# Optimal Decision Making via Stochastic Modeling and Machine Learning: Applications to Resource Allocation Problems and Sequential Decision Problems

Emmanuel C. Ekwedike

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Operations Research and Financial Engineering
Advisers: Han Liu and William Massey

September 2020

# Abstract

This thesis is about optimal decision making for resource allocation problems and sequential decision problems. There are two parts to this thesis: the first part focuses on developing a new stochastic analysis of queues and algorithms necessary to assist in the management of bike-sharing systems. The second part focuses on developing a general learning framework for sequential decision problems.

The first part proposes a new transient dynamics analysis of the bike-sharing station queueing model using complex analysis. Based on our queueing analysis, we develop a new algorithm for the optimal allocation of bikes in a bike-sharing system. Our algorithm takes in the rental rate, the return rate, and the number of docks as its input and outputs the optimal allocation given by our objective function. To demonstrate the practicality of our approach, extensive computational results are included.

For the second part, we propose a tree-based method for solving sequential decision problems using Monte Carlo Tree Search (MCTS). Our method works by iteratively applying MCTS on small, finite-horizon versions of the original infinite horizon Markov decision process. To demonstrate the efficiency of our approach, we provide the first sample complexity analysis of the batch MCTS-based Reinforcement Learning method. Lastly, we tested the neural network implementation of our methods in a challenging video game environment to demonstrate the power of our approach.

# Acknowledgements

I would like to thank my doctoral advisors: Han Liu and William Massey, without whom the completion of this dissertation would not have been possible. I benefited immensely from their unsurpassed knowledge; I could not have imagined having better advisors for my doctoral studies. My sincere gratitude to Prof. Liu for his motivation, enthusiasm, and support throughout my research and graduate studies. Furthermore, I am grateful to Prof. Massey for his unwavering support, immense knowledge and insightful comments throughout my studies.

This thesis was also shaped by the insightful suggestions of my committee members and readers. I am sincerely grateful to Professor Jamol Pender for not only serving on my dissertation committee but also advising part of my dissertation; his valuable suggestions greatly improved the quality of this dissertation. I also grateful to Professor Alain Kornhauser for serving on my dissertation committee; his valuable feedbacks greatly improved the quality of this dissertation. Moreover, I am truly grateful to Professors Conrad Tucker and Daniel Jiang for the precious time they spent on reading my thesis; their vast knowledge and thought-provoking comments have greatly improved my work. I would like to thank Professors Robert Hampshire and Alfred Noel for their thoughtful discussions and their support. Moreover, special gratitude goes to Tencent America LLC for generously sponsoring my dissertation and providing computational resources during my internship.

Thank you to the Department of Operations Research & Financial Engineering for creating an intellectually stimulating atmosphere in which I was able to complete my dissertation work. I am very thankful to Professors Robert Vanderbei, Mengdi Wang, Amir Ali Ahmadi, Ludovic Tangpi, John Mulvey, Samory Kpotufe, Warren Powell, Rene Carmona, and Ronnie Sircar who offered generous help along my graduate studies. I would also like to express my gratitude to the entire ORFE faculty and staff for their direct and indirect support. A special thank you to Kim Lupinacci

for her constant support and help with any administrative issues I have. Also thank you to Michael Bino for his constant tech support. My time at Princeton has been much more enjoyable thanks to an amazing group of friends and colleagues: Kobby, Parfait, Felix, Yair, Bachir, Mrs. Blessing, Mr. James, Levon, Pierre Yves, Kaizheng, Wenyan, Zongxi, Carson, Galen, and Raymond. To my fellow graduate students in the department and all my friends outside of the department; it is their encouragement that helped me achieve this far. They have left me many unforgettable memories.

Most importantly, I must express my very profound gratitude to my family for providing me with unfailing support and continuous encouragement throughout my studies. Special thank you to Chinyere, Godwin, Emeka, Benjamin, Ngozi, Deacon Okafor, Chioma and Roseline. This accomplishment would not have been possible without them. Finally, I dedicate this dissertation to my late father Chief Ambrose A. K. Ekwedike.

To Chief Ambrose A.K. Ekwedike.

# Contents

# List of Tables

# List of Figures

# Part I

# THESIS OVERVIEW

# Chapter 1

# Thesis Overview

## 1.1 Motivation and Thesis Statement

This thesis is about optimal decision making via stochastic modeling and machine learning. There is a growing demand for decision support systems. Many important problems involve decision making under uncertainty. These include inventory management, wildfire management, vehicle routing, supply chain management, self-driving cars, aircraft collision avoidance, and robot locomotion control systems (Kochenderfer et al., 2015; Sutton and Barto, 1998; Shalev-Shwartz and Ben-David, 2014). This thesis will focus on two application areas: resource allocation problems and sequential decision problems. The resource allocation problems and the sequential decision problems are connected since both involve decision making under uncertainty.

The first part of this thesis will explore the resource allocation problem. The resource allocation problems seek to find an optimal allocation of a fixed amount of resources to a given number of activities to minimize the cost incurred by the allocation. The number of resources to be allocated to each activity can be treated as a continuous variable, a discrete variable, or even both, depending on the application setting. The resource allocation problem is encountered in a variety of application

areas in operations research and management science, including load distribution, production planning, computer resource, portfolio selection, and apportionment. In this thesis, we will consider the inventory allocation problem in a bike-sharing system. Specifically, we consider a bike-share inventory management problem where the operator of the system must determine a cost-effective inventory allocation to meet the quality of service requirements at each location.

The second part of this thesis will explore sequential decision problems. Sequential decision problems describe a situation where the decision-maker makes successive observations of a process before a final decision is made. Sequential decision problems are typically modeled by the Markov decision processes. The main idea of the model is that a decision-maker, or an agent (computer algorithm), inhabits an environment, which changes the state randomly in response to action choices made by the decision-maker over time. The state of the environment affects the immediate reward obtained by the agent, as well as the probabilities of future state transitions. The decision-makers objective, or the agent's objective, is to select actions to maximize a long-term measure of total reward. The sequential decision-making problem is encountered in a variety of application areas in computer science, operations research, and finance, including robot locomotion control systems, aircraft collision avoidance, self-driving cars, personalized recommendation systems, games, bidding, and advertising.

Moreover, we consider the sequential decision problem in the context of stochastic games. Specifically, we consider the task of learning complex action control in the Multiplayer Online Battle Arena (MOBA) which is a challenging real-time strategy game. Anthony et al. (2017) proposes a general framework, for sequential decision problems, called *expert iteration* that combines supervised learning with tree search-based planning. The methods described in Guo et al. (2014), Silver et al. (2017), and the second part of this thesis can all be (at least loosely) expressed under the expert iteration framework. However, no theoretical insights were given in any of these

previous works and the second part of this thesis intends to fill this gap by providing a full theoretical analysis of an iterative, MCTS-based RL algorithm. Our analysis relies on the *concentrability coefficient* idea of Munos (2007) for approximate value iteration and builds upon the work on classification based policy iteration (Lazaric et al., 2016), approximate modified policy iteration (Scherrer et al., 2015), and fitted value iteration (Munos and Szepesvári, 2008).

This thesis has two main goals: The first goal is to develop a flexible optimization methodology for modeling resource allocation problems. We do this by understanding the transient distributions of the queueing models for resource allocation problems. We develop as our second goal a flexible learning methodology for solving sequential decision problems efficiently. Moreover, we give the first sample complexity analysis for the proposed methodology.

## 1.2 Main Results

In this section, we highlight the main results and provide a high-level summary of the contributions of this thesis.

**Contributions in PART I:**

We propose a new transient dynamics analysis of the bike-sharing station queueing model using complex analysis. Moreover, we present a method for approximating the transient probabilities dynamics for the non-constate rate queueing model for the bike-sharing station. Based on our queueing analysis, we develop a new algorithm for the optimal allocation of bikes in a bike-sharing system. Our algorithm takes in the rental rate, the return rate, and the number of docks as its input and outputs the optimal allocation given by our objective function. To demonstrate

the practicality of our approach, we give computational results using synthetic datasets.

**Contributions in PART II:**

First, we propose a batch, Monte Carlo Tree Search (MCTS) based reinforcement learning (RL) method that operates on the continuous state, finite action Markov Decision Processes (MDPs) and exploits the idea that leaf-evaluators can be updated to produce a stronger tree search using *previous tree search results*. Function approximators are used to track policy and value function approximations, where the latter is used to reduce the length of the tree search rollout (oftentimes, the rollout of the policy becomes a computational bottle-neck in complex environments).

Second, we demonstrate a supervised learning approach for learning complex policies for Multiplayer Online Battle Arena (MOBA) games with Parametrized Action Spaces from relatively small numbers of human demonstrations. Our technique uses a hybrid loss function to learn both continuous and discrete components of the policies. And show that a completely data-driven resampling technique can be used to significantly improve the performance of the learned policy. We tested our approach on a challenging video game environment, the popular MOBA game `King of Glory` (a North American version of the same game is titled `Arena of Valor`), where we build a competitive AI agent for the one-versus-one (1v1) mode of the game.

Third, we provide a full sample complexity analysis of the method and show that with large enough sample sizes and sufficiently large tree search effort, the performance of the estimated policies can be made close to optimal, up to some unavoidable approximation error. To our knowledge, batch MCTS-based RL methods have not been theoretically analyzed.

Finally, the feedback-based tree search algorithm is tested on the popular MOBA game `King of Glory` (a North American version of the same game is titled `Arena of Valor`), where we build a competitive AI agent for the 1v1 mode of the game.

## 1.3 Outline

We organize the rest of the thesis into the following chapters. Chapter 2 discusses optimal inventory repositioning in bike-sharing systems. Chapter 3 presents the supervised learning-based decision making for infinite horizon problems. This can be viewed as a queueing based decision making for the finite horizon decision problem. In Chapter 4, we present the MCTS-based decision making for infinite horizon problems. Moreover, in Appendix A, we present the supplementary materials to Chapter 2. In Appendix B, we present the supplementary materials to Chapter 3. In Chapter C, we present the supplementary materials to Chapter 4.

# Part II

# OPTIMIZATION FRAMEWORK FOR RESOURCE ALLOCATION PROBLEMS

# Chapter 2

# Optimal Inventory Repositioning in Bike-Sharing Systems

This chapter studies a new stochastic analysis of queues and presents the complete transient distribution analysis for the bike-sharing station queueing models. Based on our queueing analysis, we develop a new algorithm for the optimal allocation of bikes in a bike-sharing system. Our algorithm takes in the rental rate, the return rate, and the number of docks as its input and outputs the optimal allocation given by our objective function. To demonstrate the practicality of our approach, we give computational results using synthetic datasets.

The material in this chapter is joint work with Jamol Pender, Robert Hampshire, and William Massey.

## 2.1 Introduction

Bike-sharing systems (BSS) have gained immense popularity in the major cities around the world for more than a decade. The number of cities adopting bike-sharing programs has rapidly increased since 2007. BSS is operating in more than 1200 cities

worldwide and that number is projected to continue growing due to heightened interest in sustainable mobility in the urban cities (Fishman, 2016; DeMaio and Meddin, 2019; Tao and Pender, 2020). A station-based BSS typically consists of a map of stations.

The term repositioning/rebalancing in BSS refers to the movement of bicycles across the system to maintain a reasonable distribution across all docking stations Fishman et al. (2014). The uneven distribution of bicycles across the system is often caused when the flow of bike-sharing trips move from certain area of the city to another area. For example, in the morning peak hours, the flow of bike-sharing trips move from residential area to commercial zone and vice versa in the evening peak hour. This leads to the problem of some stations being empty while others are full. The periodic repositioning of inventory helps maintain even distribution of both bikes and docks in BSS. The demand for bikes is for those customers who wish to check out a bike; whereas the demand for docks is for those customers who which to return a bike at the BSS station and a station consists of a set of docks. A user is then allowed to rent a bike from any station and return the bike at any station in the system.

The unbalanced flow in the BSS network not only reduces the usability of certain stations as this could lead to a lack of reliability in service but it also imposes a cost on the bike-sharing operators, the party in-charge of managing and maintaining the BSS inventories, to manually redistribute the inventories Fishman et al. (2014). In managing a BSS, rebalancing operations account for a sizable percentage of the total operational costly; repositioning/rebalancing operation is the act of replenishing a station with bikes when it becomes empty and the act of removing bikes from a station when it is full. Ideally, you would like to rebalance as little as possible.

The ultimate goal of the bike-sharing system operators is two-fold: first is to maintain a desired quality of service in the system by minimizing the number of unsatisfied customers, that is the number of customers either blocked from retrieving

a bike at a station or blocked from returning a bike at a station. The second is to reduce costs due to frequent rebalancing of the inventories at each bike station in the system by optimally allocating initial inventory at each station in the remainder of the planning period. We develop a method to enable the operator of BSS to optimally allocate bikes to each station during the planning period of one day.

The contributions of this work can be summarized as follows. We propose a new stochastic analysis of queues and presents the complete transient distribution analysis for the bike-sharing station queueing models. Our approach bypasses the sample path argument, traditionally used to obtain the transient probabilities of the model, and reduce this analysis to simply computing a real integral. Based on our queueing analysis, we develop a new algorithm for the optimal allocation of bikes in a bike-sharing system. Our algorithm takes in the rental rate, the return rate, and the number of docks as its input and outputs the optimal allocation given by our objective function. To demonstrate the practicality of our approach, we give computational results using synthetic datasets.

**Notations:** We now briefly outline a commonly used notation for queueing system. Other notations will be defined as needed throughout the paper. A general notation for fully characterizing a queueing system is the Kendall's notation. Denote a queueing system by:

$$A/B/C/D/E, \tag{2.1.1}$$

where

$A$ : the arrival/inter-arrival time distribution

$B$ : the service times distribution

$C$ : number of servers in the system

$D$ : the capacity of the system. (i.e. the maximum number of customers in the system)

$E$ : the service discipline. (e.g. PS, FIFO and FCFS).

### 2.1.1 Related Work

Rebalancing inventory in a bike-sharing system has received considerable attention in the literature in recent years. In the literature, this problem is commonly known as the bike-sharing rebalancing, also known as bike-sharing repositioning problem (BRP). BRP typically consists of two parts: first, determining the desired inventory level at each bike station. Second, planning truck routes to pickup inventories from stations with surplus inventories and deliver them to stations that require replenishment. Both problems are incurred when a bike-sharing system is unbalanced. A burgeoning area of research in bike-sharing focuses on solving the twin problems. In their work, Schuijbroek et al. (2017) proposed a framework *cluster-first route-second huristic* that handles two problems of BRP of determining the service level requirements at each bike-sharing station and approximates routing costs for rebalancing the inventory. On the other hand, the idea of formulating the static bike repositioning (SBR) problem as a bilevel programming model was explored by Tang et al. (2019). The authors developed an iterated local search and tabu search to solve the bi-level programming model.

Our approach for determining the desired inventory level at each bike station is a variant (and in some respects, generalization) of the bi-level formulation in Tang et al.

(2019). The key differences are the following: (1) We further modeled the upper-level allocation problem as a queueing system and used queueing performance measures to define the user dis-satisfaction objective function as explored by (Raviv and Kolka, 2013; Raviv et al., 2014), and (2) we present a full analysis of the dynamics for the finite capacity queueing model used for modeling inventories at a single bike station in the BSS. Our approach bypasses the sample path argument, traditionally used to obtain the transient probabilities of the model, and reduce this analysis to simply computing a real integral.

Several studies in the literature focus on bike-sharing demand analysis, which consists of forecasting future demand. In their paper, Pan et al. (2019) proposes a real-time method for predicting bike renting and returning in different areas of a city during a future period based on historical data, weather data, and time data. Also, Wang and Kim (2018) mainly focused on the short-term forecasting for docking station usage. Demand analysis is useful in improving the quality of service objective function (user dissatisfaction function) used in this paper.

## 2.2 Spectral Analysis of the Transient Distribution of Queues

In this section, we introduce 5 queueing models summarized in Figure 2.1. Moreover, we also derive transition probabilities for each of the 5 queueing models. The purpose of introducing these models and deriving their transition probabilities is to eventually study bike-sharing systems.

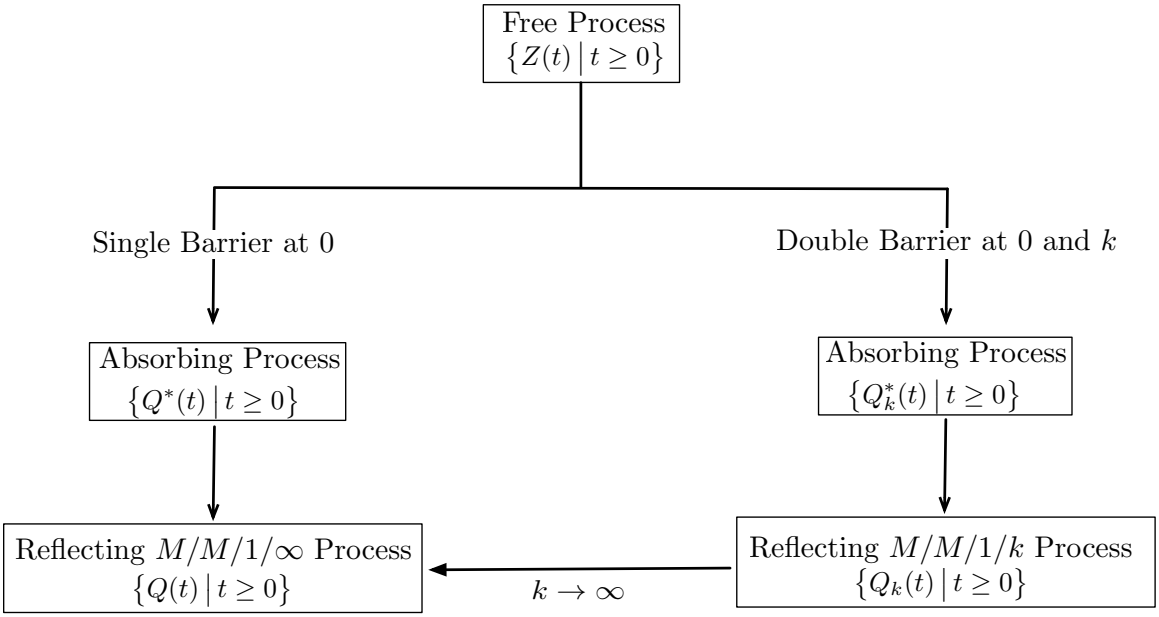



Figure 2.1: The relationship between the 5 queueing models studied in this chapter.

### 2.2.1 The Free Process

We first introduce an important process called the free process, also known as the zero-barrier free process. Studying this simple process is vital as it would help us further understand complicated processes.

**Definition 2.2.1.** The free process is the process $\{Z(t) \mid t \geq 0\}$ defined by

$$Z(t) = Z(0) + \Pi_\lambda(t) - \Pi_\mu(t), \tag{2.2.1}$$

where $\Pi_\lambda(t)$ and $\Pi_\mu(t)$ are two independent Poisson processes with rates $\lambda$ and $\mu$ respectively. The mean and variance of the free process are linear functions of the following form:

$$\mathbb{E}_m\big[Z(t)\big] = m + (\lambda - \mu)t \quad \text{and} \quad \mathrm{Var}\big[Z(t)\big] = (\lambda + \mu)t. \tag{2.2.2}$$

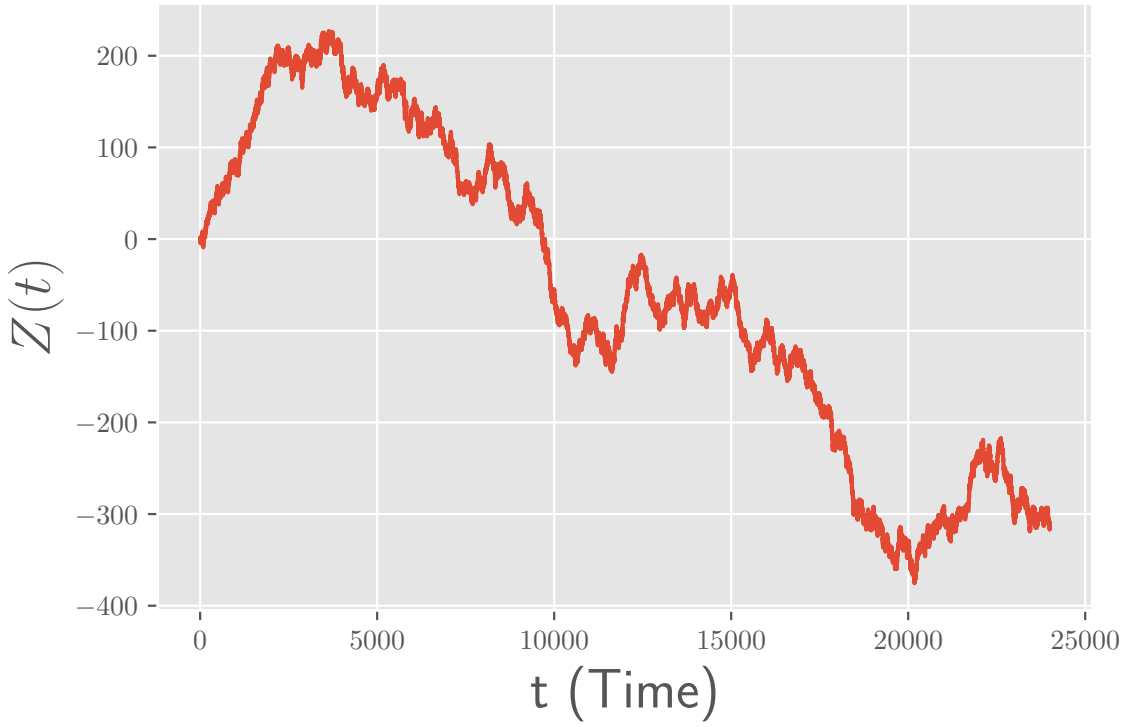



Figure 2.2: The realizations of the free process with $\lambda = \mu = 1$, $Z(0) = 0$, $\Delta t = 10^{-3}$, and $0 < t < 24,000$.

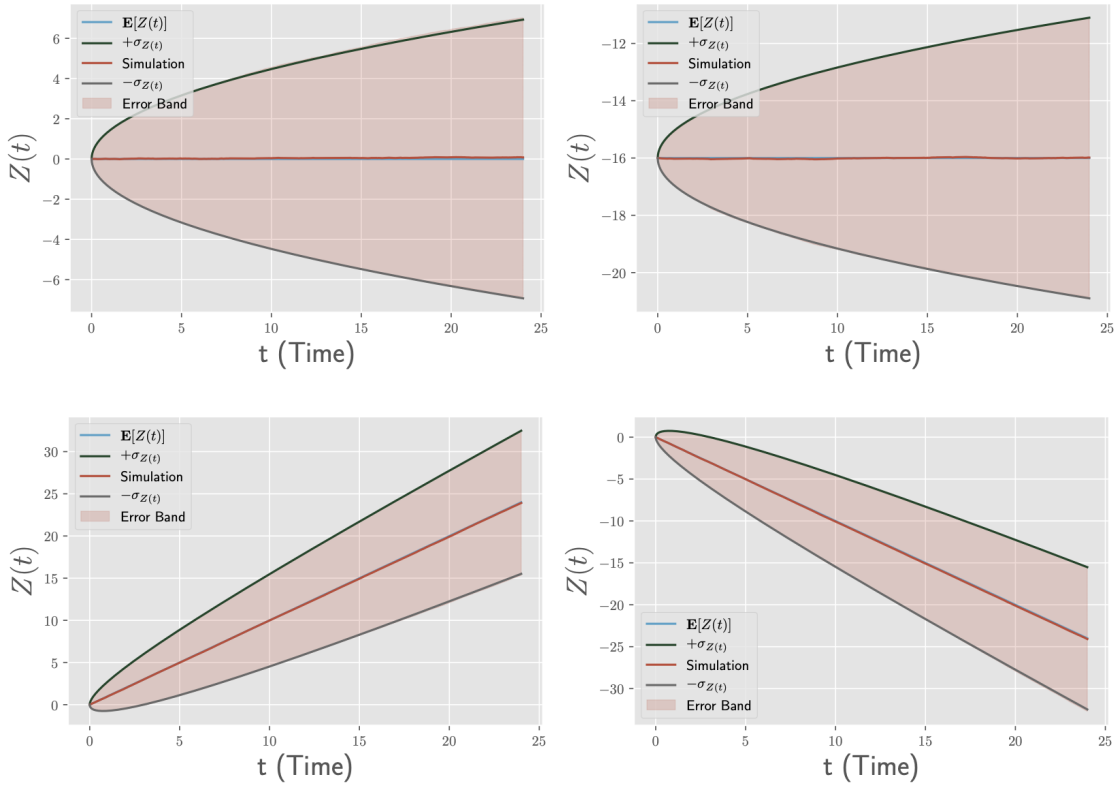



Figure 2.3: The plot of the steady-state mean $\mathbb{E}\big[Z(t)\big]$ and the steady-state standard deviation $\sigma_{Z(t)}$ of the free process. For the simulation, we set $\Delta t = 10^{-3}$, $N = 10^4$, and $0 < t < 24$. In (a) $\lambda = \mu = 1$, and $Z(0) = 0$. In (b) $\lambda = \mu = 0.5$, and $Z(0) = -16$. In (c) $\lambda = 2$, $\mu = 1$, and $Z(0) = 0$. In (d) $\lambda = 1$, $\mu = 2$, and $Z(0) = 0$.

In Figure 2.2, we show the realization of the free process, starting at the origin, as a function of time. As you can see, the free process is not constrained to either positive or negative states. To validate the simulation of the free process given by the Algorithm 1, we compare the simulation results with the steady-state closed-form mean and standard deviation of the free process as shown in Figure 2.3. The simulation results closely approximate the theoretical closed-form results.

**Algorithm 1:** The free process simulator

**Input:** Given arrival rate $\lambda$, service rate $\mu$, initial state $m$ and stopping time $T$.

**Output:** the free process $\boldsymbol{Z}$

Initialize time $t = 0$, starting state $z = m$, and create an empty list $\boldsymbol{Z}$

**while** $t \leq T$ **do**

  $U_1 \sim \mathcal{U}(0,1)$

  $t \leftarrow t - \frac{\log(U_1)}{\lambda+\mu}$

  **if** $t > T$ **then**

    Break

  **else**

    $U_2 \sim \mathcal{U}(0,1)$

    **if** $U_2 < \frac{\lambda}{\lambda+\mu}$ **then**

      $z = z + 1$

    **else**

      $z = z - 1$

    **end**

    $\boldsymbol{Z}$.append(z)

  **end**

**end**

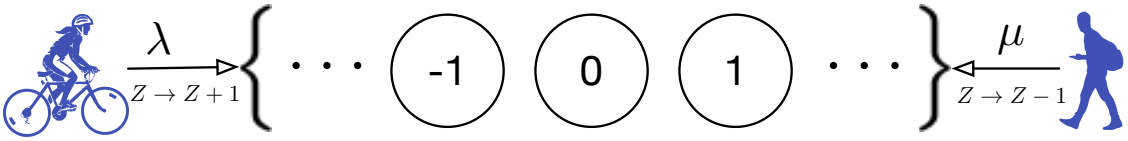



Figure 2.4: The transient states of the free process $Z(t)$. The values $\lambda$ and $\mu$ are the rates in which the state transitions in the positive direction and negative direction respectively.

The state-space of the free process is set $\mathbb{Z} = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$. Let $\mathbb{P}_m\Big\{Z(t) = n\Big\}$ be the transition probability for $Z(t)$ starting at state $m$ and terminating at state $n$, for time $t \geq 0$ and for $m, n \in \mathbb{Z}$. The transient states of the free process are given in Figure 2.4. It is well known, see (Ledermann et al., 1954; Baccelli and Massey, 1989), that the transition probabilities for the free process $\mathbb{P}_m\Big\{Z(t) = n\Big\}$ can be solved explicitly in terms of the modified Bessel functions. The exact solution for the transient distribution of the free process has been proved in the past using various advanced methods (Champernowne, 1956; Clarke, 1956; Ledermann et al., 1954; Baccelli and Massey, 1989). In their work, Baccelli and Massey (1989) describe $Z(t)$ as a nearest-neighbor random walk on the integers and derived the following result through sample-path arguments:

$$\mathbb{P}_m\Big\{Z(t) = n\Big\} \equiv \mathbb{P}\Big\{Z(t) = n\Big|Z(0) = m\Big\} = e^{-(\lambda+\mu)t} \cdot \left(\frac{\lambda}{\mu}\right)^{(n-m)/2} \cdot I_{n-m}\left(2t\sqrt{\lambda\mu}\right)$$

for all integers $m$ and $n$, where $I_n(\cdot)$ is the $n^{th}$ modified Bessel function as shown in Figure 2.5. The explicit formula above also describes processes associated with the $M_\lambda/M_\mu/1/\infty$ queue length process when we restrict $m, n \in \mathbb{Z}_+$ such that $Z(s) \neq 0$ for $0 \leq s \leq t$. Many explicit solutions in queueing theory involve Bessel functions.

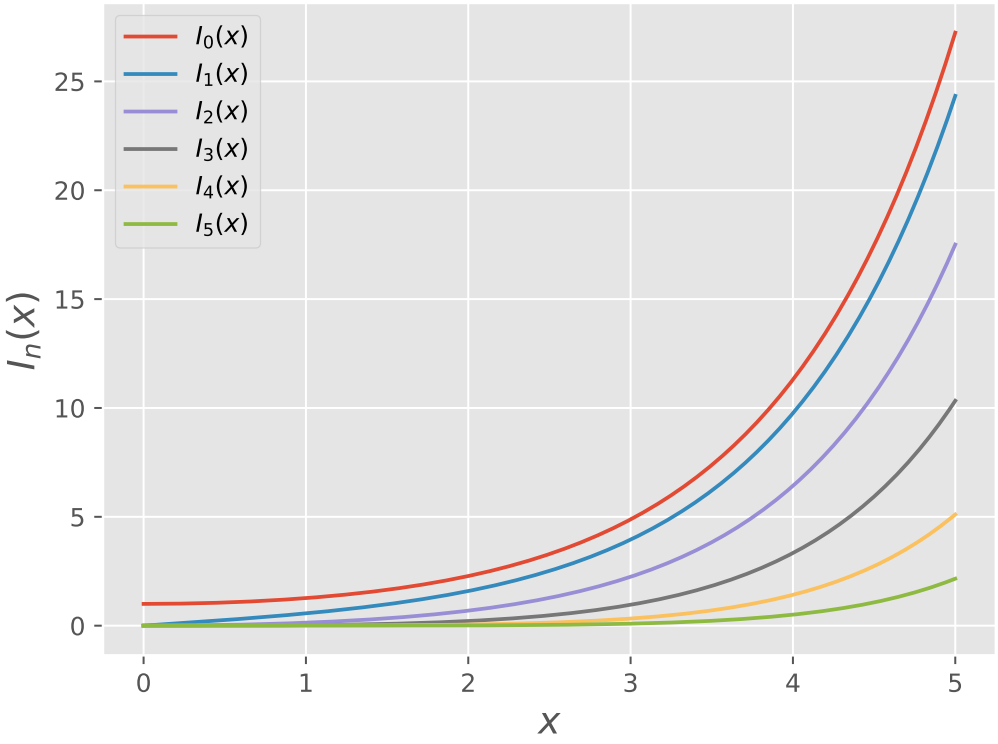



Figure 2.5: The first six modified Bessel functions

The modified Bessel function of order $n > -1$ for all real $x$ is defined as follows:

$$I_n(x) = \sum_{k=0}^{\infty} \frac{1}{k!\Gamma(k+n+1)} \cdot \left(\frac{x}{2}\right)^{2k+n}. \tag{2.2.3}$$

$I_n$ is often referred to as the modified Bessel function or Bessel function with imaginary argument. It turns out that the exact solution of the transition probabilities of the free process can be expressed in a simpler form. In what follows, we develop a new method for deriving the transition probability for the free process that bypasses the use of Bessel functions or transform methods such as Laplace transforms. In developing this method, we will exploit the natural group symmetries of the process and will use simple integration.

| | |
|---|---|
| $\alpha \equiv \frac{\lambda+\mu}{2}$ | $\delta(e^{i\theta}) = \cos\theta$ |
| $\beta \equiv \sqrt{\frac{\lambda}{\mu}}$ | $\delta(\omega) = \delta(\frac{1}{\omega})$ |
| $\gamma \equiv \sqrt{\lambda\mu}$ | $\epsilon_n(\omega) \equiv \frac{\omega^n - \frac{1}{\omega^n}}{2}$ |
| $\rho \equiv \beta^2 = \frac{\lambda}{\mu}$ | $\epsilon_n(\omega) = -\epsilon_n(\frac{1}{\omega})$ |
| $\delta(\omega) \equiv \frac{\omega+\frac{1}{\omega}}{2}$ | $\epsilon_n(1) = \epsilon_n(-1) = 0$ |
| $\gamma \cdot \delta(\beta) = \gamma \cdot \delta(1/\beta) = \alpha$ | $\frac{\epsilon_n(e^{i\theta})}{i} = \sin\theta$ |
| $\delta(1) = 1$ | |

Table 2.1: Useful parameters, functions, and identities

Next, we give a useful proposition and lemma's that would be helpful in the derivation of the transition probabilities.

**Proposition 2.2.1.** We have the following representation of the sum of $\epsilon_n(\cdot)$ function:

$$\sum_{\ell=1}^{n} a^{\ell} \cdot \epsilon_n(\omega) = \frac{\epsilon_1(\omega) + a^{n+1} \cdot \epsilon_n(\omega) - a^n \cdot \epsilon_{n+1}(\omega)}{2 \cdot \Big(\delta(a) - \delta(\omega)\Big)} \tag{2.2.4}$$

*Proof.*

$$\begin{aligned}
\sum_{\ell=1}^{n} a^{\ell} \cdot \epsilon_n(\omega) &= \frac{1}{2} \cdot \sum_{\ell=1}^{n} (a\omega)^{\ell} - (a/w)^{\ell} \\
&= \frac{1}{2} \cdot \left( \frac{1-(a\omega)^{n+1}}{1-a\omega} - \frac{1-(a/\omega)^{n+1}}{1-a/\omega} \right) \\
&= \frac{1}{2} \cdot \frac{\Big(1-(a\omega)^{n+1}\Big) \cdot \Big(1-a/\omega\Big) - \Big(1-a\omega\Big) \cdot \Big(1-(a/\omega)^{n+1}\Big)}{(1-a\omega) \cdot (1-a/\omega)} \\
&= \frac{1}{2} \cdot \frac{1-(a\omega)^{n+1} - a/\omega + a^2 \cdot (a\omega)^n - 1 + (a/\omega)^{n+1} + a\omega - a^2 \cdot (a/\omega)^n}{(1-a\omega) \cdot (1-a/\omega)} \\
&= \frac{a \cdot \epsilon_1(\omega) + a^{n+2} \cdot \epsilon_n(\omega) - a^{n+1} \cdot \epsilon_{n+1}(\omega)}{1 + a^2 - a \cdot (\omega + 1/\omega)} \\
&= \frac{a \cdot \epsilon_1(\omega) + a^{n+1} \cdot \epsilon_n(\omega) - a^{n} \cdot \epsilon_{n+1}(\omega)}{a + 1/a - (\omega + 1/\omega)} \\
&= \frac{\epsilon_1(\omega) + a^{n+1} \cdot \epsilon_n(\omega) - a^{n} \cdot \epsilon_{n+1}(\omega)}{2 \cdot \Big(\delta(a) - \delta(\omega)\Big)}
\end{aligned} \tag{2.2.5}$$

□

**Lemma 2.2.2.** The moment generating function for the free process $Z(t)$ has the form:

$$\mathbb{E}_m \left[ \left( \frac{\omega}{\beta} \right)^{Z(t)-n} \right] = \left( \frac{\omega}{\beta} \right)^{m-n} \cdot e^{-2\gamma \cdot \left(\delta(\beta) - \delta(\omega)\right) t} \tag{2.2.6}$$

*Proof.* From the functional Kolmogorov forward equation, we have

$$\frac{d}{dt} \mathbb{E}_m \left[ f(Z_t) \right] = \lambda \cdot \mathbb{E}_m \left[ f(Z_t + 1) - f(Z_t) \right] + \mu \cdot \mathbb{E}_m \left[ f(Z_t - 1) - f(Z_t) \right] \tag{2.2.7}$$

where $f(Z_t) = \left(\frac{\omega}{\beta}\right)^{Z_t - n}$ with $\beta = \sqrt{\frac{\lambda}{\mu}}$ and $\gamma = \sqrt{\lambda\mu}$. So $\lambda = \beta\gamma$ and $\mu = \frac{\gamma}{\beta}$, then the functional forward equation becomes

$$\begin{aligned}
\frac{d}{dt}\mathbb{E}_m\left[\left(\frac{\omega}{\beta}\right)^{Z(t)-n}\right] &= \lambda \cdot \mathbb{E}_m\left[\left(\frac{\omega}{\beta}\right)^{Z(t)+1-n} - \left(\frac{\omega}{\beta}\right)^{Z(t)-n}\right] \\
&\quad + \mu \cdot \mathbb{E}_m\left[\left(\frac{\omega}{\beta}\right)^{Z(t)-1-n} - \left(\frac{\omega}{\beta}\right)^{Z(t)-n}\right] \\
&= \beta\gamma \cdot \mathbb{E}_m\left[\left(\frac{\omega}{\beta}\right)^{Z(t)-n}\right] \cdot \left(\frac{\omega}{\beta} - 1\right) + \frac{\gamma}{\beta} \cdot \mathbb{E}_m\left[\left(\frac{\omega}{\beta}\right)^{Z(t)-n}\right] \cdot \left(\frac{\beta}{\omega} - 1\right) \\
&= \mathbb{E}_m\left[\left(\frac{\omega}{\beta}\right)^{Z(t)-n}\right] \cdot \left[\beta\gamma\left(\frac{\omega}{\beta} - 1\right) + \frac{\gamma}{\beta}\left(\frac{\beta}{\omega} - 1\right)\right] \\
&= \mathbb{E}_m\left[\left(\frac{\omega}{\beta}\right)^{Z(t)-n}\right] \cdot \left[\gamma\omega - \beta\gamma + \frac{\gamma}{\omega} - \frac{\gamma}{\beta}\right] \\
&= \mathbb{E}_m\left[\left(\frac{\omega}{\beta}\right)^{Z(t)-n}\right] \cdot \left[\gamma\left(\omega - \frac{1}{\omega}\right) - \left(\beta\gamma + \frac{\gamma}{\beta}\right)\right] \\
&= \mathbb{E}_m\left[\left(\frac{\omega}{\beta}\right)^{Z(t)-n}\right] \cdot \left[\gamma\left(\omega - \frac{1}{\omega}\right) - (\lambda + \mu)\right] \qquad (2.2.8) \\
&= \mathbb{E}_m\left[\left(\frac{\omega}{\beta}\right)^{Z(t)-n}\right] \cdot 2\left[\gamma\left(\frac{\omega - \frac{1}{\omega}}{2}\right) - \left(\frac{\lambda + \mu}{2}\right)\right] \\
&= \mathbb{E}_m\left[\left(\frac{\omega}{\beta}\right)^{Z(t)-n}\right] \cdot 2\Big[\gamma\delta(\omega) - \alpha\Big] \qquad (2.2.9) \\
&= \mathbb{E}_m\left[\left(\frac{\omega}{\beta}\right)^{Z(t)-n}\right] \cdot \Big[-2\big(\alpha - \gamma\delta(\omega)\big)\Big] \qquad (2.2.10) \\
&= \mathbb{E}_m\left[\left(\frac{\omega}{\beta}\right)^{Z(t)-n}\right] \cdot \Big[-2\gamma \cdot \big(\delta(\beta) - \delta(\omega)\big)\Big]
\end{aligned}$$

In Equation 2.2.8, we used the fact that $\lambda = \beta\gamma$ and $\mu = \frac{\gamma}{\beta}$ . Also in Equation 2.2.9, we used $\alpha = \frac{\lambda+\mu}{2}$ and $\delta(\omega) = \frac{\omega - \frac{1}{\omega}}{2}$. So we have the following differential equation

$$\frac{d}{dt}\mathbb{E}_m\left[\left(\frac{\omega}{\beta}\right)^{Z(t)-n}\right] = \mathbb{E}_m\left[\left(\frac{\omega}{\beta}\right)^{Z(t)-n}\right] \cdot \Big[-2\gamma \cdot \big(\delta(\beta) - \delta(\omega)\big)\Big]. \qquad (2.2.11)$$

Using the method of separation of variable, the solution of the above differential equation has the form:

$$\mathbb{E}_m\left[\left(\frac{\omega}{\beta}\right)^{Z(t)-n}\right] = C \cdot e^{-2\gamma\cdot\left(\delta(\beta)-\delta(\omega)\right)t}. \tag{2.2.12}$$

and the exact constant $C = \left(\frac{\omega}{\beta}\right)^{Z(0)-n}$ is obtained by setting $t = 0$ in the solution of the differential equation.

$$\mathbb{E}_m\left[\left(\frac{\omega}{\beta}\right)^{Z(t)-n}\right] = \left(\frac{\omega}{\beta}\right)^{Z(0)-n} \cdot e^{-2\gamma\cdot\left(\delta(\beta)-\delta(\omega)\right)t}. \tag{2.2.13}$$

□

Now that we have the moment generating function for the free process. We are ready to give the results for the transition probabilities for the free process. Before we do that, we define a few useful complex integration concepts.

**Definition 2.2.2** (General Cauchy's Theorem)**.**

$$\text{We have } \int_\gamma f(z)dz = 0 \text{ if either:} \tag{2.2.14}$$

1. The function $f$ has an analytic domain (complex differentiable) $\Omega$, where $\gamma = \partial\Omega$. Analytic domain is a complex differentiable function $f : \mathbb{C} \to \mathbb{C}$ over an open set $\Omega \subset \mathbb{C}$

2. The cycle (continuous function) $\gamma$ has an analytic funtion $F$ where $f = F'$. A cycle is a continuous funtion $\gamma : [0, 1] \to \mathbb{C}$ with the same boundary point $\gamma(0) = \gamma(1)$.

