpha}{1-\beta_2}\right) \\ &= \lambda_1(1-\alpha)\left(1-\frac{\beta_1}{1-\beta_2}\right) = \lambda_1(1-\alpha)\frac{1-\beta_1-\beta_2}{1-\beta_2}. \end{aligned}$$

Hence,

$$\lambda_1 = \frac{\gamma}{1-\alpha}\frac{1-\beta_2}{1-\beta_1-\beta_2}, \qquad \lambda_2 = \frac{1-\alpha}{1-\beta_2}\lambda_1 = \frac{\gamma}{1-\beta_1-\beta_2}.$$

We want of course that $\lambda_1 < \mu_1$ and $\lambda_2 < \mu_2$. With the above expressions this leads to conditions on α, β_1 and β_2. Note that we have three parameters, and two equations; there is no single condition that guarantees stability. If $\alpha \uparrow 1$, the arrival rate at node 1 explodes. If $\beta_1 = 0$ no jobs are sent from node 2 to node 1.

s.7.3.4. The rate into state $(0,0)$ is $\mu_2 p(0,1) = \mu_2\rho_2$. The rate out of state $(0,0)$ is $\lambda p(0,0) = \lambda$. Since $\rho_2 = \lambda/\mu_2$, these two rates are the same.

s.7.3.5.

$$\begin{aligned} \text{rate out } &= (\lambda+\mu_1)p(i,0) = \mu_1\rho_1^i + \lambda\rho_1^i = \lambda\rho_1^{i-1} + \mu_2\rho_1^i\rho_2 \\ &= \lambda p(i-1,0) + \mu_2 p(i,1) = \text{ rate in.} \end{aligned}$$

s.7.3.6. We show that the rate out is the rate in.

$$\begin{aligned}\text{rate out } &= (\lambda+\mu_2)p(0,j) = \mu_2\rho_2^j + \lambda\rho_2^j = \mu_1\rho_1\rho_2^{j-1} + \mu_2\rho_2^{j+1} \\ &= \mu_1 p(1,j-1) + \mu_2 p(0,j+1) = \text{ rate in.}\end{aligned}$$

s.7.4.1. $P_{i,i+1} = 1$, and $P_{ij} = 0$ for $j \neq i+1$. As P is a strictly upper-triangular matrix of dimension $M \times M$, $P^M = 0$.

s.7.4.2. Since P is transient and the number of states is finite, there is an M such that every row sum of $Q = P^M$ is strictly smaller than 1. Hence, for some $\epsilon > 0$, $Q\mathbf{1} \leq (1-\epsilon)\mathbf{1}$, where $\mathbf{1}$ is the vector of ones. By induction, $Q^k\mathbf{1} \leq (1-\epsilon)^k\mathbf{1}$. Now write $n = kM + r$, where $0 \leq r < M$. Since P is substochastic,

$$P^n\mathbf{1} = P^r Q^k \mathbf{1} \leq (1-\epsilon)^k P^r \mathbf{1} \leq (1-\epsilon)^{\lfloor n/M \rfloor}\mathbf{1}.$$

Thus, every entry of P^n converges to 0 exponentially fast.

s.7.4.3. The above argument is not necessarily valid for matrices P that are infinite, since $\inf\left\{P_{ij}^M\right\}$ need not be strictly positive for all i, j.

BIBLIOGRAPHY

S. Asmussen. *Applied Probability and Queues*. Springer-Verlag, Berlin, 2003.

J.C. Baez and J. Biamonte. Quantum techniques for stochastic mechanics, 2019.

P. Doyle and J. Laurie Snell. *Random Walks and Electrical Networks*. Mathematical Association of America, 1984.

M. El-Taha and S. Stidham Jr. *Sample-Path Analysis of Queueing Systems*. Kluwer Academic Publishers, 1998.

J.R. Norris. *Markov Chains*. Cambridge University Press, 1997.

A.A. Yushkevich and E.B. Dynkin. *Markov Processes: Theorems and Problems*. Plenum Press, 1969.

7.6 ARRIVALS AND EVENT PROCESSES

Symbol	Meaning
a_k	Arrivals in period k.
$A(t)$	Arrivals in $[0, t]$.
A_k	Arrival time of job k.
$D(t)$	Departures in $[0, t]$.
D_k	Departure time of job k.
d_k	Departures in period k.
δ	Departure rate.
λ	Arrival rate.
$N(t)$	Events of a Poisson process in $[0, t]$.
$N(s, t)$	Arrivals in $(s, t]$.
X	Generic inter-arrival time.
X_k	Inter-arrival time between jobs $k-1$ and k.

7.7 SERVICE AND CAPACITY

Symbol	Meaning
$\tilde{A}_k$	Start of service of job k.
B	Generic batch size.
B_k	Batch size at arrival time k.
c_k	Service or production capacity in period k.
c	Number of servers.
C	Generic batch size during a service time.
F	Service-time distribution.
μ	Service rate.
S	Generic service time.
S_k	Service time of job k.

7.8 STATE AND PERFORMANCE

Symbol	Meaning
I	Idle time of a single server.
J_k	Sojourn time of job k.
$L(t)$	Jobs in the system at time t.
L_k	Jobs in the system at the end of period k.
$L_s(t)$	Jobs in service at time t.
$p(n)$	Long-run time-average probability of n jobs.
$\pi(n)$	Stationary probability that an arrival sees n jobs.
$Q(t)$	Jobs in queue at time t.
Q_k	Queue length seen by job k, or at the end of period k.
ρ	Utilization of a single server.
U	Busy time of a single server.
W	Generic waiting time in queue.
W_k	Waiting time in queue of job k.

7.9 VARIABILITY

Symbol	Meaning
C_a^2	Squared coefficient of variation of inter-arrival times.
C_s^2	Squared coefficient of variation of service times.