**Definition 2.2.3** (Cauchy Integral Formula)**.** Let $f$ have an analytic domain $\Omega$ with $\gamma = \partial\Omega$. For all $a \in \Omega$, we then have

$$f(a) = \frac{1}{2\pi i} \int_{\gamma} \frac{f(z)}{z-a} dz. \tag{2.2.15}$$

**Remark 2.2.1** (Integral Identity)**.** We have the following integral identity via the Cauchy Theorem and Integal formula:

$$\frac{1}{2\pi i} \int_{|z|=r} z^n \frac{dz}{z} = \frac{1}{2\pi i} \int_{|z|=r} z^{n-1} dz = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \ . \end{cases} \tag{2.2.16}$$

For all $n \neq 0$, $z^{n-1} = \frac{d}{dz}\frac{z^n}{n}$ and $\frac{z^n}{n}$ is analytic on $\{z \big| \, |z| = r\}$. Also notice that $z^{-1} = \frac{d}{dz} \log z$, but the logarithm cannot be single valued and continuous on $\{z \big| \, |z| = r\}$. Hence, it is not analytic here. For case of $n = 0$, just use the Cauchy integral formula. Moreover, we also have the following integral identity for the event $\{Z(t) = n\}$ :

$$\{Z(t) = n\} = \frac{1}{2\pi i} \oint_{|\omega|=r} \left(\frac{\omega}{\beta}\right)^{Z(t)-n} \frac{d\omega}{\omega} = \begin{cases} 1 & Z(t) = n \\ 0 & Z(t) \neq n \ . \end{cases} \tag{2.2.17}$$

When $Z(t) = n$ then following complex integral evaluates to 1:

$$\begin{aligned} \frac{1}{2\pi i} \oint_{|\omega|=r} \left(\frac{\omega}{\beta}\right)^{Z(t)-n} \frac{d\omega}{\omega} &= \frac{1}{2\pi i} \oint_{|\omega|=r} \frac{d\omega}{\omega} \\ &= \frac{1}{2\pi i} \int_0^{2\pi} \frac{1}{\omega(t)} \frac{d\omega(t)}{dt} dt \\ &= \frac{1}{2\pi i} \int_0^{2\pi} \frac{ire^{it}}{re^{it}} dt \\ &= \frac{2\pi i}{2\pi i} \\ &= 1. \end{aligned} \tag{2.2.18}$$

We used the fact that $\omega(t) = re^{it}$ for $0 \leq t \leq 2\pi$ where $r$ is any positive real number. Similarly when $Z(t) \neq n$, the complex integral evaluates to 0 by invoking the Cauchy's residues theorem Bak and Newman (1997). Hence, we have

$$\frac{1}{2\pi i} \oint_{|\omega|=r} \left(\frac{\omega}{\beta}\right)^{Z(t)-n} \frac{d\omega}{\omega} = 0 \tag{2.2.19}$$

Therefore the event $\{Z(t) = n\}$ can be represented as follows:

$$\{Z(t) = n\} = \frac{1}{2\pi i} \oint_{|\omega|=r} \left(\frac{\omega}{\beta}\right)^{Z(t)-n} \frac{d\omega}{\omega} \tag{2.2.20}$$

**Theorem 2.2.3** (Free Process Transition Probabilities as Complex Integrals)**.** The Free Process transition probabilities as complex integrals is given by the following integral:

$$\mathbb{P}_m\Big\{Z(t) = n\Big\} = \frac{\beta^{n-m}}{2\pi i} \cdot \oint_{|\omega|=r} \omega^{m-n} \cdot e^{-2\gamma \cdot \left(\delta(\beta)-\delta(\omega)\right)t} \frac{d\omega}{\omega} \tag{2.2.21}$$

*Proof.*

$$\begin{aligned}
\mathbb{P}_m\Big\{Z(t) = n\Big\} &= \mathbb{E}_m \left[\frac{1}{2\pi i} \oint_{|\omega|=r} \left(\frac{\omega}{\beta}\right)^{Z(t)-n} \frac{d\omega}{\omega}\right] \\
&= \frac{1}{2\pi i} \oint_{|\omega|=r} \mathbb{E}_m \left[\left(\frac{\omega}{\beta}\right)^{Z(t)-n}\right] \frac{d\omega}{\omega} \\
&= \frac{1}{2\pi i} \oint_{|\omega|=r} \left(\frac{\omega}{\beta}\right)^{m-n} \cdot e^{-2\gamma \cdot \left(\delta(\beta)-\delta(\omega)\right)t} \frac{d\omega}{\omega} \\
&= \frac{\beta^{n-m}}{2\pi i} \cdot \oint_{|\omega|=r} \omega^{m-n} \cdot e^{-2\gamma \cdot \left(\delta(\beta)-\delta(\omega)\right)t} \frac{d\omega}{\omega}
\end{aligned} \tag{2.2.22}$$

□

**Theorem 2.2.4** (Free Process Transition Probabilities as Real Integrals)**.** The Free Process transition probabilities as real integrals is given by the following integral

$$\mathbb{P}_m\Big\{Z(t)=n\Big\} = \frac{\beta^{n-m}}{\pi}\cdot\int_0^{\pi}\cos\Big(\big(m-n\big)\cdot\theta\Big)\cdot e^{-2\gamma\cdot\left(\delta(\beta)-\cos\theta\right)t}\,d\theta \tag{2.2.23}$$

*Proof.*

$$\begin{aligned}
\mathbb{P}_m\Big\{Z(t)=n\Big\} &= \mathbb{E}_m\left[\frac{1}{2\pi i}\oint_{|\omega|=r}\left(\frac{\omega}{\beta}\right)^{Z(t)-n}\frac{d\omega}{\omega}\right]\\
&= \frac{\beta^{n-m}}{2\pi i}\cdot\oint_{|\omega|=r}\omega^{m-n}\cdot e^{-2\gamma\cdot\left(\delta(\beta)-\delta(\omega)\right)t}\,\frac{d\omega}{\omega}\\
&= \frac{\beta^{n-m}}{2\pi i}\cdot\oint_{|\omega|=1}\frac{\omega^{m-n}+1/\omega^{m-n}}{2}\cdot e^{-2\gamma\cdot\left(\delta(\beta)-\delta(\omega)\right)t}\,\frac{d\omega}{\omega}\qquad\left(\text{since }\delta(\omega)=\delta(\frac{1}{\omega})\right)\\
&= \frac{\beta^{n-m}}{2\pi i}\cdot\oint_{|\omega|=1}\delta(\omega^{m-n})\cdot e^{-2\gamma\cdot\left(\delta(\beta)-\delta(\omega)\right)t}\,\frac{d\omega}{\omega}\\
&= \frac{\beta^{n-m}}{2\pi i}\int_{-\pi}^{\pi}\delta(e^{i(m-n)\theta})\cdot e^{-2\gamma\cdot\left(\delta(\beta)-\delta(e^{i\theta})\right)t}\,id\theta\\
&= \frac{\beta^{n-m}}{2\pi}\cdot\int_{-\pi}^{\pi}\cos\Big(\big(m-n\big)\cdot\theta\Big)\cdot e^{-2\gamma\cdot\left(\delta(\beta)-\delta(e^{i\theta})\right)t}\,d\theta\\
&= \frac{\beta^{n-m}}{\pi}\cdot\int_{0}^{\pi}\cos\Big(\big(m-n\big)\cdot\theta\Big)\cdot e^{-2\gamma\cdot\left(\delta(\beta)-\cos\theta\right)t}\,d\theta
\end{aligned}\tag{2.2.24}$$

In the above derivation, we set $\omega = r\cdot e^{i\theta}$. Then $|\omega| = |r\cdot e^{i\theta}| = r|e^{i\theta}| = r$, $\frac{d\omega}{\omega} = id\theta$ with $r=1$. Moreover, we also used the fact that $e^{i\theta}=\cos\theta + i\sin\theta$ and the real part of $e^{i\theta}$ is $\cos\theta$. We used the fact that cos is an even function to change the limit of integration. And since we are interested in probability we limit our attention to the unit circle by setting $r=1$, which is an optimal choice of contours. This essentially transforms this complex integral over the unit complex in $\mathbb{C}$ into the real integral over the real line $\mathbb{R}$. □

The major contribution in Theorems 2.2.3 and 2.2.4 is new derivation using complex analysis. Another closed-form representation of the free process transition probabilities exists Skellam (1946); Feller (1968). The free process transition probabilies

given in Theorem 2.2.4 has the following symmetries:

$$\mathbb{P}_m\big\{Z(t)=n\big\}=\mathbb{P}_{m+\ell}\big\{Z(t)=n+\ell\big\}=\mathbb{P}_{-n}\big\{Z(t)=-m\big\} \tag{2.2.25}$$

and

$$\beta^{m-n}\cdot\mathbb{P}_m\big\{Z(t)=n\big\}=\beta^{n-m}\cdot\mathbb{P}_n\big\{Z(t)=m\big\} \tag{2.2.26}$$

for all integers $m$ and $n$. Which implies

$$\frac{\mathbb{P}_m\big\{Z(t)=n\big\}}{\beta^{n-m}}=\frac{\mathbb{P}_{m+\ell}\big\{Z(t)=n+\ell\big\}}{\beta^{n-m}}=\frac{\mathbb{P}_{-n}\big\{Z(t)=-m\big\}}{\beta^{n-m}}=\frac{\mathbb{P}_n\big\{Z(t)=m\big\}}{\beta^{m-n}}$$

These symmetries would be useful in deriving the transition probabilities for the other processes we study in this chapter. Next, we study the single barrier absorbing process.

### 2.2.2 Single Barrier Absorbing Process

An absorbing process at the origin is a Markov chain in which it is impossible to leave the origin, and any state could reach the origin with some positive probability. The absorbing process is the same as the free process that starts either positive or negative and when the process hits zero, it stays there forever. The origin is called an absorbing state. We classify the state space $\mathbb{Z}=\{\ldots,-3,-2,-1,0,1,2,3,\ldots\}$ of the free process into two classes: Absorbing state $\{0\}$ and transient states $\{\ldots,-3,-2,-1\}$ and $\{1,2,3,\ldots\}$. So there are two transient states which depend on the starting state: if the process starts positive, then the transient states are $\{1,2,3,\ldots\}$. On the other hand, if the process starts on a negative state then the transient states are $\{\ldots,-3,-2,-1\}$.

**Positive starting state:** Our transportation interpretation of the absorbing process, in the context of public transportation serves, is a bike-sharing station where customers are infinitely patient and are waiting in line. And the bike station starts off with some people already in the line waiting for service. Figure 2.6 shows the transient states of the absorbing free process at the origin given a positive starting state. One interesting question is this setting is how long does it take to clear the queue, in other words, how long does it take to serve all waiting customer.

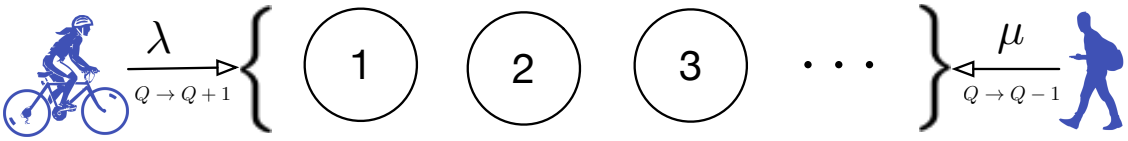



Figure 2.6: The transient states of the absorbing free process at the origin given a positive starting state. The values $\lambda$ and $\mu$ are the rates in which the state transitions in the positive direction and negative direction respectively.

**Negative starting state:** By symmetry of the free process, we can also talk about negative transient states. One interpretation of the absorbing process, in the context of public transportation serves, is a bike-sharing station where "negative customers" are infinitely patient and are waiting in line. In other words, bikes are sitting at the station waiting to be rented. And the bike station starts off with some initial bike already at the station waiting to be rented. Figure 2.7 shows the transient states of the absorbing free process at the origin given a negative starting state. One interesting question is this setting is how long does it take to clear the queue, in other words, how long does it take for the station to run out of bikes.

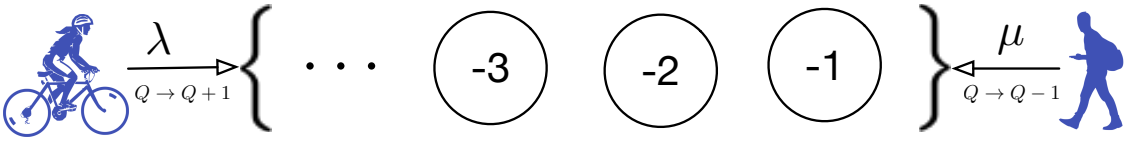



Figure 2.7: The transient states of the absorbing free process at the origin given a negative starting state. The values $\lambda$ and $\mu$ are the rates in which the state transitions in the positive direction and negative direction respectively.

**Definition 2.2.4** (Absorbing Process )**.** Given the free process $\{Z(t) \big| t \geq 0\}$ and the stopping time $\mathcal{T}_0 = \min\{t \big| Z(t) = 0\}$, define the single (barrier) absorbing process $\{Q^*(t) \big| t \geq 0\}$ to be

$$Q^*(t) \equiv Z\big(\mathcal{T}_0 \wedge t\big) \equiv Z\big(\min(\mathcal{T}_0, t)\big) \tag{2.2.27}$$

**Definition 2.2.5** (Single Barrier Absorbing Transition Probabilities)**.** The single barrier absorbing transition probabilitie are defined for all positive integers $m$ and $n$ to be

$$\begin{aligned}\mathbb{P}_m\{Q^*(t) = n\} &\equiv \mathbb{P}\{Q^*(t) = n | Q^*(0) = n\} \\ &= \begin{cases} \mathbb{P}_m\{Q^*(t) = n, \mathcal{T}_0 > t\} & \text{if } m > 0 \text{ and } n > 0, \\ 0 & \text{if } m = 0 \text{ and } n > 0, \\ \mathbb{P}_m\{\mathcal{T}_0 \leq t\} & \text{if } m > 0 \text{ and } n = 0, \\ 1 & \text{if } m = n = 0. \end{cases}\end{aligned}$$

$$\frac{d}{dt}\mathbb{P}_m\{\mathcal{T}_0 \leq t\} = \frac{d}{dt}\mathbb{P}_m\{Q^*(t) = 0\} = \mu \cdot \mathbb{P}_m\{Q^*(t) = 1\} \tag{2.2.28}$$

This implies that

$$\mathbb{P}_m\{\mathcal{T}_0 \leq t\} = \mu \cdot \int_0^t \mathbb{P}_m\{Q^*(s) = 1\}ds \tag{2.2.29}$$

Using linearity and symmetry, the absorbing process transition probabilities solve the same forward equations as the free process.

$$\mathbb{P}_m\{Q^*(t) = n\} = \mathbb{P}_m\{Z(t) = n\} - \beta^{-2m} \cdot \mathbb{P}_{-m}\{Z(t) = n\} \tag{2.2.30}$$

This implies

$$\begin{aligned}\frac{d}{dt}\mathbb{P}_m\{Q^*(t) = n\} =& \lambda \cdot \mathbb{P}_m\{Q^*(t) = n-1\} + \mu \cdot \mathbb{P}_m\{Q^*(t) = n+1\} \\ & - (\lambda + \mu) \cdot \mathbb{P}_m\{Q^*(t) = n\}\end{aligned}$$

and

$$\frac{d}{dt}\mathbb{P}_m\{Q^*(t) = 1\} = \mu \cdot \mathbb{P}_m\{Q^*(t) = 2\} - (\lambda + \mu) \cdot \mathbb{P}_m\{Q^*(t) = 1\} \tag{2.2.31}$$

Where Equation 2.2.31 is obtained using the following symmetries:

$$\beta^{-m-n} \cdot \mathbb{P}_{-m}\{Z(t) = n\} = \beta^{n+m} \cdot \mathbb{P}_n\{Z(t) = -m\} = \beta^{n+m} \cdot \mathbb{P}_m\{Z(t) = -n\}$$

which implies

$$\begin{aligned}\mathbb{P}_m\{Q^*(t) = n\} &= \mathbb{P}_m\{Z(t) = n\} - \beta^{2n} \cdot \mathbb{P}_m\{Z(t) = -n\} \\ \text{and } \mathbb{P}_m\{Q^*(t) = 0\} &= \mathbb{P}_m\{Z(t) = 0\} - \beta^0 \cdot \mathbb{P}_m\{Z(t) = 0\} = 0\end{aligned}$$

Similarly using linearity and symmetry, the absorbing process transition probabilities solve the same backward equations as the free process.

$$\mathbb{P}_m\{Q^*(t) = n\} = \mathbb{P}_m\{Z(t) = n\} - \beta^{2n} \cdot \mathbb{P}_m\{Z(t) = -n\} \tag{2.2.32}$$

This implies

$$\begin{aligned}\frac{d}{dt}\mathbb{P}_m\{Q^*(t) = n\} =& \lambda \cdot \mathbb{P}_{m+1}\{Q^*(t) = n\} + \mu \cdot \mathbb{P}_{m-1}\{Q^*(t) = n\} \\ & - (\lambda + \mu) \cdot \mathbb{P}_m\{Q^*(t) = n\}\end{aligned} \tag{2.2.33}$$

and

$$\frac{d}{dt}\mathbb{P}_1\{Q^*(t) = n\} = \lambda \cdot \mathbb{P}_2\{Q^*(t) = n\} - (\lambda + \mu) \cdot \mathbb{P}_1\{Q^*(t) = n\}$$

Where Equation 2.2.2 is obtained using the following symmetries:

$$\beta^{-m-n} \cdot \mathbb{P}_{-m}\{Z(t) = n\} = \beta^{n+m} \cdot \mathbb{P}_n\{Z(t) = -m\} = \beta^{n+m} \cdot \mathbb{P}_m\{Z(t) = -n\}$$

which implies

$$\begin{aligned}\mathbb{P}_m\{Q^*(t) = n\} &= \mathbb{P}_m\{Z(t) = n\} - \beta^{-2m} \cdot \mathbb{P}_{-m}\{Z(t) = n\} \\ \text{and } \mathbb{P}_0\{Q^*(t) = n\} &= \mathbb{P}_0\{Z(t) = n\} - \beta^0 \cdot \mathbb{P}_0\{Z(t) = n\} = 0\end{aligned}$$

Next, we give the transition probabilities result for the single barrier absorbing process. Moreover, we also give the absorbing process tail distribution.

**Theorem 2.2.5** (Single Barrier Absorbing Transition Probabilities as Complex Integrals)**.** The Absorbing Process transition probabilities as complex integrals is given

by the following integral:

$$\mathbb{P}_m\Big\{Q^*(t)=n\Big\} = -\frac{\beta^{n-m}}{\pi i}\cdot\oint_{|\omega|=1}\epsilon_m(\omega)\cdot\epsilon_n(\omega)\cdot e^{-2\gamma\cdot\left(\delta(\beta)-\delta(\omega)\right)t}\,\frac{d\omega}{\omega} \tag{2.2.34}$$

*Proof.*

$$\begin{aligned}
\mathbb{P}_m\Big\{Q^*(t)=n\Big\} &= \mathbb{P}_m\Big\{Z(t)=n\Big\} - \beta^{-2m}\cdot\mathbb{P}_{-m}\Big\{Z(t)=n\Big\} \\
&= \frac{1}{2\pi i}\cdot\oint_{|\omega|=r}\left[\left(\frac{\omega}{\beta}\right)^{m-n} - \beta^{-2m}\cdot\left(\frac{\omega}{\beta}\right)^{-m-n}\right]\cdot e^{-2\gamma\cdot\left(\delta(\beta)-\delta(\omega)\right)t}\,\frac{d\omega}{\omega} \\
&= \frac{\beta^{-m}}{2\pi i}\cdot\oint_{|\omega|=r}\left(\omega^m-\omega^{-m}\right)\cdot\left(\frac{\omega}{\beta}\right)^{-n}\cdot e^{-2\gamma\cdot\left(\delta(\beta)-\delta(\omega)\right)t}\,\frac{d\omega}{\omega} \\
&= \frac{\beta^{n-m}}{\pi i}\cdot\oint_{|\omega|=r}\epsilon_m(\omega)\cdot\omega^{-n}\cdot e^{-2\gamma\cdot\left(\delta(\beta)-\delta(\omega)\right)t}\,\frac{d\omega}{\omega} \\
&= \frac{\beta^{n-m}}{\pi i}\cdot\oint_{|\omega|=1}\epsilon_m(\omega)\cdot\frac{\omega^{-n}-\omega^n}{2}\cdot e^{-2\gamma\cdot\left(\delta(\beta)-\delta(\omega)\right)t}\,\frac{d\omega}{\omega} \\
&= -\frac{\beta^{n-m}}{\pi i}\cdot\oint_{|\omega|=1}\epsilon_m(\omega)\cdot\epsilon_n(\omega)\cdot e^{-2\gamma\cdot\left(\delta(\beta)-\delta(\omega)\right)t}\,\frac{d\omega}{\omega}
\end{aligned} \tag{2.2.35}$$

In the above equations, we performed the following transformation $\omega \leftrightarrow 1/\omega$ using the symmetry functions $\delta(\omega)=\delta(1/\omega)$ and $\epsilon_n(\omega)=\epsilon_n(1/\omega)$. □

**Theorem 2.2.6** (Single Barrier Absorbing Transition Probabilities as Real Integrals)**.** The Absorbing Process transition probabilities as complex integrals is given by the following integral:

$$\mathbb{P}_m\Big\{Q^*(t)=n\Big\} = \frac{\beta^{n-m}}{\pi}\cdot\int_{-\pi}^{\pi}\sin m\theta\cdot\sin n\theta\cdot e^{-2\gamma\cdot\left(\delta(\beta)-\cos\theta\right)t}\,d\theta \tag{2.2.36}$$

*Proof.*

$$\begin{aligned}
\mathbb{P}_m\Big\{Q^*(t)=n\Big\} &= -\frac{\beta^{n-m}}{\pi i}\cdot\oint_{|\omega|=1}\epsilon_m(\omega)\cdot\epsilon_n(\omega)\cdot e^{-2\gamma\cdot\Big(\delta(\beta)-\delta(\omega)\Big)t}\,\frac{d\omega}{\omega}\\
&= -\frac{\beta^{n-m}}{\pi i}\cdot\int_{-\pi}^{\pi}\epsilon_m(e^{i\theta})\cdot\epsilon_n(e^{i\theta})\cdot e^{-2\gamma\cdot\Big(\delta(\beta)-\delta(e^{i\theta})\Big)t}\,\frac{ie^{i\theta}d\theta}{e^{i\theta}}\\
&= \frac{\beta^{n-m}}{\pi}\cdot\int_{-\pi}^{\pi}\sin m\theta\cdot\sin n\theta\cdot e^{-2\gamma\cdot\Big(\delta(\beta)-\cos\theta\Big)t}\,d\theta \qquad (2.2.37)
\end{aligned}$$

□

**Theorem 2.2.7** (Single Barrier Absorbing Time Distribution as Complex Integrals)**.** The Absorbing Process tail distribution as complex integrals is given by the following integral:

$$\mathbb{P}_m\Big\{\mathcal{T}_0>t\Big\}=\left(1-\frac{1}{\rho^m}\right)^+-\frac{\beta^{-m}}{\pi i}\cdot\oint_{|\omega|=1}\frac{\epsilon_m(\omega)\cdot\epsilon_1(\omega)\cdot e^{-2\gamma\cdot\Big(\delta(\beta)-\delta(\omega)\Big)t}}{\delta(\beta)-\delta(\omega)}\,\frac{d\omega}{\omega}$$

*Proof.*

$$\begin{aligned}
\mathbb{P}_m\Big\{\mathcal{T}_0 > t\Big\} &= \sum_{n=0}^{\infty}\left[\frac{\beta^{-m}}{\pi i}\cdot\oint\limits_{|\omega|=r>\beta}\epsilon_m(\omega)\cdot\left(\frac{\omega}{\beta}\right)^{-n}\cdot e^{-2\gamma\cdot\Big(\delta(\beta)-\delta(\omega)\Big)t}\,\frac{d\omega}{\omega}\right] \\
&= \frac{\beta^{-m}}{\pi i}\cdot\oint\limits_{|\omega|=r>\beta}\epsilon_m(\omega)\cdot\sum_{n=0}^{\infty}\left(\frac{\beta}{\omega}\right)^{n}\cdot e^{-2\gamma\cdot\Big(\delta(\beta)-\delta(\omega)\Big)t}\,\frac{d\omega}{\omega} \\
&= \frac{2\beta^{-m}}{2\pi i}\cdot\oint\limits_{|\omega|=r>\beta}\epsilon_m(\omega)\cdot\frac{1}{1-\beta/\omega}\cdot e^{-2\gamma\cdot\Big(\delta(\beta)-\delta(\omega)\Big)t}\,\frac{d\omega}{\omega} \\
&= 2\beta^{-m}\cdot\epsilon(\beta)^{+} + \frac{2\beta^{-m}}{2\pi i}\cdot\oint\limits_{|\omega|=1}\frac{\epsilon_m(\omega)\cdot e^{-2\gamma\cdot\Big(\delta(\beta)-\delta(\omega)\Big)t}}{1-\beta/\omega}\,\frac{d\omega}{\omega} \\
&= \left(1-\frac{1}{\beta^{2m}}\right)^{+} + \frac{\beta^{-m}}{2\pi i}\cdot\oint\limits_{|\omega|=1}\epsilon_m(\omega)\left(\frac{1}{1-\beta/\omega}-\frac{1}{1-\beta\omega}\right)\cdot e^{-2\gamma\cdot\Big(\delta(\beta)-\delta(\omega)\Big)t}\,\frac{d\omega}{\omega} \\
&= \left(1-\frac{1}{\beta^{2m}}\right)^{+} + \frac{\beta^{-m}}{2\pi i}\cdot\oint\limits_{|\omega|=1}\epsilon_m(\omega)\cdot\frac{\beta\cdot(1/\omega-\omega)}{(1-\beta/\omega)\cdot(1-\beta\omega)}\cdot e^{-2\gamma\cdot\Big(\delta(\beta)-\delta(\omega)\Big)t}\,\frac{d\omega}{\omega} \\
&= \left(1-\frac{1}{\rho^{m}}\right)^{+} - \frac{\beta^{-m}}{\pi i}\cdot\oint_{|\omega|=1}\frac{\epsilon_m(\omega)\cdot\epsilon_1(\omega)\cdot e^{-2\gamma\cdot\Big(\delta(\beta)-\delta(\omega)\Big)t}}{\delta(\beta)-\delta(\omega)}\,\frac{d\omega}{\omega}
\end{aligned}$$

□

**Theorem 2.2.8** (Absorbing Process Tail Distribution as Real Integrals)**.** The Absorbing Process tail distribution as real integrals is given by the following integral:

$$\mathbb{P}_m\Big\{\mathcal{T}_0 > t\Big\} = \left(1-\frac{1}{\rho^{m}}\right)^{+} + \frac{\beta^{-m}}{\pi}\cdot\int_{-\pi}^{\pi}\frac{\sin m\theta\cdot\sin\theta\cdot e^{-2\gamma\cdot\Big(\delta(\beta)-\cos\theta\Big)t}}{\delta(\beta)-\cos\theta}\,d\theta \quad (2.2.38)$$

*Proof.*

$$\begin{aligned}
\mathbb{P}_m\Big\{\mathcal{T}_0 > t\Big\} &= \left(1 - \frac{1}{\rho^m}\right)^+ - \frac{\beta^{-m}}{\pi i} \cdot \oint_{|\omega|=1} \frac{\epsilon_m(\omega) \cdot \epsilon_1(\omega) \cdot e^{-2\gamma \cdot \left(\delta(\beta) - \delta(\omega)\right) t}}{\delta(\beta) - \delta(\omega)} \frac{d\omega}{\omega} \\
&= \left(1 - \frac{1}{\rho^m}\right)^+ - \frac{\beta^{-m}}{\pi} \cdot \oint_{|\omega|=1} \frac{\epsilon_m(e^{i\theta}) \cdot \epsilon_1(e^{i\theta}) \cdot e^{-2\gamma \cdot \left(\delta(\beta) - \delta(e^{i\theta})\right) t}}{\delta(\beta) - \delta(e^{i\theta})} d\theta \\
&= \left(1 - \frac{1}{\rho^m}\right)^+ + \frac{\beta^{-m}}{\pi} \cdot \int_{-\pi}^{\pi} \frac{\sin m\theta \cdot \sin\theta \cdot e^{-2\gamma \cdot \left(\delta(\beta) - \cos\theta\right) t}}{\delta(\beta) - \cos\theta} d\theta
\end{aligned}$$

Moreover,

$$\min(\rho^{-m}, 1) = \frac{\beta^{-m}}{\pi i} \cdot \oint_{|\omega|=1} \frac{\epsilon_m(\omega) \cdot \epsilon_1(\omega)}{\delta(\beta) - \delta(\omega)} \frac{d\omega}{\omega} = -\frac{\beta^{-m}}{2\pi} \cdot \int_{-\pi}^{\pi} \frac{\sin m\theta \cdot \sin\theta}{\delta(\beta) - \cos\theta} d\theta. \quad (2.2.39)$$

□

Theorems 2.2.5, 2.2.6, 2.2.7, and 2.2.8 are special case of Baccelli et al. (1994). The major contribution is new derivation using complex analysis.

### 2.2.3 $M_\lambda/M_\mu/1/\infty$ Queue Length (Reflecting) Process

Now consider the free process that gets reflected whenever it hits the origin and then either gets constrained to either the positive integers $\mathbb{Z}_+ \equiv \{0, 1, 2, \ldots\}$ or the negative integers $\mathbb{Z}_- \equiv \{\ldots, -2, -1, 0\}$.

**Constrained on positive integers:** Our transportation interpretation of the reflecting process is the same as the absorbing process except that we now added the zero state. In the context of public transportation serves the reflecting process could represent a bike-sharing station where customers are infinitely patient and are waiting in line. And the bike station starts off with some people already in the line waiting for service. Figure 2.8 shows the transient states of the absorbing free process at the origin given a positive starting state. One interesting question is this setting is how

long does it take to clear the queue, in other words, how long does it take to serve all waiting customer.

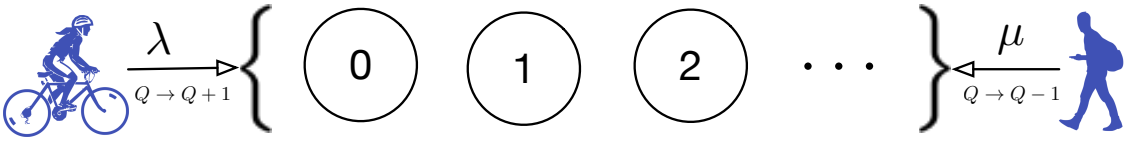



Figure 2.8: The reflecting process at the origin constrained on the positive integers. The values $\lambda$ and $\mu$ are the rates in which the state transitions in the positive direction and negative direction respectively.

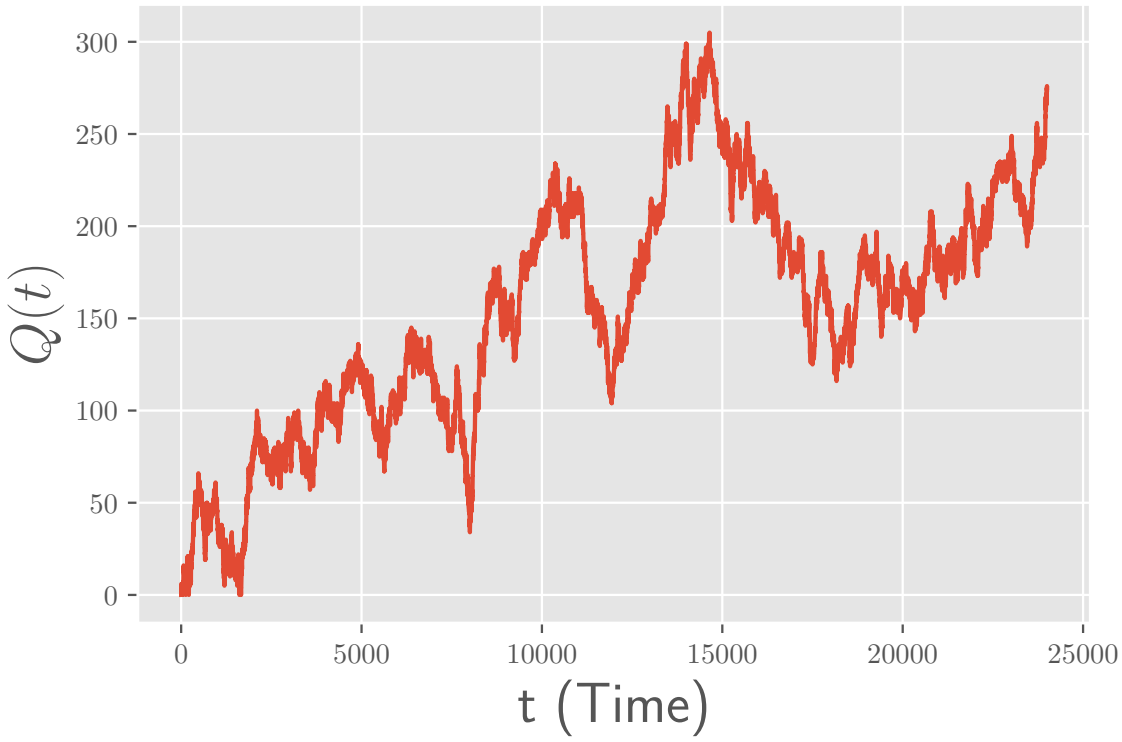



Figure 2.9: The realizations of the reflected free process constrained on the positive integers with $\lambda = \mu = 1$, $Z(0) = 0$, and $0 < t < 24,000$.

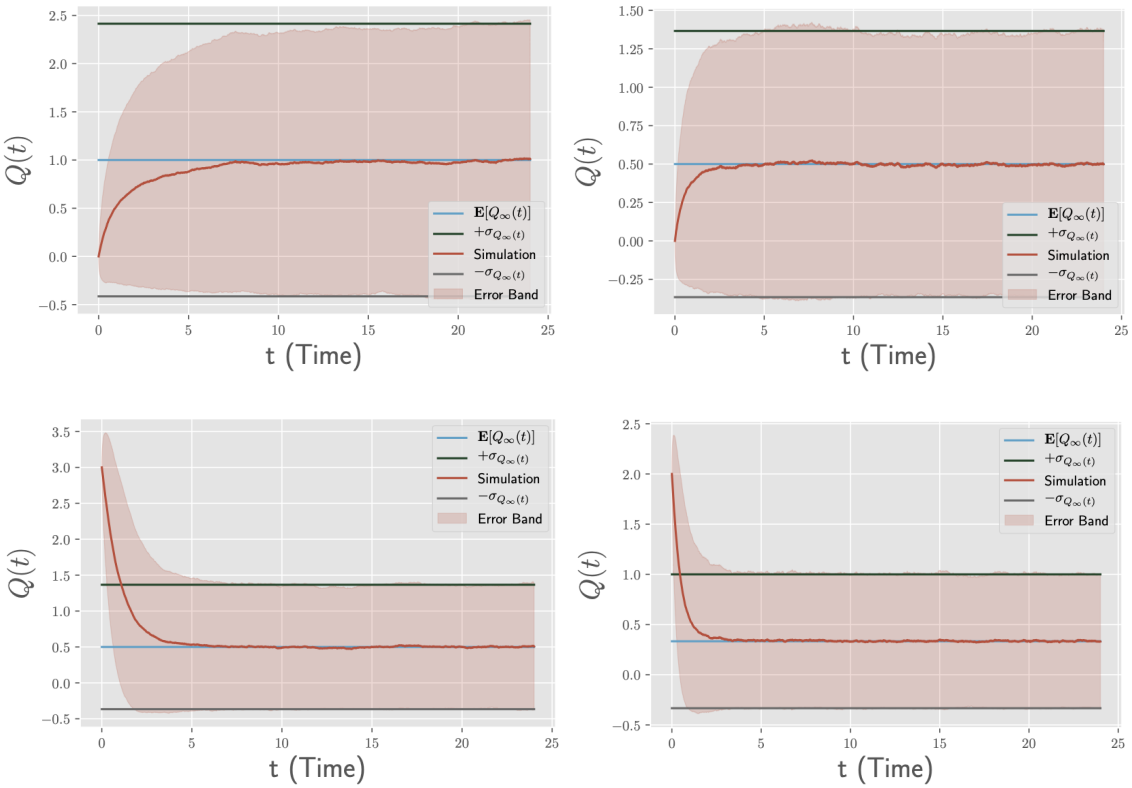



Figure 2.10: The plot of the steady-state mean and standard deviation of the reflected free process constrained on the positive intergers where $\mathbb{E}\big[Q_\infty(t)\big]$ is the steady-state mean, and $\sigma_{Q_\infty}$ is the steady-state standard deviation. For the simulation $\Delta t = 10^{-3}$, $N = 10^4$ and $0 < t < 24$. In (a) $\lambda = 1$, $\mu = 2$ and $Q(0) = 0$. In (b) $\lambda = 1$, $\mu = 3$ and $Q(0) = 0$. In (c) $\lambda = 1$, $\mu = 3$ and $Q(0) = 3$. In (d) $\lambda = 1$, $\mu = 4$ and $Q(0) = 2$.

**Constrained on negative integers:** By symmetry of the free process, we can also talk about the free process constrained on the negative integers. The interpretation of the reflecting process is the same as the absorbing process except that we now added the zero state. In the context of public transportation serves, the reflecting process could represent a bikes-hare station where "negative customers" are infinitely patient and are waiting in line. In other words, bikes are sitting at the station waiting to be rented. And the bike station starts off with some initial bike already at the station waiting to be rented. Figure 2.11 shows the transient states of the absorbing Free

Process at the origin given a negative starting state. One interesting question is this setting is how long does it take to clear the queue, in other words, how long does it take for the station to run out of bikes.

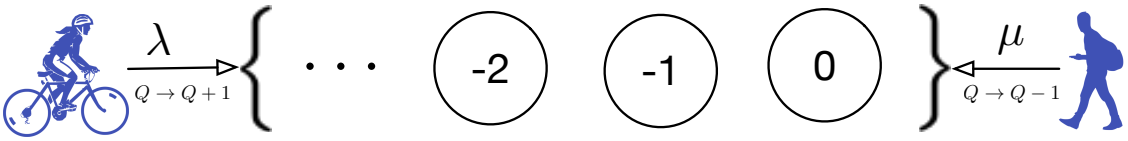



Figure 2.11: The reflecting process at the origin constrained on the negative integers. The values $\lambda$ and $\mu$ are the rates in which the state transitions in the positive direction and negative direction respectively.

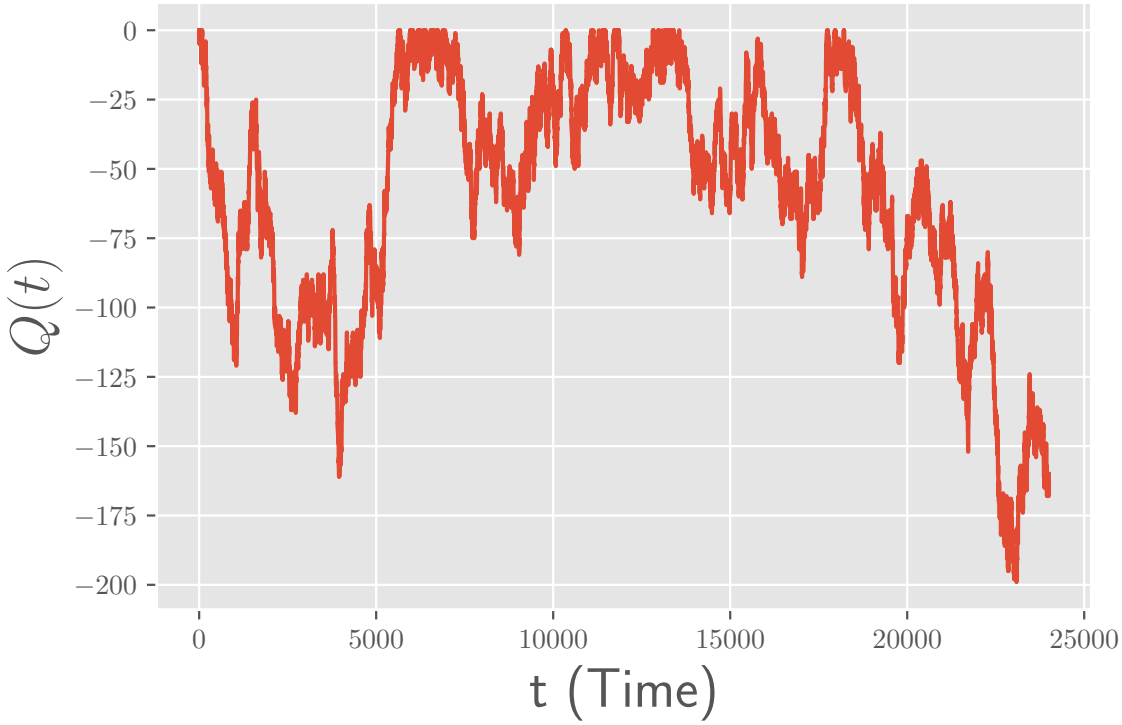



Figure 2.12: The realizations of the reflected free process constrained on the positive integers with $\lambda = \mu = 1$ and $Z(0) = 0$, and $0 < t < 24,000$.

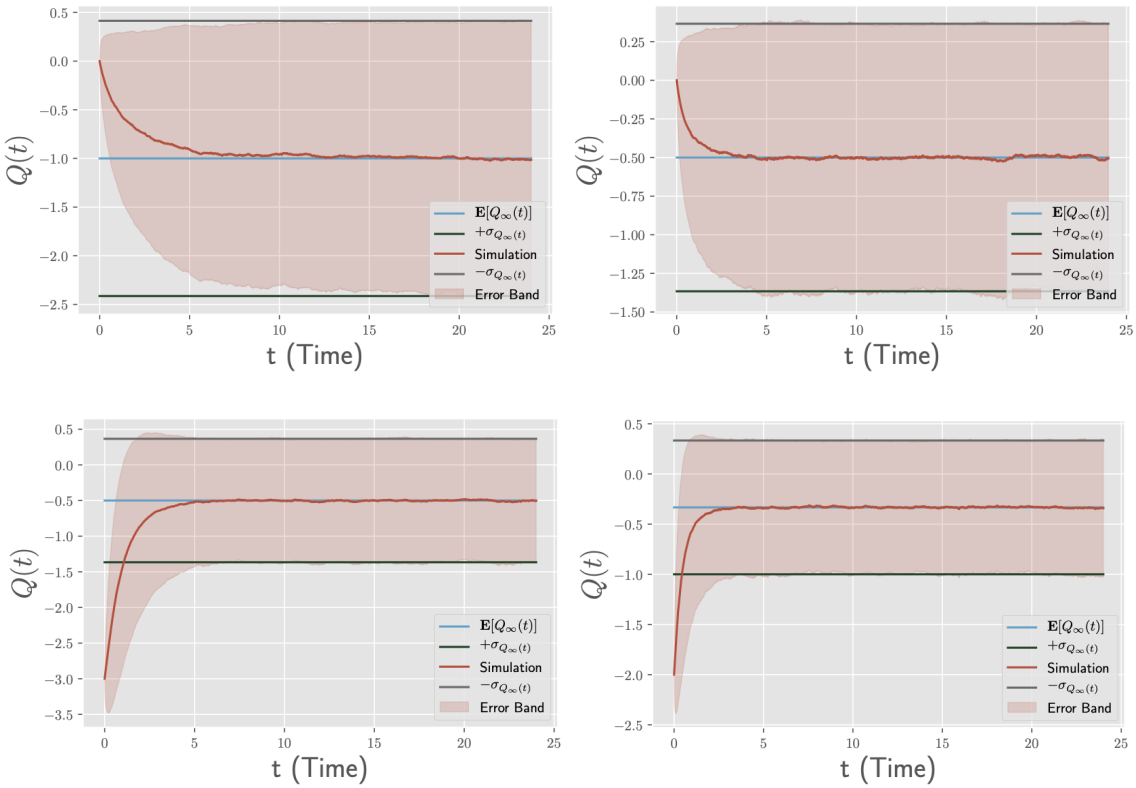



Figure 2.13: The plot of the steady-state mean and standard deviation of the reflected free process constrained on the negative intergers where $\mathbb{E}\big[Q_\infty(t)\big]$ is the steady-state mean, and $\sigma_{Q_\infty}$ is the steady-state standard deviation. For the simulation $\Delta t = 10^{-3}$, $N = 10^4$ and $0 < t < 24$. In (a) $\lambda = 2$, $\mu = 1$ and $Q(0) = 0$. In (b) $\lambda = 3$, $\mu = 1$ and $Q(0) = 0$. In (c) $\lambda = 3$, $\mu = 1$ and $Q(0) = -3$. In (d) $\lambda = 3$, $\mu = 1$ and $Q(0) = -2$.

**Algorithm 2:** The $M/M/1/\infty$ queue simulator

**Input:** Given arrival rate $\lambda$, service rate $\mu$, initial state $m$, and stopping time $T$.

**Output:** queue length process $\boldsymbol{Q}$

Initialize time $t = 0$, starting state $q = m$, and create an empty list $\boldsymbol{Q}$
**while** $t \leq T$ **do**
  $U_1 \sim \mathcal{U}(0, 1)$
  $t \leftarrow t - \frac{\log(U_1)}{\lambda+\mu}$
  **if** $t > T$ **then**
    Break
  **else**
    $U_2 \sim \mathcal{U}(0, 1)$
    **if** $U_2 < \frac{\lambda}{\lambda+\mu}$ **then**
      $q = q + 1$
    **else**
      $q = \max(q - 1, 0)$
    **end**
    $\boldsymbol{Q}$.append(q)
  **end**
**end**

Figures 2.9 and 2.12 show the realization of the single barrier reflecting process, constrained on positive and negative integers respectively. To validate the simulation of the single barrier reflecting process given by the Algorithm 2, we compare the simulation results with the steady-state closed-form mean and standard deviation of the free process as shown in Figures 2.10 and 2.13. As you can see, the simulation results

closely approximate the theoretical closed-form results. We have the following probabilistic symmetry (generally proven in Massey (1987)), also known as the reflection around the boundary point for $Z(0) = 0$

$$\begin{aligned}
\mathbb{P}_0\Big\{Z(t) = n\Big\} &\equiv \mathbb{P}\Big\{Z(t) = n\Big|Z(0) = 0\Big\} \\
&= \frac{\beta^n}{\pi} \cdot \int_0^\pi \cos\big(-n \cdot \theta\big) \cdot e^{-2 \cdot \left(\alpha - \gamma \cdot \cos(\theta)\right) t}\, d\theta \\
&= \frac{\beta^n}{\pi} \cdot \int_0^\pi \cos\big(n \cdot \theta\big) \cdot e^{-2 \cdot \left(\alpha - \gamma \cdot \cos(\theta)\right) t}\, d\theta \\
&= \beta^{2n} \cdot \left(\frac{\beta^{-n}}{\pi}\right) \cdot \int_0^\pi \cos\big(n \cdot \theta\big) \cdot e^{-2 \cdot \left(\alpha - \gamma \cdot \cos(\theta)\right) t}\, d\theta \\
&= \beta^{2n} \cdot \mathbb{P}\Big\{Z(t) = -n\Big|Z(0) = 0\Big\} \\
&\equiv \beta^{2n} \cdot \mathbb{P}_0\Big\{Z(t) = -n\Big\} \qquad (2.2.40)
\end{aligned}$$

Using linearity, the reflecting process transition probabilites solve the same Kolmogorov backward equations as the free process.

$$\mathbb{P}_m\Big\{Q(t) < n\Big\} = \mathbb{P}_m\Big\{Z(t) < n\Big\} - \beta^{2n} \cdot \mathbb{P}_m\Big\{Z(t) < -n\Big\} \qquad (2.2.41)$$

which implies that

$$\begin{aligned}
\mathbb{P}_{-1}\Big\{Q(t) < n\Big\} &= \mathbb{P}_{-1}\Big\{Z(t) < n\Big\} - \beta^{2n} \cdot \mathbb{P}_{-1}\Big\{Z(t) < -n\Big\} \\
&= \mathbb{P}_0\Big\{Z(t) < n + 1\Big\} - \beta^{2n} \cdot \mathbb{P}_0\Big\{Z(t) < -n + 1\Big\} \\
&= \mathbb{P}_0\Big\{Z(t) < n\Big\} - \beta^{2n} \cdot \mathbb{P}_0\Big\{Z(t) < -n\Big\} + \mathbb{P}_0\Big\{Z(t) = n\Big\} \\
&\quad - \beta^{2n} \cdot \mathbb{P}_0\Big\{Z(t) = -n\Big\} \\
&= \mathbb{P}_0\Big\{Q(t) < n\Big\} + \mathbb{P}_0\Big\{Z(t) = n, \mathcal{T}_0 > t\Big\} \\
&= \mathbb{P}_0\Big\{Q(t) < n\Big\} \qquad (2.2.42)
\end{aligned}$$

Similarly, using linearity, the reflecting process transition probabilities solve the same Kolmogorov backward equations as the free process.

$$\mathbb{P}_m\Big\{Q(t) < n\Big\} \equiv \mathbb{P}_m\Big\{Z(t) < n\Big\} - \beta^{2n} \cdot \mathbb{P}_m\Big\{Z(t) < -n\Big\},$$

$$\mathbb{P}_{-1}\Big\{Q(t) < n\Big\} = \mathbb{P}_0\Big\{Q(t) < n\Big\},$$

and

$$\begin{aligned}\frac{d}{dt}\mathbb{P}_m\Big\{Z(t) < n\Big\} =& \lambda \cdot \Big(\mathbb{P}_{m+1}\Big\{Z(t) < n\Big\} - \mathbb{P}_m\Big\{Z(t) < n\Big\}\Big) \\ &+ \mu \cdot \Big(\mathbb{P}_{m-1}\Big\{Z(t) < n\Big\} - \mathbb{P}_m\Big\{Z(t) < n\Big\}\Big)\end{aligned}$$

which implies

$$\begin{aligned}\frac{d}{dt}\mathbb{P}_m\Big\{Q(t) < n\Big\} =& \lambda \cdot \Big(\mathbb{P}_{m+1}\Big\{Q(t) < n\Big\} - \mathbb{P}_m\Big\{Q(t) < n\Big\}\Big) \\ &+ \mu \cdot \Big(\mathbb{P}_{m-1}\Big\{Q(t) < n\Big\} - \mathbb{P}_m\Big\{Q(t) < n\Big\}\Big)\end{aligned}$$

and

$$\begin{aligned}\frac{d}{dt}\mathbb{P}_0\Big\{Q(t) < n\Big\} &= \lambda \cdot \Big(\mathbb{P}_1\Big\{Q(t) < n\Big\} - \mathbb{P}_0\Big\{Q(t) < n\Big\}\Big) \\ &\quad + \mu \cdot \Big(\mathbb{P}_{-1}\Big\{Q(t) < n\Big\} - \mathbb{P}_0\Big\{Q(t) < n\Big\}\Big) \\ &= \lambda \cdot \Big(\mathbb{P}_1\Big\{Q(t) < n\Big\} - \mathbb{P}_0\Big\{Q(t) < n\Big\}\Big)\end{aligned}$$

Expressing reflecting process transition probabilities in terms of absorbing process transition probabilities

$$\mathbb{P}_m\Big\{Q(t) < n\Big\} \equiv \mathbb{P}_m\Big\{Z(t) < n\Big\} - \beta^{2n} \cdot \mathbb{P}_m\Big\{Z(t) < -n\Big\},$$

this implies

$$\begin{aligned}
\mathbb{P}_m\Big\{Q(t) < n\Big\} &= \mathbb{P}_m\Big\{Z(t) < n\Big\} - \beta^{2n} \cdot \mathbb{P}_m\Big\{Z(t) < -n\Big\} \\
&= \mathbb{P}_{m+1}\Big\{Z(t) < n+1\Big\} - \beta^{2n} \cdot \mathbb{P}_{m+1}\Big\{Z(t) < -n+1\Big\} \\
&= \mathbb{P}_{m+1}\Big\{Z(t) < n\Big\} - \beta^{2n} \cdot \mathbb{P}_{m+1}\Big\{Z(t) < -n\Big\} \\
&\quad + \mathbb{P}_{m+1}\Big\{Z(t) = n\Big\} - \beta^{2n} \cdot \mathbb{P}_{m+1}\Big\{Z(t) = -n\Big\} \\
&= \mathbb{P}_{m+1}\Big\{Q(t) < n\Big\} + \mathbb{P}_{m+1}\Big\{Z(t) = n, \mathcal{T}_0 > t\Big\}
\end{aligned}$$

hence

$$\mathbb{P}_m\Big\{Q(t) < n\Big\} = \sum_{\ell=m+1}^{\infty} \mathbb{P}_\ell\Big\{Z(t) = n, \mathcal{T}_0 > t\Big\} \tag{2.2.43}$$

Recall that the absorbing process probabilities as complex integrals

$$\begin{aligned}
\mathbb{P}_m\Big\{Z(t) = n, \mathcal{T}_0 > t\Big\} &= \mathbb{P}_m\Big\{Z(t) = n\Big\} - \beta^{2n} \cdot \mathbb{P}_m\Big\{Z(t) = -n\Big\} \\
&= \frac{1}{2\pi i} \cdot \oint_{|\omega|=r} \left[ \left(\frac{\omega}{\beta}\right)^{m-n} - \beta^{2n} \cdot \left(\frac{\omega}{\beta}\right)^{m+n} \right] \cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega)\right)t} \, \frac{d\omega}{\omega} \\
&= \frac{\beta^n}{2\pi i} \cdot \oint_{|\omega|=r} \left(\frac{\omega}{\beta}\right)^{m} \cdot \left(\omega^{-n} - \omega^n\right) \cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega)\right)t} \, \frac{d\omega}{\omega} \\
&= -\frac{\beta^{n-m}}{\pi i} \cdot \oint_{|\omega|=r} \omega^m \cdot \epsilon_n(\omega) \cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega)\right)t} \, \frac{d\omega}{\omega}
\end{aligned} \tag{2.2.44}$$

**Theorem 2.2.9** (Reflecting Process Transition Probabilities as Complex Integrals)**.** The reflecting process transition probabilities as complex integrals is given by the following integral:

$$\mathbb{P}_m\Big\{Q(t) < n\Big\} = (1-\rho^n)^+ + \frac{\beta^{n-m}}{\pi i} \cdot \oint\limits_{|\omega|=1} \epsilon_n(\omega) \cdot \frac{\epsilon_m(\omega) - \beta \cdot \epsilon_{m+1}(\omega)}{(\omega-\beta)\cdot(1/\omega-\beta)} \cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega)\right)t} \, \frac{d\omega}{\omega}$$

*Proof.*

$$\begin{aligned}
\mathbb{P}_m\Big\{Q(t) < n\Big\} &= \sum_{\ell=m+1}^{\infty} \mathbb{P}_\ell\Big\{Z(t) = n, \mathcal{T}_0 > t\Big\} \\
&= -\sum_{\ell=m+1}^{\infty} \frac{\beta^{n-\ell}}{\pi i} \cdot \oint_{|\omega|=r} \omega^\ell \cdot \epsilon_n(\omega) \cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega)\right)t} \, \frac{d\omega}{\omega} \\
&= -\frac{\beta^n}{2\pi i} \cdot \oint_{|\omega|=r<\beta} \left(\sum_{\ell=m+1}^{\infty} \left(\frac{\omega}{\beta}\right)^\ell\right) \cdot \epsilon_n(\omega) \cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega)\right)t} \, \frac{d\omega}{\omega} \\
&= -\frac{\beta^n}{2\pi i} \cdot \oint_{|\omega|=r<\beta} \frac{\omega^{m+1}/\beta^{m+1}}{1-\omega/\beta} \cdot \epsilon_n(\omega) \cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega)\right)t} \, \frac{d\omega}{\omega} \\
&= \frac{\beta^{n-m}}{\pi i} \cdot \oint_{|\omega|=r<\beta} \epsilon_n(\omega) \cdot \frac{\omega^{m+1}}{\omega-\beta} \cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega)\right)t} \, \frac{d\omega}{\omega} \\
&= 2\beta^{n-m} \cdot \left(-\epsilon_n(\omega)^+ \cdot \beta^m + \frac{1}{2\pi i} \cdot \oint_{|\omega|=1} \frac{\epsilon_n(\omega) \cdot \omega^m \cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega)\right)t}}{\omega-\beta} \, d\omega\right) \\
&= \left(1-\rho^{2n}\right)^+ + \frac{\beta^{n-m}}{2\pi i} \cdot \oint_{|\omega|=1} \epsilon_n(\omega) \cdot \left(\frac{\omega^{m+1}}{\omega-\beta} - \frac{\omega^{-m-1}}{1/\omega-\beta}\right) \cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega)\right)t} \, \frac{d\omega}{\omega} \\
&= (1-\rho^n)^+ + \frac{\beta^{n-m}}{2\pi i} \cdot \oint_{|\omega|=1} \epsilon_n(\omega) \cdot \frac{\omega^m - \omega^{-m} - \beta\cdot(\omega^{m+1} - \omega^{-m-1})}{(\omega-\beta)\cdot(1/\omega-\beta)} \cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega)\right)t} \, \frac{d\omega}{\omega} \\
&= (1-\rho^n)^+ + \frac{\beta^{n-m}}{\pi i} \cdot \oint_{|\omega|=1} \epsilon_n(\omega) \cdot \frac{\epsilon_m(\omega) - \beta\cdot\epsilon_{m+1}(\omega)}{(\omega-\beta)\cdot(1/\omega-\beta)} \cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega)\right)t} \, \frac{d\omega}{\omega}
\end{aligned}$$

□

**Theorem 2.2.10** (Reflecting Process Transition Probabilities as Real Integrals)**.** The reflecting process transition probabilities as real integrals is given by the following integral:

$$\begin{aligned}
\mathbb{P}_m\Big\{Q(t) = n\Big\} &= \Big(1-\rho\Big)^+ \cdot \rho^n \\
&+ \frac{\beta^{n-m}}{\pi} \cdot \int_{-\pi}^{\pi} \frac{\Big(\sin n\theta - \beta\cdot\sin(n+1)\theta\Big)\cdot\Big(\sin m\theta - \beta\sin(m+1)\theta\Big)\cdot e^{-2\cdot\left(\alpha-\gamma\cdot\cos(\theta)\right)t}}{1-2\beta\cos\theta+\beta^2} d\theta
\end{aligned}$$

*Proof.*

$$\begin{aligned}
\mathbb{P}_m\Big\{Q(t) < n\Big\} &= \Big(1-\rho^n\Big)^+ + \frac{\beta^{n-m}}{\pi i}\cdot \oint\limits_{|\omega|=1} \epsilon_n(\omega)\cdot \frac{\epsilon_m(\omega)-\beta\cdot\epsilon_{m+1}(\omega)}{(\omega-\beta)\cdot(1/\omega-\beta)}\cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega)\right)t}\,\frac{d\omega}{\omega} \\
&= \Big(1-\rho^n\Big)^+ + \frac{\beta^{n-m}}{\pi}\cdot \oint\limits_{-\pi}^{\pi} \epsilon_n(e^{i\theta})\cdot \frac{\epsilon_m(e^{i\theta})-\beta\cdot\epsilon_{m+1}(e^{i\theta})}{(e^{i\theta}-\beta)\cdot(e^{-i\theta}-\beta)}\cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(e^{i\theta})\right)t}\,d\theta \\
&= \Big(1-\rho^n\Big)^+ - \frac{\beta^{n-m}}{\pi}\cdot \oint\limits_{-\pi}^{\pi} \sin n\theta\cdot \frac{\sin m\theta-\beta\cdot\sin(m+1)\theta}{1-2\beta\cos\theta+\beta^2}\cdot e^{-2\cdot\left(\alpha-\gamma\cdot\cos\theta\right)t}\,d\theta
\end{aligned}$$

and

$$\mathbb{P}_m\Big\{Q(t)=n\Big\} = \mathbb{P}_m\Big\{Q(t)<n+1\Big\} - \mathbb{P}_m\Big\{Q(t)<n\Big\} \tag{2.2.45}$$

Hence

$$\begin{aligned}
\mathbb{P}_m\Big\{Q(t)=n\Big\} &= \Big(1-\rho\Big)^+\cdot\rho^n \\
&+ \frac{\beta^{n-m}}{\pi}\cdot\int\limits_{-\pi}^{\pi} \frac{\Big(\sin n\theta-\beta\cdot\sin(n+1)\theta\Big)\cdot\Big(\sin m\theta-\beta\sin(m+1)\theta\Big)\cdot e^{-2\cdot\left(\alpha-\gamma\cdot\cos(\theta)\right)t}}{1-2\beta\cos\theta+\beta^2} d\theta
\end{aligned}$$

□

The major contribution in Theorems 2.2.9, and 2.2.10 is new derivation using complex analysis. Another closed-form representation of the free process transition probabilities exists (Lajos, 1962). Next we explore operator approach to computing these transition probabilities of the reflected process.

**Definition 2.2.6** (Left and Right Shift Operators for Row Vectors)**.**

$$\boldsymbol{L} = \begin{bmatrix} 0 & 0 & 0 & \cdots \\ 1 & 0 & 0 & \ddots \\ 0 & 1 & 0 & \ddots \\ \vdots & \ddots & \ddots & \ddots \end{bmatrix} \quad \text{and} \quad \boldsymbol{R} = \begin{bmatrix} 0 & 1 & 0 & \cdots \\ 0 & 0 & 1 & \ddots \\ 0 & 0 & 0 & \ddots \\ \vdots & \ddots & \ddots & \ddots \end{bmatrix}$$

and

$$\big(\boldsymbol{I} - \boldsymbol{L}\big)^{-1} = \begin{bmatrix} 1 & 0 & 0 & \cdots \\ 1 & 1 & 0 & \ddots \\ 1 & 1 & 1 & \ddots \\ \vdots & \ddots & \ddots & \ddots \end{bmatrix} \quad \text{and} \quad (\boldsymbol{I} - \boldsymbol{R})^{-1} = \begin{bmatrix} 1 & 1 & 1 & \cdots \\ 0 & 1 & 1 & \ddots \\ 0 & 0 & 1 & \ddots \\ \vdots & \ddots & \ddots & \ddots \end{bmatrix}$$

where $\boldsymbol{L}^T = \boldsymbol{R}$ and $\boldsymbol{RL} = \boldsymbol{I} \neq \boldsymbol{LR}$

We have the following

$$\Big[\big\{Q = 0\big\} \ \big\{Q = 1\big\} \ \cdots\Big] \cdot \big(\boldsymbol{I} - \boldsymbol{L}\big)^{-1} = \Big[\big\{Q \geq 0\big\} \ \big\{Q \geq 1\big\} \ \cdots\Big] \tag{2.2.46}$$

$$\Big[\big\{Q = 0\big\} \ \big\{Q = 1\big\} \ \cdots\Big] \cdot \big(\boldsymbol{I} - \boldsymbol{R}\big)^{-1} = \Big[\big\{Q \leq 0\big\} \ \big\{Q \leq 1\big\} \ \cdots\Big] \tag{2.2.47}$$

The Markov Generators for $M/M/1$ Queues or Single Barrier Reflecting Process

$$\mathbf{A} = \begin{pmatrix} -\lambda & \lambda & 0 & \cdots \\ \mu & -(\lambda+\mu) & \lambda & \ddots \\ 0 & \mu & -(\lambda+\mu) & \ddots \\ \ddots & \ddots & \ddots & \ddots \end{pmatrix}$$

$$= \lambda \boldsymbol{R} + \mu \boldsymbol{L} - \lambda \boldsymbol{I} - \mu \boldsymbol{L}\boldsymbol{R}$$

$$= \lambda \boldsymbol{R} + \mu \boldsymbol{L} - \lambda \boldsymbol{R}\boldsymbol{L} - \mu \boldsymbol{L}\boldsymbol{R}$$

$$= \big(\lambda \boldsymbol{R} - \mu \boldsymbol{L}\boldsymbol{R}\big) \cdot \big(\boldsymbol{I} - \boldsymbol{L}\big)$$

The sub-Markov generators for single barrier absorbing process as the dual to single barrier relfection process

$$(\boldsymbol{I} - \boldsymbol{L})^{-1} \cdot \boldsymbol{A}^T = \boldsymbol{A} \cdot (\boldsymbol{I} - \boldsymbol{R})^{-1} = \lambda \boldsymbol{R} - \mu \boldsymbol{L}\boldsymbol{R}$$

So

$$\boldsymbol{A}^T = (\boldsymbol{I} - \boldsymbol{L}) \cdot \Big(\lambda \boldsymbol{R} - \mu \boldsymbol{L}\boldsymbol{R}\Big) \cdot (\boldsymbol{I} - \boldsymbol{L}) = \lambda \boldsymbol{R} - \mu \boldsymbol{L}\boldsymbol{R}^2 - (\lambda + \mu)\boldsymbol{L}\boldsymbol{R}$$

which implies that

$$\boldsymbol{A} = \lambda \boldsymbol{L} - \mu \boldsymbol{L}^2 \boldsymbol{R} - (\lambda + \mu)\boldsymbol{L}\boldsymbol{R}$$

$$= \begin{pmatrix} 0 & 0 & 0 & \cdots \\ \lambda & -(\lambda+\mu) & \mu & \ddots \\ 0 & \lambda & -(\lambda+\mu) & \ddots \\ \ddots & \ddots & \ddots & \ddots \end{pmatrix}$$

Duality between a single barrier reflecting process and its complimentary single barrier absorbing process

$$\mathbb{P}_m\big\{Q(t) \geq n\big\} = \mathbb{P}_n\big\{Q^*(t) \leq m\big\}$$

We get the following similarity transformation equivalence

$$\boldsymbol{A} \cdot \big(\boldsymbol{I} - \boldsymbol{R}\big)^{-1} = \big(\boldsymbol{I} - \boldsymbol{R}\big)^{-1} \boldsymbol{A}^T$$
$$\text{and } \exp t\boldsymbol{A} \cdot \big(\boldsymbol{I} - \boldsymbol{R}\big)^{-1} = \big(\boldsymbol{I} - \boldsymbol{R}\big)^{-1} \exp t\boldsymbol{A}^T$$

Again we now verify that the single Barrier reflecting transition probabilities using the operator approach.

**Theorem 2.2.11** (Single Barrier Reflecting Process Transition probabilities As a Complex Integral via Duality)**.** The single barrier reflecting process transition probabilities are given by the following integral:

$$\mathbb{P}_m\big\{Q(t) = n\big\} = (1-\rho)^+ \rho^n$$
$$+ \frac{\beta^{n-m}}{\pi i} \cdot \oint\limits_{|\omega|=1} \frac{\Big(\epsilon_m(\omega) - \beta \cdot \epsilon_{n+1}(\omega)\Big) \cdot \Big(\epsilon_m(\omega) - \beta \cdot \epsilon_{m+1}(\omega)\Big)}{2 \cdot \beta\Big(\delta(\beta) - \delta(\omega)\Big)} \cdot e^{-2 \cdot \left(\alpha - \gamma \cdot \delta(\omega)\right) t} \frac{d\omega}{\omega}$$

*Proof.*

$$
\begin{aligned}
\mathbb{P}_m\big\{Q(t) \geq n\big\} &= \mathbb{P}_n\big\{Q^*(t) = \ell\big\} \\
&= \mathbb{P}_n\big\{Q^*(t) = 0\big\} + \sum_{\ell=1}^{m} \mathbb{P}_n\big\{Q^*(t) \leq m\big\} \\
&= \lambda \int_0^t \mathbb{P}_n\big\{Q^*(s) = 1\big\}\, ds + \sum_{\ell=1}^{m} \mathbb{P}_n\big\{Q^*(t) \leq m\big\} \\
&= -\lambda \int_0^t \left( \frac{\beta^{n-1}}{\pi i} \cdot \oint_{|\omega|=1} \epsilon_n(\omega) \cdot \epsilon_1(\omega) \cdot e^{-2\gamma \cdot \left(\delta(\beta) - \delta(\omega)\right) s}\, \frac{d\omega}{\omega} \right) ds \\
&\quad - \sum_{\ell=1}^{m} \left( \frac{\beta^{n-m}}{\pi i} \cdot \oint_{|\omega|=1} \epsilon_n(\omega) \cdot \epsilon_\ell(\omega) \cdot e^{-2\gamma \cdot \left(\delta(\beta) - \delta(\omega)\right) t}\, \frac{d\omega}{\omega} \right) \\
&= -\lambda \cdot \frac{\beta^{n-1}}{\pi i} \oint_{|\omega|=1} \epsilon_n(\omega) \cdot \epsilon_1(\omega) \left( \int_0^t e^{-2\gamma \cdot \left(\delta(\beta) - \delta(\omega)\right) s}\, ds \right) \frac{d\omega}{\omega} \\
&\quad - \frac{\beta^n}{\pi i} \oint_{|\omega|=1} \epsilon_n(\omega) \cdot \left( \sum_{\ell=1}^{m} \beta^{-\ell} \epsilon_\ell(\omega) \right) \cdot e^{-2\gamma \cdot \left(\delta(\beta) - \delta(\omega)\right) t}\, \frac{d\omega}{\omega} \\
&= -\lambda \cdot \frac{\beta^{n-1}}{\pi i} \oint_{|\omega|=1} \epsilon_n(\omega) \cdot \epsilon_1(\omega) \left( \frac{1 - e^{-2 \cdot \left(\alpha - \gamma \cdot \delta(\omega)\right) t}}{2\gamma \cdot \Big(\delta(\beta) - \delta(\omega)\Big)} \right) \frac{d\omega}{\omega} \\
&\quad - \frac{\beta^n}{\pi i} \oint_{|\omega|=1} \epsilon_n(\omega) \cdot \left( \frac{\epsilon_1(\omega) + \beta^{-m-1} \cdot \epsilon_m(\omega) - \beta^{-m} \cdot \epsilon_{m+1}(\omega)}{2 \cdot \Big(\delta(\beta) - \delta(\omega)\Big)} \right) \cdot e^{-2\gamma \cdot \left(\delta(\beta) - \delta(\omega)\right) t}\, \frac{d\omega}{\omega} \\
&= \min(1, \rho^n) \\
&\quad + \frac{\beta^{n-m}}{\pi i} \cdot \oint_{|\omega|=1} \epsilon_n(\omega) \cdot \frac{\Big(\epsilon_m(\omega) - \beta \cdot \epsilon_{m+1}(\omega)\Big)}{2 \cdot \beta \Big(\delta(\beta) - \delta(\omega)\Big)} \cdot e^{-2 \cdot \left(\alpha - \gamma \cdot \delta(\omega)\right) t}\, \frac{d\omega}{\omega}
\end{aligned}
$$

which implies that

$$
\begin{aligned}
\mathbb{P}_m\big\{Q(t) = n\big\} &= (1-\rho)^+ \rho^n \qquad (2.2.48) \\
&\quad + \frac{\beta^{n-m}}{\pi i} \cdot \oint_{|\omega|=1} \frac{\Big(\epsilon_m(\omega) - \beta \cdot \epsilon_{n+1}(\omega)\Big) \cdot \Big(\epsilon_m(\omega) - \beta \cdot \epsilon_{m+1}(\omega)\Big)}{2 \cdot \beta \Big(\delta(\beta) - \delta(\omega)\Big)} \cdot e^{-2 \cdot \left(\alpha - \gamma \cdot \delta(\omega)\right) t}\, \frac{d\omega}{\omega}
\end{aligned}
$$

□

**Theorem 2.2.12** (Single Barrier Reflecting Process Transition probabilities As a Real Integral via Duality)**.** The single barrier reflecting process transition probabilities are given by the following integral:

$$\mathbb{P}_m\big\{Q(t)=n\big\} = (1-\rho)^+\,\rho^n + \frac{\beta^{n-m}}{\pi i}\cdot\oint\limits_{|\omega|=1}\frac{\Big(\epsilon_m(\omega)-\beta\cdot\epsilon_{n+1}(\omega)\Big)\cdot\Big(\epsilon_m(\omega)-\beta\cdot\epsilon_{m+1}(\omega)\Big)}{2\cdot\beta\Big(\delta(\beta)-\delta(\omega)\Big)}\cdot e^{-2\cdot\Big(\alpha-\gamma\cdot\delta(\omega)\Big)t}\,\frac{d\omega}{\omega}$$

*Proof.*

$$\begin{aligned}\mathbb{P}_m\big\{Q(t)=n\big\} &= (1-\rho)^+\,\rho^n \\ &+ \frac{\beta^{n-m}}{\pi i}\cdot\oint\limits_{|\omega|=1}\frac{\Big(\epsilon_m(\omega)-\beta\cdot\epsilon_{n+1}(\omega)\Big)\cdot\Big(\epsilon_m(\omega)-\beta\cdot\epsilon_{m+1}(\omega)\Big)}{2\cdot\beta\Big(\delta(\beta)-\delta(\omega)\Big)}\cdot e^{-2\cdot\Big(\alpha-\gamma\cdot\delta(\omega)\Big)t}\,\frac{d\omega}{\omega} \\ &= (1-\rho)^+\,\rho^n \\ &+ \frac{\beta^{n-m}}{\pi i}\int\limits_{-\pi}^{\pi}\frac{\Big(\sin n\theta-\beta\cdot\sin(n+1)\theta\Big)\cdot\Big(\sin m\theta-\beta\cdot\sin(m+1)\theta\Big)}{1+\beta^2-2\beta\cos\theta}\cdot e^{-2\gamma\cdot\Big(\delta(\beta)-\delta(\omega)\Big)t}\,d\theta\end{aligned}$$

□

### 2.2.4 Two Barrier Absorbing Process

An absorbing process at the boundary points $\{0,k\}$ is a Markov chain in which it is impossible to leave the boundary states $\{0,k\}$, and any state could reach the boundary states after some number of steps, with positive probability. We classify the state space $Z=\{\ldots,-3,-2,-1,0,1,2,3,\ldots\}$ of the free process into two classes: Absorbing state $0,k$ and transient states $\{1,2,\ldots,k-2,k-1\}$. The absorbing process is the same as the free process that starts positive and when the process hits boundary states $\{0,k\}$, it stays there forever. If the process starts positive on the state space $\{0,1,2,\ldots,k-1,k\}$, then the process would never get to visit the states

$\{\ldots,-3,-2,-1\}$ or the states $\{k+1,k+2,k+3\ldots\}$, so we could also consider those states as implicitly absorbing.

Our transportation interpretation of the absorbing free process at the boundary points $\{0,k\}$, in the context of public transportation serves, is a bike-sharing station model having $k$ secured bicycle parking docks. One group of customers arrive at rate $\lambda$ to return a rented bicycle if an empty parking dock is available. And the second group of customers arrives at rate $\mu$ to rent an available bicycle. Moreover, the bike station starts with some initial number of bicycles already at the station but the station also has some empty bicycle parking dock. Absorption at state zero means there is no available bike. Whereas absorption at state $k$ means that there is no available parking dock at the bicycle station. Figure 2.14 shows the transient states of the absorbing Free Process at the boundary points 0 and $k$ given a positive starting state. One interesting question is this setting, relating to measure of the quality of service at a bike-sharing station, is how long does it take for there to be empty bicycles at the station or how long does it take for there to be no available docks at the station.

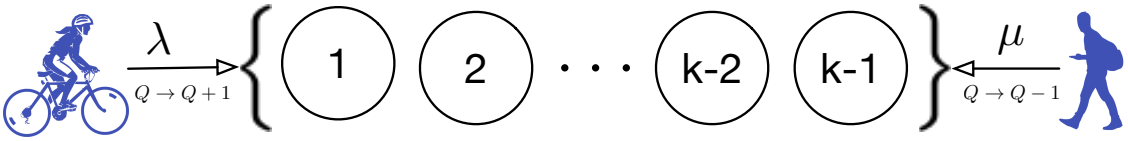



Figure 2.14: The transient states of the absorbing free process at the boundary points 0 and $k$. The values $\lambda$ and $\mu$ are the rates in which the state transitions in the positive direction and negative direction respectively.

**Definition 2.2.7** (Two Barrier Absorbing Process)**.** Given the free process $\{Z(t)\,|\,t\geq 0\}$ and the stopping time $\mathcal{T}_{0,k}=\min\{t|\,Z(t)=0 \text{ or } k\}$, define the double barrier

absorbing process $\Big\{Q_k^*(t)\Big|\, t \geq 0\Big\}$ to be

$$Q_k^*(t) \equiv Z\big(\mathcal{T}_{0,k} \wedge t\big) \equiv Z\big(\min(\mathcal{T}_{0,k}, t)\big) \tag{2.2.49}$$

**Definition 2.2.8** (Double Barrier Absorbing Transition Probabilities)**.** The double barrier absorbing transition probabilitie are defined for all positive integers $0 \leq m \leq k$ and $0 \leq n \leq k$ to be

$$\begin{aligned}\mathbb{P}_m\big\{Q_k^*(t) = n\big\} &\equiv \mathbb{P}\big\{Q_k^*(t) = n | Q_k^*(0) = n\big\} \\ &= \begin{cases} \mathbb{P}_m\big\{Q_k^*(t) = n, \mathcal{T}_{0,k} > t\big\} & \text{if } 0 < m < k \text{ and } 0 < n < k, \\ 0 & \text{if } m = 0 \text{ or } k \text{ with } n \neq m, \\ \mathbb{P}_m\big\{Q_k^*(t) = n, \mathcal{T}_{0,k} \leq t\big\} & \text{if } m \neq n \text{ with } n = 0 \text{ or } k, \\ 1 & \text{if } m = n = 0 \text{ or } k. \end{cases}\end{aligned}$$

By the linearity and symmetry, the two barrier absorbing transition probabilities solve the same forward equations as the free process

$$\mathbb{P}_m\big\{Z(t) = -n\big\} = \mathbb{P}_n\big\{Z(t) = -m\big\} = \beta^{-2(m+n)} \cdot \mathbb{P}_{-m}\big\{Z(t) = n\big\}$$

implies

$$\begin{aligned}\mathbb{P}_m\big\{Q_k^*(t) = n\big\} &= \sum_{\ell=-\infty}^{\infty} \beta^{2k\ell} \cdot \Big(\mathbb{P}_{m+2k\ell}\big\{Z(t) = n\big\} - \beta^{-2m} \cdot \mathbb{P}_{-m+2k\ell}\big\{Z(t) = n\big\}\Big) \\ &= \sum_{\ell=-\infty}^{\infty} \Big(\beta^{2k\ell} \cdot \mathbb{P}_m\big\{Z(t) = n\big\} - \beta^{-2m+2k\ell} \cdot \beta^{-2(-m-n+2k\ell)} \cdot \mathbb{P}_m\big\{Z(t) = -n + 2k\ell\big\}\Big) \\ &= \sum_{\ell=-\infty}^{\infty} \beta^{-2k\ell} \cdot \Big(\mathbb{P}_m\big\{Z(t) = n + 2k\ell\big\} - \beta^{2n} \cdot \mathbb{P}_m\big\{Z(t) = -n + 2k\ell\big\}\Big).\end{aligned}$$

So

$$\mathbb{P}_m\big\{Q_k^*(t)=0\big\}=\sum_{\ell=-\infty}^{\infty}\beta^{-2k\ell}\cdot\Big(\mathbb{P}_m\big\{Z(t)=2k\ell\big\}-\mathbb{P}_m\big\{Z(t)=2k\ell\big\}\Big)=0$$

and

$$\mathbb{P}_m\big\{Q_k^*(t)=k\big\}=\sum_{\ell=-\infty}^{\infty}\beta^{-2k\ell}\cdot\Big(\mathbb{P}_m\big\{Z(t)=k+2k\ell\big\}-\beta^{2k}\cdot\mathbb{P}_m\big\{Z(t)=-k+2k\ell\big\}\Big)$$

implies

$$\begin{aligned}\sum_{\ell=-\infty}^{\infty}\beta^{-2k\ell}\cdot\mathbb{P}_m\big\{Z(t)=k+2k\ell\big\}&=\sum_{\ell=-\infty}^{\infty}\beta^{-2k(\ell-1)}\cdot\mathbb{P}_m\big\{Z(t)=k+2k(\ell-1)\big\}\\&=\sum_{\ell=-\infty}^{\infty}\beta^{-2k\ell+2k}\cdot\mathbb{P}_m\big\{Z(t)=-k+2k\ell\big\}.\end{aligned}\tag{2.2.50}$$

Hence , we have

$$\mathbb{P}_m\big\{Q_k^*(t)=k\big\}=0.$$

Similarly, by the linearity and symmetry, the two barrier absorbing transition probabilities solve the same backward equations as the free process

$$\mathbb{P}_0\big\{Q_k^*(t)=n\big\}=\sum_{\ell=-\infty}^{\infty}\beta^{2k\ell}\cdot\Big(\mathbb{P}_{2k\ell}\big\{Z(t)=n\big\}-\mathbb{P}_{2k\ell}\big\{Z(t)=n\big\}\Big)=0$$

and

$$\mathbb{P}_m\big\{Q_k^*(t)=n\big\}=\sum_{\ell=-\infty}^{\infty}\beta^{2k\ell}\cdot\Big(\mathbb{P}_{k+2k\ell}\big\{Z(t)=n\big\}-\beta^{-2k}\cdot\mathbb{P}_{-k+2k}\big\{Z(t)=n\big\}\Big)$$

implies

$$\sum_{\ell=-\infty}^{\infty} \beta^{2k\ell} \cdot \mathbb{P}_{k+2k\ell}\big\{Z(t) = n\big\} = \sum_{\ell=-\infty}^{\infty} \beta^{2k(\ell-1)} \cdot \mathbb{P}_{k+2k(\ell-1)}\big\{Z(t) = n\big\}$$
$$= \sum_{\ell=-\infty}^{\infty} \beta^{2k\ell-2k} \cdot \mathbb{P}_{-k+2k\ell}\big\{Z(t) = n\big\}.$$

Hence, we have

$$\mathbb{P}_m\big\{Q_k^*(t) = n\big\} = 0.$$

We define the following argument principle that we will use to derive the transition probabilities of the absorbing process.

**Definition 2.2.9** (Argument principle)**.** Let $f$ and $g$ have the same analytic domain $\Omega$ with $\Omega \equiv \partial\Omega$. For all $a \in \Omega$ where $f(a)$ is a unique value for $\Omega$, then we have

$$g(a) = \frac{1}{2\pi i} \int_{\gamma} \frac{g(z) \cdot f'(z)}{f(z) - f(a)} dz \tag{2.2.51}$$

In what follows, we present the two barrier absorbing transition probabilities as both complex and real integrals.

**Theorem 2.2.13** (Two Barrier Absorbing Transition Probabilities as a Complex Sum)**.** The two barrier absorbing transition probabilities are given by the following sum:

$$\mathbb{P}_m\big\{Q_k^*(t) = n\big\} = -\frac{\beta^{n-m}}{\pi i} \cdot \left( \oint_{|\omega|=r_+>1} \frac{\omega^{2k}}{\omega^{2k}} \cdot \frac{\omega^m \cdot \epsilon_n(\omega) \cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega)\right)\cdot t}}{1-\omega^{-2k}} \frac{d\omega}{\omega} \right.$$
$$\left. + \oint_{|\omega|=r_+<1} \frac{\omega^{m+2k} \cdot \epsilon_n(\omega) \cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega)\right)\cdot t}}{1-\omega^{2k}} \frac{d\omega}{\omega} \right)$$

*Proof.*

$$\begin{aligned}
\mathbb{P}_m\big\{Q_k^*(t)=n\big\} &= \sum_{\ell=-\infty}^{\infty} \beta^{-2k\ell}\cdot\Big(\mathbb{P}_m\big\{Z(t)=n+2k\ell\big\}-\beta^{2n}\cdot\mathbb{P}_m\big\{Z(t)=-n+2k\ell\big\}\Big)\\
&= \frac{\beta^{n-m}}{2\pi i}\cdot\sum_{\ell=-\infty}^{\infty}\oint_{|\omega|=r(\ell)}\Big(\omega^{m-2k\ell-n}-\omega^{m-2k\ell+n}\Big)\cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega)\right)\cdot t}\,\frac{d\omega}{\omega}\\
&= -\frac{\beta^{n-m}}{\pi i}\cdot\sum_{\ell=-\infty}^{\infty}\oint_{|\omega|=r(\ell)}\omega^{m-2k\ell}\epsilon_n(\omega)\cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega)\right)\cdot t}\,\frac{d\omega}{\omega}\\
&= -\frac{\beta^{n-m}}{\pi i}\cdot\left(\oint_{|\omega|=r_+>1}\omega^m\left(\sum_{\ell=0}^{\infty}\omega^{-2k\ell}\right)\cdot\epsilon_n(\omega)\cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega)\right)\cdot t}\,\frac{d\omega}{\omega}\right.\\
&\quad\left.+\oint_{|\omega|=r_+<1}\omega^m\left(\sum_{\ell=-\infty}^{-1}\omega^{-2k\ell}\right)\cdot\epsilon_n(\omega)\cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega)\right)\cdot t}\,\frac{d\omega}{\omega}\right)\\
&= -\frac{\beta^{n-m}}{\pi i}\cdot\left(\oint_{|\omega|=r_+>1}\frac{\omega^{2k}}{\omega^{2k}}\cdot\frac{\omega^m\cdot\epsilon_n(\omega)\cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega)\right)\cdot t}}{1-\omega^{-2k}}\,\frac{d\omega}{\omega}\right.\\
&\quad\left.+\oint_{|\omega|=r_+<1}\frac{\omega^{m+2k}\cdot\epsilon_n(\omega)\cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega)\right)\cdot t}}{1-\omega^{2k}}\,\frac{d\omega}{\omega}\right)
\end{aligned}$$

□

**Theorem 2.2.14** (Two Barrier Absorbing Transition Probabilities as a Real Sum)**.** The two barrier absorbing transition probabilities are given by the following sum:

$$\mathbb{P}_m\big\{Q_k^*(t)=n\big\}=\frac{2\beta^{n-m}}{k}\cdot\sum_{\ell=0}^{k-1}\sin\frac{m\pi\ell}{k}\cdot\sin\frac{n\pi\ell}{k}\cdot e^{-2\cdot\left(\alpha-\gamma\cdot\cos\frac{\pi\ell}{k}\right)t}$$

*Proof.*

$$\begin{aligned}
\mathbb{P}_m\big\{Q_k^*(t)=n\big\} &= -\frac{\beta^{n-m}}{\pi i}\cdot\left(\oint\limits_{|\omega|=r_+>1}\frac{\omega^{2k}}{\omega^{2k}}\cdot\frac{\omega^m\cdot\epsilon_n(\omega)\cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega)\right)\cdot t}}{1-\omega^{-2k}}\,\frac{d\omega}{\omega}\right.\\
&\qquad\left.+\oint\limits_{|\omega|=r_+<1}\frac{\omega^{m+2k}\cdot\epsilon_n(\omega)\cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega)\right)\cdot t}}{1-\omega^{2k}}\,\frac{d\omega}{\omega}\right)\\
&= -\frac{\beta^{n-m}}{\pi i}\cdot\sum_{\ell=0}^{k-1}\frac{1}{2\pi i}\oint\limits_{|\omega-\omega_{2k}^{\ell}|=\eta}\omega^m\cdot\epsilon_n(\omega)\cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega)\right)\cdot t}\cdot\frac{2k\omega^{2k-1}}{\omega^{2k}-1}\,d\omega\\
&= -\frac{\beta^{n-m}}{\pi i}\cdot\sum_{\ell=0}^{k-1}\left(\omega_{2k}^{\ell}\right)^m\cdot\epsilon_n(\omega_{2k}^{\ell})\cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega_{2k}^{\ell})\right)\cdot t}\quad\text{(Argument Principle)}\\
&= -\frac{2\beta^{n-m}}{\pi i}\cdot\sum_{\ell=0}^{k-1}\epsilon_m(\omega_{2k}^{\ell})\cdot\epsilon_n(\omega_{2k}^{\ell})\cdot e^{-2\cdot\left(\alpha-\gamma\cdot\delta(\omega_{2k}^{\ell})\right)\cdot t}\\
&= \frac{2\beta^{n-m}}{k}\cdot\sum_{\ell=0}^{k-1}\sin\frac{m\pi\ell}{k}\cdot\sin\frac{n\pi\ell}{k}\cdot e^{-2\cdot\left(\alpha-\gamma\cdot\cos\frac{\pi\ell}{k}\right)t}.
\end{aligned}$$

□

Moreover, the two barriers absorbing time distribution is given as follows:

**Theorem 2.2.15** (Two Barrier Absorbing Time Distributions as a Real Sum)**.** The two barrier absorbing time distributions are given by the following sum:

$$\mathbb{P}_m\big\{\mathcal{T}_{0,k}>t\big\}=\frac{1}{k\cdot\beta^m}\cdot\sum_{\ell=1}^{k-1}\sin\frac{\pi\ell m}{k}\cdot\sin\frac{\pi\ell}{k}\cdot\frac{\left(1+\beta^k\cdot(-1)^{\ell}\right)\cdot e^{-2\gamma\cdot\left(\delta(\beta)-\cos\frac{\pi\ell}{k}\right)t}}{\delta(\beta)-\cos\frac{\pi\ell}{k}}\tag{2.2.52}$$

*Proof.*

$$-\frac{d}{dt}\mathbb{P}_m\big\{\mathcal{T}_{0,k}>t\big\}=\lambda\mathbb{P}_m\big\{Q_k^*(t)=k-1\big\}-\mu\mathbb{P}_m\big\{Q_k^*(t)=1\big\}\tag{2.2.53}$$

implies that

$$\begin{aligned}
\mathbb{P}_m\{\mathcal{T}_{0,k} > t\} &= \lambda \cdot \int_t^\infty \mathbb{P}_m\{Q_k^*(s) = k-1\}ds - \mu \cdot \int_t^\infty \mathbb{P}_m\{Q_k^*(s) = 1\}ds \\
&= \lambda \cdot \frac{\beta^{k-1-m}}{k} \cdot \sum_{\ell=1}^{k-1} \frac{\sin\frac{\pi\ell m}{k} \cdot \sin\frac{\pi\ell(k-1)}{k}}{\alpha - \gamma \cdot \cos\frac{\pi\ell}{k}} \cdot e^{-2\cdot\left(\alpha-\gamma\cdot\cos\frac{\pi\ell}{k}\right)t} \\
&\quad + \mu \cdot \frac{\beta^{1-m}}{k} \cdot \sum_{\ell=1}^{k-1} \frac{\sin\frac{\pi\ell m}{k} \cdot \sin\frac{\pi\ell}{k}}{\alpha - \gamma \cdot \cos\frac{\pi\ell}{k}} \cdot e^{-2\cdot\left(\alpha-\gamma\cdot\cos\frac{\pi\ell}{k}\right)t} \\
&= \frac{1}{k \cdot \beta^m} \cdot \sum_{\ell=1}^{k-1} \sin\frac{\pi\ell m}{k} \cdot \sin\frac{\pi\ell}{k} \cdot \frac{\left(1 + \beta^k \cdot (-1)^\ell\right) \cdot e^{-2\gamma\cdot\left(\delta(\beta)-\cos\frac{\pi\ell}{k}\right)t}}{\delta(\beta) - \cos\frac{\pi\ell}{k}}.
\end{aligned}$$

□

Theorems 2.2.13, 2.2.14, and 2.2.15 are special case of Baccelli et al. (1994). The major contribution is new derivation using complex analysis.

### 2.2.5 $M_\lambda/M_\mu/1/k$ Queue Length (Reflecting) Process

Now consider the free process that gets reflected whenever it hits the boundary points 0 and $k$ and then gets constrained to the positive integers $\{0, 1, 2, \ldots, k-1, k\}$. Our interpretation of the reflecting process at the boundary points 0 and $k$ is the same as the absorbing process at the boundary points 0 and $k$ except that we now added two states $\{0, k\}$ to the state space. In the context of public transportation serves the reflecting process could represent a bike-sharing station model having $k$ secured bicycle parking docks. One group of customers arrive at rate $\lambda$ to return a rented bicycle if an empty parking dock is available. And the second group of customers arrives at rate $\mu$ to rent an available bicycle. Moreover, the bike station starts with some bicycle already at the station but the station also has some empty bicycle parking dock. The reflection at the boundary states means we restrict the free process from the original state space $\mathbb{Z} \equiv \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ to the new state

space $\{0, 1, 2, \ldots, k-1, k\}$. Figure 2.15 shows the transient states of the absorbing Free Process at the boundary points 0 and $k$ given a positive starting state.

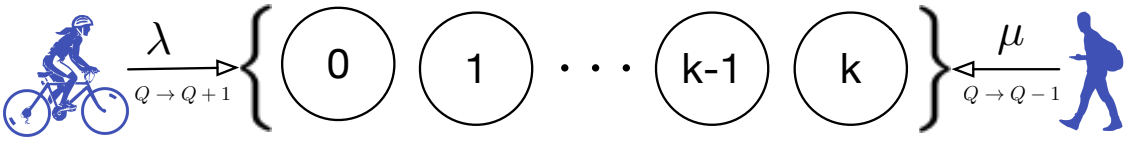



Figure 2.15: The transient states of the reflecting process at the boundary points 0 and $k$. The values $\lambda$ and $\mu$ are the rates in which the state transitions in the positive direction and negative direction respectively.

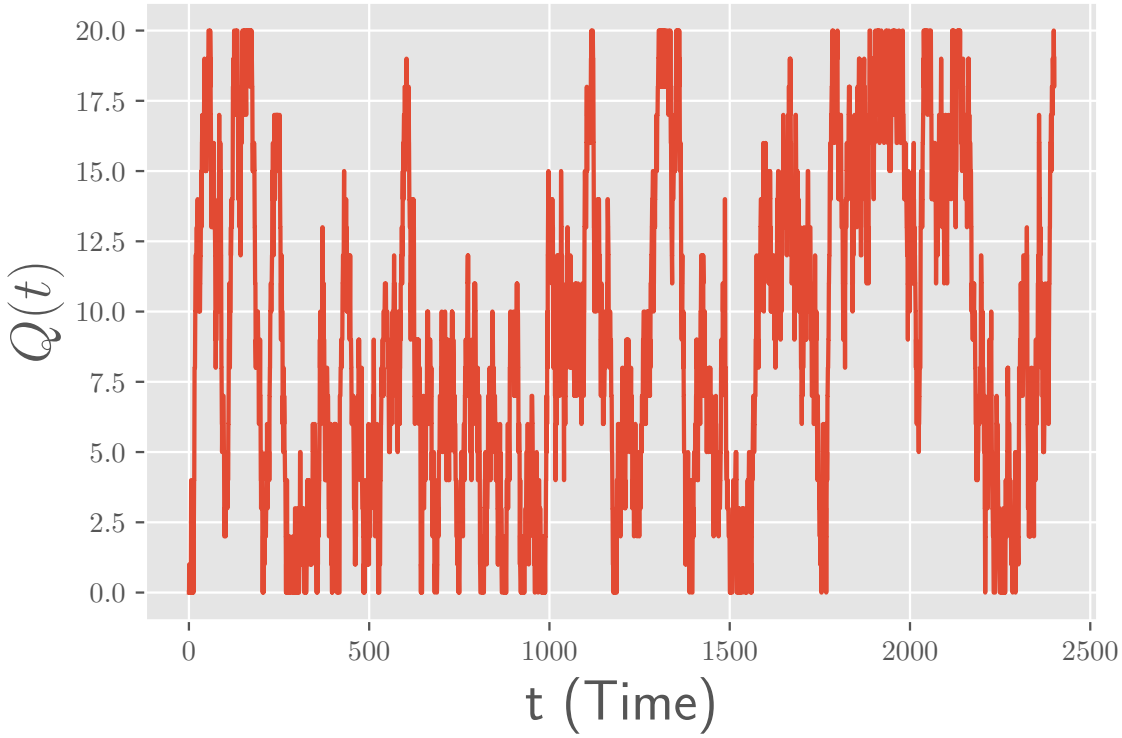



Figure 2.16: The realizations of the reflected free process with $\lambda = \mu = 1$, $Z(0) = 0$, $0 < t < 2400$ and capacity $k = 20$.

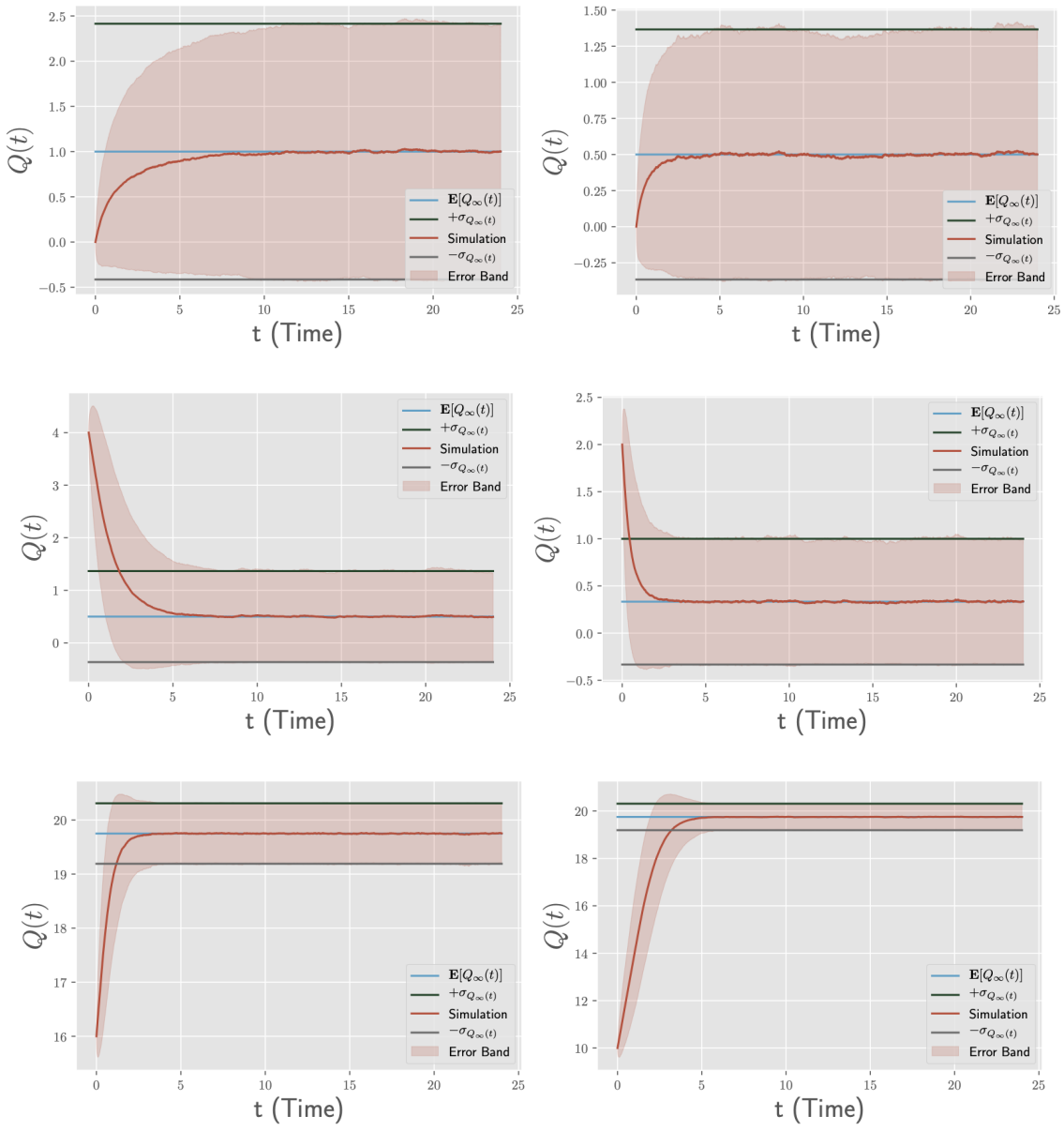



Figure 2.17: The plot of the steady-state mean and standard deviation of the reflected free process where $\mathbb{E}\big[Q_\infty(t)\big]$ is the steady-state mean, and $\sigma_{Q_\infty}$ is the steady-state standard deviation. For the simulation $\Delta t = 10^{-3}$, sample size $N = 10^4$, time range $0 < t < 24$ and capacity $k = 20$. In (a) , $\lambda = 1$, $\mu = 2$, and $Q(0) = 0$. In (b) , $\lambda = 1$, $\mu = 3$, and $Q(0) = 0$. In (c) , $\lambda = 1$, $\mu = 3$, and $Q(0) = 4$. In (d) , $\lambda = 1$, $\mu = 4$, and $Q(0) = 2$. In (e) , $\lambda = 5$, $\mu = 1$, and $Q(0) = 16$. In (f) , $\lambda = 5$, $\mu = 1$, and $Q(0) = 10$.

**Algorithm 3:** The $M_\lambda/M_\mu/1/k$ queue simulator

**Input:** Given arrival rate $\lambda$, service rate $\mu$, initial state $m$, capacity of the queue $k$ and stopping time $T$.

**Output:** queue length process $\boldsymbol{Q}$

Initialize time $t = 0$, starting state $q = m$ and create an empty list $\boldsymbol{Q}$
**while** $t \leq T$ **do**
  $U_1 \sim \mathcal{U}(0, 1)$
  $t \leftarrow t - \frac{\log(U_1)}{\lambda+\mu}$
  **if** $t > T$ **then**
    Break
  **else**
    $U_2 \sim \mathcal{U}(0, 1)$
    **if** $U_2 < \frac{\lambda}{\lambda+\mu}$ **then**
      $q = \min(q + 1, k)$
    **else**
      $q = \max(q - 1, 0)$
    **end**
    $\boldsymbol{Q}$.append(q)
  **end**
**end**

Figure 2.16 shows the realization of the two barrier reflecting process, starting at the origin, as a function of time. To validate the simulation of the two barrier reflecting process given by the Algorithm 3, we compare the simulation results with the steady-state closed-form mean and standard deviation of the free process as shown in Figure 2.17. As you can see, the simulation results closely approximate the the-

oretical closed-form results. Next, we explore the operator approach to computing transition probabilities of the two barrier reflected process. To do this, we first define shift operators and their properties.

**Definition 2.2.10** (Left and Right Shift Operators for k+1 Dimensional Row Vectors)**.**

$$\boldsymbol{L} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & 0 & \ddots & 0 & 0 \\ 0 & 1 & 0 & \ddots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & \ddots & 0 & 0 \\ 0 & 0 & 0 & \ddots & 1 & 0 \end{bmatrix} \quad \text{and} \quad \boldsymbol{R} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \ddots & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix}$$

and

$$\left(\boldsymbol{I} - \boldsymbol{L}\right)^{-1} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 1 & 1 & 0 & \ddots & 0 & 0 \\ 1 & 1 & 1 & \ddots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 1 & 1 & 1 & \ddots & 1 & 0 \\ 1 & 1 & 1 & \ddots & 1 & 1 \end{bmatrix} \quad \text{and} \quad (\boldsymbol{I} - \boldsymbol{R})^{-1} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 0 & 1 & 1 & \ddots & 1 & 1 \\ 0 & 0 & 0 & \ddots & 1 & 1 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & \cdots & 1 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}$$

where $\boldsymbol{L}^T = \boldsymbol{R}$, $\boldsymbol{LRL} = \boldsymbol{L}$, $\boldsymbol{RLR} = \boldsymbol{R}$, and $\boldsymbol{L}^{k+1} = \boldsymbol{R}^{k+1} = 0$.

We have the following

$$\begin{bmatrix} \{Q = 0\} & \{Q = 1\} & \cdots & \{Q = k\} \end{bmatrix} \cdot \left(\boldsymbol{I} - \boldsymbol{L}\right)^{-1} = \begin{bmatrix} \{Q \geq 0\} & \{Q \geq 1\} & \cdots & \{Q \geq k\} \end{bmatrix}$$
$$\begin{bmatrix} \{Q = 0\} & \{Q = 1\} & \cdots & \{Q = k\} \end{bmatrix} \cdot \left(\boldsymbol{I} - \boldsymbol{R}\right)^{-1} = \begin{bmatrix} \{Q \leq 0\} & \{Q \leq 1\} & \cdots & \{Q \leq k\} \end{bmatrix}$$

The Markov Generators for $M/M/1/k$ Queues or Double Barrier Reflecting Process or $k^{th}$ Two barrier reflecting process

$$\mathbf{A} = \begin{pmatrix} -\lambda & \lambda & 0 & \cdots & 0 & 0 \\ \mu & -(\lambda+\mu) & \lambda & \ddots & 0 & 0 \\ 0 & \mu & -(\lambda+\mu) & \ddots & 0 & 0 \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & 0 & -(\lambda+\mu) & \lambda \\ 0 & 0 & 0 & 0 & \mu & -\mu \end{pmatrix}$$

$$= \lambda \boldsymbol{R} + \mu \boldsymbol{L} - \lambda \boldsymbol{R}\boldsymbol{L} - \mu \boldsymbol{L}\boldsymbol{R}$$

$$= \big(\lambda \boldsymbol{R} - \mu \boldsymbol{L}\boldsymbol{R}\big) \cdot \big(\boldsymbol{I} - \boldsymbol{L}\big) \tag{2.2.54}$$

The sub-Markov generators for $k+1^{st}$ two barrier absorbing process as the dual to $k^{th}$ double barrier relfection process

$$\boldsymbol{A} = (\boldsymbol{I} - \boldsymbol{R})^{-1} \cdot \boldsymbol{A}^T \cdot (\boldsymbol{I} - \boldsymbol{R}) = \big(\lambda \boldsymbol{R} - \mu \boldsymbol{L}\boldsymbol{R}\big) \cdot (\boldsymbol{I} - \boldsymbol{R}) = \lambda \boldsymbol{R} + \mu \boldsymbol{L}\boldsymbol{R}^2.$$

Transition probabilities for the dual process. We get the following similarity transformation equivalence

$$\boldsymbol{A} \cdot \big(\boldsymbol{I} - \boldsymbol{R}\big)^{-1} = \big(\boldsymbol{I} - \boldsymbol{R}\big)^{-1} \boldsymbol{A}^T \tag{2.2.55}$$

Equation 2.2.55 implies

$$\exp t\boldsymbol{A} \cdot \big(\boldsymbol{I} - \boldsymbol{L}\big)^{-1} = \Big(\exp t\boldsymbol{A} \cdot \big(\boldsymbol{I} - \boldsymbol{R}\big)^{-1}\Big)^T \tag{2.2.56}$$

And Equation 2.2.56 implies

$$\mathbb{P}_m\{Q_k(t) \geq n\} = \mathbb{P}_n\{k - Q^*_{k+1}(t) \leq m\} = \mathbb{P}_n\{Q^*_{k+1}(t) \geq k - m\}.$$

The dual of a $k^{th}$ two barrier reflecting process and its complimentary $k+1^{st}$ two barrier absorbing process is given by:

$$\mathbb{P}_m\{Q_k(t) \geq n\} = \mathbb{P}_n\{k + 1 - Q^*_{k+1}(t) \leq m\} = \mathbb{P}_n\{Q^*_{k+1}(t) \geq k + 1 - m\}. \quad (2.2.57)$$

Moreover, the following two Equations are equivalent:

$$\begin{gathered}\frac{d}{dt}\mathbb{P}_n\{k + 1 - Q^*_{k+1}(t) = 0\} = \lambda \cdot \mathbb{P}_n\{k + 1 - Q^*_{k+1}(t) = 1\} \\ \Updownarrow \\ \frac{d}{dt}\mathbb{P}_n\{Q^*_{k+1}(t) = k + 1\} = \lambda \cdot \mathbb{P}_n\{Q^*_{k+1}(t) = k\}.\end{gathered} \quad (2.2.58)$$

We also have

$$\mathbb{P}_0\{Q_{k+1}(t) \geq n\} = \mathbb{P}_n\{k + 1 - Q^*_{k+1}(t) = 0\} = \lambda \cdot \int_0^t \mathbb{P}_n\{k + 1 - Q^*_{k+1}(s) = 1\}ds$$

and (2.2.59)

$$\frac{d}{dt}\mathbb{P}_n\{Q^*_{k+1}(t) = k + 1\} = \lambda \cdot \mathbb{P}_n\{Q^*_{k+1}(t) = k\}. \quad (2.2.60)$$

Hence,

$$\begin{aligned}&\mathbb{P}_m\{Q_k(t) \geq n\} = \mathbb{P}_n\{k + 1 - Q^*_{k+1}(t) \leq m\} \\ &= \lambda \cdot \int_0^t \mathbb{P}_n\{k + 1 - Q^*_{k+1}(s) = 1\}ds + \sum_{\ell=1}^{m} \mathbb{P}_n\{k + 1 - Q^*_{k+1}(s) = \ell\}.\end{aligned}$$

Finally, we now give the transition probabilities of the two barrier reflecting process as the finite sum of trigonometric functions.

**Theorem 2.2.16** (Double Barrier Reflecting ProcessTransition Probabilities As a Real Sum)**.** The double barrier reflecting process transition probabilities are given by the following sums:

$$\begin{aligned}\mathbb{P}_m\{Q_k(t) = n\} = &\frac{(1-\rho)\cdot\rho^n}{1-\rho^{k+1}} \\ &- \frac{2\beta^{n-m}}{k+1}\cdot\sum_{\ell=1}^{k}\frac{\left(\sin\frac{n\pi\ell}{k+1} - \beta\cdot\sin\frac{(n+1)\pi\ell}{k+1}\right)\cdot\left(\sin\frac{m\pi\ell}{k+1} - \beta\cdot\sin\frac{(m+1)\pi\ell}{k+1}\right)}{1+\beta^2-2\beta\cos\frac{\pi\ell}{k+1}} \\ &\cdot e^{-2\gamma\cdot\left(\delta(\beta)-\cos\frac{\pi\ell}{k+1}\right)t}\end{aligned}$$

*Proof.*

$$
\begin{aligned}
&\mathbb{P}_m\big\{Q_k(t)\geq n\big\}=\mathbb{P}_{k+1-n}\big\{k+1-Q^*_{k+1}(t)\leq m\big\}\\
&\quad=\mu\cdot\int_0^t\mathbb{P}_{k+1-n}\big\{Q^*_{k+1}(s)=k\big\}ds+\sum_{\nu}^{m}\mathbb{P}_{k+1-n}\big\{Q^*_{k+1}(s)=k+1-\nu\big\}\\
&\quad=-\mu\cdot\int_0^t\left(\frac{2\beta^{n-1}}{k+1}\cdot\sum_{\ell=1}^{k}\epsilon_{k+1-n}\big(\omega^\ell_{2k+2}\big)\cdot\epsilon_k\big(\omega^\ell_{2k+2}\big)\cdot e^{-2\gamma\cdot\left(\delta(\beta)-\delta(\omega^\ell_{2k+2})\right)s}\right)ds\\
&\qquad-\sum_{\nu=1}^{m}\frac{2\beta^{n-\nu}}{k+1}\cdot\sum_{\ell=1}^{k}\epsilon_{k+1-n}\big(\omega^\ell_{2k+2}\big)\cdot\epsilon_{k+1-\nu}\big(\omega^\ell_{2k+2}\big)\cdot e^{-2\gamma\cdot\left(\delta(\beta)-\delta(\omega^\ell_{2k+2})\right)s}\\
&\quad=-\mu\cdot\int_0^t\left(\frac{2\beta^{n-1}}{k+1}\cdot\sum_{\ell=1}^{k}\epsilon_{n}\big(\omega^\ell_{2k+2}\big)\cdot\epsilon_1\big(\omega^\ell_{2k+2}\big)\cdot e^{-2\gamma\cdot\left(\delta(\beta)-\delta(\omega^\ell_{2k+2})\right)s}\right)ds\\
&\qquad-\sum_{\nu=1}^{m}\frac{2\beta^{n-\nu}}{k+1}\cdot\sum_{\ell=1}^{k}\epsilon_{n}\big(\omega^\ell_{2k+2}\big)\cdot\epsilon_\nu\big(\omega^\ell_{2k+2}\big)\cdot e^{-2\gamma\cdot\left(\delta(\beta)-\delta(\omega^\ell_{2k+2})\right)s}\\
&\quad=-\lambda\cdot\frac{2\beta^{n-1}}{k+1}\cdot\sum_{\ell=1}^{k}\frac{\epsilon_n\big(\omega^\ell_{2k+2}\big)\cdot\epsilon_1\big(\omega^\ell_{2k+2}\big)\cdot\left(1-e^{-2\gamma\cdot\left(\delta(\beta)-\delta(\omega^\ell_{2k+2})\right)s}\right)}{2\gamma\cdot\big(\delta(\beta)-\delta(\omega^\ell_{2k+2})\big)}\\
&\qquad-\sum_{\nu=1}^{m}\frac{2\beta^{n}}{k+1}\cdot\sum_{\ell=1}^{k}\epsilon_{n}\big(\omega^\ell_{2k+2}\big)\cdot\left(\sum_{\nu=1}^{m}\beta^{-\nu}\epsilon_\nu(\omega^\ell_{2k+2})\right)\cdot e^{-2\gamma\cdot\left(\delta(\beta)-\delta(\omega^\ell_{2k+2})\right)t}\\
&\quad=-\lambda\cdot\frac{2\beta^{n-1}}{k+1}\cdot\sum_{\ell=1}^{k}\frac{\epsilon_n\big(\omega^\ell_{2k+2}\big)\cdot\epsilon_1\big(\omega^\ell_{2k+2}\big)\cdot\left(1-e^{-2\gamma\cdot\left(\delta(\beta)-\delta(\omega^\ell_{2k+2})\right)s}\right)}{2\gamma\cdot\big(\delta(\beta)-\delta(\omega^\ell_{2k+2})\big)}\\
&\qquad-\frac{2\beta^{n}}{k+1}\cdot\sum_{\ell=1}^{k}\epsilon_{n}\big(\omega^\ell_{2k+2}\big)\cdot\left(\frac{\epsilon_1\big(\omega^\ell_{2k+2}\big)+\beta^{-m-1}\cdot\epsilon_m\big(\omega^\ell_{2k+2}\big)-\beta^{-m}\cdot\epsilon_{m+1}\big(\omega^\ell_{2k+2}\big)}{2\cdot\big(\delta(a)-\delta(\omega^\ell_{2k+2})\big)}\right)\\
&\qquad\cdot e^{-2\gamma\cdot\left(\delta(\beta)-\delta(\omega^\ell_{2k+2})\right)t}\\
&\quad=-\frac{2\beta^{n-1}}{k+1}\cdot\sum_{\ell=1}^{k}\frac{\epsilon_n\big(\omega^\ell_{2k+2}\big)\cdot\epsilon_1\big(\omega^\ell_{2k+2}\big)}{2\beta\cdot\big(\delta(\beta)-\delta(\omega^\ell_{2k+2})\big)}\\
&\qquad-\frac{2\beta^{n-m}}{k+1}\cdot\sum_{\ell=1}^{k}\epsilon_{n}\big(\omega^\ell_{2k+2}\big)\cdot\left(\frac{\epsilon_m\big(\omega^\ell_{2k+2}\big)-\beta\cdot\epsilon_{m+1}\big(\omega^\ell_{2k+2}\big)}{2\beta\cdot\big(\delta(\beta)-\delta(\omega^\ell_{2k+2})\big)}\right)\cdot e^{-2\gamma\cdot\left(\delta(\beta)-\delta(\omega^\ell_{2k+2})\right)t}\\
&\quad=\frac{\rho^n\cdot\big(1-\rho^{k+1-n}\big)}{1-\rho^{k+1}}\\
&\qquad-\frac{2\beta^{n-m}}{k+1}\cdot\sum_{\ell=1}^{k}\epsilon_{n}\big(\omega^\ell_{2k+2}\big)\cdot\left(\frac{\epsilon_m\big(\omega^\ell_{2k+2}\big)-\beta\cdot\epsilon_{m+1}\big(\omega^\ell_{2k+2}\big)}{2\beta\cdot\big(\delta(\beta)-\delta(\omega^\ell_{2k+2})\big)}\right)\cdot e^{-2\gamma\cdot\left(\delta(\beta)-\delta(\omega^\ell_{2k+2})\right)t}
\end{aligned}
$$

□

The major contribution in Theorem 2.2.16 is new derivation using complex analysis. Another closed-form representation of the free process transition probabilities exists (Lajos, 1962).

### Operator Approach to Queue Analysis

We further explore the operator approach to $M_\lambda/M_\mu/1/k$ analysis. The ultimate goal of introducing this operator approach is to map some of the results developed for the reflecting processes to other types of queueing process, specifically absorbing processes.

**Definition 2.2.11** (Similar matrices)**.** In linear algebra, two n-by-n matrices $A$ and $B$ are similar if there exists an invertible n-by-n matrix $Q$ such that

$$B = Q^{-1}AQ \tag{2.2.61}$$

### Similarity Transformation

Define $\boldsymbol{e}_n$ for $n \in \mathbb{Z}$ to be the $n$th unit basis vector. In addition, we define the right and the left shift operators which we denote as $\boldsymbol{R}$ and $\boldsymbol{L}$ respectively. The right and the left shift operators can be defined in terms of $\boldsymbol{e}_n$'s , where $\boldsymbol{e}_n\boldsymbol{R} = \boldsymbol{e}_{n+1}$ and $\boldsymbol{e}_n\boldsymbol{L} = \boldsymbol{e}_{n-1}$. For the finite capacity $M_\lambda/M_\mu/1/k$, the right shift and the left shift operators can be defined in terms of $\boldsymbol{e}_n$'s as follows:

$$\boldsymbol{e}_n\boldsymbol{R} = \begin{cases} \boldsymbol{e}_{n+1} & n < k \\ 0 & n = k \end{cases}$$

and the left shift is given by

$$\boldsymbol{e}_n\boldsymbol{L} = \begin{cases} \boldsymbol{e}_{n-1} & n > 0 \\ 0 & n = 0, \end{cases}$$

where $\boldsymbol{e}_n$ is the standard basis in $\mathbb{R}^K$, in other words $\boldsymbol{e}_n = [0, \ldots, 0, 1, 0, \ldots, 0]$ is the unit vector with all components equal to 0, except the $n^{th}$ component which is 1.

For example the left-shift and right-shift matrix for the identity matrix $\boldsymbol{I}_{6\times 6}$

$$\boldsymbol{I}_{6\times 6}\boldsymbol{L} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{and} \quad \boldsymbol{I}_{6\times 6}\boldsymbol{R} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.2.62}$$

**Corollary 2.2.17.** The left-shift and right-shift operators satisfy the following identities $\boldsymbol{RLR} = \boldsymbol{R}$ and $\boldsymbol{LRL} = \boldsymbol{L}$.

*Proof.* From the definition the left-shift and right-shift operators, it follows that $\boldsymbol{RLR} = \boldsymbol{R}$, and $\boldsymbol{LRL} = \boldsymbol{L}$. □

**Remark 2.2.2.** It turns out that the left $\boldsymbol{L}$ and right $\boldsymbol{R}$ shift operators are nilpotent matrix. In linear algebra, a nilpotent matrix is a square matrix $\boldsymbol{M}$ such that $N^k = 0$ for some positive integer $k$. The smallest such $k$ is often called the index of the matrix $\boldsymbol{M}$. So we have $(\boldsymbol{I} - \boldsymbol{L})$ and $(\boldsymbol{I} - \boldsymbol{R})$ are invertible.

**Lemma 2.2.18.** Let $\boldsymbol{A}$ be the Markov generator for the queue length process $\{Q(t)|t \geq 0\}$ associated with $M_\lambda/M_\mu/1/k$ queue. There exists matrices $\boldsymbol{U} \equiv \boldsymbol{I} - \boldsymbol{R}$

and $\boldsymbol{V} \equiv \boldsymbol{I} - \boldsymbol{L}$, such that the generator can be decomposed as

$$\boldsymbol{U}\boldsymbol{A}\boldsymbol{U}^{-1} = \begin{pmatrix} 0 & \boldsymbol{q}_\lambda^T \\ \boldsymbol{0} & \boldsymbol{Q} \end{pmatrix} \text{ and } \boldsymbol{V}\boldsymbol{A}\boldsymbol{V}^{-1} = \begin{pmatrix} \boldsymbol{Q} & \boldsymbol{0} \\ \boldsymbol{q}_\mu^T & 0 \end{pmatrix} \tag{2.2.63}$$

where

$$\boldsymbol{q}_\lambda = \begin{bmatrix} \lambda \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \boldsymbol{q}_\mu = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \mu \end{bmatrix}, \tag{2.2.64}$$

and

$$\mathbf{Q} = \begin{pmatrix} -(\lambda+\mu) & \lambda & 0 & 0 & \cdots & 0 \\ \mu & -(\lambda+\mu) & \lambda & 0 & \cdots & 0 \\ 0 & \mu & -(\lambda+\mu) & \lambda & \cdots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & \mu & -(\lambda+\mu) & \lambda \\ 0 & \cdots & 0 & 0 & \mu & -(\lambda+\mu) \end{pmatrix} \tag{2.2.65}$$

*Proof.* Using the left-shift and right-shift operators, we can rewrite the Markov generator $A$ for the queue-length process for $M_\lambda/M_\mu/1/k$ as follows:

$$\begin{aligned} \boldsymbol{A} &= \mu\boldsymbol{R} + \lambda\boldsymbol{L} - \mu\boldsymbol{R}\boldsymbol{L} - \lambda\boldsymbol{L}\boldsymbol{R} \\ &= \mu(\boldsymbol{R}\boldsymbol{L}\boldsymbol{R}) + \lambda\boldsymbol{L} - \mu\boldsymbol{R}\boldsymbol{L} - \lambda\boldsymbol{L}\boldsymbol{R} \quad \text{(by the identity of right operator)} \\ &= (\lambda\boldsymbol{L} - \mu\boldsymbol{R}\boldsymbol{L})(\boldsymbol{I} - \boldsymbol{R}) \quad \text{(by factoring like terms)} \end{aligned} \tag{2.2.66}$$

Let

$$\begin{aligned}\boldsymbol{U} &= \boldsymbol{I} - \boldsymbol{R} \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ -1 & 1 & 0 & 0 & \cdots & 0 \\ 0 & -1 & 1 & 0 & \cdots & 0 \\ \vdots & & \ddots & \ddots & & \vdots \\ 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}\end{aligned} \tag{2.2.67}$$

and

$$\begin{aligned}\boldsymbol{U}^{-1} &= \left(\boldsymbol{I} - \boldsymbol{R}\right)^{-1} \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 1 & 0 & \cdots & 0 \\ \vdots & & \ddots & \ddots & & \vdots \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}\end{aligned} \tag{2.2.68}$$

This implies that

$$
\begin{aligned}
\boldsymbol{U}\boldsymbol{A}\boldsymbol{U}^{-1} &= (\boldsymbol{I}-\boldsymbol{R})\boldsymbol{A}(\boldsymbol{I}-\boldsymbol{R})^{-1} \\
&= (I-\boldsymbol{R})\Big[(\lambda\boldsymbol{L}-\mu\boldsymbol{R}\boldsymbol{L})(\boldsymbol{I}-\boldsymbol{R})\Big](\boldsymbol{I}-\boldsymbol{R})^{-1} \\
&= (\boldsymbol{I}-\boldsymbol{R})\big(\lambda\boldsymbol{L}-\mu\boldsymbol{R}\boldsymbol{L}\big) \\
&= \begin{pmatrix}
0 & \lambda & 0 & 0 & 0 & \cdots & 0 \\
0 & -(\lambda+\mu) & \lambda & 0 & 0 & \cdots & 0 \\
0 & \mu & -(\lambda+\mu) & \lambda & 0 & \cdots & 0 \\
0 & 0 & \mu & -(\lambda+\mu) & \lambda & \cdots & 0 \\
0 & \vdots & & \ddots & \ddots & \ddots & \vdots \\
0 & 0 & \cdots & 0 & \mu & -(\lambda+\mu) & \lambda \\
0 & 0 & \cdots & 0 & 0 & \mu & -(\lambda+\mu)
\end{pmatrix} \\
&= \begin{pmatrix} 0 & \boldsymbol{q}_\lambda^T \\ \boldsymbol{0} & \boldsymbol{Q} \end{pmatrix}
\end{aligned}
\tag{2.2.69}
$$

Similarly, using the left-shift and right-shift operators, we can also rewrite the Markov generator $A$ for the queue-length process for $M_\lambda/M_\mu/1/k$ as follows:

$$
\begin{aligned}
\boldsymbol{A} &= \mu\boldsymbol{R}+\lambda\boldsymbol{L}-\mu\boldsymbol{R}\boldsymbol{L}-\lambda\boldsymbol{L}\boldsymbol{R} \\
&= \mu\boldsymbol{R}+\lambda(\boldsymbol{L}\boldsymbol{R}\boldsymbol{L})-\mu\boldsymbol{R}\boldsymbol{L}-\lambda\boldsymbol{L}\boldsymbol{R} \quad \text{(by the identity of right operator)} \\
&= (\mu\boldsymbol{R}-\lambda\boldsymbol{L}\boldsymbol{R})(\boldsymbol{I}-\boldsymbol{L}) \quad \text{(by factoring like terms)}
\end{aligned}
\tag{2.2.70}
$$

Let

$$\boldsymbol{V} = \boldsymbol{I} - \boldsymbol{L} \tag{2.2.71}$$

$$= \begin{bmatrix} 1 & -1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & -1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & -1 & \cdots & 0 \\ \vdots & & & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2.72}$$

and

$$\boldsymbol{V}^{-1} = \left(\boldsymbol{I} - \boldsymbol{L}\right)^{-1} \tag{2.2.73}$$

$$= \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 0 & 1 & 1 & 1 & \cdots & 1 \\ 0 & 0 & 1 & 1 & \cdots & 1 \\ \vdots & & & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2.74}$$

This implies that

$$
\begin{aligned}
\boldsymbol{V}\boldsymbol{A}\boldsymbol{V}^{-1} &= (\boldsymbol{I}-\boldsymbol{L})\boldsymbol{A}(\boldsymbol{I}-\boldsymbol{L})^{-1} \\
&= (\boldsymbol{I}-\boldsymbol{L})\Big[(\mu\boldsymbol{R}-\lambda\boldsymbol{L}\boldsymbol{R})(\boldsymbol{I}-\boldsymbol{L})\Big](\boldsymbol{I}-\boldsymbol{L})^{-1} \\
&= (\boldsymbol{I}-\boldsymbol{L})\big(\mu\boldsymbol{R}-\lambda\boldsymbol{L}\boldsymbol{R}\big) \\
&= \begin{pmatrix}
-(\lambda+\mu) & \lambda & 0 & 0 & \cdots & & 0 \\
\mu & -(\lambda+\mu) & \lambda & 0 & \cdots & & 0 \\
0 & \mu & -(\lambda+\mu) & \lambda & \cdots & & 0 \\
\vdots & & \ddots & \ddots & \ddots & & \vdots \\
0 & \cdots & 0 & \mu & -(\lambda+\mu) & \lambda & 0 \\
0 & \cdots & 0 & 0 & \mu & -(\lambda+\mu) & 0 \\
0 & \cdots & 0 & 0 & 0 & \mu & 0
\end{pmatrix} \\
&= \begin{pmatrix} \boldsymbol{Q} & \boldsymbol{0} \\ \boldsymbol{q}_{\mu}^{T} & 0 \end{pmatrix}
\end{aligned}
\tag{2.2.75}
$$

□

Consider the matrix $\widetilde{\boldsymbol{A}}$ obtained from the similiarity transformation of the generator $\boldsymbol{A}$ of a reflecting queueing model $M_\lambda/M_\mu/1/k$ defined by

$$\widetilde{\boldsymbol{A}} = (\boldsymbol{I} - \boldsymbol{R})\boldsymbol{A}(\boldsymbol{I} - \boldsymbol{R})^{-1}$$

$$= \begin{pmatrix} 0 & \lambda & 0 & 0 & 0 & \cdots & 0 \\ 0 & -(\lambda+\mu) & \lambda & 0 & 0 & \cdots & 0 \\ 0 & \mu & -(\lambda+\mu) & \lambda & 0 & \cdots & 0 \\ 0 & 0 & \mu & -(\lambda+\mu) & \lambda & \cdots & 0 \\ 0 & \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \mu & -(\lambda+\mu) & \lambda \\ 0 & 0 & \cdots & 0 & 0 & \mu & -(\lambda+\mu) \end{pmatrix}$$

**Remark 2.2.3.** $\widetilde{\boldsymbol{A}}$ is not a yet a generator of a Markov chain because $\widetilde{\boldsymbol{A}}$ does not satisfy all the conditions of a Markov generator, all row entries does not sum to zero.

Let's look at the transpose of $\widetilde{\boldsymbol{A}}$.

$$\widetilde{\boldsymbol{A}}^T = \begin{pmatrix} 0 & \boldsymbol{0}^T \\ \boldsymbol{q}_\lambda & \boldsymbol{Q}^T \end{pmatrix} \tag{2.2.76}$$

$$= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ \lambda & -(\lambda+\mu) & \mu & 0 & 0 & \cdots & 0 \\ 0 & \lambda & -(\lambda+\mu) & \mu & 0 & \cdots & 0 \\ 0 & 0 & \lambda & -(\lambda+\mu) & \mu & \cdots & 0 \\ \vdots & \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \lambda & -(\lambda+\mu) & \mu \\ 0 & 0 & \cdots & 0 & 0 & \lambda & -(\lambda+\mu) \end{pmatrix} \tag{2.2.77}$$

**Remark 2.2.4.** $\widetilde{\boldsymbol{A}}^T$ is not a yet a generator of a Markov chain because $\widetilde{\boldsymbol{A}}^T$ does not satisfy all the conditions of a Markov generator, all row entries does not sum to zero. But now it is more straightforward to construct Markov generator from $\widetilde{\boldsymbol{A}}^T$.

We will now construct a Markov generator from $\widetilde{\boldsymbol{A}}^T$. To do this we need to first augment $\mu \boldsymbol{e}_k$ to the right of $\widetilde{\boldsymbol{A}}^T$, where

$$\mu \boldsymbol{e}_k = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ \mu \end{bmatrix}. \tag{2.2.78}$$

Additionally, to maintain a square matrix, we need to augment zero row to the bottom of $\widetilde{\boldsymbol{A}}^T$. Denote the resulting generator by $\widetilde{\boldsymbol{Q}}$ defined as

$$\widetilde{\boldsymbol{Q}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \lambda & -(\lambda+\mu) & \mu & 0 & 0 & \cdots & 0 & 0 \\ 0 & \lambda & -(\lambda+\mu) & \mu & 0 & \cdots & 0 & 0 \\ 0 & 0 & \lambda & -(\lambda+\mu) & \mu & \cdots & 0 & 0 \\ \vdots & \vdots & & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & \lambda & -(\lambda+\mu) & \mu & 0 \\ 0 & 0 & \cdots & 0 & 0 & \lambda & -(\lambda+\mu) & \mu \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

So $\widetilde{\boldsymbol{Q}}$ is a $k+2$ dimension square transition. We have constructed a transition rate matrix for a Markov chain that is absorbing in states 0 and state $k+1$. This gives us a great connection between a reflecting queueing process at states $\{0, k\}$ and an absorbing queueing process at states $\{0, k+1\}$, also known as two barrier absorbing

process. Since we constructed $\widetilde{\boldsymbol{Q}}$ from $\widetilde{\boldsymbol{A}}^T$ by adding a column and a row where $\widetilde{\boldsymbol{A}}$ is defined by

$$\widetilde{\boldsymbol{A}} = (\boldsymbol{I} - \boldsymbol{R})\boldsymbol{A}(\boldsymbol{I} - \boldsymbol{R})^{-1}$$

**Remark 2.2.5.** We can get the transient probabilities of the non absorbing states of the absorbing queueing process from the transition probabilities of the reflecting queueing process with generator $A$ by applying similarity transformation using the right shift operator and matrix exponential for block diagonal matrix. Since $\widetilde{\boldsymbol{A}} = U\boldsymbol{A}U^{-1}$ implies that the exponential has the form $e^{\widetilde{\boldsymbol{A}}} = Ue^{\boldsymbol{A}}U^{-1}$.

Similarly, consider the matrix $\widetilde{\boldsymbol{B}}$ obtained from the similiarity transformation of the generator $\boldsymbol{A}$ of a reflecting queueing model $M_\lambda/M_\mu/1/k$ defined by

$$\begin{aligned}\widetilde{\boldsymbol{B}} &= (\boldsymbol{I} - \boldsymbol{L})\boldsymbol{A}(\boldsymbol{I} - \boldsymbol{L})^{-1} \\ &= \begin{pmatrix} -(\lambda+\mu) & \lambda & 0 & 0 & \cdots & & 0 \\ \mu & -(\lambda+\mu) & \lambda & 0 & \cdots & & 0 \\ 0 & \mu & -(\lambda+\mu) & \lambda & \cdots & & 0 \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ 0 & \cdots & 0 & \mu & -(\lambda+\mu) & \lambda & 0 \\ 0 & \cdots & 0 & 0 & \mu & -(\lambda+\mu) & 0 \\ 0 & \cdots & 0 & 0 & 0 & \mu & 0 \end{pmatrix}\end{aligned} \tag{2.2.79}$$

**Remark 2.2.6.** $\widetilde{\boldsymbol{B}}$ is not a yet a generator of a Markov chain because $\widetilde{\boldsymbol{B}}$ does not satisfy all the conditions of a Markov generator, all row entries does not sum to zero.

Let's look at the transpose of $\widetilde{\boldsymbol{B}}$.

$$\widetilde{\boldsymbol{B}}^T = \begin{pmatrix} \boldsymbol{Q}^T & \boldsymbol{q}_\mu \\ \boldsymbol{0}^T & 0 \end{pmatrix}$$

$$= \begin{pmatrix} -(\lambda+\mu) & \mu & 0 & 0 & \cdots & & 0 \\ \lambda & -(\lambda+\mu) & \mu & 0 & \cdots & & 0 \\ 0 & \lambda & -(\lambda+\mu) & \mu & \cdots & & 0 \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ 0 & \cdots & 0 & \lambda & -(\lambda+\mu) & \mu & 0 \\ 0 & \cdots & 0 & 0 & \lambda & -(\lambda+\mu) & \mu \\ 0 & \cdots & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

**Remark 2.2.7.** $\widetilde{\boldsymbol{B}}^T$ is not a yet a generator of a Markov chain because $\widetilde{\boldsymbol{B}}^T$ does not satisfy all the conditions of a Markov generator, all row entries does not sum to zero. But straightforward to construct Markov generator from $\widetilde{\boldsymbol{B}}^T$.

We will now construct a Markov generator from $\widetilde{\boldsymbol{B}}^T$. To do this we need to first augment $\lambda \boldsymbol{e}_0$ to the right of $\widetilde{\boldsymbol{B}}^T$, where

$$\lambda \boldsymbol{e}_0 = \begin{bmatrix} \lambda \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{2.2.80}$$

Additionally, to maintain a square matrix, we need to augment zero row to the top of $\widetilde{\boldsymbol{B}}^T$. Denote the resulting generator by $\widetilde{\boldsymbol{Q}}_{\boldsymbol{B}}$ defined as

$$\widetilde{\boldsymbol{Q}}_{\boldsymbol{B}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & \cdots & & 0 \\ \lambda & -(\lambda+\mu) & \mu & 0 & 0 & \cdots & & 0 \\ 0 & \lambda & -(\lambda+\mu) & \mu & 0 & \cdots & & 0 \\ 0 & 0 & \lambda & -(\lambda+\mu) & \mu & \cdots & & 0 \\ \vdots & \vdots & & \ddots & \ddots & \ddots & & \vdots \\ 0 & 0 & \cdots & 0 & \lambda & -(\lambda+\mu) & \mu & 0 \\ 0 & 0 & \cdots & 0 & 0 & \lambda & -(\lambda+\mu) & \mu \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

So $\widetilde{\boldsymbol{Q}}_{\boldsymbol{B}}$ is a $k+2$ dimension square transition. We have constructed a transition rate matrix for a Markov chain that is absorbing in states $-1$ and state $K$. This gives us a great connection between a reflecting queueing process at states $\{0, k\}$ and an absorbing queueing process at states $\{-1, k\}$, also known as two barrier absorbing process. Since we constructed $\widetilde{\boldsymbol{Q}}_{\boldsymbol{B}}$ from $\widetilde{\boldsymbol{B}}^T$ by adding a column and a row where $\widetilde{\boldsymbol{B}}$ is defined by

$$\widetilde{\boldsymbol{B}} = (\boldsymbol{I} - \boldsymbol{L})\boldsymbol{A}(\boldsymbol{I} - \boldsymbol{L})^{-1}$$

**Remark 2.2.8.** We also get the transient probabilities of the non-absorbing states of the absorbing queueing process from the transition probabilities of the reflecting queueing process with generator $A$ by applying similarity transformation using the left shift operator and matrix exponential for block diagonal matrix. Since $\widetilde{\boldsymbol{B}} = V\boldsymbol{A}V^{-1}$ implies that the exponential has the form $e^{\widetilde{\boldsymbol{B}}} = Ve^{\boldsymbol{A}}V^{-1}$.

## 2.3 Allocation in a Bike-Sharing System

We now explore the optimal allocation problem for a station-based bike-sharing system. The bike-sharing system consists of a map of stations and a station consists of a set of docks. A user is then allowed to rent a bike from any station and return the bike at any station in the system.

In managing a BSS, rebalancing operations account for more than 50 percent of the operational costly; The term repositioning/rebalancing operation, the act of replenishing a station with bikes when it becomes empty and the act of removing bikes from a station when it is full, redistributes bicycles across the system to maintain a reasonable distribution across all docking stations Fishman et al. (2014). Ideally, you would like to rebalance as little as possible. In what follows, we develop a method to enable the operator of BSS to optimally allocate bikes to each station during the planning period of one day.

### 2.3.1 $M_{\lambda(t)}/M_{\mu(t)}/1/k$ Queueing Model for a Bike Station

We model a bike-sharing station as a double barrier reflecting $M_{\lambda(t)}/M_{\mu(t)}/1/k$ process with non-constant rates. In the previous section, we discussed the time constant rate analog of the double barrier reflecting process. In a bike-sharing station, there are two sets of customers: one group of customers is interested in renting a bike and the other group of customers is interested in returning a bike. Here we model both traffic as a Poisson process (counting process) with non-constant $\mu(t)$ and $\lambda(t)$ representing the rate people are arriving to rent and return a bike respectively.

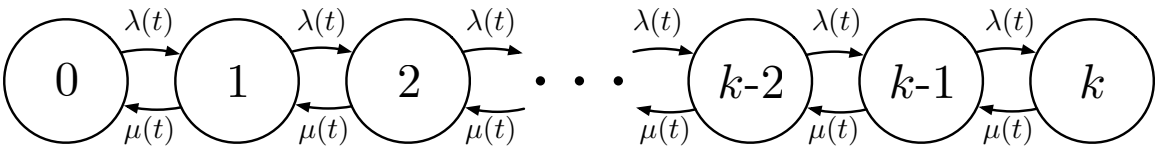



Figure 2.18: A single station inventory process state dynamics. Customers retrieve bicycles with rate $\mu(t)$ and return bicycles with rate $\lambda(t)$.

**Definition 2.3.1.** A stochastic process $\{X(t) : t \geq 0\}$ with discrete state space $\mathcal{S}$ is called a continous-time Markov chain (CTMC) if for all $t \geq 0$, $s \geq 0$, $i \in \mathcal{S}$, $i \in \mathcal{S}$

$$\begin{aligned} \mathbb{P}\Big(X(s+t) = j \Big| X(s) = i, \{X(u) : 0 \leq u < s\}\Big) &= \mathbb{P}\Big(X(s+t) = j \Big| X(s) = i\Big) \\ &= P_{i,j}(t) \end{aligned} \tag{2.3.1}$$

**Lemma 2.3.1.** The stochastic process $\{Q(t) : t \geq 0\}$ defined on the state space $\{0, 1, \ldots, k\}$ representing the inventory at a single station at time $t$, under the $M_{\lambda(t)}/M_{\mu(t)}/1/k$ queueing model, is a continous state Markov chain.

*Proof.* See Lajos (1962) for a proof. □

### 2.3.2 The Transition Probabilities for $M_{\lambda(t)}/M_{\mu(t)}/1/k$ Queues

We first give an important lemma on the multiplicative property of stochastic matrices popularly known as the ChapmanKolmogorov equation.

**Lemma 2.3.2.** The product of two row-stochastic square matrices is also a row-stochastic matrix. Similarly, the product of two column-stochastic square matrices is also a column-stochastic matrix. Moreover, the product of two doubly-stochastic square matrices is also a doubly-stochastic matrix.

*Proof.* Suppose matrices $A \in \mathbb{R}^{k\times k}$ and $B \in \mathbb{R}^{k\times k}$ are row stochastic. Then $\sum_{j}^{k} A_{ij} = 1$ for each row $i \in \{1, 2, \ldots, k\}$ and $\sum_{j}^{k} B_{ij} = 1$ for each row $i \in \{1, 2, \ldots, k\}$. Now consider the sum of the elements on each row $i \in \{1, 2, \ldots, k\}$ of the dot product of the two matrices A and B. Denote the dot product by AB.

$$\begin{aligned}
\sum_{j=1}^{k}(AB)_{ij} &= \sum_{j=1}^{k}\left(\sum_{\ell=1}^{k} A_{i\ell}B_{\ell j}\right) \\
&= \sum_{\ell=1}^{k}\left(A_{i\ell}\Big(\sum_{j=1}^{k} B_{\ell j}\Big)\right) \\
&= \sum_{\ell=1}^{k} A_{i\ell}\cdot 1 \\
&= 1. \qquad (2.3.2)
\end{aligned}$$

In equation (2.3.2), we used the fact that we have finite summation to swap the order of summation and $A_{i\ell}$ does not depend $j$. The proof for column stochastic is analogous. Suppose matrices $A \in \mathbb{R}^{k\times k}$ and $B \in \mathbb{R}^{k\times k}$ are column stochastic. Then $\sum_{i}^{k} A_{ij} = 1$ for each row $j \in \{1, 2, \ldots, k\}$ and $\sum_{i}^{k} B_{ij} = 1$ for each row $j \in \{1, 2, \ldots, k\}$. Now consider the sum of the elements on each column $j \in \{1, 2, \ldots, k\}$ of the dot product of the two matrices A and B. Denote the dot product by AB.

$$\begin{aligned}
\sum_{i=1}^{k}(AB)_{ij} &= \sum_{i=1}^{k}\left(\sum_{\ell=1}^{k} A_{i\ell}B_{\ell j}\right) \\
&= \sum_{\ell=1}^{k}\left(B_{\ell j}\Big(\sum_{i=1}^{k} A_{i\ell}\Big)\right) \\
&= \sum_{\ell=1}^{k} B_{\ell j}\cdot 1 \\
&= 1. \qquad (2.3.3)
\end{aligned}$$

For the doubly stochastic matrices $A \in \mathbb{R}^{k\times k}$ and $B \in \mathbb{R}^{k\times k}$, it remains to show that $0 \leq (AB)_{ij} \leq 1 \ \forall i, j \in \{1, 2, \ldots, k\}$.

$$\begin{aligned} 0 &\leq (AB)_{ij} \\ &= \sum_{\ell=1}^{k} A_{i\ell} B_{\ell j} \\ &\leq \sum_{\ell=1}^{k} A_{i\ell} \\ &= 1. \end{aligned} \tag{2.3.4}$$

In equation (2.3.4), we used the fact that we have $0 \leq B_{\ell j} \leq 1 \quad \forall \ell, j \in \{1, 2, \ldots, k\}$ and $0 \leq A_{i\ell} B_{\ell j} \leq A_{i\ell} \quad \forall i, j, \ell \in \{1, 2, \ldots, k\}$. □

We now discuss three approximation methods to obtain the transition probabilities for the $M_{\lambda(t)}/M_{\mu(t)}/1/k$.

**Numerical Approximation:**

We first approximate the transient probabilities for the non-constant rate $M_{\lambda(t)}/M_{\mu(t)}/1/k$ queueing model using numerical integration of the first-order Kolmogorov equations. The Kolmogorov forward equations for $M_{\lambda(t)}/M_{\mu(t)}/1/k$ queueing model is given by:

$$\frac{d}{dt} p_m(t, 0) = \mu(t) \cdot p_m(t, 1) - \lambda(t) \cdot p_m(t, 0) \tag{2.3.5}$$

$$\frac{d}{dt} p_m(t, \ell) = \mu(t) \cdot p_m(t, \ell + 1) + \lambda(t) \cdot p_m(t, \ell - 1) - \big(\lambda(t) + \mu(t)\big) \cdot p_m(t, \ell) \tag{2.3.6}$$

$$\forall \ell \in \{1, \ldots, k - 1\}$$

$$\frac{d}{dt} p_m(t, k) = \lambda(t) \cdot p_m(t, k - 1) - \mu(t) \cdot p_m(t, k). \tag{2.3.7}$$

Since the above differential equations are linear, we use the forward Euler method, which is a first-order numerical procedure for solving ordinary differential equa-

tions (ODEs) with a given initial value, to find numerical approximations to the solutions of the system of ODEs. Let $f\big(p_m(t), \lambda(t), \mu(t)\big)$, $g\big(p_m(t), \lambda(t), \mu(t)\big)$, and $h\big(p_m(t), \lambda(t), \mu(t)\big)$ represent the right hand sides of Equations 2.3.5, 2.3.6, and 2.3.7 respectively. Moreover let $\Delta t$ be the time step, then the Euler method gives the following updates:

$$p_m(t + \Delta t, 0) \leftarrow p_m(t, 0) + \Delta t \cdot f\big(p_m(t), \lambda(t), \mu(t)\big) \tag{2.3.8}$$

$$p_m(t + \Delta t, \ell) \leftarrow p_m(t, \ell) + \Delta t \cdot g\big(p_k(t), \lambda(t), \mu(t)\big) \quad \forall m \in \{1, \dots, k-1\} \tag{2.3.9}$$

$$p_m(t + \Delta t, k) \leftarrow p_m(t, k) + \Delta t \cdot h\big(p_m(t), \lambda(t), \mu(t)\big) \tag{2.3.10}$$

The error due to the forward Euler approximation is of order $O(\Delta t)$ and becomes smaller as the time step $\Delta t$ becomes finer. For our numerical experiments, we set $\Delta t = 10^{-3}$. Moreover, the solution of the forward equations in the non-stationary case is the solution to a system of $k+1$ ordinary differential equations. The solution can be written in matrix notation by defining

$$\boldsymbol{P}(t) = \begin{bmatrix} p_0(t,0) & \dots & p_0(t,m) & \dots & p_0(t,k) \\ \vdots & \dots & \vdots & \dots & \vdots \\ p_i(t,0) & \dots & p_i(t,m) & \dots & p_i(t,k) \\ \vdots & \dots & \vdots & \dots & \vdots \\ p_k(t,0) & \dots & p_K(t,m) & \dots & p_k(t,k) \end{bmatrix} \in \mathbb{R}^{(k+1)\times(k+1)} \tag{2.3.11}$$

and

$$\mathbf{A}(t) = \begin{bmatrix} a_{00} & \dots & a_{0m} & \dots & a_{0k} \\ \vdots & \dots & \vdots & \dots & \vdots \\ a_{i0} & \dots & a_{im} & \dots & a_{ik} \\ \vdots & \dots & \vdots & \dots & \vdots \\ a_{k0} & \dots & a_{km} & \dots & a_{kk} \end{bmatrix} \in \mathbb{R}^{(k+1)\times(k+1)} \tag{2.3.12}$$

where $a_{ii} = -(\lambda + \mu)$, $a_{i,i-1} = \mu$, $a_{i,i+1} = \lambda$ for $i = 1, 2, \ldots, k-1$, with $a_{00} = -\lambda$, $a_{01} = \lambda$ and $a_{kk} = -\mu$, $a_{k,k-1} = \mu$ and $a_{i,\ell} = 0$ for $\{i, \ell : |\ell - i| > 1\}$. Then ODEs can be written in matrix notation

$$\frac{d}{dt}\boldsymbol{P}(t) = \boldsymbol{P}(t)\boldsymbol{A}(t) \tag{2.3.13}$$

and the initial condition is $\boldsymbol{P}(0) = \boldsymbol{I}$. The notation in Equation 2.3.13 is often referred to as the operator notation. Since the differential equation is linear, we can write a solution to the differential equation in closed-form, when the operator does not depend on time. The solution to the differential equation given in the Equation 2.3.13 can be written as

$$\boldsymbol{P}(t) \equiv \boldsymbol{P}(0)e^{\boldsymbol{A}t} \tag{2.3.14}$$

where $e^{\boldsymbol{A}t}$ is the exponential of a matrix defined as the element by element sum of exponential series. A solution written in terms of trigonometric functions is also available (Morse, 1958; Lajos, 1962).

**Matrix approximation:**

The second method is directly related to the previous numerical method. We saw the closed-form solution $\boldsymbol{P}(t) = \boldsymbol{P}(0)e^{\boldsymbol{A}t}$, where $\boldsymbol{A}$ is the generator for the $M_{\lambda(t)}/M_{\mu(t)}/1/k$ queueing model with constant the rates are constant (i.e when $\lambda(t) = \lambda$ and $\mu(t) = \mu$) for the planning horizon $[0, T]$. We will use also use the results from Lemma 2.3.2, which states that the dot product of two row-stochastic squared matrices is also a row-stochastic matrix.

Now, to approximate the transition probabilities for the $M_{\lambda(t)}/M_{\mu(t)}/1/k$ queueing model in the time interval $[0, T]$, we first partition the closed time interval $[0, T]$ into sub-intervals. Consider the a finite sequence $t_0, t_1, t_2, \ldots, t_{p-1}, t_p$ of real numbers such

that $0 = t_0 < t_1, < t_2, \ldots, t_{p-1} < t_p = T$, we obtain a $p$ partitions, of the closed time interval $[0, T]$, denoted by $\mathrm{P} = \{(t_0, t_1], (t_1, t_2], \ldots, (t_{p-2}, t_{p-1}], (t_{p-1}, t_p]\}$.

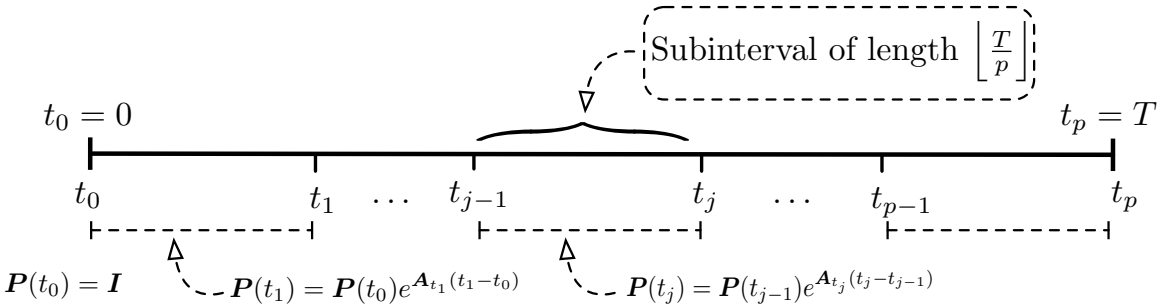



Figure 2.19: Illustration of the partition of finite time interval $[0, T]$

where each sub-interval $(t_{j-1}, t_j]$ has length $\left\lfloor \frac{T}{p} \right\rfloor$ for $j \in \{1, 2, \ldots, p-1, p\}$. We assume the arrival and servise rate are constant in each partition and set them to the average rate in that interval. For each sub-interval $(t_{j-1}, t_j]$ for $j \in \{1, 2, \ldots, p-1, p\}$, we denote the rate matrix in the interval $(t_{j-1}, t_j]$ by $A_{t_j}$. Then by the result of Lemma 2.3.2, we can recursively calculate the transition probabilities as follows:

$$
\begin{aligned}
\boldsymbol{P}(t_1) &= \boldsymbol{P}(t_0)e^{\boldsymbol{A}_{t_1}(t_1 - t_0)} \\
\boldsymbol{P}(t_2) &= \boldsymbol{P}(t_1)e^{\boldsymbol{A}_{t_2}(t_2 - t_1)} \\
&\vdots \\
\boldsymbol{P}(t_j) &= \boldsymbol{P}(t_{j-1})e^{\boldsymbol{A}_{t_j}(t_j - t_{j-1})} \quad \text{for } j \in \{1, 2, \ldots, p-1, p\}.
\end{aligned} \tag{2.3.15}
$$

**Simulation Approximation:**

The last method for estimating the transition probabilities for the time-varying $M_{\lambda(t)}/M_{\mu(t)}/1/k$ queueing model we considered is the Monte Carlo simulation. This approach is essentially using an empirical average estimator given in Equation 2.3.16 to approximate the transition probabilities.

$$\widehat{p}_m(t,n) = \frac{1}{N}\sum_{i=1}^{N}\{Q_i(t) = n\} \qquad \forall m,n \in \{0,1,\ldots,k\} \tag{2.3.16}$$

were the initial queue length is $Q(0) = m$.

### 2.3.3 Performance Metrics for $M_{\lambda(t)}/M_{\mu(t)}/1/k$ Queueing Model

| Parameters | Meaning |
|---|---|
| $\mathcal{S}$ | The set of all station in the network |
| $m$ | The initial inventory level |
| $m^\star$ | The optimal initial inventory level after the rebalancing process |
| $k$ | The dock capacity of the station |
| $\pi_e$ | The penalty charged for each customer lost due to a shortage of bicycles |
| $\pi_f$ | The penalty charged for each customer lost due to a shortage of vacant docks |
| P | The discretized set of the time interval $[0,T]$ into $p$ partitions |
| $J(x)$ | The customer dissatisfaction for the station |
| $\widehat{J}(m)$ | The discretized customer dissatisfaction function for the station |
| $p_m(t,n)$ | the transition probability of starting at state $m$ and ending at state $n$ |
| $\widehat{p}_m(t,n)$ | the approximate transition probability of starting at state $m$ and ending at state |

Table 2.2: The notation table for the allocation problem

As a performance metric for the double barrier reflecting $M_{\lambda(t)}/M_{\mu(t)}/1/k$ queueing model, we measure the expected total number of unserved/upset users in the rest of the planning period. Two events mostly lead to upset customers in a bike-sharing station. The first event occurs when a station is empty, preventing an arriving customer who is seeking to rent a bike from receiving service. The second event occurs when a station is full, preventing an arriving customer who is seeking to return a bike from

receiving service. This performance metric is popularly known as the user dissatisfaction penalty function; the user dissatisfaction was inspired and first made popular by Raviv and Kolka (2013) and is becoming a unified metric in the literature Fishman et al. (2014); Schuijbroek et al. (2017); Tang et al. (2019). The user dissatisfaction function $J(m)$, also known as the penalty function, is defined as follows:

$$\begin{aligned} J(m) &= \int_0^T \Big[\pi_e \cdot \mathbb{P}\big(Q(t)=0|Q(0)=m\big)\mu(t) + \pi_f \cdot \mathbb{P}\big(Q(t)=k|Q(0)=m\big)\lambda(t)\Big]dt. \\ &= \int_0^T \Big[\pi_e \cdot p_m(t,0)\cdot\mu(t) \;+\; \pi_f \cdot p_m(t,k)\cdot\lambda(t)\Big]dt. \end{aligned} \tag{2.3.17}$$

The user dissatisfaction definition has two terms in the integral. The first term in the integral represents the cost incurred due to shortage of bicycles (cost incurred due to abandonments of users hoping to rent a bicycle) and the second term in the integral represents the cost incurred due to shortage of parking docks (cost incurred due to abandonments of users hoping to return a bicycle). So the user dissatisfaction function represents the expected total cost incurred for unserved users during the remainder of the planning period $[0,T]$. However when $\pi_e = \pi_f = 1$ then the user dissatisfaction function represents the expected number of unserved users in the planning period $[0,T]$. To maintain a desired quality of service (QoS) at each self-serving bike station, the service operator needs to reduce the user-dissatisfaction at each station. This is equivalent to solving the following unconstrained optimization problem for each station:

$$m^* \in \underset{0\le m\le k}{\operatorname{argmin}} \; J(m) \tag{2.3.18}$$

The optimization problem in Equation 2.3.18 is fairly straight forward, except that the the transition probabilities $p_m(t,0)$ and $p_m(t,k)$ are not known and hence need to be approximated. However, by discretizing the time interval $[0,T]$ we can approximate those transition probabilities and hence obtain an approximation of the user-dissatisfaction function. We first partition the closed time interval $[0,T]$, on the real-line, into a finite sequence $t_0, t_1, t_2, \ldots, t_{p-1}, t_p$ of real numbers such that $0 = t_0 < t_1, < t_2, \ldots, t_{p-1} < t_p = T$. So we have $p$ partitions of the form $\mathrm{P} = \big\{(t_0,t_1],(t_1,t_2],\ldots,(t_{p-2},t_{p-1}],(t_{p-1},t_p]\big\}$. Below, we give a straight forward Riemann approximation of user-dissatisfaction function by a finite sum.

$$\begin{aligned}\widehat{J}(m) &\triangleq \sum_{i=1}^{p}\Big(\pi_e \cdot p_m(t_i,0)\mu(t_i) + \pi_f \cdot p_m(t_i,k)\lambda(t_i)\Big)\Delta t_i \\ &\approx \frac{T}{p}\sum_{i=1}^{p}\Big(\pi_e \cdot \widehat{p}_m(t_i,0)\mu(t_i) + \pi_f \cdot \widehat{p}_m(t_i,k)\lambda(t_i)\Big) \end{aligned} \tag{2.3.19}$$

were $\Delta t_i = t_i - t_{i-1} = \frac{T}{p}$ is the length of sub-interval $(t_{i-1}, t_i]$. As $\Delta t_i$ approaches zero, we expect to get a better approximation. In subsection 2.3.2, we will discuss in detail how to approximate the time-inhomogenous transition probability within each subinterval of the partition, which is critical to perform the optimization discussed in this section. This approximation of the user dissatisfaction function allows us to easily and accurately minimize the user-dissatisfaction function. Hence, we have

$$m^* \in \underset{0\le m\le k}{\operatorname{argmin}}\, J(m) \approx \underset{0\le m\le k}{\operatorname{argmin}}\, \widehat{J}(m). \tag{2.3.20}$$

Next, we discuss how we use the user dissatisfaction performance metric to guide data generation and train a policy recommender.

### 2.3.4 Allocation Policy Recommender via Machine Learning

In this section, we use machine learning to develop a policy for recommending the number of bikes to allocate in a bike-sharing station during the planning period. Figure 2.20, summarizes the architecture of the recommendation policy. The recommendation policy takes as input the rental rate, the return rate, and the number of docks (the parameter of the $M_\lambda/M_\mu/1/k$ queue) and outputs the optimal allocation given by our objective function. And the notion of optimal is relative to the user dissatisfaction function performance metric.

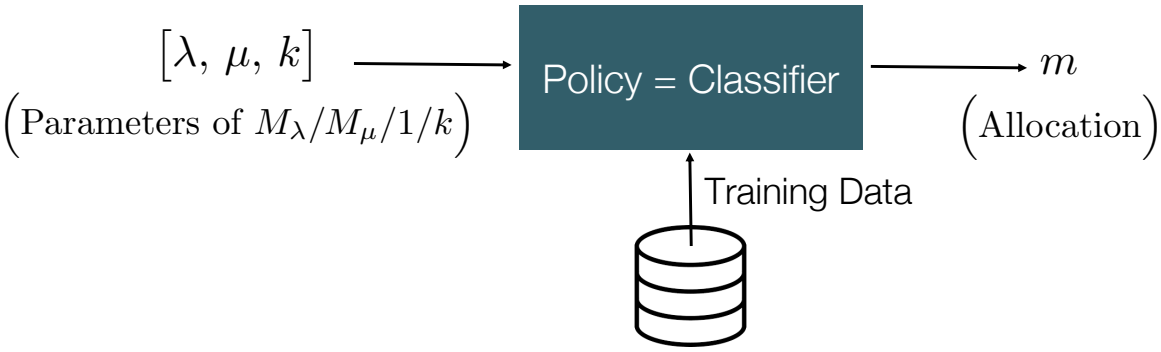



Figure 2.20: Illustration of the allocation recommender pipeline

Next, we discuss how to generate quality data for training the allocation recommender policy.

| | |
|---|---|
| $\lambda_i$ | The return rate of bikes for the $i^{th}$ sample |
| $\mu_i$ | The renting rate of bikes for the $i^{th}$ sample |
| $k_i$ | The dock capacity of bike station for the $i^{th}$ sample |
| $N$ | The total number of training sample |
| $x^{(i)}$ | The list of queue parameters (feature list) for the $i^{th}$ sample |
| $y^{(i)}$ | The allocation (label) corresponding to $x^{(i)}$ |

Table 2.3: The notation table for data generation

**Algorithm 4:** ML Data Generation

**Input:** Given the minimum and maximum arrival rate values $\lambda_{\min}$, $\lambda_{\max}$ respectively, the minimum and maximum service rate values $\mu_{\min}$, $\mu_{\max}$ respectively, the minimum and maximum capacity values $k_{\min}$, $k_{\max}$ respectively, and the number of sample $N$.

**Output:** Generated data $\mathcal{S} = \left\{ \left(x^{(i)}, y^{(i)}\right) \right\}_{i=1}^{N}$ .

**for** $i = 1, 2, \ldots, N$ **do**

1 Sample:

$\lambda_i \sim \mathcal{U}(\lambda_{\min}, \lambda_{\max})$ continuous uniform ,

$\mu_i \sim \mathcal{U}(\mu_{\min}, \mu_{\max})$ continuous uniform,

$k_i \sim \mathcal{U}(k_{\min}, k_{\max})$ discrete uniform.

2 Set the feature list $x^{(i)} = [\lambda_i,\ \mu_i,\ k_i]$.

3 Set the label value $y^{(i)} = \underset{0 \le m \le k_i}{\operatorname{argmin}} J_{\lambda_i, \mu_i, k_i}(m)$.

**end**

4 $\mathcal{S} = \left\{ \left(x^{(i)}, y^{(i)}\right) \right\}_{i=1}^{N}$

Algorithm 4 delineats the process of data generation. The algorithm outputs a labeled data of the form $\mathcal{S} = \left\{ \left(x^{(i)}, y^{(i)}\right) \right\}_{i=1}^{N}$. Next, we discuss a subroutine for training the policy recommender. This subroutine takes the training samples as input and outputs a mapping, often called a hypothesis function, $H_\theta : \mathcal{X} \to \mathcal{Y}$ such that $H_\theta\left(x^{(i)}\right) \approx y^{(i)}$. For our computational work, we will use a neural network base hypothesis function as the classifier, however, other reasonable hypothesis functions for multiclass classification would suffice.

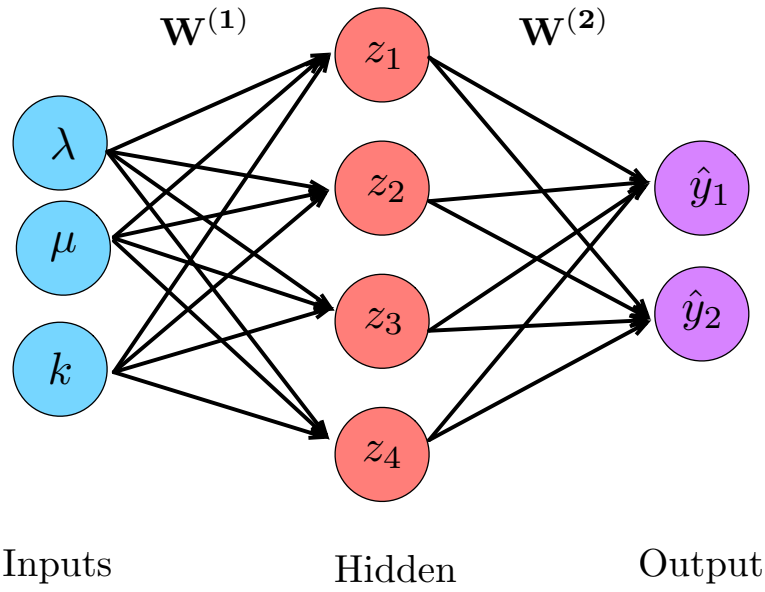



Figure 2.21: Example of a simple feed-forward neural network architecture

Figure 2.21 shows an example a feedforward neural network architecture, where the hidden nodes $z_i$ and the output nodes $y_i$ are defined as follows:

$$\begin{aligned} z_i &= \boldsymbol{W}_{0,i}^{(1)} + \lambda \boldsymbol{W}_{1,i}^{(1)} + \mu \boldsymbol{W}_{2,i}^{(1)} + k \boldsymbol{W}_{3,i}^{(1)} \\ &= \boldsymbol{W}_{0,i}^{(1)} + \sum_{j=1}^{3} x_j \boldsymbol{W}_{j,i}^{(1)} \end{aligned}$$

where

$$\boldsymbol{X} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \lambda \\ \mu \\ k \end{bmatrix}$$

and the outputs

$$\widehat{y}_i = g\left( \boldsymbol{W}_{0,i}^{(2)} + \sum_{j=1}^{4} g(z_j) \boldsymbol{W}_{j,i}^{(2)} \right) \tag{2.3.21}$$

where $g(\cdot)$ is the non-linear activation function, for example $g(z) = \tanh(z)$.

## 2.4 Numerical and Computational Results

We present the computational and numerical results in this section.

### 2.4.1 Transition Probabilities Results

Figures 2.22 shows the plots of the transition probabilities, for the free process, as a function of time using three methods: Integration, Bessel function, and Monte-Carlo simulation. The integration and Bessel function methods give an exact form for the transition probabilities and the simulation method gives an approximation of the transient probabilities. As expected, the simulation results closely approximate the exact results for the transition probabilities.

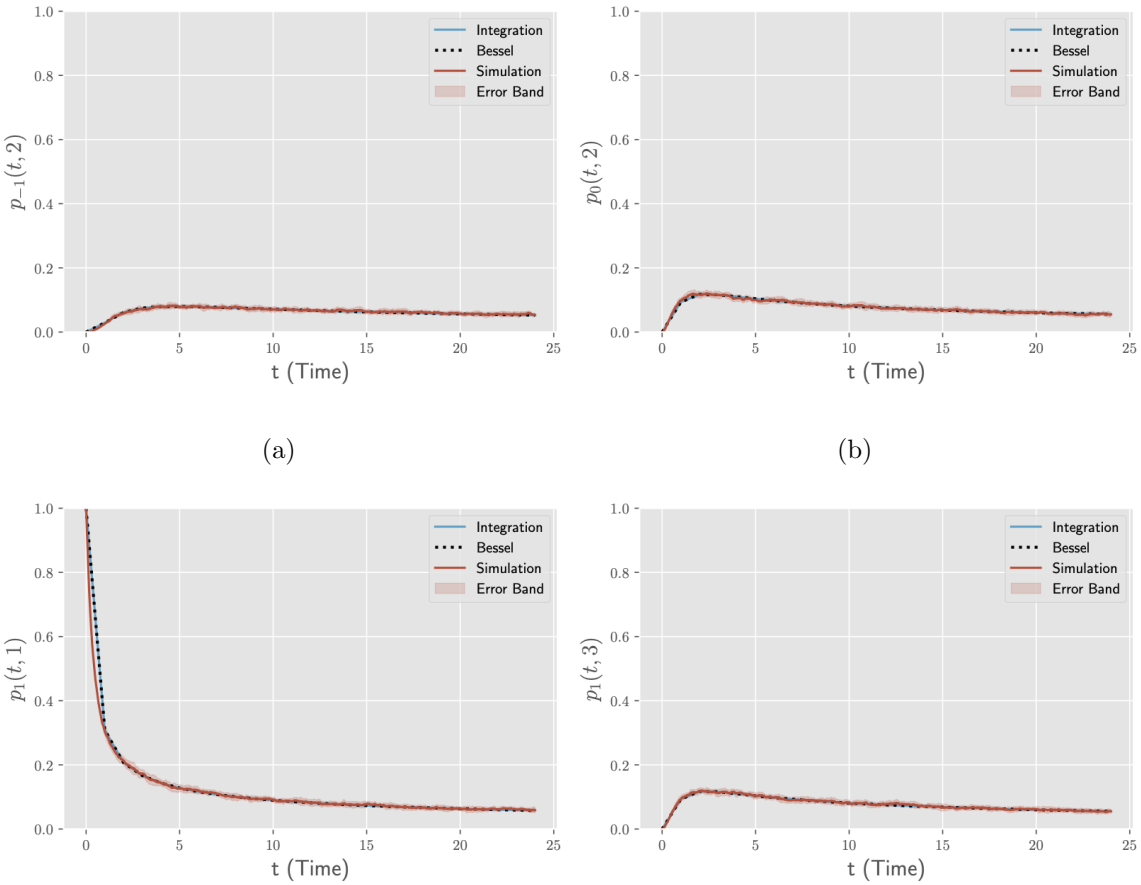



Figure 2.22: The transition probabilities of the free process as a function of time using three methods: Integration, Bessel function, and Monte-Carlo simulation. The rates $\lambda = \mu = 1$, time steps $\Delta t = 10^{-3}$, and sample size $N = 10^3$.

Figures 2.23 shows the plots of the transition probabilities, for the constant rate $M_\lambda/M_\mu/1/k$ queue, as a function of time using three methods: Integration, Exponentiation, and Monte-Carlo simulation. The integration and Exponentiation methods give an exact form for the transition probabilities and the simulation method gives an approximation of the transient probabilities. As expected, the simulation results closely approximate the exact results for the transition probabilities.

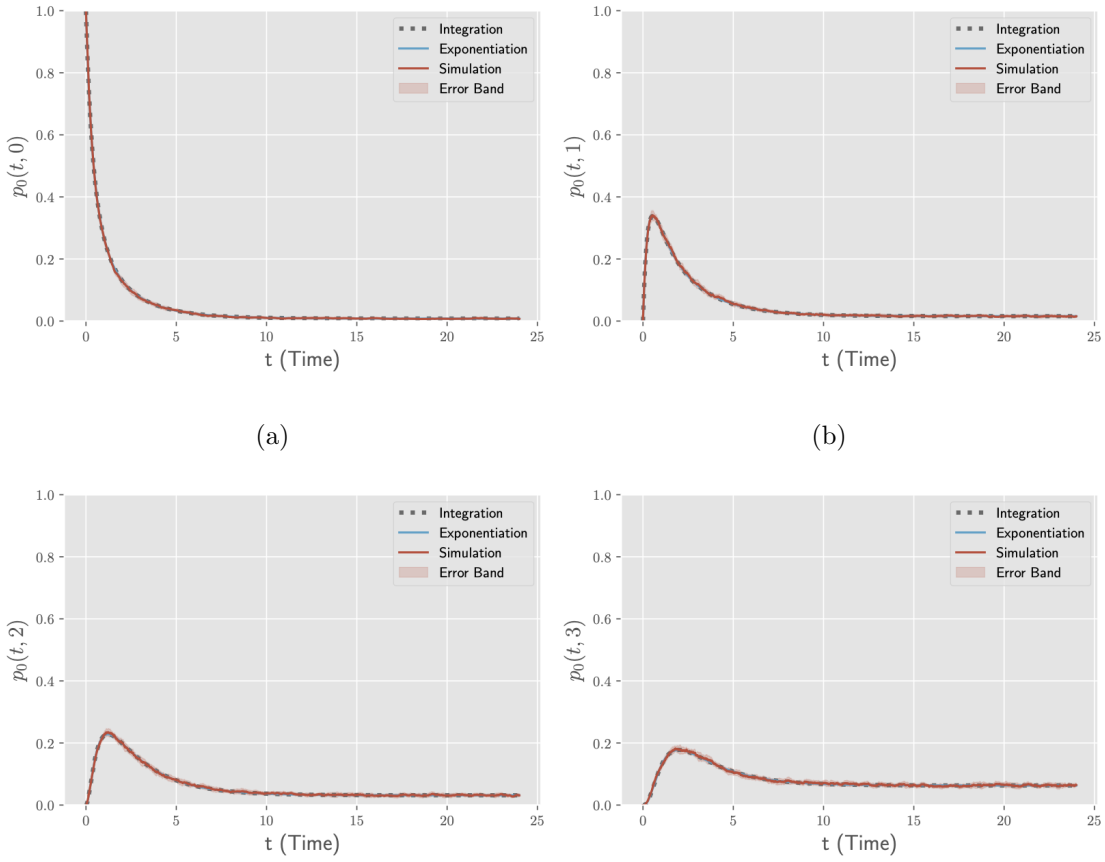
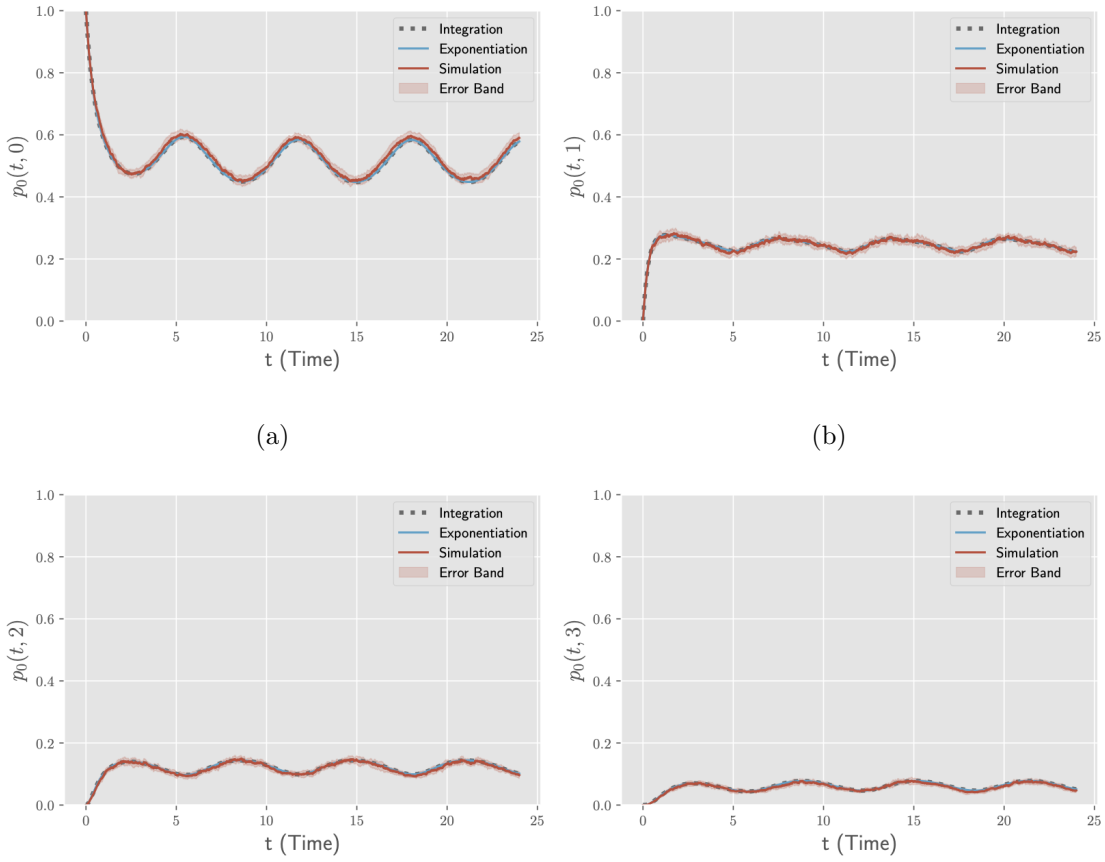



Figure 2.23: The transition probability estimation for $M_\lambda/M_\mu/1/k$ queue as a function of time. Using the numerical approximation technique, the matrix exponentiation technique, and simulation technique. We use constant rates $\lambda = 2$, $\mu = 1$, capacity $k = 6$, time steps $\Delta t = 10^{-3}$, and sample size $N = 10^3$.

Figures 2.24 shows the plots of the transition probabilities, for the non-constant rate $M_{\lambda(t)}/M_{\mu(t)}/1/k$ queue, as a function of time. We show the results for the exponentiation estimation technique and the numerical approximation technique and the simulation approximation technique. As expected the exponentiation approximation technique seems to match the numerical approximation procedure for the transient probabilities.



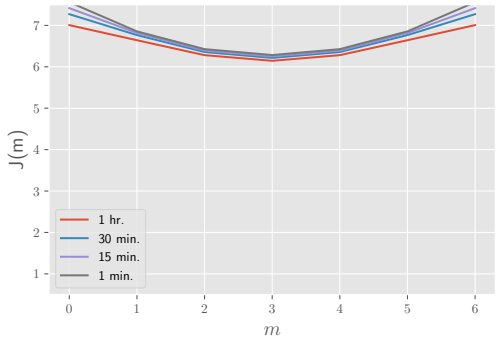

(a) (b)

(c) (d)

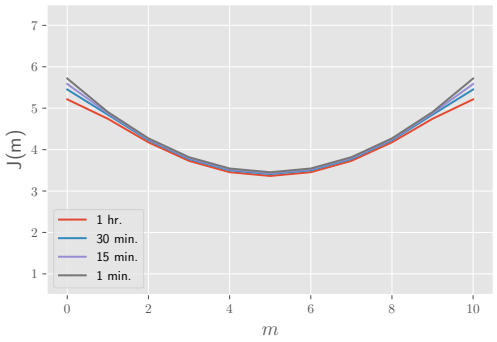
Figure 2.24: The transition probability estimation for $M_{\lambda(t)}/M_{\mu(t)}/1/k$ queue as a function of time. Using the numerical approximation technique, the matrix exponentiation technique, and simulation technique. We use time varying rates $\lambda = 1 + 0.5\sin(t)$, $\mu = 2 + 0.5\sin(t)$, capacity $k = 6$, $\Delta t = 10^{-3}$, and $N = 10^3$ samples.

### 2.4.2 Allocation Insights via Case Studies

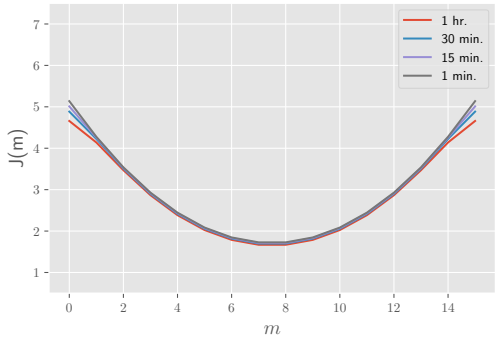
We first show the plots for the performance matric objective function $J(m)$, in the constant rate setting, highlighting cases when the pickup and the return rates are either symmetric or non-symmetric in Figures 2.25, 2.26, and 2.27.



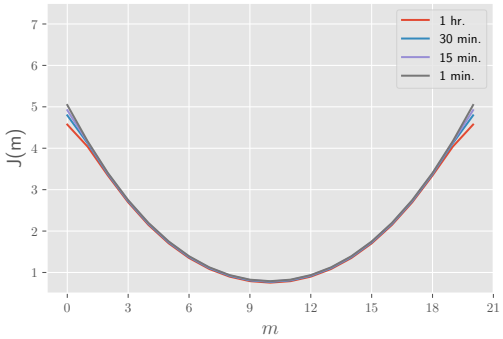

(a) $k = 6$




(b) $k = 10$

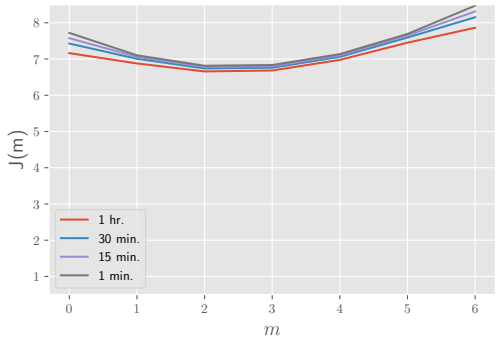



(c) $k = 15$

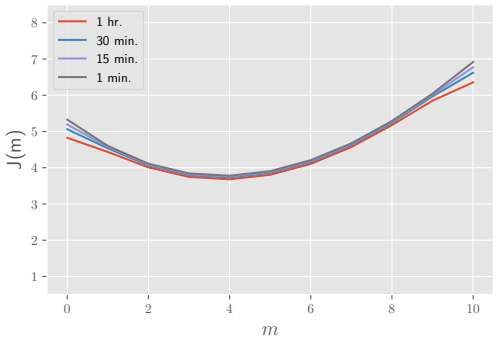



(d) $k = 20$

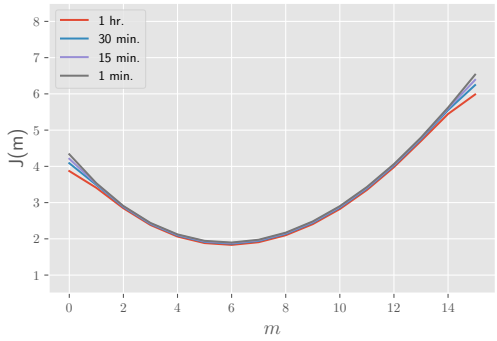
Figure 2.25: The plot of the objective function with constant pickup and return rates. The pickup rate equals the return rate $\mu = \lambda = 1$. Under four different discretizations of time: $\Delta t = 1\,hour$, $\Delta t = 30\,minutes$, and $\Delta t = 1\,minute$.



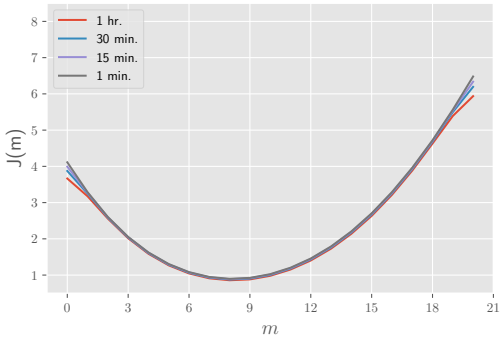

(a) $k = 6$




(b) $k = 10$

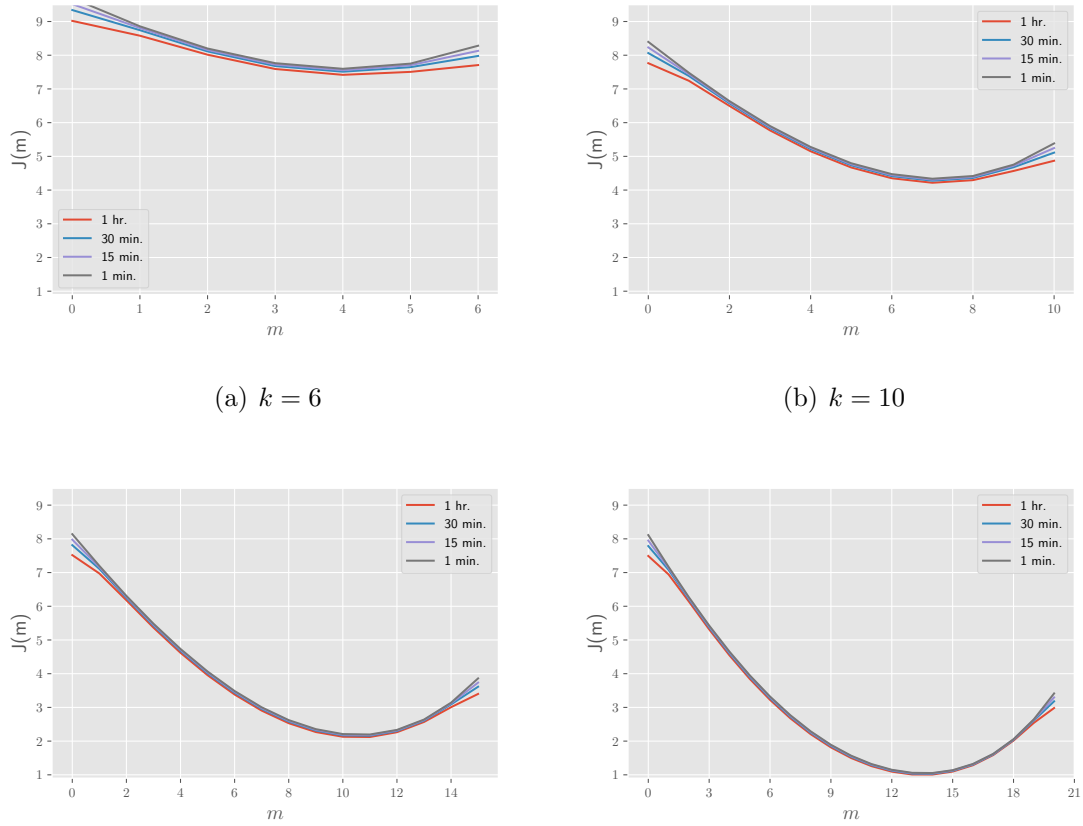



(c) $k = 15$




(d) $k = 20$

Figure 2.26: The plot of the objective function with constant pickup and return rates. The pickup rate is less than the return rate $\mu = 1$ and $\lambda = 1.1$. Under four different discretizations of time: $\Delta t = 1\,hour$, $\Delta t = 30\,minutes$, and $\Delta t = 1\,minute$.

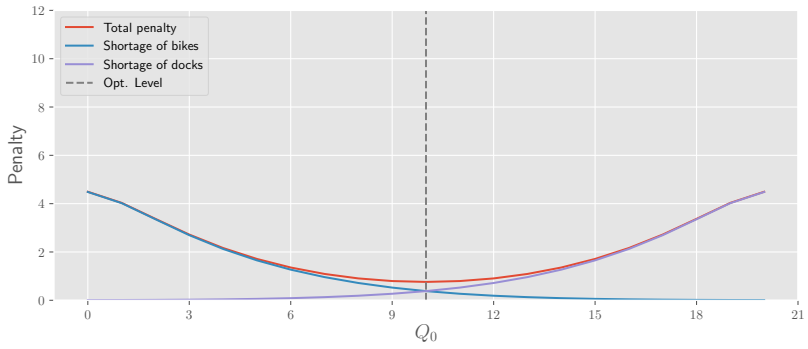



(a) $k = 6$ (b) $k = 10$

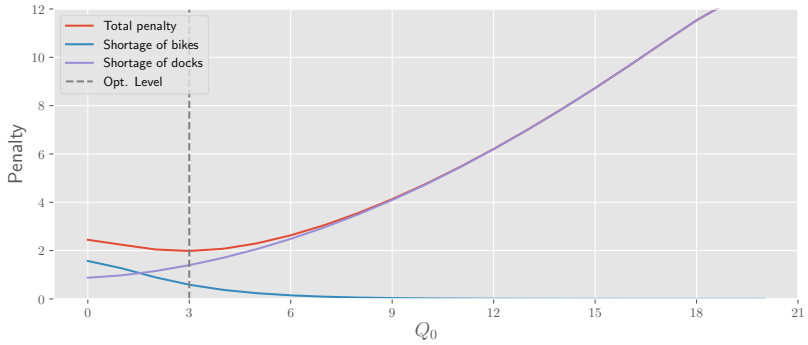
(c) $k = 15$ (d) $k = 20$

Figure 2.27: The plot of the objective function with constant pickup and return rates. The pickup rate is greater than the return rate $\mu = 1.2$ and $\lambda = 1$. Under four different discretizations of time: $\Delta t = 1\,hour$, $\Delta t = 30\,minutes$, and $\Delta t = 1\,minute$.

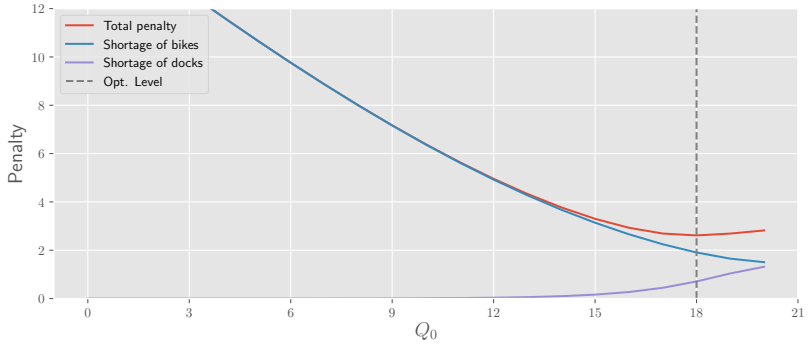
Next, we show the plots for the performance matric objective function $J(m)$, in the non-constant rate setting, highlighting cases when the pickup and the return rates are either symmetric or non-symmetric in Figure 2.28. The time step is set to one hour increment. Moreover, the station capacity $k = 20$; the penalty charge for each customer lost due to a shortage of bicycles and docks is the same $\pi_e = \pi_f = 1$.




(a) Symmetric rates: $\lambda(t) \equiv \mu(t) = 1 + 0.5\sin(t)$




(b) Non-symmetric rates: $\lambda(t) = 1.5 + 0.5\sin(t)$ and $\mu(t) = 1 + 0.5\sin(t)$

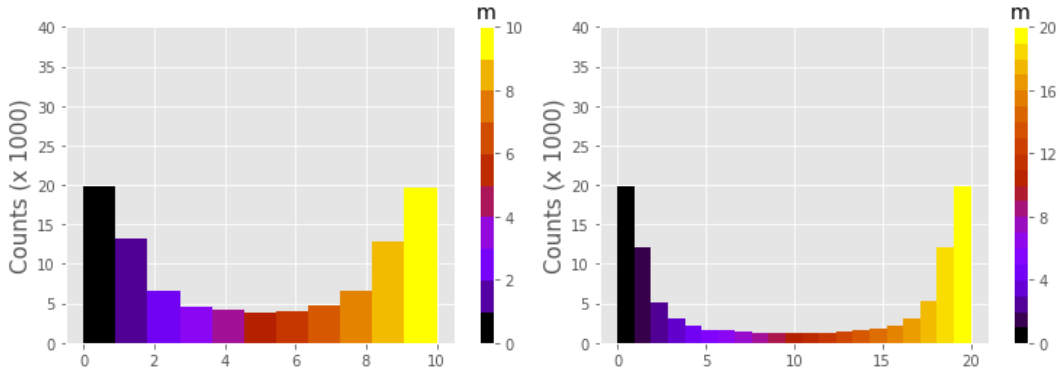



(c) Non-symmetric rates: $\lambda(t) = 1 + 0.5\sin(t)$ and $\mu(t) = 1.6 + 0.5\sin(t)$

Figure 2.28: The plot of different penalties incurred due to lost demands at a single station. In (a), the optimal objective value is 0.760, which was achieved at $Q_0^* = 10$. In (b), the optimal objective value is 1.983, which was achieved at $Q_0^* = 3$. In (c), the optimal objective value is 2.612, which was achieved at $Q_0^* = 18$.

### ML Insights

In what follows, we generate 2 datasets using the Algorithm 4 with the following parameters: $\lambda_{\min} = 1$, $\lambda_{\max} = 10$ respectively, the minimum and maximum service rate values $\mu_{\min} = 1$, $\mu_{\max} = 10$ respectively, and the number of samples $N = 10^5$. The above parameters are fixed for all the data sets. The variable parameters are the minimum and maximum capacity values $k_{\min} = k_{\max} = k$, where $k = 10$ and $k = 20$.




(a) $k = 10$ (b) $k = 20$

Figure 2.29: The distribution of the target variable for the 2 datasets from constant rate $M_\lambda/M_\mu/1/k$ queue. The variable $m$ represents the class label.

Similarly, we also generated an example data for time-varying case where $\lambda(t) = \lambda + \bar{\lambda}\sin(\alpha t)$ and $\mu(t) = \mu$. We use a variant of Algorithm 4, where the $i^{th}$ feature list is $x^{(i)} = [\lambda_i, \bar{\lambda}_i, \alpha_i, \mu_i]$. We sample $\alpha \sim \mathcal{U}(\alpha_{\min}, \alpha_{\max})$, $\alpha_{\min} = 1$, $\alpha_{\max} = 10$, $\bar{\alpha} \sim \mathcal{U}(\lambda_{\min}, \lambda)$. And $\lambda_{\min} = 1$, $\lambda_{\max} = 10$ respectively, the minimum and maximum service rate values $\mu_{\min} = 1$, $\mu_{\max} = 10$ respectively, and the number of samples $N = 10^5$. Finally the minimum and maximum capacity values $k_{\min} = k_{\max} = 10$, and the label $y^{(i)}$ is the same as previous defined in the algorithm.

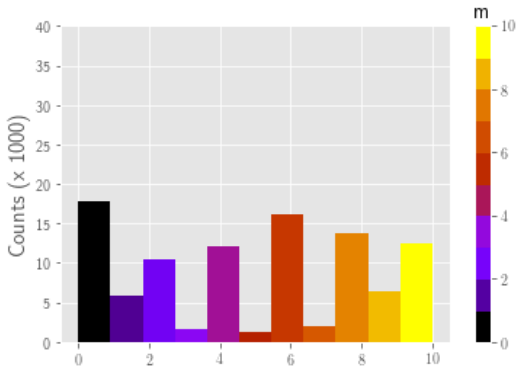
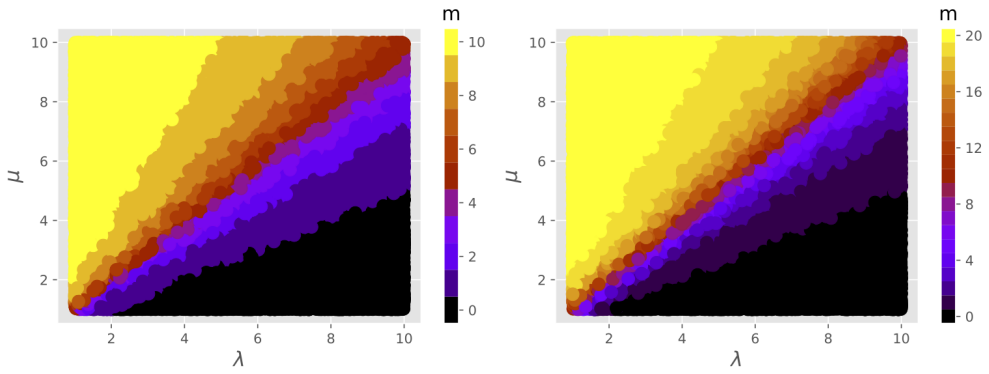



Figure 2.30: The distribution of the target variable for a dataset from non-constant rate of $M_{\lambda(t)}/M_{\mu(t)}/1/k$ queue with the capacity $k = 10$. The variable $m$ represents the class label.

Figures 2.29 show the distribution of the target allocation variable $m$ for the 2 datasets for constant rate $M_{\lambda}/M_{\mu}/1/k$ queue. The boundary points seem to be the most frequent occurrence. To understand why we look closely at the values of the traffic rates for those allocations as shown in Figure 2.31. It turns out that when the renting rate and returning rate are significantly different, the user dissatisfaction performance metric recommends allocating at one of the boundary points. In other words, there is a clear separation of the data when $\lambda$ and $\mu$ are far apart. Similarly, Figure 2.30 shows the distribution of the target allocation variable $m$ variable for one example non-constant rate $M_{\lambda(t)}/M_{\mu(t)}/1/k$ queue. And Figure 2.32 shows the distribution of features colored by the target allocation variable $m$.




(a) $k = 10$ (b) $k = 20$

Figure 2.31: The distribution of features colored by the target allocation variable $m$ for the 2 datasets from constant rate of $M_\lambda/M_\mu/1/k$ queue.

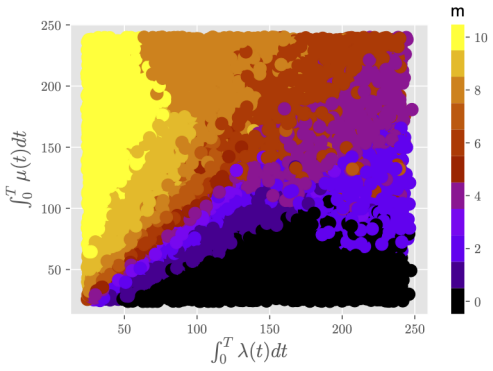



Figure 2.32: The distribution of features (total rent and return rates) colored by the target allocation variable $m$ for the dataset from non-constant rate of $M_{\lambda(t)}/M_{\mu(t)}/1/k$ queue with the capacity $k = 10$.

We apply a feed-forward neural network to learn a mapping between the queue parameters (the rent and the return rates of bikes $\mu$, $\lambda$, and the station dock capacity $k$) and the optimal allocation for that queue. We use 90% of the data as the training set and 10% of the data as the test set. Neural networks have two main hyperparameters that control the architecture of the network: the number of layers and the

number of nodes in each hidden layer. The tables below summarize the performance of the different specifications of the hyperparameter of the neural network.

| 1 Hid. Layer | Train S. | Test S. |
|---|---|---|
| 2 nodes | .198 | .198 |
| 5 nodes | .971 | .973 |
| 10 nodes | .982 | .981 |

| 2 Hid. Layers | Train S. | Test S. |
|---|---|---|
| 2 nodes | .850 | .849 |
| 5 nodes | .974 | .972 |
| 10 nodes | .985 | .983 |

Table 2.4: The performance of the neural network on the train and test dataset from constant rate of $M_\lambda/M_\mu/1/k$ queue with $k = 10$. The score is calculated based on the percentage of the correct prediction out of all predictions.

| 1 Hid. Layer | Train S. | Test S. |
|---|---|---|
| 2 nodes | .726 | .725 |
| 5 nodes | .828 | .825 |
| 10 nodes | .936 | .934 |

| 2 Hid. Layers | Train S. | Test S. |
|---|---|---|
| 2 nodes | .688 | .686 |
| 5 nodes | .932 | .933 |
| 10 nodes | .962 | .961 |

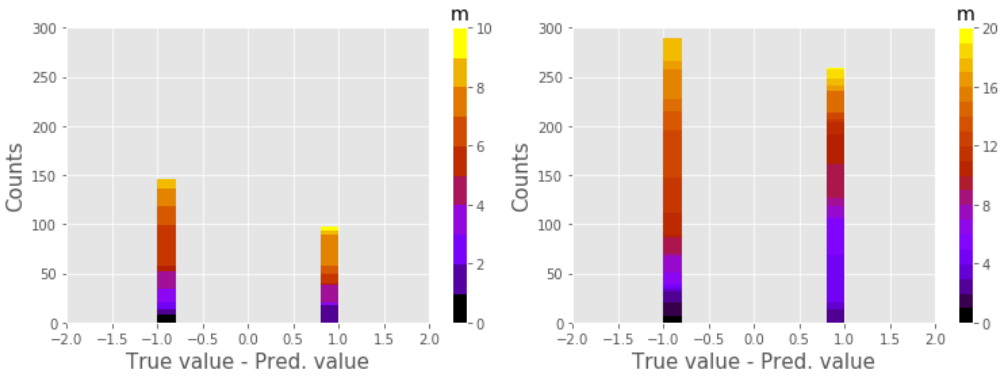
Table 2.5: The performance of the neural network on the train and test dataset from constant rate of $M_\lambda/M_\mu/1/k$ queue with $k = 20$. The score is calculated based on the percentage of the correct prediction out of all predictions.

| 1 Hid. Layer | Train S. | Test S. |
|---|---|---|
| 2 nodes | .701 | .698 |
| 5 nodes | .835 | .829 |
| 10 nodes | .887 | .884 |

| 2 Hid. Layers | Train S. | Test S. |
|---|---|---|
| 2 nodes | .764 | .761 |
| 5 nodes | .890 | .8858 |
| 10 nodes | .928 | .926 |

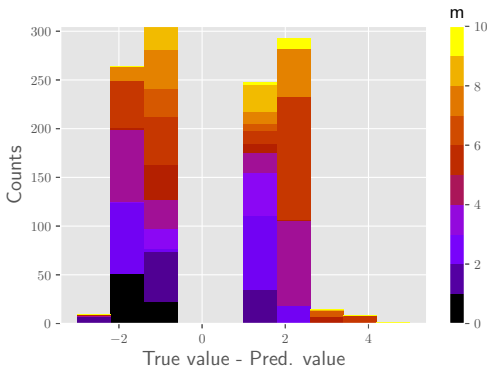
Table 2.6: The performance of the neural network on the train and test dataset from non-constant rate of $M_{\lambda(t)}/M_{\mu(t)}/1/k$ queue with $k = 10$. The score is calculated based on the percentage of the correct prediction out of all predictions.

Tables 2.4,2.5, and 2.6 show the performance, the mean accuracy, of the neural network models on different datasets. This metric of accuracy gives insight into the hyperparameter tuning of the network architecture.




(a) $k = 10$ (b) $k = 20$

Figure 2.33: The difference between neural network prediction and the actual value for the 2 datasets from constant rate of $M_\lambda/M_\mu/1/k$ queue. The neural network architectures are the same with 2 hidden layers and 5 hidden units in each layer.




(a) $k = 10$

Figure 2.34: The difference between neural network prediction and the actual value for a dataset from non-constant rate of $M_{\lambda(t)}/M_{\mu(t)}/1/k$ queue. The neural network architecture has 2 hidden layers and 5 hidden units in each layer.

Figure 2.33 and 2.34 show the distribution of the difference between the neural network predictions and the actual values for 2 datasets with constant rates and

a dataset with non-constant rates. As we can see, for the constant rate the absolute value of the misclassification difference is mostly less than 1. The non-constant example seems to have a high misclassification rate and the absolute value of the misclassification difference is mostly less than 4. This is not so surprising, since the non-constant rate example is expected to have more variability than the constate rate case.

## 2.5 Conclusion and Future Work

We proposed a new stochastic analysis of queues and presents the complete transient distribution analysis for the bike-sharing station queueing models. Our approach bypassed the sample path argument, traditionally used to obtain the transient probabilities of the model, and reduce this analysis to simply computing a real integral. Based on our queueing analysis, we developed a new algorithm for the optimal allocation of bikes in a bike-sharing system. Our algorithm takes in the rental rate, the return rate, and the number of docks as its input and outputs the optimal allocation given by our objective function. To demonstrate the practicality of our approach, computational results using synthetic datasets were included.

### Future Direction

The results of this work may lead to a different direction. In what follows, we summarize some in terms of application and theory.

1. An immediate goal is to identify applications with suitable structures to which our methodology applies. Our model and analysis are not just limited to bike-sharing systems, it could be generally used to analyze other on-demand product rental network. One example is the on-demand car-share service, where the parking lot represents dock station and rental cars replace rental bicycles.

2. As bike-sharing continues to be adapted by many cities in conjunction with technological advancements, the future of bike-sharing programs are likely to be moving towards dockless systems. Specifically, with the increased affordability of the Global Positioning Systems (GPS) devices, it's never been easier to track on-demand inventories. Research shows that GPS may reduce the need for physical docks Parkes et al. (2013). This will also serve other benefits to both the user and the operator: For the user, it will provide real-time information on bike availabilities which may save a user a trip; For the operator, GPS could assist in redistributing bikes across the fleet. Our analysis of the self-serving dock bike-sharing system remains useful even with the potential future adoption of the dockless bike-sharing systems.

3. Some of the theoretical analysis of the model dynamics presented in this work assumed the queueing system is markovian. Queueing system is markovian when the inter-arrival and service distribution is assumed to be exponentially distributed. Poisson arrival and service distribution were chosen for the model analysis, because of it's wide applicability and it's inherent simplicity. Although the Poisson assumption restricts the problem to the markovian case. Future work would explore general arrival and service distribution and non-markovian systems as an extension of this work. Moreover, future work will include exploring different representations of the user dissatisfaction function and consider the dynamic bike repositioning problem which is more complex than the static repositioning problem.

# Part III

# LEARNING FRAMEWORK FOR SEQUENTIAL DECISION PROBLEMS

# Chapter 3

# Supervised Learning-based Decision Making

This chapter studies Supervised Learning (SL) method for learning a near-optimal policy for sequential decision-making problems. Specifically, we used deep learning to learn the parameterized action space of a Multiplayer Online Mobile Arena (MOBA) game using a relatively small dataset of expert human demonstrations. Our approach uses a hybrid loss function defined as the weighted sum of regression loss and classification loss. This loss is then used to train the multioutput policy neural network. To demonstrate the effectiveness of our approach, we tested our supervised learning method on the popular MOBA game `King of Glory` (a North American version of the same game is titled `Arena of Valor`), where we build a competitive AI agent for the 1v1 mode of the game. One main emphasis is on the uniqueness of the application. Our SL-based implementation is one of the first attempts to design an SL-based AI for the 1v1 version of this game.

The material in this chapter is joint work with Xiangru Lian, Carson Eisenach, Daniel Jiang and Han Liu.

## 3.1 Introduction

Artificial intelligence (AI) systems that could learn to perform a certain challenging task from human demonstrations would revolutionize many industries. Significant progress towards artificial intelligence has been made using supervised learning systems that are trained to replicate the decisions of human experts (He et al., 2015; Krizhevsky et al.). Progress has also been made towards artificial intelligence, in gameplay, using reinforcement learning systems that are trained to learn their own experience over time (Mnih et al., 2015; Silver et al., 2016b). Both approaches typically require a huge amount of reliable data to achieve reasonable performance and work well when the state space and action space are discrete. In this work, we consider the problem of building a competitive AI agent to master the complex actions of a Multiplayer Online Battle Arena(MOBA) game. Action space in MOBA games usually contains both discrete and continuous components making it challenging to use an off-the-shelf algorithm to successfully learn. Also, the presence of continuous state space in the MOBA games further makes them challenging learning tasks. Our work falls under the general framework of imitation learning. Imitation learning techniques aim to mimic human behavior in a given task. An agent or a learning machine is trained to perform a task from demonstrations of good behavior by learning a mapping between observations and actions Hussein et al. (2017). Imitation learning is widely applicable in many domains and has been adapted for computer games applications (Thurau et al., 2004; Ross and Bagnell, 2010; Gorman, 2009). The primary contributions of this paper are summarized below:

1. We demonstrate a supervised learning method of learning the parameterized action space of a MOBA game using a relatively small dataset of expert human demonstrations. Our approach uses a hybrid loss function defined as the

weighted sum of regression loss and classification loss. This loss is then used to train the multioutput policy neural network.

2. The SL-based approach is tested on the popular MOBA game `King of Glory` (a North American version of the same game is titled `Arena of Valor`), where we build a competitive AI agent for the 1v1 mode of the game. Our SL-based implementation is one of the first attempts to design an SL-based AI for the 1v1 version of this game.

### 3.1.1 Related Work

The idea of teaching, a learning agent, by imitation has been around for many years, however, the field is gaining attention recently due to advances in computing and rising interest in the intelligent applications in robotics and gameplay domains Hussein et al. (2017). The process of learning by imitation is also gaining popularity because it promotes teaching complex tasks, to a learning agent, with minimal expert knowledge of the tasks.

Successfully learning policies for MOBA games is a challenging problem. Significant progress has been made for games with small action spaces– such as Go and Atari 2600 games Silver et al. (2016b); Mnih et al. (2015), using a combination of imitation and reinforcement learning. However, the action spaces in MOBA games typically consist of both discrete and continuous components – specifically, the discrete actions have continuous parameters. Hausknecht and Stone (2016) investigate the Deep reinforcement learning approach for the parametrized continuous action spaces in the RoboCup Soccer gameplay domain. The reinforcement learning approach requires reward functions that are designed specifically for each task. For instance, the number of a possible sequence of action grows exponentially even simple tasks and defining rewards for such problems is difficult.

Ross et al. (2011) propose an iterative algorithm, which trains a stationary deterministic policy, that can be seen as a no-regret algorithm in an online learning setting that learns the expert policy using traditional machine learning. This work is similar to the DAgger Algorithm proposed by Ross et al. (2011); our work is different in that we only use a dataset of expert human demonstrations without collecting and aggregating additional data during training. Instead, we used a resampling technique to obtain a better policy fit from the expert policy. The resampling technique clusters observation in the training data based on the labels and weight state, action pairs inversely to the size of the cluster to which the state belongs.

Imitation learning is a popular technique that aims to mimic expert behavior for a given task Hussein et al. (2017). Imitation learning techniques have proven very useful in practice and have led to a state-of-the-art performance in many application domains (Ross et al., 2011; Schaal, 1999). A typical approach to imitation learning is to train a classifier or a regressor to predict an experts behavior given training data of the encountered observations and actions performed by the expert. In our work, we combined both the classifier and regressor due to the nature of the input; needing to predict both continuous and discrete values. This is done by training a multi-output neural network architecture that combines related regression and classification tasks that rely on the same input data. Our work demonstrates a supervised learning approach for learning policies for MOBA games with Parametrized Action Spaces from relatively small numbers of human demonstrations.

## 3.2 Methodology

In this section, we define our custom loss function for learning the parameterized action space of a MOBA game using a relatively small dataset of expert human

demonstrations. Moreover, we also discuss the optimization process for this loss function using a variant of stochastic gradient descent (SGD).

### 3.2.1 Hybrid Loss Function

We now define the hybrid loss function $\mathcal{L}(\boldsymbol{y}, \widehat{\boldsymbol{y}})$ as follows:

$$\begin{aligned}\mathcal{L}(\boldsymbol{y}, \widehat{\boldsymbol{y}}) &= \gamma_1 \left[ \frac{||\boldsymbol{y}_c||_2 ||\widehat{\boldsymbol{y}}_c||_2 - \boldsymbol{y}_c^\top \widehat{\boldsymbol{y}}_c}{||\boldsymbol{y}_c||_2 \cdot ||\widehat{\boldsymbol{y}}_c||_2} \right] - \gamma_2 \sum_i y_d^{(i)} \log\left(\widehat{y}_d^{(i)}\right) \\ &= \gamma_1 \left[ 1 - \frac{\sum_i y_c^{(i)} \cdot \widehat{y}_c^{(i)}}{\sqrt{\sum_i \left(y_c^{(i)}\right)^2} \cdot \sqrt{\sum_i \left(\widehat{y}_c^{(i)}\right)^2}} \right] - \gamma_2 \sum_i y_d^{(i)} \log\left(\widehat{y}_d^{(i)}\right), \end{aligned} \tag{3.2.1}$$

where

$$\boldsymbol{y} = \begin{bmatrix} \boldsymbol{y}_d \\ \boldsymbol{y}_c \end{bmatrix} \in \mathbb{R}^n \text{ and } \widehat{\boldsymbol{y}} = \begin{bmatrix} \widehat{\boldsymbol{y}}_d \\ \widehat{\boldsymbol{y}}_c \end{bmatrix} \in \mathbb{R}^n.$$

So for the continuous component, we used cosine proximity loss to compute the cosine proximity between predicted value and actual value. While for the discrete component, we used negative logarithmic Likelihood loss to measures the accuracy of the classifier, since our model outputs a probability for each discrete class. Equation 3.2.1 is essentially the weighted sum of the regression loss and classification loss for some predefined weights $\gamma_1$ and $\gamma_2$. This overall loss function allows for easy computation of gradients. Loss function is an important part of neural networks; It is used to measure the inconsistency between predicted value $\widehat{y}$ and actual label $y$. Given the training sample set $\left\{(x^{(i)}, y^{(i)})\right\}_{i=1}^m \sim \mathcal{D}$ then the risk of the network is

given by:

$$\begin{aligned}\mathcal{L}_{\mathcal{D}}(\boldsymbol{W}) &= \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\Big[L\big(h_{\boldsymbol{W}}(\boldsymbol{x}),\boldsymbol{y}\big)\Big] \\ &\approx \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}\Big(h_{\boldsymbol{W}}\left(x^{(i)}\right),y^{(i)}\Big) \\ &= \widehat{\mathcal{L}_{\mathcal{D}}}(\boldsymbol{W}). \end{aligned} \tag{3.2.2}$$

The cost function of our model consists of two terms: the empirical risk term and the regularization term.

$$J(\boldsymbol{W}) = \widehat{\mathcal{L}_{\mathcal{D}}}(\boldsymbol{W}) + \lambda\Phi(\boldsymbol{W}), \tag{3.2.3}$$

where $\Phi(\cdot) = \frac{\lambda||\cdot||^2}{2}$ is the regularization term and $\widehat{\mathcal{L}_{\mathcal{D}}}(\boldsymbol{W})$ represent the empirical risk function. The goal is to find $\boldsymbol{W}^\star$ that minimizes this cost function,

$$\begin{aligned}\boldsymbol{W}^\star &= \underset{\boldsymbol{W}\in\boldsymbol{\Pi}}{\operatorname{argmin}}\, J(\boldsymbol{W}) \\ &= \underset{\boldsymbol{W}\in\boldsymbol{\Pi}}{\operatorname{argmin}}\left[\widehat{\mathcal{L}_{\mathcal{D}}}(\boldsymbol{W}) + \lambda\Phi(\boldsymbol{W})\right] \\ &= \underset{\boldsymbol{W}\in\boldsymbol{\Pi}}{\operatorname{argmin}}\left[\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}\left(y^{(i)},\widehat{y}^{(i)}\right) + \lambda\Phi(\boldsymbol{W})\right] \\ &= \underset{\boldsymbol{W}\in\boldsymbol{\Pi}}{\operatorname{argmin}}\left[\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}\Big(y^{(i)},h_{\boldsymbol{W}}\left(x^{(i)}\right)\Big) + \lambda\Phi(\boldsymbol{W})\right]. \end{aligned} \tag{3.2.4}$$

This cost function is highly non-convex because the hypothesis function is nonlinear. Nevertheless, we can still implement the SGD algorithm and hope it will find a reasonable near-optimal solution Shalev-Shwartz and Ben-David (2014). Next, we briefly discuss how we optimize the loss function using a variant of SGD.

### 3.2.2 Training the Neural Network

**Loss Optimization:** We want to find the network weights that achieve the lowest loss. Denote the loss of our network by:

$$\mathcal{L}\Big(h(x^{(i)}; \boldsymbol{W}), y^{(i)}\Big) \text{ where } \boldsymbol{W} = \Big\{\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}, \dots\Big\}. \tag{3.2.5}$$

The loss measures the cost incurred from incorrect predictions. The empirical Loss, shown in Equation 3.2.6, measures the total loss over our entire training dataset.

$$J(\boldsymbol{W}) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}\Big(h\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\Big). \tag{3.2.6}$$

Usually, we want to find the network weights that achieve the lowest loss as shown bellow in Equation 3.2.7.

$$\begin{aligned}\boldsymbol{W}^{\star} &\in \underset{\boldsymbol{W}}{\operatorname{argmin}}\, J(\boldsymbol{W}) \\ &= \underset{\boldsymbol{W}}{\operatorname{argmin}}\, \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}\Big(h\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\Big).\end{aligned} \tag{3.2.7}$$

The optimization through gradient descent updates as follows:

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}. \tag{3.2.8}$$

In practice computing the gradient for all points $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$ on every iteration is very expensive. On the other extreme, we could compute the gradient for a single point $\frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}$, on every iteration but it will be very noisy. So in practice, we pick a middle ground and use a mini-batches $B$ and compute the gradient for that batch as follows:

$$\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B}\sum_{k=1}^{B}\frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}} \tag{3.2.9}$$

This gives us a much more accurate estimation of the gradient than the vanilla stochastic gradient descent. This approach also allows us to parallelize the training process.

## 3.3 Experiments and Results

### 3.3.1 Case Study: King of Glory MOBA SL-Based Agent

We implemented the *SL-Based Agent* within a new and challenging environment, the recently popular MOBA game *King of Glory* by Tencent (the game is also known as *Honor of Kings* and a North American release of the game is titled *Arena of Valor*). Each character (or "hero") in *King of Glory* contains a tailor-designed AI agent, which we will refer to as the *internal AI*. Our implementation of the algorithm is one of the first attempts to design an AI agent for the one-versus-one version of this game. The initial goal here is to produce an agent that outperforms the internal rule-based AIs. Subsequently, in the next chapter, The developed *SL-Based Agent* will be tested against other competitive state-of-the-art agents. The *SL-Based Agent* is constructed via *supervised learning* on a small dataset of approximately 100,000 state/action pairs of expert human gameplay data.

**Game Description.** In the *King of Glory*, players are divided into two opposing teams and each team has a base located on the opposite corners of the game map (similar to other MOBA games, like *League of Legends* or *Dota 2*). The bases are guarded by towers, which can attack the enemies when they are within a certain attack range. The goal of each team is to overcome the towers and eventually destroy the opposing team's "crystal," located at the enemy base. In the standard version of the game, each team consists of five players (5v5), but smaller versions, 3v3 and 1v1, are also available. For this paper, we only consider 1v1 mode, where each player controls a primary "hero" alongside less powerful game-controlled characters called "minions." These units guard the path to the crystal and will automatically fire

(weak) attacks at enemies within range. Figure 3.1 shows the annotated version of the game map with the two heroes and their minions; the upper-left corner shows the map, with the blue and red markers pinpointing the towers and crystals.

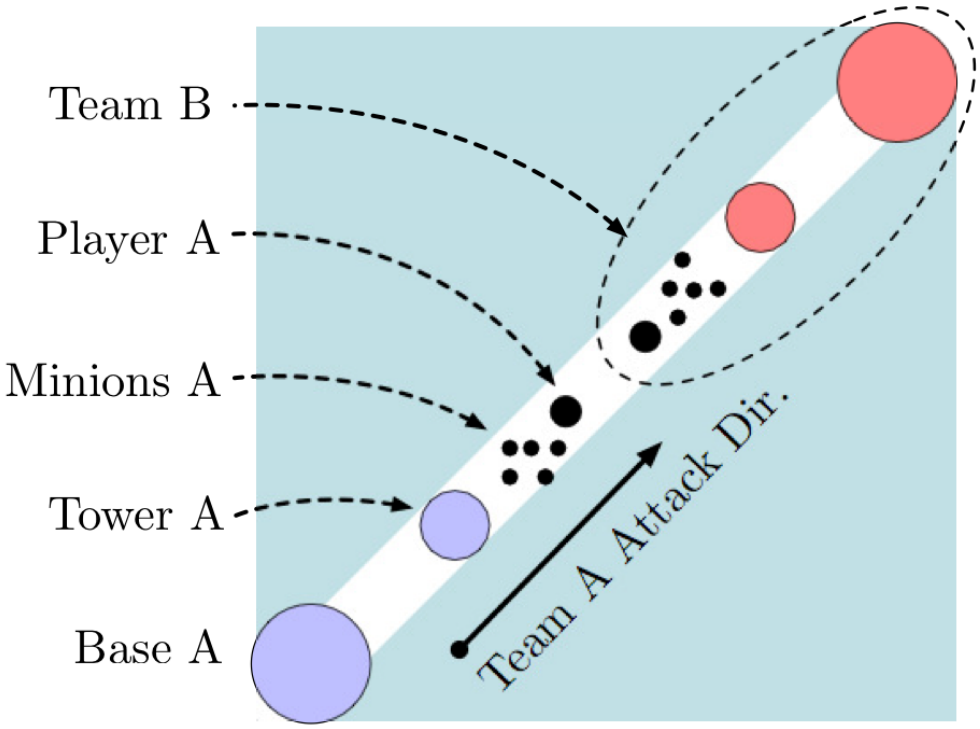



Figure 3.1: Annotated battle field for Moba 1v1 game

**Resampling Weights.** During training of the policy network, we first cluster all the observations in the training dataset based on the action types. And weight state, action pairs inversely to the size of the cluster to which the state belongs. This is helpful to help resolve the inherent class imbalance in the training dataset. In MOBA games, some state, action pair occur frequently. For example, a common action is forward movement and attack; whereas actions that involve using special skills or replenishing health are less common. As a result, sampling the dataset uniformly leads to model underfitting of the data and hence yield poor performance.

### Features of the State Space and the Action Space

**State Space.** In our experiment, the state of the game is represented by a 41-dimensional feature vector, which was constructed using the output from the game engine and API, as shown in Table 3.1. The features consist of basic attributes of the two heroes, the computer-controlled units and the game structures. The feature list also has information on the relative positions of the opposing team's units and structures with respect to the hero controlled by our algorithm.

Table 3.1: The state space features for the supervised learning agent

| No. | Feature | Dim. |
|---|---|---|
| 1 | Location of Hero 1 | 2 |
| 2 | Location of Hero 2 | 2 |
| 3 | HP of Hero 1 | 1 |
| 4 | HP of Hero 2 | 1 |
| 5 | Hero 1 skill cooldowns | 5 |
| 6 | Hero 2 skill cooldowns | 5 |
| 7 | Direction to enemy hero | 3 |
| 8 | Direction to enemy tower | 4 |
| 9 | Direction to enemy minion | 3 |
| 10 | Enemy tower HP | 1 |
| 11 | Enemy minion HP | 1 |
| 12 | Direction to the spring | 3 |
| 13 | Total HP of allied minions | 1 |
| 14 | Enemy's tower attacking Hero 1 | 3 |
| 15 | Hero 1 in range of enemy towers | 3 |
| 16 | Hero 2 in range of enemy towers | 3 |

**Action Space.** The action space consists of three kinds of actions: "normal" actions (moving and attacking), "learn" actions (learning new skills), "purchase" actions (purchasing new equipments). Let $\mathcal{A} = \mathcal{A}_{\text{act}} \cup \mathcal{A}_{\text{buy}} \cup \mathcal{A}_{\text{learn}}$ denote the action space. The action space $\mathcal{A}$ of KOG consists of both discrete and continuous actions. Specifically, an action $a \in \mathcal{A}$ can be written as $a = (k, x_k)$, where $k \in \{1, 2, \ldots, K\}$ (discrete action types) and $x_k \in \mathcal{X}_k$ (continuous parameter space). Formally the parametrized action space is given below:

$$\mathcal{A} = \left\{ (k, x_k) \,\middle|\, x_k \in \mathcal{X}_k, \quad \forall k \in \{1, 2, \ldots, K\} \right\} \tag{3.3.1}$$

Practically, an action $a = (k, x_k) \in \mathcal{A}$ is chosen in two steps: first choose $k$ from the discrete action set $\{1, 2, \ldots, K\}$. Then choose $x_k$ from the continuous action space $\mathcal{X}_k$. For simplicity, we assume the action space is the same for all non-terminal states. In our experiments, we set $K = 7$ for the discrete action types summarized in Table 3.2. We also restrict the continuous action space to $\mathcal{X}_k = \mathbb{R}^2$. Moreover, the continuous parameters are discretized to avoid exhaustive search. Finally, we note that some of the continuous parameter could be deficient.

Table 3.2: The discrete action types

| k | ActionType | Meaning |
|---|---|---|
| 1 | NormalAttack | Attack the current target |
| 2 | MoveDir($x_2$) | Move in the direction $x_2$ |
| 3 | Skill$_1$($x_3$) | Use Skill 1 at target position $x_3$ |
| 4 | Skill$_2$($x_4$) | Use Skill 2 at target position $x_4$ |
| 5 | Skill$_3$($x_5$) | Use Skill 3 at target position $x_5$ |
| 6 | Recover | Replenish health at the base |
| 7 | NoAction | Perform no action |

### Implementation details and setup

**Experimental Setup.** We now give a summary of our experimental setup and describe our practical implementation of the algorithm. Our agent plays the hero *DiRenJie*, a *marksman* type hero (i.e., he shoots a projectile), out of approximately 75 available as the opponent. When testing against other internal AIs, we choose heroes other than *DiRenJie* that we did not use for training.

1. The "SL" agent denoting supervised agent was trained with resampled weights. This agent was trained on the a dataset of 100,000 state-action pairs from human experts games

2. The second agent is labeled "NRW" for no resampled weight, which uses the same parameters as the SL agent except that it doesn't use resampled weights. This agent was trained on the same dataset of 100,000 state-action pairs from human expert games.

In fact, the policy function is consistent across all agents. The policy function approximation uses fully-connected neural networks with five and two hidden layers and SELU (scaled exponential linear unit) activation (Klambauer et al., 2017). The learning rate $r = 0.003$ was chosen carefully due to non-convexity and was tuned via trial and error. A major challenge in implementing SL agent for *King of Glory* game is that the action space is parametrized by both continuous and discrete parameters. We employed a hybrid loss function defined as a linear combination of cosine proximity loss and the negative log-likelihood loss. For the continuous component of the action space, we used cosine proximity between predicted value and actual value. While for the discrete component of the action space, we used the negative logarithmic likelihood loss to measure the accuracy of the classifier.

**Expert Data:** Given approximately 200 games of expert human games, we first extract relevant important features for the hero *DiRenJie*, a marksman type hero.

Let $\mathcal{S} = \left\{(x^{(i)}, y^{(i)})\right\}_{i=1}^{m}$ denote the extracted data, where $m \approx 100,000$. From each frame, we extract the state feature vector described in Table 3.1, and the corresponding action(s) taken by the expert player. For better performance, we normalized and scaled the features to have the same scale across all features. In practice, feed-forward neural networks work best if the inputs are centered. We train a supervised learning (SL) policy network $p_{\boldsymbol{W}}$ directly from $\mathcal{S} = \left\{(x^{(i)}, y^{(i)})\right\}_{i=1}^{m}$, where $\boldsymbol{W}$ denotes the vector of policy parameters. Here a policy is represented by a neural network whose input is a representation of the state, whose output is action selection probabilities over all legal actions, and whose weights are the policy parameters. The policy network is trained on randomly sampled state-action pairs using stochastic gradient descent (SGD) to maximize the likelihood of selecting a particular action in a particular state. For better performance, we initialize the initial policy to a randomly chosen vector whose entries are very close to zero. This works well and can potentially lead to a good local minimum as each run of SGD has a unique initialization. This initialization approach ensures that the weights of the hidden layers are not similar.

### Results and Discussion

**Results against Internal AI.** To test our agent against internal hand-crafted AIs, we ask the question: "how much faster can we beat an opponent compared to the internal AI that controls our hero?" We first played the internal DiRenJie AI against other internal AIs and selected six heroes of the *marksman* type that internal DiRenJie is able to defeat (unlike in 5v5, where each hero plays a specific role on the team, the 1v1 game should be played with heroes that are of a similar type to avoid mismatches). As shown in Figure 3.2, our SL agents are able to win the game significantly faster than the hand-crafted AIs. In order to collect additional data beyond a single game (as the game is nearly deterministic), we subsample 80% of the dataset and trained 14 additional SL agents, for a total of 15 comparisons. Similarly, we subsample 80% of

the dataset and trained 14 additional NRW agents, for a total of 15 comparisons. This variation is captured by the error bars in both the SL and NRW agents as depicted in Figure 3.2.

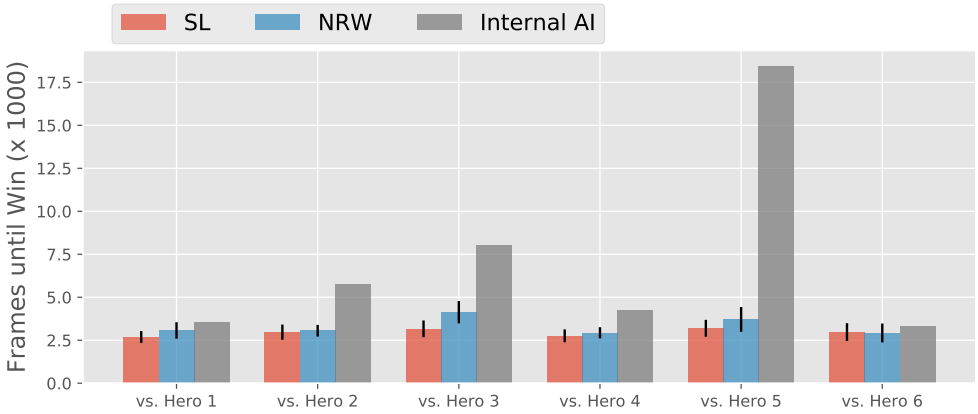
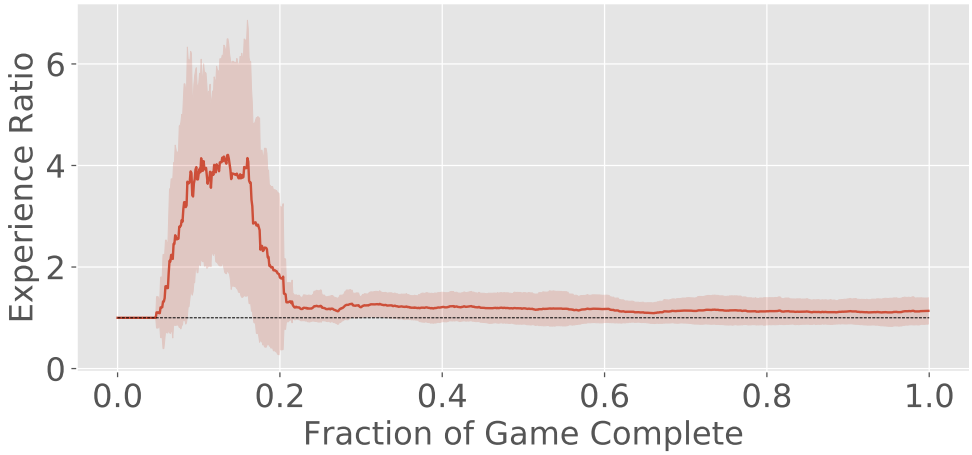
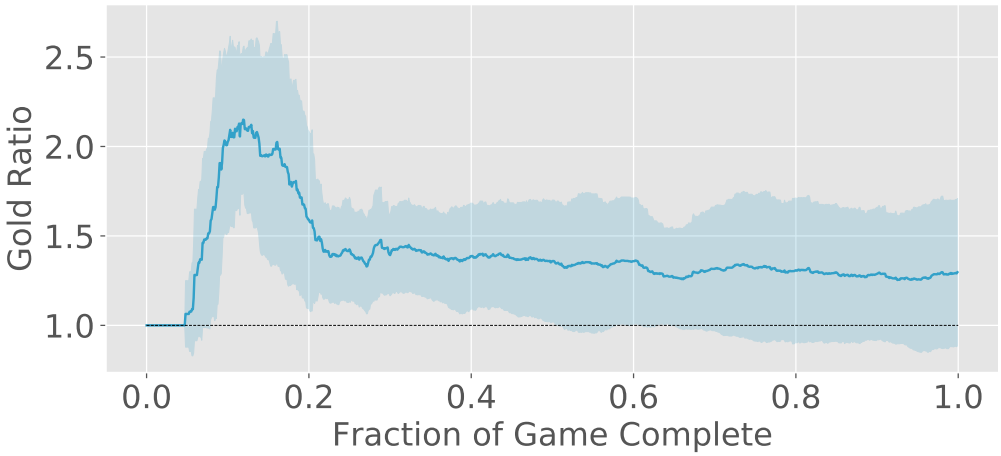



Figure 3.2: Number of Frames to Defeat Marksman Heroes

**Results for SL versus NRW.** The SL agent Defeats the NRW trained on the same dataset of 100,000 state-action pairs from human expert games. We give an aggregate view of these games by computing the ratio of SL agents' experience points to the NRW agents' experience and the ratio of our gold to the NRW agents' gold (both experience and gold are collected throughout as a hero deals damage and defeats enemies and are positively correlated with victories). Since the lengths of the 15 games vary, we compare the ratios to the fraction of the game completed; It is interesting to note that the SL agent tends to end the game with approximately 1.5x the amount of experience and gold as the NRW agents. The initial aggressiveness of the agent (first 20% of the game) may partly explain its success against the NWR agents.




(a) Experience




(b) Gold

Figure 3.3: Plot of In-game behavior between SL Agent versus NRW Agents. In (a), we show the In-game experience. In (b), we plot the In-game gold (reward) collected.

Figure 3.4, shows both the training and validation loss curves. The loss curve is often useful when debugging a neural network during training. It gives us a snapshot of the training process and the direction in which the network learns. We can see that the training loss seems to be slowing decreasing over time given the low learning rate ($r = 0.003$). The validation loss seems to be decreasing until 50 epochs and

increasing thereafter. This tells us to stop training around 50 epochs to avoid overfitting. Similarly, Figure 3.5, presents both training and validation accuracy curves. The accuracy curve is useful to understand the progress of neural networks. We can see both accuracy curves are increasing until around 50 epochs.


Figure 3.4: The Training and Validation Loss

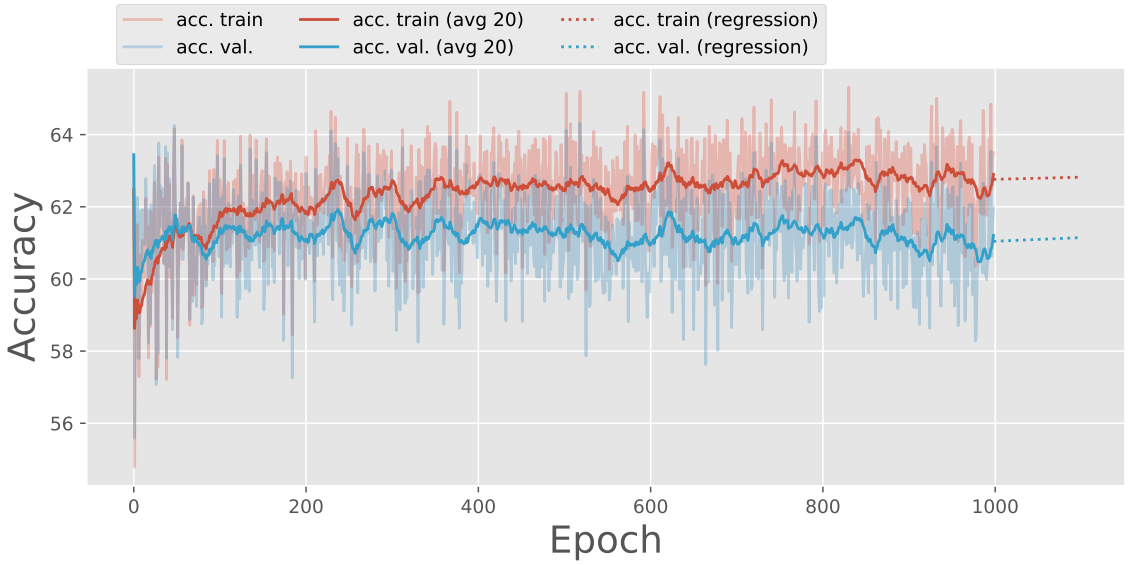



Figure 3.5: The Training and Validation Accuracy

## 3.4 Conclusion and Future Work

We have demonstrated a supervised learning method of learning the parameterized action space of a MOBA game using a relatively small dataset of expert human demonstrations. Our approach used a hybrid loss function defined as the weighted sum of regression loss and classification loss. Which was then used to train the multioutput policy neural network. Moreover, the SL-based approach is tested on the popular MOBA game `King of Glory` (a North American version of the same game is titled `Arena of Valor`), where we build a competitive AI agent for the 1v1 mode of the game. Our SL-based implementation is one of the first attempts to design an SL-based AI for the 1v1 version of this game. In the next chapter, we further compare the developed *SL-Based Agent* against other competitive state-of-the-art agents.

### Future Direction

The results of this work may lead to a different direction. In what follows, we summarize some in terms of application and theory. Our primary methodological avenue for future work, in the supervised learning setting, is to incorporate an unsupervised policy improvement. Although the supervised approach works well, it is limited to the quality of the available dataset. Future direction involves a two-step policy learning: the first step remains the same as the proposed supervised approach. While the second step takes the first step as initialization and further improves the policy with reinforcement learning.

We outlined the improvement step to the supervised approach. Currently, we have supervised policy $\pi_{SL}$ which was built by analyzing the frequency of moves from a set of human gameplay data $\mathcal{D}$. We would like to use reinforcement learning (RL) to further improve the policy $\pi_{SL}$. The hope is that the resultant policy $\pi_{RL}$ would outperform the old one. In other words, we use RL to improve the policy $\pi_{RL}$ until

the resulting policy is significantly better than the supervised policy $\pi_{SL}$. The two steps are summarized as follows:

**Step 1 (Supervised).** Let $S = \left\{(x^{(i)}, y^{(i)})\right\}_{i=1}^{n} \sim \mathcal{D}$ be a sample training data from the expert gameplay data. We would solve

$$\begin{aligned}
\boldsymbol{W}^{\star} &= \underset{\boldsymbol{W}\in\boldsymbol{\Pi}}{\operatorname{argmin}}\, J(\boldsymbol{W}) \\
&= \underset{\boldsymbol{W}\in\boldsymbol{\Pi}}{\operatorname{argmin}} \left[\widehat{\mathcal{L}_{\mathcal{D}}}(\boldsymbol{W}) + \lambda\Phi(\boldsymbol{W})\right] \\
&= \underset{\boldsymbol{W}\in\boldsymbol{\Pi}}{\operatorname{argmin}} \left[\frac{1}{n}\sum_{i=1}^{n} L(y^{(i)}, \widehat{y}^{(i)}) + \lambda\Phi(\boldsymbol{W})\right] \\
&= \underset{\boldsymbol{W}\in\boldsymbol{\Pi}}{\operatorname{argmin}} \left[\frac{1}{n}\sum_{i=1}^{n} L(y^{(i)}, h_{\boldsymbol{W}}(x^{(i)})) + \lambda\Phi(\boldsymbol{W})\right] \qquad (3.4.1)
\end{aligned}$$

using the supervised learning approach outlined in the previous section to get $\pi_{SL} \equiv \pi_{\boldsymbol{W}^{\star}}$.

**Step 2 (Reinforcement).** We use direct policy search to further tune $\boldsymbol{W}$ such that the policy $\pi_{\boldsymbol{W}}$ produces the greatest reward. Here we initialize reinforcement learning algorithm by the policy produced in Step 1. The next chapter discusses how to design such a reinforcement learning algorithm.

# Chapter 4

# Reinforcement Learning-based Decision Making

This chapter studies approximation methods for solving sequential decision problems. Inspired by recent successes of the forward-planning technique, Monte-Carlo tree search (MCTS), in several artificial intelligence (AI) application domains, we propose a model-based reinforcement learning (RL) technique that iteratively applies MCTS on batches of small, finite-horizon versions of the original infinite-horizon Markov decision process. The terminal condition of the finite-horizon problems, or the *leaf-node evaluator* of the decision tree generated by MCTS, is specified using a combination of an estimated value function and an estimated policy function. The recommendations generated by the MCTS procedure are then provided as feedback to refine, through classification and regression, the leaf-node evaluator for the next iteration. We provide the first sample complexity bounds for a tree search-based RL algorithm. Furthermore, we show that a deep neural network implementation of the technique can create a competitive AI agent for the popular multi-player online battle arena (MOBA) game *King of Glory*.

The material in this chapter is joint work with Daniel Jiang and Han Liu. The work has been submitted to 2018 ICML conference Jiang et al. (2018).

## 4.1 Introduction

Monte Carlo Tree Search (MCTS) is a heuristic search algorithm for decision processes. MCTS, introduced in Coulom (2006) and surveyed in detail by Browne et al. (2012b), has received attention in recent years for its successes in gameplay artificial intelligence (AI), culminating in the Go-playing AI AlphaGo (Silver et al., 2016a). MCTS seeks to iteratively build the decision tree associated with a given Markov decision process (MDP) so that attention is focused on "important" areas of the state space, assuming a given initial state (or *root node* of the decision tree). The intuition behind MCTS is that if rough estimates of state or action values are given, then it is only necessary to expand the decision tree in the direction of states and actions with high estimated value. To accomplish this, MCTS utilizes the guidance of *leaf-node evaluators* (either a policy function Chaslot et al. (2006) rollout, a value function evaluation (Campbell et al., 2002; Enzenberger, 2004), or a mixture of both (Silver et al., 2016a)) to produce estimates of downstream values once the tree has reached a certain depth (Browne et al., 2012b). The information from the leaf-nodes are then *backpropagated* up the tree. The performance of MCTS depends heavily on the quality of the policy/value approximations (Gelly and Silver, 2007), and at the same time, the successes of MCTS in Go show that MCTS *improves upon a given policy* when the policy is used for leaf evaluation, and in fact, it can be viewed as a policy improvement operator (Silver et al., 2017). In this paper, we study a new feedback-based framework, wherein MCTS updates its own leaf-node evaluators using observations generated at the root node.

MCTS is typically viewed as an *online planner*, where a decision tree is built starting from the current state as the root node (Chaslot et al., 2006; 2008; Hingston and Masek, 2007; Maîtrepierre et al., 2008; Cazenave, 2009; Méhat and Cazenave, 2010; Gelly and Silver, 2011; Gelly et al., 2012; Silver et al., 2016a). The standard goal of MCTS is to recommend an action for the *root node only*. After the action is taken, the system moves forward and a new tree is created from the next state (statistics from the old tree may be partially saved or completely discarded). MCTS is thus a "local" procedure (in that it only returns an action for a given state) and is inherently different from value function approximation or policy function approximation approaches where a "global" policy (one that contains policy information about all states) is built. One concern with MCTS methods is that they are generally not well-suited to real-time decision-making applications: because MCTS builds "on-the-fly" local approximations, it is slower than playing a game using a policy which has been pre-trained. For games like Chess or Go, online planning using MCTS may be appropriate, but in games where fast decisions are necessary (e.g., Atari or MOBA games), tree search methods are too slow (Guo et al., 2014). The proposed algorithm is intended to be used in an *off-policy* fashion during the reinforcement learning (RL) *training phase*. Once the training is complete, the policies associated with leaf-node evaluation can be implemented to make fast, real-time decisions without any further need for tree search.

**Main Contributions.** These characteristics of MCTS motivate our proposed method, which attempts to leverage the *local* properties of MCTS into a training procedure to iteratively build *global* policy across all states. The idea is to apply MCTS on batches of small, finite-horizon versions of the original infinite-horizon Markov decision process (MDP). A rough summary is as follows: (1) initialize an arbitrary value function and a policy function; (2) start (possibly in parallel) a batch of MCTS instances, limited in search-depth, initialized from a set of sampled states,

while incorporating a combination of the value and policy function as leaf-node evaluators; (3) update both the value and policy functions using the latest MCTS root node observations; (4) Repeat starting from step (2). This method exploits the idea that an MCTS policy is better than either of the leaf-node evaluator policies alone (Silver et al., 2016a), yet improved leaf-node evaluators also improve the quality of MCTS (Gelly and Silver, 2007). The primary contributions of this paper are summarized below.

1. We propose a batch, MCTS-based RL method that operates on continuous state, finite action MDPs and exploits the idea that leaf-evaluators can be updated to produce a stronger tree search using *previous tree search results*. Function approximators are used to track policy and value function approximations, where the latter is used to reduce the length of the tree search rollout (oftentimes, the rollout of the policy becomes a computational bottle-neck in complex environments).

2. We provide a full sample complexity analysis of the method and show that with large enough sample sizes and sufficiently large tree search effort, the performance of the estimated policies can be made close to optimal, up to some unavoidable approximation error. To our knowledge, batch MCTS-based RL methods have not been theoretically analyzed.

3. The feedback-based tree search algorithm is tested on the popular MOBA game `King of Glory` (a North American version of the same game is titled `Arena of Valor`), where we build a competitive AI agent for the 1v1 mode of the game.

### 4.1.1 Related Work

The question of *whether tree search could be leveraged during training for situations when it is too slow for online planning* was first explored by Guo et al. (2014) in the context of Atari games, where MCTS was used to generate offline training data for a supervised learning (classification) procedure. The authors showed that by using the power of tree search offline, the resulting policy was able to outperform the deep Q-network approach of Mnih et al. (2013).

A natural next step is to repeatedly apply the procedure of Guo et al. (2014). In building AlphaGo Zero, Silver et al. (2017) extends the ideas of Guo et al. (2014) into an iterative procedure, where the neural network policy is updated after every episode and then reincorporated into tree search. The technique was able to produce a superhuman Go-playing AI (and improves upon the previous AlphaGo versions) without any human replay data. Our proposed algorithm is a *provably near-optimal* variant (and in some respects, generalization) of the AlphaGo Zero algorithm. The key differences are: (1) our theoretical results cover a continuous, rather than finite, state space setting, (2) the environment is a stochastic MDP rather than a sequential deterministic two player game, (3) we use batch updates, (4) the feedback of previous results to the leaf-evaluator manifests as both policy and value updates rather than just the value (as Silver et al. (2017) does not use policy rollouts).

Anthony et al. (2017) proposes a general framework called *expert iteration* that combines supervised learning with tree search-based planning. The methods described in (Guo et al., 2014; Silver et al., 2017), and the current paper can all be (at least loosely) expressed under the expert iteration framework. However, no theoretical insights were given in these previous works and our paper intends to fill this gap by providing a full theoretical analysis of an iterative, MCTS-based RL algorithm. Our analysis relies on the *concentrability coefficient* idea of Munos (2007) for approximate value iteration and builds upon the work on classification based policy iteration

(Lazaric et al., 2016), approximate modified policy iteration (Scherrer et al., 2015), and fitted value iteration (Munos and Szepesvári, 2008).

Sample complexity results for MCTS are relatively sparse. Teraoka et al. (2014) gives a high probability upper bound on the number of playouts needed to achieve $\epsilon$-accuracy at the root node for a stylized version of MCTS called `FindTopWinner`. More recently, Kaufmann and Koolen (2017) provided high probability bounds on the sample complexity of two other variants of MCTS called `UGapE-MCTS` and `LUCB-MCTS`. In this paper, we do not require any particular implementation of MCTS, but make a generic assumption on its accuracy that is inspired by these results.

## 4.2 Methodology

### 4.2.1 Problem Formulation

Consider a discounted, infinite-horizon MDP with a continuous state space $\mathcal{S}$ and finite action space $\mathcal{A}$. For all $(s, a) \in \mathcal{S} \times \mathcal{A}$, the *reward function* $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ satisfies $r(s, a) \in [0, R_{\max}]$.

The *transition kernel*, which describes transitions to the next state given current state $s$ and action $a$, is written $p(\cdot \,|\, s, a)$ — a probability measure over $\mathcal{S}$. Given a *discount factor* $\gamma \in [0, 1)$, the value function $V^\pi$ of a *policy* $\pi : \mathcal{S} \to \mathcal{A}$ starting in $s = s_0 \in \mathcal{S}$ is

$$V^\pi(s) = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t \, r(s_t, \pi_t(s_t))\right], \tag{4.2.1}$$

where $s_t$ is the state visited at time $t$. The *optimal value function* is obtained by maximizing over all policies: $V^*(s) = \sup_{\pi \in \Pi} V^\pi(s)$, where $\Pi$ is the set of all deterministic policies (i.e., mappings from state to action).

Both $V^\pi$ and $V^*$ are bounded by $V_{\max} = R_{\max}/(1 - \gamma)$. We let $\mathcal{F}$ be the set of bounded, real-valued functions mapping $\mathcal{S}$ to $[0, V_{\max}]$. We frequently make use of

the shorthand operator $T_\pi : \mathcal{F} \to \mathcal{F}$, where the quantity $(T_\pi V)(s)$ is be interpreted as the reward gained by taking an action according to $\pi$, receiving the reward $r(s, \pi(s))$, and then receiving an expected terminal reward according to the argument $V$:

$$(T_\pi V)(s) = r(s, \pi(s)) + \gamma \int_{\mathcal{S}} V(\widetilde{s})\, p(d\widetilde{s} \,|\, s, \pi(s)).$$

It is well-known that $V^\pi$ is the unique fixed-point of $T_\pi$, meaning $T_\pi V^\pi = V^\pi$ (Puterman, 2014). We will occasionally write $T_a = T_{\pi_a}$ where $\pi_a$ maps all states to the action $a \in \mathcal{A}$. The *Bellman operator* $T : \mathcal{F} \to \mathcal{F}$ is similarly defined using the maximizing action:

$$(TV)(s) = \max_{a \in \mathcal{A}} \left[ r(s, a) + \gamma \int_{\mathcal{S}} V(\widetilde{s})\, p(d\widetilde{s} \,|\, s, a) \right].$$

It is also known that $V^*$ is the unique fixed-point of $T$ (Puterman, 2014) and that acting greedy with respect to the optimal value function $V^*$ produces an *optimal policy*

$$\pi^*(s) \in \operatorname*{argmax}_{a \in A} \left[ r(s, a) + \gamma \int_{\mathcal{S}} V^*(\widetilde{s})\, p(d\widetilde{s} \,|\, s, a) \right].$$

Lastly, let $V \in \mathcal{F}$ and let $\nu$ be a distribution over $\mathcal{S}$. We define left and right versions of an operator $P_\pi$:

$$(P_\pi V)(s) = \int_{\mathcal{S}} V(\widetilde{s})\, p(d\widetilde{s} \,|\, s, \pi(s)),$$
$$(\nu P_\pi)(d\widetilde{s}) = \int_{\mathcal{S}} p(d\widetilde{s} \,|\, s, \pi(s))\, \nu(ds).$$

Note that $P_\pi V \in \mathcal{F}$ and $\mu P_\pi$ is another distribution over $\mathcal{S}$.

### 4.2.2 Feedback-Based Tree Search Algorithm

We now formally describe the proposed Algorithm 5. The parameters are as follows. Let $\bar{\Pi} \subseteq \Pi$ be a space of approximate policies and $\bar{\mathcal{F}} \subseteq \mathcal{F}$ be a space of approximate value functions (e.g., classes of neural network architectures). We let $\pi_k \in \bar{\Pi}$ be the policy function approximation (PFA) and $V_k \in \bar{\mathcal{F}}$ be the value function approximation (VFA) at iteration $k$ of the algorithm. Parameters subscripted with '0' are used in the value function approximation (regression) phase and parameters subscripted with '1' are used in the tree search phase. The full description of the procedure is given in Algorithm 5, but we first summarize the two phases, VFA (Steps 2 and 3) and MCTS (Steps 4, 5, and 6).

**VFA Phase.** Given a policy $\pi_k$, we wish to approximate its value by fitting a function using subroutine `Regress` on $N_0$ states sampled from a distribution $\rho_0$. The idea here is that because full rollouts during tree search are expensive, we can perform regression first on a smaller set of rollouts and use the resulting VFA on the next batch of tree search runs to reduce computation. For each sampled state $s$, we estimate its value using $M_0$ full rollouts, which can be obtained using the absorption time formulation of an infinite horizon MDP (Puterman, 2014, Proposition 5.3.1).

**MCTS Phase.** On iteration $k$, we first sample a set of $N_1$ i.i.d. states from a distribution $\rho_1$ over $\mathcal{S}$. From each state, a tree search algorithm, denoted `MCTS`, is executed for $M_1$ iterations on a search tree of maximum depth $d$. We assume here that the leaf evaluator is a general function of the PFA and VFA from the previous iteration, $\pi_k$ and $V_k$, and it is denoted as a "subroutine" `LeafEval`. The results of the `MCTS` procedure are piped into a subroutine `Classify`, which fits a new policy $\pi_{k+1}$ using classification (from continuous states to discrete actions) on the new data. As discussed more in Assumption 4.2.4, `Classify` uses $L_1$ observations (one-step rollouts) to compute a loss function.

**Algorithm 5:** Feedback-Based Tree Search

**Input:** Initial estimates $V^0$ and $\pi^0$.

**Subalgorithm:** **Regress**, **MCTS**, **LeafEval** and **Classify**

**Output:** Value and policy approximations $\{V^k\}$ and $\{\pi^k\}$.

**for** $k = 1, 2, \ldots$ **do**

1. Sample a set of $N_0$ i.i.d. states $\mathcal{S}_{0,k}$ from $\rho_0$ and $N_1$ i.i.d. states $\mathcal{S}_{1,k}$ from $\rho_1$.
2. Compute a sample average $\widehat{Y}_k(s)$ of $M_0$ independent rollouts of $\pi_k$ for each $s \in \mathcal{S}_{0,k}$. See Assumption 4.2.1.
3. Use `Regress` on the set $\{\widehat{Y}_k(s) : s \in \mathcal{S}_{0,k}\}$ to obtain a value function $V_k \in \bar{\mathcal{F}}$. See Assumption 4.2.1.
4. From each $s \in \mathcal{S}_{1,k}$, run `MCTS` with parameters $M_1$, $d$, and evaluator `LeafEval`. Return estimated value of each $s$, denoted $\widehat{U}_k(s)$. See Assumption 4.2.3.
5. For each $s \in \mathcal{S}_{1,k}$ and $a \in \mathcal{A}$, create estimate $\widehat{Q}_k(s, a) \approx (T_a V_k)(s)$ by averaging $L_1$ transitions from $p(\cdot \,|\, s, a)$. See Assumption 4.2.4.
6. Use `Classify` to solve a cost-sensitive classification problem and obtain $\pi_{k+1} \in \bar{\Pi}$. Costs are defined using $\{\widehat{U}_k(s) : s \in \mathcal{S}_{1,k}\}$ and $\{\widehat{Q}_k(s, \pi_{k+1}(s)) : s \in \mathcal{S}_{1,k}\}$. See Assumption 4.2.4. Increment $k$ and return to Step 1.

**end**

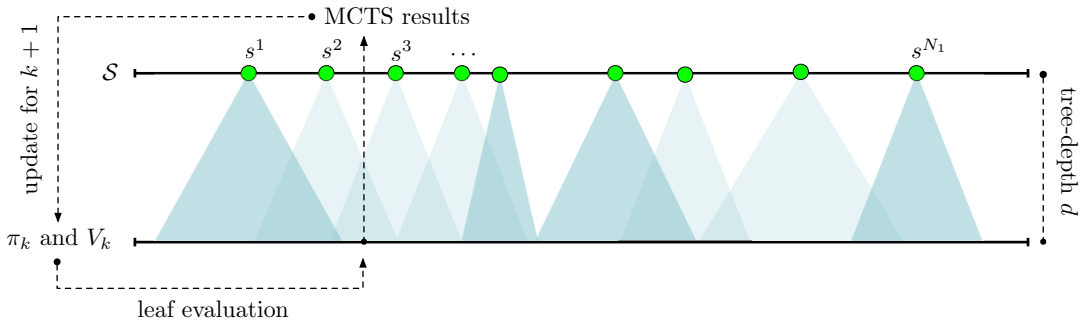



Figure 4.1: Illustration of Feedback-Based Tree Search

The illustration given in Figure 4.1 shows the interactions (and feedback loop) of the basic components of the algorithm: (1) a set of tree search runs initiated from a batch of sampled states (triangles), (2) leaf evaluation using $\pi_k$ and $V_k$ is used during tree search, and (3) updated PFA and VFA $\pi_{k+1}$ and $V_{k+1}$ using tree search results. Notice that depending on MDP dynamics from the initial state, tree search may expand differently sized trees (as indicated by the wide/narrow triangles).

### 4.2.3 Assumptions

Algorithm 5 shows the algorithm written with general subroutines `Regress`, `MCTS`, `LeafEval`, and `Classify`, allowing for variations in implementation suited for different problems. However, our analysis assumes specific choices and properties of these subroutines, which we describe now. The regression step solves a least absolute deviation problem to minimize an empirical version of

$$\|f - V^{\pi_k}\|_{1,\rho_0} = \int_{\mathcal{S}} |f(s) - V^{\pi_k}(s)| \rho_0(ds),$$

as described in the first assumption.

**Assumption 4.2.1** (`Regress` Subroutine)**.** For each $s^i \in \mathcal{S}_{0,k}$, define $s^i = s_0^{ij}$ for all $j$. Let $\widehat{Y}_k(s^i)$ be an estimate of $V^{\pi_k}(s^i)$ using $M_0$ rollouts and $V_k$, the VFA resulting

from `Regress`, be obtained via least absolute deviation regression:

$$\widehat{Y}_k(s_0^i) = \frac{1}{M_0} \sum_{j=1}^{M_0} \sum_{t=0}^{\infty} \gamma^t \, r(s_t^{ij}, \pi_k(s_t^{ij})), \tag{4.2.2}$$

$$V_k \in \operatorname*{argmin}_{f \in \bar{\mathcal{F}}} \frac{1}{N_0} \sum_{i=1}^{N_0} \left| f(s^i) - \widehat{Y}_k(s^i) \right|. \tag{4.2.3}$$

There are many ways that `LeafEval` may be defined. The standard leaf evaluator for MCTS is to simulate a default or "rollout" policy (Browne et al., 2012b) until the end of the game, though in related tree search techniques, authors have also opted for a value function approximation (Campbell et al., 2002; Enzenberger, 2004). It is also possible to combine the two approximations: Silver et al. (2016a) uses a weighted combination of a full rollout from a pre-trained policy and a pre-trained value function approximation.

**Assumption 4.2.2** (`LeafEval` Subroutine)**.** Our approach uses a partial rollout of length $h \geq 0$ and a value estimation at the end. `LeafEval` produces unbiased observations of

$$J_k(s) = \mathbf{E}\left[ \sum_{t=0}^{h-1} \gamma^t \, r(\widetilde{s}_t, \pi_k(\widetilde{s}_t)) + \gamma^h \, V_k(\widetilde{s}_h) \right], \tag{4.2.4}$$

where $\widetilde{s}_0 = s$.

This is motivated by our MOBA application, where we observed that even short rollouts (as opposed to simply using a VFA) are immensely helpful in determining local outcomes (e.g., dodging attacks, eliminating minions, health regeneration). At the same time, we found that numerous full rollouts simulated using the relatively slow and complex game engine is far too time-consuming within tree search.

We also need to make an assumption on the sample complexity of `MCTS`, of which there are many possible variations/implementations (Chaslot et al., 2006; Coulom, 2006; Kocsis and Szepesvári, 2006; Gelly and Silver, 2007; Couëtoux et al., 2011a;b; Al-Kanj et al., 2016; Jiang et al., 2017). Particularly relevant to our continuous-state

setting are tree expansion techniques called *progressive widening* and *double progressive widening*, proposed in Couëtoux et al. (2011a), which have proven successful in problems with continuous state/action spaces. To our knowledge, precise analysis of the sample complexity is only available for stylized versions of MCTS on finite problems, like Teraoka et al. (2014) and Kaufmann and Koolen (2017). Theorems from these two papers show upper bounds on the number of iterations needed so that with high probability (greater than $1-\delta$), the value at the root node is accurate within a tolerance of $\epsilon$. Fortunately, there are ways to discretize continuous state MDPs that enjoy error guarantees, such as Bertsekas (1975), Dufour and Prieto-Rumeau (2012), or Saldi et al. (2017). These error bounds can be combined with the MCTS guarantees of Teraoka et al. (2014) and Kaufmann and Koolen (2017) to produce a sample complexity bound for MCTS on continuous problems. The next assumption captures the essence of these results (and if desired, can be made precise for specific implementations through the references above).

**Assumption 4.2.3** (`MCTS` Subroutine)**.** Consider a $d$-stage, finite-horizon subproblem of (4.2.1) with terminal value function $J$ and initial state is $s$. Let the result of `MCTS` be denoted $\widehat{U}(s)$. We assume that there exists a function $m(\epsilon, \delta)$, such that if $m(\epsilon, \delta)$ iterations of `MCTS` are used, the inequality $|\widehat{U}(s) - (T^d J)(s)| \leq \epsilon$ holds with probability at least $1-\delta$.

Now, we are ready to discuss the `Classify` subroutine. Our goal is to select a policy $\pi \in \bar{\Pi}$ that *closely mimics* the performance of the `MCTS` result, similar to practical implementations in existing work (Guo et al., 2014; Silver et al., 2017; Anthony et al., 2017). The question is: given a candidate $\pi$, how do we measure "closeness" to the `MCTS` policy? We take inspiration from previous work in classification-based RL and use a cost-based penalization of classification errors (Langford and Zadrozny, 2005; Li et al., 2007; Lazaric et al., 2016). Since $\widehat{U}(s^i)$ is an approximation of the performance of the `MCTS` policy, we should try to select a policy $\pi$ with similar performance. To

estimate the performance of some candidate policy $\pi$, we use a one-step rollout and evaluate the downstream cost using $V_k$.

**Assumption 4.2.4** (`Classify` Subroutine)**.** For each $s^i \in \mathcal{S}_{1,k}$ and $a \in \mathcal{A}$, let $\widehat{Q}_k(s^i, a)$ be an estimate of the value of state-action pair $(s^i, a)$ using $L_1$ samples.

$$\widehat{Q}_k(s^i, a) = \frac{1}{L_1} \sum_{j=1}^{L_1} \left[ r(s^i, a) + \gamma V_k(\widetilde{s}^j(a)) \right].$$

Let $\pi_{k+1}$, the result of `Classify`, be obtained by minimizing the discrepancy between the `MCTS` result $\widehat{U}_k$ and the estimated value of the policy under approximations $\widehat{Q}_k$:

$$\pi_{k+1} \in \operatorname*{argmin}_{\pi \in \bar{\Pi}} \frac{1}{N_1} \sum_{i=1}^{N_1} \left| \widehat{U}_k(s^i) - \widehat{Q}_k(s^i, \pi(s^i)) \right|,$$

where $\widetilde{s}^j(a)$ are i.i.d. samples from $p(\cdot \,|\, s^i, a)$.

An issue that arises during the analysis is that even though we can control the distribution from which states are sampled, this distribution is transformed by the transition kernel of the policies used for rollout/lookahead. Let us now introduce the *concentrability coefficient* idea of Munos (2007) (and used subsequently by many authors, including Munos and Szepesvári (2008), Lazaric et al. (2016), Scherrer et al. (2015), and Haskell et al. (2016)).

**Assumption 4.2.5** (Concentrability)**.** Consider any sequence of $m$ policies $\mu_1, \mu_2, \ldots, \mu_m \in \Pi$. Suppose we start in distribution $\nu$ and that the state distribution attained after applying the $m$ policies in succession, $\nu P_{\mu_1} P_{\mu_2} \cdots P_{\mu_m}$, is absolutely continuous with respect to $\rho_1$. We define an $m$-step concentrability coefficient

$$A_m = \sup_{\mu_1, \ldots, \mu_m} \left\| \frac{d\nu P_{\mu_1} P_{\mu_2} \cdots P_{\mu_m}}{d\rho_1} \right\|_\infty,$$

and assume that $\sum_{i,j=0}^{\infty} \gamma^{i+j} A_{i+j} < \infty$. Similarly, we assume $\rho_1 P_{\mu_1} P_{\mu_2} \cdots P_{\mu_m}$, is absolutely continuous with respect to $\rho_0$ and assume that

$$A'_m = \sup_{\mu_1,\ldots,\mu_m} \left\| \frac{d\rho_1 P_{\mu_1} P_{\mu_2} \cdots P_{\mu_m}}{d\rho_0} \right\|_\infty$$

is finite for any $m$.

The concentrability coefficient describes how the state distribution changes after $m$ steps of arbitrary policies and how it relates to a given reference distribution. Assumptions 4.2.1-4.2.5 are used for the remainder of the paper.

## 4.3 Experiments and Results

### 4.3.1 Case Study: King of Glory MOBA AI

We implemented *Feedback-Based Tree Search* within a new and challenging environment, the recently popular MOBA game *King of Glory* by Tencent (the game is also known as *Honor of Kings* and a North American release of the game is titled *Arena of Valor*). Our implementation of the algorithm is one of the first attempts to design an AI for the 1v1 version of this game.


Figure 4.2: Screenshot from 1v1 *King of Glory*

**Game Description.** In the *King of Glory*, players are divided into two opposing teams and each team has a base located on the opposite corners of the game map (similar to other MOBA games, like *League of Legends* or *Dota 2*). The bases are guarded by towers, which can attack the enemies when they are within a certain attack range. The goal of each team is to overcome the towers and eventually destroy the opposing team's "crystal," located at the enemy's base. For this paper, we only consider the 1v1 mode, where each player controls a primary "hero" alongside less powerful game-controlled characters called "minions." These units guard the path to the crystal and will automatically fire (weak) attacks at enemies within range. Figure 4.2 shows the two heroes and their minions; the upper-left corner shows the map, with the blue and red markers pinpointing the towers and crystals.

**Experimental Setup.** We now give a brief summary of our experimental setup and describe our practical implementation of the algorithm; due to space constraints, more details are provided in the Appendix C.3. The state of the system is a 41-

dimensional vector containing information obtained directly from the game engine, including *hero locations*, *hero health*, *minion health*, *hero skill states*, and *relative locations to various structures*. There are 22 actions, including move, attack, heal, and special skill actions, some of which are associated with (discretized) directions. The reward function is designed to mimic *reward shaping* (Ng et al., 1999) and uses a combination of signals including *health*, *kills*, *damage dealt*, and *proximity to crystal*. We trained five King of Glory agents, using the hero *DiRenJie*:

1. The "FBTS" agent is trained using our feedback-based tree search algorithm for $K = 7$ iterations of 50 games each. The search depth is $d = 7$ and rollout length is $h = 5$. Each call to `MCTS` ran for 400 iterations.

2. The second agent is labeled "NR" for *no rollouts*. It uses the same parameters as the FBTS agent except no rollouts are used. At a high level, this bears some similarity to the AlphaGo Zero algorithm (Silver et al., 2017) in a batch setting.

3. The "DPI" agent uses the *direct policy iteration* technique of Lazaric et al. (2016) for $K = 10$ iterations. There is no value function and no tree search (due to computational limitations, more iterations are possible when tree search is not used).

4. We then have the "AVI" agent, which implements *approximate value iteration* (De Farias and Van Roy, 2000; Van Roy, 2006; Munos, 2007; Munos and Szepesvári, 2008) for $K = 10$ iterations. This algorithm can be considered a batch version of DQN (Mnih et al., 2013).

5. Lastly, we consider an "SL" agent trained via *supervised learning* on a dataset of approximately 100,000 state/action pairs of human gameplay data. Notably, the policy architecture used here is consistent with the previous agents.

In fact, both the policy and value function approximations are consistent across all agents; they use fully-connected neural networks with five and two hidden layers, respectively, and SELU (scaled exponential linear unit) activation (Klambauer et al., 2017). The initial policy $\pi_0$ takes random actions: move (w.p. 0.5), directional attack (w.p. 0.2), or a special skill (w.p. 0.3). Besides biasing the move direction toward the forward direction, no other heuristic information is used by $\pi_0$. `MCTS` was chosen to be a variant of UCT (Kocsis and Szepesvári, 2006) that is more amenable toward parallel simulations: instead of using the argmax of the UCB scores, we sample actions according to the distribution obtained by applying softmax to the UCB scores.

In the practical implementation of the algorithm, `Regress` uses a mean squared error loss while `Classify` combines a negative log-likelihood loss with a cosine proximity loss (due to continuous action parameters; see supplementary material), differing from the theoretical specifications. Due to the inability to "rewind" or "fast-forward" the game environment to arbitrary states, the sampling distribution $\rho_0$ is implemented by first taking random actions (for a random number of steps) to arrive at an initial state and then following $\pi_k$ until the end of the game. To reduce correlation during value approximation, we discard 2/3 of the states encountered in these trajectories. For $\rho_1$, we follow the `MCTS` policy while occasionally injecting noise (in the form of random actions and random switches to the default policy) to reduce correlation. During rollouts, we use the internal AI for the hero *DiRenJie* as the opponent.

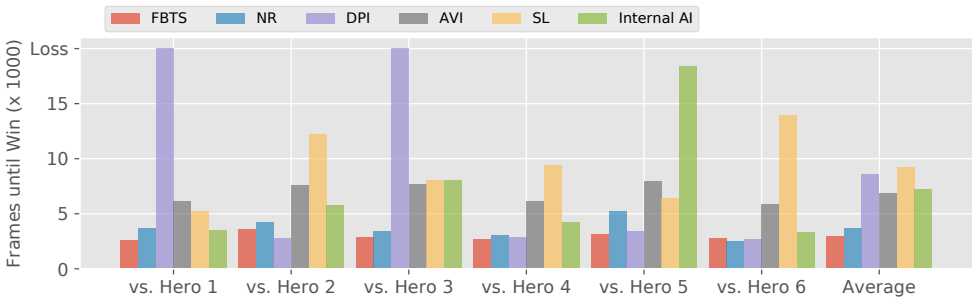



Figure 4.3: Number of Frames to Defeat Marksman Heroes

**Results.** As the game is nearly deterministic, our main test methodology is to compare the agents' effectiveness against a common set of opponents chosen from the internal AIs. We also added the internal DiRenJie AI as a "sanity check" baseline agent. To select the test opponents, we played the internal DiRenJie AI against other internal AIs (i.e., other heroes) and selected six heroes of the *marksman* type that the internal DiRenJie AI is able to defeat. Each of our agents, including the internal DiRenJie AI, was then played against every test opponent. Figure 4.3 shows the length of time, measured in frames, for each agent to defeat the test opponents (a value of 20,000 frames is assigned if the opponent won). Against the set of common opponents, FBTS significantly outperforms DPI, AVI, SL, and the internal AI. However, FBTS only slightly outperforms NR on average (which is perhaps not surprising as NR is the only other agent that also uses MCTS). Our second set of results help to visualize head-to-head battles played between FBTS and the four baselines (all of which are won by FBTS): Figure 4.4 shows the ratio of the FBTS agent's gold to its opponent's gold as a function of time. Gold is collected throughout the game as heroes deal damage and defeat enemies, so a ratio above 1.0 (above the red region) indicates good relative performance by FBTS. As the figure shows, each game ends with FBTS achieving a gold ratio in the range of $[1.25, 1.75]$.

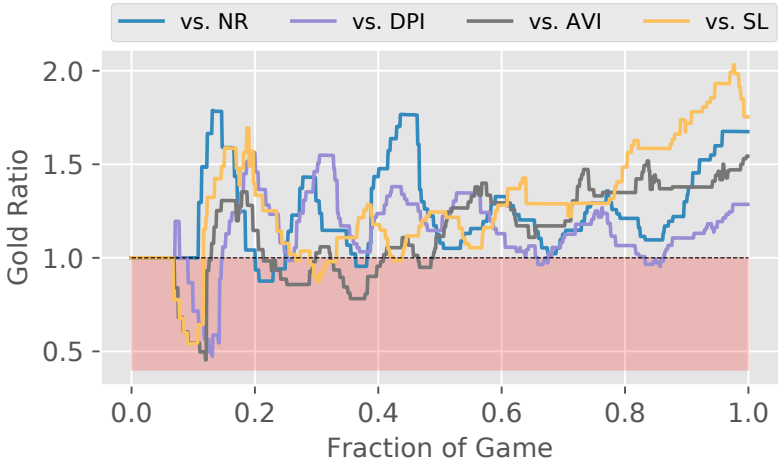



Figure 4.4: In-game Behavior: FBTS vs Competitors

The implementation of the algorithm in the 1v1 MOBA game *King of Glory* provided us encouraging results against several related algorithms; however, significant work remains for the agent to become competitive with humans. In the next section, we provide a sample complexity analysis for feedback-based tree search algorithm .

## 4.4 Theoretical Analysis

In this section, we provide a full sample complexity analysis of the MCTS-based reinforcement learning method and show that with large enough sample sizes and sufficiently large tree search effort, the performance of the estimated policies can be made close to optimal, up to some unavoidable approximation error. To our knowledge, batch MCTS-based RL methods have not been theoretically analyzed.

Sample complexity results for MCTS are relatively sparse. Teraoka et al. (2014) gives a high probability upper bound on the number of playouts needed to achieve $\epsilon$-accuracy at the root node for a stylized version of MCTS called `FindTopWinner`. More recently, Kaufmann and Koolen (2017) provided high probability bounds on the sample complexity of two other variants of MCTS called `UGapE-MCTS` and `LUCB-MCTS`. In this paper, we do not require any particular implementation of MCTS, but make a generic assumption on its accuracy that is inspired by these results.

### 4.4.1 Technical Lemmas

In this section, we present some technical lemmas that would be used in subsequent proofs.

**Lemma 4.4.1** (Section 4, Corollary 2 of Haussler (1992))**.** Let $\mathcal{G}$ be a set of functions from $\mathcal{X}$ to $[0, B]$ with pseudo-dimension $\mathrm{d}_{\mathcal{G}} < \infty$. Then for all $0 < \epsilon \leq B$, it holds

that

$$\mathbf{P}\left(\sup_{g\in\mathcal{G}}\left|\frac{1}{m}\sum_{i=1}^{m}g(X^{(i)})-\mathbf{E}\big[g(X)\big]\right|>\epsilon\right)\leq 8\left(\frac{32eB}{\epsilon}\log\frac{32eB}{\epsilon}\right)^{\mathrm{d}_{\mathcal{G}}}\exp\left(-\frac{\epsilon^2 m}{64B^2}\right), \tag{4.4.1}$$

where $X^{(i)}$ are i.i.d. draws from the distribution of the random variable $X$.

**Lemma 4.4.2.** Consider a policy $\mu\in\Pi$ and suppose each $s^i$ is sampled i.i.d. from $\rho_0$. Define initial states $s_0^{ij}=s^i$ for all $j$. Analogous to Step 5 of the algorithm and Assumption 1, let:

$$\widehat{Y}(s^i)=\frac{1}{M_0}\sum_{j=1}^{M_0}\sum_{t=0}^{\infty}\gamma^t\, r(s_t^{ij},\mu(s_t^{ij}))\quad\text{and}\quad V\in\operatorname*{argmin}_{f\in\bar{\mathcal{F}}}\frac{1}{N_0}\sum_{i=1}^{N_0}\big|f(s^i)-\widehat{Y}(s^i)\big|.$$

For $\delta\in(0,1)$ and $\epsilon\in(0,V_{\max})$, if the number of sampled states $N_0$ satisfies the condition:

$$N_0\geq\left(\frac{32V_{\max}}{\epsilon}\right)^2\left[\log\frac{32}{\delta}+2\mathrm{d}_{\bar{\mathcal{F}}}\log\frac{64eV_{\max}}{\epsilon}\right]=:\Gamma_{\mathrm{a}}(\epsilon,\delta),$$

and the number of rollouts performed from each state $M_0$ satisfies:

$$M_0\geq 8\left(\frac{V_{\max}}{\epsilon}\right)^2\log\frac{8N_0}{\delta}=:\Gamma_{\mathrm{b}}(\epsilon,\delta),$$

then we have the following bound on the error of the value function approximation:

$$\|V-V^{\mu}\|_{1,\rho_0}\leq\min_{f\in\bar{\mathcal{F}}}\|f-V^{\mu}\|_{1,\rho_0}+\epsilon,$$

with probabilty at least $1-\delta$.

*Proof.* Recall that the estimated value function $V$ satisfies

$$V\in\operatorname*{argmin}_{f\in\widehat{F}}\frac{1}{N_{\mathrm{v}}}\sum_{i=1}^{N_{\mathrm{v}}}\left|f(s_{\mathrm{v}}^i)-\frac{1}{M_0}\sum_{j=1}^{M_0}\Big[V^{\pi}(s_0^i)+\xi^j(s_0^i)\Big]\right|,$$

where for each $i$, the terms $\xi^j(s_0^i)$ are i.i.d. mean zero error. The inner summation over $j$ is an equivalent way to write $\widehat{Y}(s_0^i)$. Noting that the rollout results $V^\mu(s_0^i)+\xi^j(s_0^i) \in [0, V_{\max}]$, we have by a union bound followed by Hoeffding's inequality:

$$\begin{aligned}
\mathbf{P}\Big(\max_i \big|\widehat{Y}(s_0^i) - V^\mu(s_0^i)\big| > \epsilon\Big) &= \mathbf{P}\Big(\bigcup_{i=1}^{N_0} \big|\widehat{Y}(s_0^i) - V^\mu(s_0^i)\big| > \epsilon\Big) \\
&\leq \sum_{i=1}^{N_0} \mathbf{P}\Big( \big|\widehat{Y}(s_0^i) - V^\mu(s_0^i)\big| > \epsilon\Big) \\
&\leq 2N_0 \exp\Big(-\frac{2M_0\epsilon^2}{(V_{\max}-0)^2}\Big) \\
&= N_0\, \Delta_1(\epsilon, M_0), \qquad (4.4.2)
\end{aligned}$$

where $\Delta_1(\epsilon, M_0) = 2\exp\left(-2\,M_0\,\epsilon^2/V_{\max}^2\right)$. Next, we define the loss minimizing function $f^* \in \operatorname{argmin}_{f\in\bar{\mathcal{F}}} \|f - V^\mu\|_{1,\rho_0}$ and define the function

$$\Delta_2(\epsilon, N_0) = 8\left(\frac{32\,e\,V_{\max}}{\epsilon}\log\frac{32\,e\,V_{\max}}{\epsilon}\right)^{\mathrm{d}_{\bar{\mathcal{F}}}} \exp\left(-\frac{\epsilon^2 N_0}{64\,V_{\max}^2}\right),$$

representing the right-hand-side of the bound in Lemma 4.4.1 with $B = V_{\max}$ and $m = N_0$. By Lemma 4.4.1, the probabilities of the events

$$\begin{aligned}
&\left\{\left|\|V - V^\mu\|_{1,\rho_0} - \frac{1}{N_0}\sum_{i=1}^{N_0}\big|V(s_0^i) - V^\mu(s_0^i)\big|\right| > \frac{\epsilon}{4}\right\} \text{ and} \\
&\left\{\left|\|f^* - V^\mu\|_{1,\rho_0} - \frac{1}{N_0}\sum_{i=1}^{N_0}\big|f^*(s_0^i) - V^\mu(s_0^i)\big|\right| > \frac{\epsilon}{4}\right\}
\end{aligned} \qquad (4.4.3)$$

are each bounded by $\Delta_2(\epsilon/4, N_0)$. Also, it follows by the definition of $V$ that

$$\frac{1}{N_0}\sum_{i=1}^{N_0}\big|V(s_0^i) - \widehat{Y}(s_0^i)\big| \leq \frac{1}{N_0}\sum_{i=1}^{N_0}\big|f^*(s_0^i) - \widehat{Y}(s_0^i)\big|.$$

Therefore, using (4.4.2) twice and (4.4.3) once, we have by a union bound that the inequality

$$\|V - V^\mu\|_{1,\rho_0} \leq \min_{f \in \bar{\mathcal{F}}} \|f - V^\mu\|_{1,\rho_0} + \epsilon$$

happens with probability greater than $1 - 2\,N_0\,\Delta_1(\epsilon/4, M_0) - 2\,\Delta_2(\epsilon/4, N_0)$. We then choose $N_0$ so that $\Delta_2(\epsilon/4, N_0) = \delta/4$ (following Haussler (1992), we utilize the inequality $\log(a \log a) < 2 \log(a/2)$ for $a \geq 5$). To conclude, we choose $M_0$ so that $\Delta_1(\epsilon/4, M_0) = \delta/(4N_0)$. □

**Lemma 4.4.3** (Sampling Error)**.** Suppose $|\mathcal{A}| = 2$ and let $\mathrm{d}_{\bar{\Pi}}$ be the VC-dimension of $\bar{\Pi}$. Consider $Z$, $V \in \mathcal{F}$ and suppose each $s^i$ is sampled i.i.d. from $\rho_1$. Also, let $w^j$ be i.i.d. samples from the standard uniform distribution and $g : \mathcal{S} \times \mathcal{A} \times [0,1] \to \mathcal{S}$ be a transition function such that $g(s, a, w)$ has the same distribution as $p(\cdot\,|\,s, a)$. For $\delta \in (0,1)$ and $\epsilon \in (0, V_{\max})$, if the number of sampled states $N_1$ satisfies the condition:

$$N_1 \geq 128 \left(\frac{V_{\max}}{\epsilon}\right)^2 \left[\log \frac{8}{\delta} + \mathrm{d}_{\bar{\Pi}} \log \frac{e\,N_1}{\mathrm{d}_{\bar{\Pi}}}\right] =: \Gamma_{\mathrm{c}}(\epsilon, \delta, N_1),$$

and the number of sampled transitions $L$ satisfies:

$$L_1 \geq 128 \left(\frac{V_{\max}}{\epsilon}\right)^2 \left[\log \frac{8}{\delta} + \mathrm{d}_{\bar{\Pi}} \log \frac{e\,L_1}{\mathrm{d}_{\bar{\Pi}}}\right] =: \Gamma_{\mathrm{d}}(\epsilon, \delta, L_1),$$

then we have the bounds:

(a) $\sup_{\pi \in \bar{\Pi}} \left| \frac{1}{N_1} \sum_{i=1}^{N_1} |Z(s^i) - (T_\pi V)(s^i)| - \|Z - T_\pi V\|_{1,\rho_1} \right| \leq \epsilon$ w.p. at least $1 - \delta$.

(b) $\sup_{\pi \in \bar{\Pi}} \left| \frac{1}{L_1} \sum_{j=1}^{L_1} \left[ r(s^i, \pi(s^i)) + \gamma V\big(g(s^i, \pi(s^i), w^j)\big) \right] - (T_\pi V)(s^i) \right| \leq \epsilon$ w.p. at least $1 - \delta$.

*Proof.* First, we remark that in both (a) and (b), the term within the absolute value is bounded between 0 and $V_{\max}$. A second remark is that we reformulated the problem

using $w^j$ to take advantage of the fact that these random samples do not depend on the policy $\pi$. Such a property is required to invoke (Györfi et al., 2006, Theorem 9.1), a result that (Lazaric et al., 2016, Lemma 3) depends on. With these two issues in mind, an argument similar to the proof of (Lazaric et al., 2016, Lemma 3) gives the conclusion for both (a) and (b). □

### 4.4.2 Sample Complexity Analysis

Before presenting the sample complexity analysis, let us consider a sequence of policies $\{\pi_0, \pi_1, \pi_2, \ldots\}$ satisfying $T_{\pi_{k+1}} T^{d-1} V^{\pi_k} = T^d V^{\pi_k}$ with no error. Bertsekas and Tsitsiklis (1996, pp. 30-31) shows that $\pi_k \to \pi^*$ in the finite state and action setting. Our proposed algorithm 5 can be viewed as *approximately satisfying* this iteration in a continuous state space setting, where `MCTS` plays the role of $T^d$ and evaluation of $\pi_k$ uses a combination of accurate rollouts (due to `Classify`) and fast VFA evaluations (due to `Regress`). The sample complexity analysis requires the effects of all errors to be systematically analyzed.

For some $K \geq 0$, our goal is to develop a high probability upper bound on the *expected suboptimality over the state space* of the performance of policy $\pi_K$ under a testing distribution $\nu$: $\|V^* - V^{\pi_K}\|_{1,\nu}$. Because there is no requirement to control errors with probability one, bounds in $\|\cdot\|_{1,\nu}$ tend to be much more useful in practice than ones in the traditional $\|\cdot\|_\infty$. Notice that:

$$\begin{aligned} \frac{1}{N_1} \sum_{i=1}^{N_1} \left| \widehat{U}_k(s^i) - \widehat{Q}_k(s^i, \pi_{k+1}(s^i)) \right| \\ \approx \left\| T^d V^{\pi_k} - T_{\pi_{k+1}} V^{\pi_k} \right\|_{1,\rho_1}, \end{aligned} \tag{4.4.4}$$

where the left-hand-side is the loss function used in the classification step from Assumption 4.2.4. It turns out that we can relate the right-hand-side (under a different distribution) to the expected suboptimality after $K$ iterations $\|V^* - V^{\pi_K}\|_{1,\nu}$, as

shown in the following lemma. Full proofs of all results are given in the supplementary material.

**Lemma 4.4.4** (Loss to Performance Relationship)**.** The expected suboptimality of $\pi_K$ can be bounded as follows:

$$\begin{aligned}\|V^* - V^{\pi_K}\|_{1,\nu} \leq\ & \gamma^{Kd} \|V^* - V^{\pi_0}\|_\infty \\ & + \sum_{k=1}^{K} \gamma^{(K-k)d} \left\| T^d V^{\pi_{k-1}} - T_{\pi_k} V^{\pi_{k-1}} \right\|_{1,\Lambda_{\nu,k}}\end{aligned}$$

where $\Lambda_{\nu,k} = \nu \left(P_{\pi^*}\right)^{(K-k)d} \left[I - (\gamma P_{\pi_k})\right]^{-1}$.

*Proof.* This proof is a modification of arguments used in (Lazaric et al., 2016, Equation 8 and Theorem 7). By the fixed point property of $T_{\pi_{k+1}}$ and the definition of the Bellman operator $T$, we have

$$\begin{aligned}V^{\pi_k} - V^{\pi_{k+1}} &= T_{\pi_k} V^{\pi_k} - T_{\pi_{k+1}} V^{\pi_{k+1}} \quad \text{(Fixed point property)} \\ &\leq T^d V^{\pi_k} - T_{\pi_{k+1}} V^{\pi_{k+1}} \quad \text{(Definition of operator } T\text{)} \\ &= T^d V^{\pi_k} - T_{\pi_{k+1}} V^{\pi_k} + T_{\pi_{k+1}} V^{\pi_k} - T_{\pi_{k+1}} V^{\pi_{k+1}} \quad \text{(Subtracting and adding } T_{\pi_{k+1}} V^{\pi_k}\text{)} \\ &= T^d V^{\pi_k} - T_{\pi_{k+1}} V^{\pi_k} + (\gamma P_{\pi_{k+1}})\left(V^{\pi_k} - V^{\pi_{k+1}}\right). \qquad (4.4.5)\end{aligned}$$

Rearranging terms in equation 4.4.5, we have that

$$\left[I - (\gamma P_{\pi_{k+1}})\right]\left(V^{\pi_k} - V^{\pi_{k+1}}\right) \leq T^d V^{\pi_k} - T_{\pi_{k+1}} V^{\pi_k}. \qquad (4.4.6)$$

The term $\left[I - (\gamma P_{\pi_{k+1}})\right]$ is invertible and $\left[I - (\gamma P_{\pi_{k+1}})\right]^{-1} = \sum_{j=0}^{\infty} (\gamma P_{\pi_{k+1}})^j$ has positive elements. Therefore,

$$V^{\pi_k} - V^{\pi_{k+1}} \leq \left[I - (\gamma P_{\pi_{k+1}})\right]^{-1}\left(T^d V^{\pi_k} - T_{\pi_{k+1}} V^{\pi_k}\right). \qquad (4.4.7)$$

Similarly, we will bound the difference between $V^*$ and $V^{\pi_{k+1}}$ in terms of the distances between $V^* - V^{\pi_k}$ and $V^{\pi_k} - V^{\pi_{k+1}}$:

$$\begin{aligned}
V^* - V^{\pi_{k+1}} &= T_{\pi^*} V^* - T_{\pi_{k=1}} V^{\pi_{k+1}} \\
&= T_{\pi^*} V^* - T^d V^{\pi_k} + T^d V^{\pi_k} - T_{\pi_{k=1}} V^{\pi_{k+1}} \\
&\leq T_{\pi^*} V^* - T^d_{\pi^*} V^{\pi_k} + T^d V^{\pi_k} - T_{\pi_{k=1}} V^{\pi_{k+1}} \\
&\leq (\gamma P_{\pi^*})^d \big(V^* - V^{\pi_k}\big) + \big(T^d V^{\pi_k} - T_{\pi_{k=1}} V^{\pi_{k+1}}\big) \\
&= (\gamma P_{\pi^*})^d \big(V^* - V^{\pi_k}\big) + T^d V^{\pi_k} - T_{\pi_{k+1}} V^{\pi_k} + (\gamma P_{\pi_{k+1}}) \big(V^{\pi_k} - V^{\pi_{k+1}}\big).
\end{aligned} \tag{4.4.8}$$

Using the bound $V^{\pi_k} - V^{\pi_{k+1}} \leq \left[I - (\gamma P_{\pi_{k+1}})\right]^{-1} \left(T^d V^{\pi_k} - T_{\pi_{k+1}} V^{\pi_k}\right)$ from (4.4.7) on the last term of the right side of (4.4.8) along with a power series expansion on the inverse, we obtain:

$$\begin{aligned}
V^* - V^{\pi_{k+1}} &\leq (\gamma P_{\pi^*})^d \left(V^* - V^{\pi_k}\right) + \left[I + (\gamma P_{\pi_{k+1}}) \sum_{j=0}^{\infty} (\gamma P_{\pi_{k+1}})^j\right] \left(T^d V^{\pi_k} - T_{\pi_{k+1}} V^{\pi_k}\right) \\
&= (\gamma P_{\pi^*})^d \left(V^* - V^{\pi_k}\right) + \left[I + \sum_{j=1}^{\infty} (\gamma P_{\pi_{k+1}})^j\right] \left(T^d V^{\pi_k} - T_{\pi_{k+1}} V^{\pi_k}\right) \\
&= (\gamma P_{\pi^*})^d \left(V^* - V^{\pi_k}\right) + \left[\sum_{j=0}^{\infty} (\gamma P_{\pi_{k+1}})^j\right] \left(T^d V^{\pi_k} - T_{\pi_{k+1}} V^{\pi_k}\right) \\
&= (\gamma P_{\pi^*})^d \left(V^* - V^{\pi_k}\right) + \left[I - (\gamma P_{\pi_{k+1}})\right]^{-1} \left(T^d V^{\pi_k} - T_{\pi_{k+1}} V^{\pi_k}\right),
\end{aligned}$$

which can be iterated to show:

$$V^* - V^{\pi_K} \leq (\gamma P_{\pi^*})^{Kd} \left(V^* - V^{\pi_0}\right) + \sum_{k=1}^{K} (\gamma P_{\pi^*})^{(K-k)d} \left[I - (\gamma P_{\pi_k})\right]^{-1} \left(T^d V^{\pi_k} - T_{\pi_{k+1}} V^{\pi_k}\right).$$

The statement from the lemma follows from taking absolute value, bounding by the maximum norm, and integrating.

$$
\begin{aligned}
\|V^* - V^{\pi_K}\|_{1,\nu} &= \int_{\mathcal{S}} \left|V^*(s) - V^{\pi_K}(s)\right| \nu(ds) \\
&\leq \int_{\mathcal{S}} \left| (\gamma P_{\pi^*})^{K\,d} \left(V^*(s) - V^{\pi_0}(s)\right)\right| \nu(ds) \\
&\quad + \int_{\mathcal{S}} \left| \sum_{k=1}^{K} (\gamma P_{\pi^*})^{(K-k)\,d} \left[I - (\gamma P_{\pi_k})\right]^{-1} \left(T^d V^{\pi_k}(s) - T_{\pi_{k+1}} V^{\pi_k}(s)\right)\right| \nu(ds) \\
&\leq \gamma^{K\,d} \int_{\mathcal{S}} (P_{\pi^*})^{K\,d} \left|V^*(s) - V^{\pi_0}(s)\right| \nu(ds) \\
&\quad + \sum_{k=1}^{K} \gamma^{(K-k)\,d} \int_{\mathcal{S}} (P_{\pi^*})^{(K-k)\,d} \left[I - (\gamma P_{\pi_k})\right]^{-1} \left|T^d V^{\pi_k}(s) - T_{\pi_{k+1}} V^{\pi_k}(s)\right| \nu(ds) \\
&= \gamma^{K\,d} \|V^* - V^{\pi_0}\|_{\nu (P_{\pi^*})^{K\,d}} + \sum_{k=1}^{K} \gamma^{(K-k)\,d} \left\| T^d V^{\pi_{k-1}} - T_{\pi_k} V^{\pi_{k-1}} \right\|_{1,\Lambda_{\nu,k}} \\
&\leq \gamma^{K\,d} \|V^* - V^{\pi_0}\|_{\infty} + \sum_{k=1}^{K} \gamma^{(K-k)\,d} \left\| T^d V^{\pi_{k-1}} - T_{\pi_k} V^{\pi_{k-1}} \right\|_{1,\Lambda_{\nu,k}}
\end{aligned}
$$

where $\Lambda_{\nu,k} = \nu \left(P_{\pi^*}\right)^{(K-k)\,d} \left[I - (\gamma P_{\pi_k})\right]^{-1}$. □

From Lemma 4.4.4, we see that the expected suboptimality at iteration $K$ can be upper bounded by the suboptimality of the initial policy $\pi_0$ (in maximum norm) plus a discounted and weighted version of $\|T^d V^{\pi_{k-1}} - T_{\pi_k} V^{\pi_{k-1}}\|_{1,\rho_1}$ accumulated over prior iterations. Hypothetically, if $(T^d V^{\pi_{k-1}})(s) - (T_{\pi_k} V^{\pi_{k-1}})(s)$ were small for all $k$ and all states $s$, then the suboptimality of $\pi_K$ diminishes geometrically fast. Hence, we may refer to $\|T^d V^{\pi_{k-1}} - T_{\pi_k} V^{\pi_{k-1}}\|_{1,\rho_1}$ as the "true loss," the target term to be minimized. We now have a starting point for the analysis: if (4.4.4) can be made precise, then the result can be combined with Lemma 4.4.4 to provide an explicit bound on $\|V^* - V^{\pi_K}\|_{1,\nu}$. Figure 4.5 shows the various errors that we incur when relating the objective of `Classify` to the true loss: the error due to regression using functions in $\bar{\mathcal{F}}$; the error due to sampling the state space according to $\rho_1$; the error of estimating $(T_\pi V_k)(s)$ using the sample average of one-step rollouts $\widehat{Q}_k(s, \pi(s))$; and of course, the error due to `MCTS`.

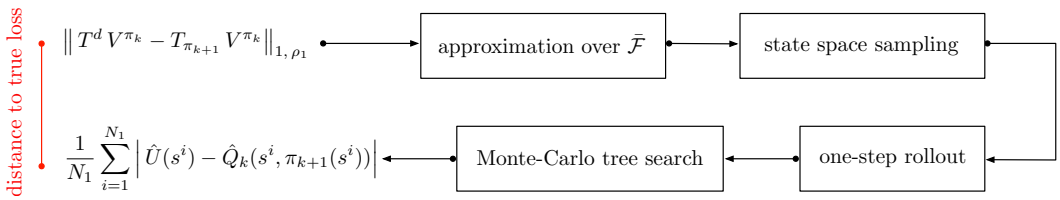



Figure 4.5: Various Errors Incurred in Feedback-Based Tree Search

We now give a series of lemmas that help us carry out the analysis outlined in Figure 4.5. In the algorithmic setting, the policy $\pi_k$ is a random quantity that depends on the samples collected in previous iterations; however, for simplicity, the lemmas that follow are stated from the perspective of a fixed policy $\mu$ or fixed value function approximation $V$ rather than $\pi_k$ or $V_k$. Conditioning arguments will be used when invoking these lemmas.

**Lemma 4.4.5** (Propagation of VFA Error)**.** Consider a policy $\mu \in \Pi$ and value function $V \in \mathcal{F}$. Analogous to (4.2.4), let $J = T_\mu^h V$. Then, under Assumption 4.2.5, we have the bounds:

(a) $\sup_{\pi \in \bar{\Pi}} \|T_\pi V - T_\pi V^\mu\|_{1,\rho_1} \leq \gamma A_1' \|V - V^\mu\|_{1,\rho_0}$,

(b) $\|T^d J - T^d V^\mu\|_{1,\rho_1} \leq \gamma^{d+h} A_{d+h}' \|V - V^\mu\|_{1,\rho_0}$.

*Proof.* For part (a), we note the following:

$$\begin{aligned}
\|T_\pi V - T_\pi V^\mu\|_{1,\rho_1} &= \gamma \int_{\mathcal{S}} \big|(P_\pi V)(s) - (P_\pi V^\mu)(s)\big| \, \rho_1(ds) \\
&= \gamma \int_{\mathcal{S}} \big|(P_\pi V)(s) - (P_\pi V^\mu)(s)\big| \, \rho_1(ds) \, \frac{d\rho_0}{d\rho_0} \\
&= \gamma \int_{\mathcal{S}} \Big| \int_{\mathcal{S}} \Big(V(\widetilde{s}) - V^\mu(s)\Big) p(d\widetilde{s} \,|\, s, \pi(s)) \Big| \, \rho_1(ds) \, \frac{d\rho_0}{d\rho_0} \\
&\le \gamma \int_{\mathcal{S}} \int_{\mathcal{S}} \Big|V(\widetilde{s}) - V^\mu(s)\Big| \, p(d\widetilde{s} \,|\, s, \pi(s)) \, \rho_1(ds) \, \frac{d\rho_0}{d\rho_0} \\
&= \gamma \int_{\mathcal{S}} \frac{d}{d\rho_0} \int_{\mathcal{S}} \Big|V(\widetilde{s}) - V^\mu(s)\Big| \, p(d\widetilde{s} \,|\, s, \pi(s)) \, \rho_1(ds) \, \rho_0(ds) \\
&= \gamma \int_{\mathcal{S}} \Big|V(\widetilde{s}) - V^\mu(s)\Big| \frac{d}{d\rho_0} \Big( \int_{\mathcal{S}} p(d\widetilde{s} \,|\, s, \pi(s)) \, \rho_1(ds) \Big) \, \rho_0(ds) \\
&= \gamma \int_{\mathcal{S}} \big|V(s) - V^\mu(s)\big| \, \frac{d(\rho_1 P_\pi)}{d\rho_0} \, \rho_0(ds) \\
&\le \gamma \left\| \frac{d(\rho_1 P_\pi)}{d\rho_0} \right\|_\infty \|V - V^\mu\|_{1,\rho_0}.
\end{aligned}$$

By the concentrability conditions of Assumption 5, the right-hand-side can be bounded by $\gamma \, A_1' \, \|V - V^\mu\|_{1,\rho_0}$. Now, we can apply the same steps with the roles of $T_\pi V$ and $T_\pi V^\mu$ reversed to see that the same inequality holds for $\|T_\pi V - T_\pi V^\mu\|_{1,\rho_1}$ and part (a) is complete.

For part (b), we partition the state space $\mathcal{S}$ into two sets:

$$\mathcal{S}^+ = \big\{ s \in \mathcal{S} : (T^d J)(s) \ge (T^d V^\mu)(s) \big\} \quad \text{and} \quad \mathcal{S}^- = \big\{ s \in \mathcal{S} : (T^d J)(s) < (T^d V^\mu)(s) \big\}.$$

We start with $\mathcal{S}^+$. Consider the finite-horizon $d$-stage MDP with the same dynamics as (4.2.1) and terminal condition $J$. Let $\pi_1^J, \pi_2^J, \ldots, \pi_d^J$ be the time-dependent optimal policy for this MDP. Thus, we have

$$T_{\pi_1^J} T_{\pi_2^J} \cdots T_{\pi_d^J} J = T^d J \quad \text{and} \quad T_{\pi_1^J} T_{\pi_2^J} \cdots T_{\pi_d^J} V^\mu \le T^d V^\mu.$$

Using similar steps as for part (a), the following hold:

$$\begin{aligned}\int_{\mathcal{S}^+} \left[(T^d J)(s) - (T^d V^\mu)(s)\right] \rho_1(ds) &\leq \int_{\mathcal{S}^+} \left[(T^d T_\mu^h V)(s) - (T_{\pi_1^J} T_{\pi_2^J} \cdots T_{\pi_d^J} T_\mu^h V^\mu)(s)\right] \rho_1(ds) \\ &\leq \gamma^{d+h} \int_{\mathcal{S}^+} \left|V(s) - V^\mu(s)\right| \frac{d(\rho_1 P_{\pi_1^J} P_{\pi_2^J} \cdots P_{\pi_d^J} P_\mu^h)}{d\rho_0} \rho_0(ds) \\ &\leq \gamma^{d+h} A'_{d+h} \int_{\mathcal{S}^+} \left|V(s) - V^\mu(s)\right| \rho_0(ds).\end{aligned}$$

Now, using the optimal policy with respect to the $d$-stage MDP with terminal condition $V^\mu$, we can repeat these steps to show that

$$\int_{\mathcal{S}^-} \left[(T^d V^\mu)(s) - (T^d J)(s)\right] \rho_1(ds) \leq \gamma^{d+h} A'_{d+h} \int_{\mathcal{S}^-} \left|V(s) - V^\mu(s)\right| \rho_0(ds).$$

Summing the two inequalities, we obtain:

$$\begin{aligned}\|T^d J - T^d V^\mu\|_{1,\rho_1} &\leq \gamma^{d+h} A'_{d+h} \left[\int_{\mathcal{S}^+} \left|V(s) - V^\mu(s)\right| \rho_0(ds) + \int_{\mathcal{S}^-} \left|V(s) - V^\mu(s)\right| \rho_0(ds)\right] \\ &= \gamma^{d+h} A'_{d+h} \|V - V^\mu\|_{1,\rho_0},\end{aligned}$$

which completes the proof. □

The lemma above addresses the fact that instead of using $V^{\pi_k}$ directly, `Classify` and `MCTS` only have access to the estimates $V_k$ and $J_k = T_{\pi_k}^h V_k$ ($h$ steps of rollout with an evaluation of $V_k$ at the end), respectively. Note that propagation of error in $V_k$ is discounted by $\gamma$ or $\gamma^{d+h}$ and since the lemma converts between $\|\cdot\|_{1,\rho_1}$ and $\|\cdot\|_{1,\rho_0}$, it is also impacted by the concentrability coefficients $A'_1$ and $A'_{d+h}$.

Let $\mathrm{d}_{\bar{\Pi}}$ be the *VC-dimension* of the class of binary classifiers $\bar{\Pi}$ and let $\mathrm{d}_{\bar{\mathcal{F}}}$ be the *pseudo-dimension* of the function class $\bar{\mathcal{F}}$. The VC-dimension is a measure of the *capacity* of $\bar{\Pi}$ and the notion of a pseudo-dimension is a generalization of the VC-dimension to real-valued functions (see, e.g., Pollard (1990), Haussler (1992), Mohri et al. (2012) for definitions of both). Similar to Lazaric et al. (2016) and Scherrer

et al. (2015), we will present results for the case of $|\mathcal{A}| = 2$ and note that the extension to multiple actions is possible by performing an analysis along the lines of Lazaric et al. (2016, Section 6). We now quantify the error illustrated in Figure 4.6. Define the quantity $B'_\gamma = \gamma A'_1 + \gamma^{d+h} A'_{d+h}$, the sum of the coefficients from Lemma 4.4.5.

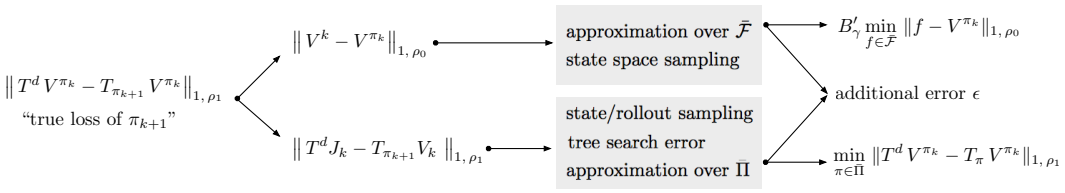



Figure 4.6: Various Errors Analyzed in Lemma 4.4.6

**Lemma 4.4.6.** Suppose the regression sample size $N_0$ is

$$\mathcal{O}\left((V_{\max} B'_\gamma)^2 \epsilon^{-2} \left[\log(1/\delta) + d_{\bar{\mathcal{F}}} \log(V_{\max} B'_\gamma/\delta)\right]\right)$$

and the sample size $M_0$, for estimating the regression targets, is

$$\mathcal{O}\left((V_{\max} B'_\gamma)^2 \epsilon^{-2} \left[\log(N_0/\delta)\right]\right).$$

Furthermore, there exist constants $C_1$, $C_2$, $C_3$, and $C_4$, such that if $N_1$ and $L_1$ are large enough to satisfy

$$N_1 \geq C_1 V_{\max}^2 \epsilon^{-2} \left[\log(C_2/\delta) + d_{\bar{\Pi}} \log(e N_1/d_{\bar{\Pi}})\right],$$
$$L_1 \geq C_1 V_{\max}^2 \epsilon^{-2} \left[\log(C_2 N_1/\delta) + d_{\bar{\Pi}} \log(e L_1/d_{\bar{\Pi}})\right],$$

and if $M_1 \geq m(C_3\epsilon, C_4\delta/N_1)$,

$$\|T^d V^{\pi_k} - T_{\pi_{k+1}} V^{\pi_k}\|_{1,\rho_1} \leq B'_\gamma \min_{f \in \bar{\mathcal{F}}} \|f - V^{\pi_k}\|_{1,\rho_0}$$
$$+ \min_{\pi \in \bar{\Pi}} \|T^d V^{\pi_k} - T_\pi V^{\pi_k}\|_{1,\rho_1} + \epsilon$$

with probability at least $1 - \delta$.

We first give an intuition for the proof of Lemma 4.4.6. By adding and subtracting terms, applying the triangle inequality, and invoking Lemma 4.4.5, we see that:

$$\begin{aligned}\|T^d V^{\pi_k} - T_{\pi_{k+1}} V^{\pi_k}\|_{1,\rho_1} \leq B'_\gamma \|V_k - V^{\pi_k}\|_{1,\rho_0} \\ + \|T^d J_k - T_{\pi_{k+1}} V_k\|_{1,\rho_1},\end{aligned}$$

Here, the error is split into two terms. The first depends on the sample $\mathcal{S}_{0,k}$ and the history through $\pi_k$ while the second term depends on the sample $\mathcal{S}_{1,k}$ and the history through $V_k$. We can thus view $\pi_k$ as fixed when analyzing the first term and $V_k$ as fixed when analyzing the second term (details in the supplementary material). The first term $\|V_k - V^{\pi_k}\|_{1,\rho_0}$ contributes the quantity $\min_{f \in \bar{\mathcal{F}}} \|f - V^{\pi_k}\|_{1,\rho_0}$ in the final bound with additional estimation error contained within $\epsilon$. The second term $\|T^d J_k - T_{\pi_{k+1}} V_k\|_{1,\rho_1}$ contributes the rest (see Figure 4.6).

The first two terms on the right-hand-side are related to the approximation power of $\bar{\mathcal{F}}$ and $\bar{\Pi}$ and can be considered unavoidable. We upper-bound these terms by maximizing over $\bar{\Pi}$, in effect removing the dependence on the random process $\pi_k$ in the analysis of the next theorem. We define:

$$\begin{aligned}\mathbb{D}_0(\bar{\Pi}, \bar{\mathcal{F}}) &= \max_{\pi \in \bar{\Pi}} \min_{f \in \bar{\mathcal{F}}} \|f - V^\pi\|_{1,\rho_0}, \\ \mathbb{D}_1^d(\bar{\Pi}) &= \max_{\pi \in \bar{\Pi}} \min_{\pi' \in \bar{\Pi}} \|T^d V^\pi - T_{\pi'} V^\pi\|_{1,\rho_1},\end{aligned}$$

two terms that are closely related to the notion of *inherent Bellman error* (Antos et al., 2008; Munos and Szepesvári, 2008; Lazaric et al., 2016; Scherrer et al., 2015; Haskell et al., 2017). Also, let $B_\gamma = \sum_{i,j=0}^{\infty} \gamma^{i+j} A_{i+j}$, which was assumed to be finite in Assumption 4.2.5. Now we present the formal proof for Lemma 4.4.6.

*Proof.* On each iteration of the the algorithm, two random samples are used: $\mathcal{S}_{0,k}$ and $\mathcal{S}_{1,k}$. From $\mathcal{S}_{0,k}$, we obtain $V_k$ and from $\mathcal{S}_{1,k}$ we obtain $\pi_{k+1}$. Let $\mathcal{S}_k = (\mathcal{S}_{0,k}, \mathcal{S}_{1,k})$ represent both of the samples at iteration $k$. We define:

$$\mathcal{G}_{k-1} = \sigma\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{k-1}\} \quad \text{and} \quad \mathcal{G}'_{k-1} = \sigma\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{k-1}, \mathcal{S}_{0,k}\}.$$

Due to the progression of the algorithm with two random samples per iteration, we will analyze each iteration in two steps. We first separate the two random samples by noting that:

$$\begin{aligned}\|T^d V^{\pi_k} - T_{\pi_{k+1}} V^{\pi_k}\|_{1,\rho_1} &\leq \|T^d V^{\pi_k} - T^d J_k\|_{1,\rho_1} + \|T_{\pi_{k+1}} V_k - T_{\pi_{k+1}} V^{\pi_k}\|_{1,\rho_1} \\ &\quad + \|T^d J_k - T_{\pi_{k+1}} V_k\|_{1,\rho_1} \\ &\leq (\gamma A'_1 + \gamma^{d+h} A'_{d+h}) \|V_k - V^{\pi_k}\|_{1,\rho_0} + \|T^d J_k - T_{\pi_{k+1}} V_k\|_{1,\rho_1},\end{aligned} \tag{4.4.9}$$

where the first inequality follows by adding and subtracting terms and the triangle inequality while the second inequality follows by Lemma 2. Now, we may analyze the first term on the right-hand-side conditional on $\mathcal{G}_{k-1}$ and the second term conditional on $\mathcal{G}'_{k-1}$.

As it is currently stated, Lemma 4.4.2 gives an unconditional probability for a fixed policy $\mu$. However, since $\mathcal{S}_{0,k}$ is independent from $\mathcal{G}_{k-1}$ and $\pi_k$ is $\mathcal{G}_{k-1}$-measurable, we can utilize Lemma 4.4.2 in a conditional setting using a well-known property of conditional expectations (Resnick, 2013, Property 12, Section 10.3). This property will be repeatedly used in this proof (without further mention). We obtain that for a sample size $N_0 \geq \Gamma_{\mathrm{a}}(\epsilon'/(\gamma A'_1 + \gamma^{d+h} A'_{d+h}), \delta')$,

$$\mathbf{P}\Big(\|V_k - V^{\pi_k}\|_{1,\rho_0} > \min_{f \in \bar{\mathcal{F}}} \|f - V^{\pi_k}\|_{1,\rho_0} + \epsilon'/(\gamma A'_1 + \gamma^{d+h} A'_{d+h}) \,\Big|\, \mathcal{G}_{k-1}\Big) \leq \delta'. \tag{4.4.10}$$

It remains for us to analyze the error of the second term $\|T^d J_k - T_{\pi_{k+1}} V_k\|_{1,\rho_1}$. By part (a) of Lemma 4.4.3 with $Z = T^d J_k$ and $V = V_k$, if $N_1 \geq \Gamma_c(\epsilon', \delta', N_1)$ and $s^i$ are sampled i.i.d. from $\rho_1$, we have

$$\mathbf{P}\left(\left|\frac{1}{N_1}\sum_{i=1}^{N_1}\left|(T^d J_k)(s^i) - (T_{\pi_{k+1}} V_k)(s^i)\right| - \left\|T^d J_k - T_{\pi_{k+1}} V_k\right\|_{1,\rho_1}\right| > \epsilon' \,\middle|\, \mathcal{G}'_{k-1}\right) \leq \delta'. \tag{4.4.11}$$

The term $(T_{\pi_{k+1}} V_k)(s^i)$ is approximated using $L_1$ samples. Part (b) of Lemma 4.4.3 along with a union bound shows that if $L_1 \geq \Gamma_d(\epsilon', \delta'/N_1, L_1)$, then

$$\mathbf{P}\left(\max_i \left|\widehat{Q}_k(s^i, \pi_{k+1}(s^i)) - (T_{\pi_{k+1}} V_k)(s^i)\right| > \epsilon' \,\middle|\, \mathcal{G}'_{k-1}\right) \leq \delta'. \tag{4.4.12}$$

Similarly, by Assumption 3, if the number of iterations of `MCTS` $M_1$ exceeds $m(\epsilon', \delta'/N_1)$, we can take a union bound to arrive at

$$\mathbf{P}\left(\max_i \left|\widehat{U}_k(s^i) - (T^d J_k)(s^i)\right| > \epsilon' \,\middle|\, \mathcal{G}'_{k-1}\right) \leq \delta'. \tag{4.4.13}$$

The maximum over $i$ can be replaced with an average over the $N_1$ samples and the conclusion of the last two bounds would remain unchanged. Since $\pi_{k+1}$ is assumed to optimize a quantity involving $\widehat{U}_k$ and $\widehat{Q}_k$, we want to relate this back to $\|T^d J_k - T_{\pi_{k+1}} V_k\|_{1,\rho_1}$. Indeed, taking expectation of both sides of inequalities (4.4.11)–(4.4.13) and then combining, we obtain that with probability at least $1 - 3\delta'$,

$$\begin{aligned}
\left\|T^d J_k - T_{\pi_{k+1}} V_k\right\|_{1,\rho_1} &\leq \frac{1}{N_1}\sum_{i=1}^{N_1}\left|\widehat{U}_k(s^i) - \widehat{Q}_k(s^i, \pi_{k+1}(s^i))\right| + 3\epsilon' \\
&\leq \frac{1}{N_1}\sum_{i=1}^{N_1}\left|\widehat{U}_k(s^i) - \widehat{Q}_k(s^i, \widetilde{\pi}(s^i))\right| + 3\epsilon'
\end{aligned}$$

where $\widetilde{\pi} \in \text{argmin}_{\pi \in \bar{\Pi}} \|T^d V^{\pi_k} - T_\pi V^{\pi_k}\|_{1,\rho_1}$. Following the same steps in reverse , we have:

$$\left\|T^d J_k - T_{\pi_{k+1}} V_k\right\|_{1,\rho_1} \leq \min_{\pi \in \bar{\Pi}} \|T^d V^{\pi_k} - T_\pi V^{\pi_k}\|_{1,\rho_1} + 6\epsilon', \tag{4.4.14}$$

with probability at least $1 - 6\,\delta'$. Finally, we take expectation of both sides of (4.4.10) and then combine with (4.4.9) and (4.4.14) while setting $\epsilon' = \epsilon/7$ and $\delta' = \delta/7$ to obtain

$$\begin{aligned}\|T^d V^{\pi_k} - T_{\pi_{k+1}} V^{\pi_k}\|_{1,\rho_1} \leq (\gamma\, A'_1 + \gamma^{d+h}\, A'_{d+h}) \min_{f \in \bar{\mathcal{F}}} \|f - V^{\pi_k}\|_{1,\rho_0} \\ + \min_{\pi \in \bar{\Pi}} \|T^d V^{\pi_k} - T_\pi V^{\pi_k}\|_{1,\rho_1} + \epsilon\end{aligned}$$

with probability at least $1 - \delta$. □

**Theorem 4.4.7.** Suppose the sample size requirements of Lemma 4.4.6 are satisfied with $\epsilon/B_\gamma$ and $\delta/K$ replacing $\epsilon$ and $\delta$, respectively. Then, the suboptimality of the policy $\pi_K$ can be bounded as follows:

$$\begin{aligned}\|V^* - V^{\pi_K}\|_{1,\nu} \leq B_\gamma\, [B'_\gamma\, \mathbb{D}_0(\bar{\Pi}, \bar{\mathcal{F}}) + \mathbb{D}_1^d(\bar{\Pi})] \\ + \gamma^{K\,d}\, \|V^* - V^{\pi_0}\|_\infty + \epsilon,\end{aligned}$$

with probability at least $1 - \delta$.

*Proof.* This proof essentially synthesizes the previous results. From the definition of $D_0(\bar{\Pi}, \bar{\mathcal{F}})$ and $\mathbb{D}_1^d(\bar{\Pi})$ from the main paper, we note that if the sample size assumptions of Lemma 3 are satisfied,

$$\|T^d V^{\pi_k} - T_{\pi_{k+1}} V^{\pi_k}\|_{1,\rho_1} \leq B'_\gamma\, \mathbb{D}_0(\bar{\Pi}, \bar{\mathcal{F}}) + \mathbb{D}_1^d(\bar{\Pi}) + \epsilon, \tag{4.4.15}$$

with probability at least $1 - \delta$. This removes any dependence on the iteration $k$ from the right-hand-side. We now integrate all results with Lemma 1 in order to find a

bound on the suboptimality $\|V^* - V^{\pi_K}\|_{1,\nu}$. Consider the distribution $\Lambda_{\nu,k}$, as defined in Lemma 1, which needs to be related to $\nu$. We can use the power series expansion to write:

$$\Lambda_{\nu,k} = \nu \, (P_{\pi^*})^{(K-k)\,d} \sum_{i=0}^{\infty} (\gamma P_{\pi_k})^i.$$

For a fixed $i$, the measure $\nu$ is transformed by applying $\pi^*$ a total of $K-k$ times and then $\pi_k$ a total of $i$ times. We see that the summation term on the right-hand-side of Lemma 1 can be upper-bounded in the following way:

$$\sum_{k=1}^{K} \gamma^{(K-k)\,d} \left\| T^d V^{\pi_{k-1}} - T_{\pi_k} V^{\pi_{k-1}} \right\|_{1,\Lambda_{\nu,k}}$$
$$\leq \left( \sum_{j=0}^{K-1} \sum_{i=0}^{\infty} \gamma^{j+i} \, A_{j+i} \right) \max_{k \leq K} \left\| T^d V^{\pi_{k-1}} - T_{\pi_k} V^{\pi_{k-1}} \right\|_{1,\rho_1},$$

where we use Assumption 5 with $m = K-k+i$, maximize over $k$ for the loss term, and then re-index with $j = K-k$. The coefficient in parentheses can be upper-bounded by $B_\gamma$ (since all $A_{j+i}$ are nonnegative). Finally, we use Inequality (4.4.15) and then a union bound over the $K$ iterations to conclude the statement of the theorem.

$$\begin{aligned}
\|V^* - V^{\pi_K}\|_{1,\nu} &\leq \gamma^{K\,d} \|V^* - V^{\pi_0}\|_\infty \; + \; \sum_{k=1}^{K} \gamma^{(K-k)\,d} \left\| T^d V^{\pi_{k-1}} - T_{\pi_k} V^{\pi_{k-1}} \right\|_{1,\Lambda_{\nu,k}} \\
&\leq \gamma^{K\,d} \|V^* - V^{\pi_0}\|_\infty \; + \left( \sum_{j=0}^{K-1} \sum_{i=0}^{\infty} \gamma^{j+i} \, A_{j+i} \right) \max_{k \leq K} \left\| T^d V^{\pi_{k-1}} - T_{\pi_k} V^{\pi_{k-1}} \right\|_{1,\rho_1} \\
&\leq \gamma^{K\,d} \|V^* - V^{\pi_0}\|_\infty \; + \sum_{i,j=0}^{\infty} \gamma^{i+j} A_{i+j} \left[ B'_\gamma \, \mathbb{D}_0(\bar{\Pi}, \bar{\mathcal{F}}) + \mathbb{D}_1^d(\bar{\Pi}) + \epsilon \right] \\
&= \gamma^{K\,d} \|V^* - V^{\pi_0}\|_\infty \; + B_\gamma \left[ B'_\gamma \, \mathbb{D}_0(\bar{\Pi}, \bar{\mathcal{F}}) + \mathbb{D}_1^d(\bar{\Pi}) + \epsilon \right]
\end{aligned}$$

□

## 4.5 Conclusion and Future Work

We modeled the long-term multiperiod sequential decision problems as Markov Decision Processes (MDPs). Moreover, we proposed a batch, Monte Carlo Tree Search (MCTS) based reinforcement learning (RL) method that operates on the continuous state, finite action Markov Decision Processes and exploits the idea that leaf-evaluators can be updated to produce a stronger tree search using *previous tree search results*. Function approximators are used to track policy and value function approximations, where the latter is used to reduce the length of the tree search rollout (oftentimes, the rollout of the policy becomes a computational bottle-neck in complex environments). In addition, we provided a full sample complexity analysis of the method and show that with large enough sample sizes and sufficiently large tree search effort, the performance of the estimated policies can be made close to optimal, up to some unavoidable approximation error. To our knowledge, batch MCTS-based RL methods have not been theoretically analyzed. Furthermore, the feedback-based tree search algorithm is tested on the popular MOBA game `King of Glory` (a North American version of the same game is titled `Arena of Valor`), where we build a competitive AI agent for the 1v1 mode of the game which demonstrates the power of our approach.

### Future Direction

The results of this work may lead to different direction. In what follows, we summarize some in terms of application and theory. Several directions appear promising for future work:

Our primary methodological avenues for future work are (1) to analyze a self-play variant of the algorithm and (2) to consider related techniques in multi-agent domains (see, e.g., Hu and Wellman (2003)). The implementation of the algorithm in the 1v1

MOBA game *King of Glory* provided us encouraging results against several related algorithms; however, significant work remains to become competitive with humans. In addition, the 5v5 version of the game remains a captivating challenge. Moreover, the next step for reinforcement learning is to leverage past experience to quickly learn new environments. Some of the current algorithms are prone to memorization and can't adapt well to new environments. While the case studies focuses on challenging video games, we hope the techniques developed would be applicable to a wide variety of domains.

# Part IV

# APPENDIX

# Appendix A

# Appendix to Chapter 2

## A.1 Simulational-Inference Framework

Simulational- Inference is a framework that uses simulation to generate data to do inference. This framework could be useful when you have limited data on the real system or limited high-quality data. Sometimes the available data may not contain some important features. Simulational- Inference deals with the issue of limited data by using simulation to generate additional data. We now describe the three major components of the simulational inference framework:

1. **Simulator Model Specification:** In this step, we specify a parametric model for the simulator.

2. **Simulator Parameter Calibration:** This step leverages real data to learn the parameters of the simulator.

3. **Simulator Validation and Inference:** This step verifies that the simulator is as close as possible to the real system. This is to ensure reliable results from the simulator. Thereafter, the simulator is ready for inference which may involve using the simulator to generate new data for inference.

## Simulator Model Specification

We will mostly use queueing models to illustrate this framework, however, this framework is much more general. Let's consider the finite capacity queueing models:$M_{\lambda(t)}/M_{\mu(t)}/1/k$ and $M_{\lambda(t)}/G/1/k$. The second queueing model has a general service distribution $G$.

## Simulator Parameter Calibration

In this section, we describe methods for estimating the intensity rate of the arrival process and the service time distribution from observed data. Estimating the arrival intensity and the distribution of service times from data are very important in practice to ensure the simulator closely mimics the actual system.

### Arrival Intensity Estimation

We first define the useful Poisson process.

**Definition A.1.1.** A random process $\{N_t\}$ is called a counting process if it satisfies the following:

- $N_0 = 0$ (i.e it starts at the origin.);
- $N_t$ is increasing function of $t$;
- $N_t$ increments by one every time that it changes.

**Definition A.1.2.** A counting process $\{N_t\}_{t\geq 0}$ is called a Poisson process with rate $\lambda > 0$, if it satisfies the following:

- $N_0 = 0$
- Independent increments i.e $N_t - N_s \quad \perp\!\!\!\perp \text{(independent)} \quad \{N_r\}_{r\leq s}$, for $t \geq s$
- Stationary increments i.e $N_t - N_s \quad \sim \quad \text{Pois}\big(\lambda(t-s)\big)$, for $t \geq s$

A Poisson process is a very popular counting process. It is used for counting the occurrences of certain random events that appear to happen at a certain rate. A random variable $X$ is said to be a Poisson random variable with parameter $\lambda$, written as $X \sim Poisson(\lambda)$, if its Probability mass function is given by

$$P_X\Big\{X = k\Big\} = \begin{cases} \frac{e^{-\lambda}\lambda^k}{k!} & \text{for } k \in R_X \\ 0 & \text{otherwise,} \end{cases}$$

where $R_X = \{0, 1, 2, \ldots\}$ is the range. Poisson process has stationary increments. In other words, the distribution of the number of arrivals in any interval depends only on the length of the interval and does not depend on the location of the interval on the real line.

**Definition A.1.3.** A counting process $\{N_t\}_{t\geq 0}$ is called a Nonhomogenous Poisson process with integrable rate function $\lambda : \mathbb{R}_+ \to \mathbb{R}_+$ if it satisfies the following:

- Independent increments i.e $N_t - N_s \quad \perp\!\!\!\perp \quad \{N_r\}_{r\leq s}$, for $t > s$
- Stationary increments i.e $N_t - N_s \quad \sim \quad \text{Pois}\big(\int_s^t \lambda(u)du\big)$, for $t > s$

A non-homogeneous Poisson process is similar to the regular Poisson process, except that the average rate of arrivals is no longer stationary with time. This is somewhat a generalization of the regular Poisson process. However, this generalization does not come without cost. The cost here is that we lose the stationary increment property of the regular Poisson process.

**Parametric Estimation:** One approach to estimating the arrival intensity function $\lambda(t)$ is via a parametric method. This approach assumes that the observed data are from a parametric family of distribution. This method involves two main steps; one must first decide on the functional form of the intensity function and thereafter estimate the parameters of the intensity function:

1. **First Step:**This step assumes the arrival intensity rate $\lambda(t)$ has a certain functional form. Some examples of common functional forms:

   - Polynomial: $\lambda(t) = a_0 + a_1 t + a_2 t^2 + \dots a_n t^n, \quad \lambda(t) \geq 0, \quad \forall t$
   - Exponentials of polynomial: $\lambda(t) = \exp\big(a_0 + a_1 t + a_2 t^2 + \dots a_n t^n\big)$
   - Fourier Series: $\lambda(t) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty}(a_n \cos(nt) + b_n \sin(nt)), \quad \lambda(t) \geq 0, \quad \forall t$
   - Exponential of Fourier: $\lambda(t) = \exp\Big(\frac{1}{2}a_0 + \sum_{n=1}^{\infty}(a_n \cos(nt) + b_n \sin(nt))\Big), \quad \lambda(t) \geq 0, \quad \forall t$

2. **Second Step:** After the parametric distribution of $\lambda(t)$ has been selected, the step estimates the corresponding parameters using the observed data. This essentially reduces the problem of estimating the intensity rate $\lambda(t)$ to the problem of estimating a set of parameters.

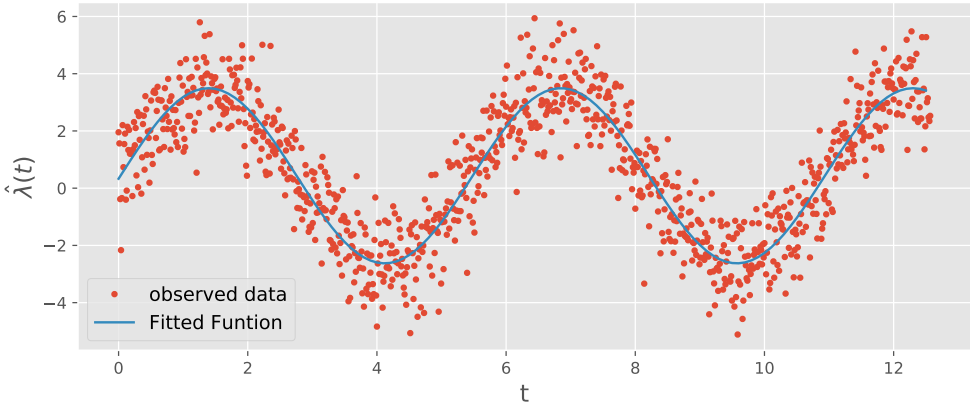



Figure A.1: In this example, the parametric form of the observe data has the form: $\lambda(t) = a_0 + a_1 \sin(\omega t + \phi)$ and estimating its parameter from the oberseved data gives $\widehat{\lambda}(t) = 0.52 + 3.04 \sin(1.15t - 0.01)$.

The parametric approach works well, as shown in Figure A.1, if the assumption that the observed data belong to the known parametric family is correct. Moreover, it

is a much easier problem to estimate a bunch of parameters than to estimate an arbitrary function. In the literature, Maximum Likelihood Estimation (MLE) and other methods can be used to estimate these parameters. However, if the parametric assumption fails, then the parametric approach becomes very restrictive. The good news is there is another approach that does not make any assumption about the parametric form of $\lambda(t)$.

**Non-parametric Estimation:** The non-parametric approach directly estimates $\lambda(t)$ from the observed sample data. it is not always the case that the data generating distribution is known a priori. Since we rarely have such knowledge, the non-parametric method is preferable. In what follows, we describe a non-parametric approach for estimating the intensity function of the Poisson process. Let $\lambda(t)$ denote the non-negative intensity function and define the cumulative intensity function:

$$\Lambda(x) = \int_0^x \lambda(t)dt, \quad x > 0, \tag{A.1.1}$$

where $\Lambda(x)$ is assumed to be monotone right continuous function. Now consider the probability density function on $0 \leq t \leq t_0$, defined as follows:

$$f(t) = \frac{\lambda(t)}{\Lambda(t_0) - \Lambda(0)}, \quad 0 \leq t \leq t_0. \tag{A.1.2}$$

Suppose we have observed a NHPP for a fixed time interval $(0, t_0]$ in which $n(n \neq 0)$ arrivals occurred at times $T_1 < T_2 < \ldots < T_n < t_0$. $f(t)$ can be estimated from a random sample $T_1, T_2, \ldots, T_n$ by the nonparametric kernel-type density estimator. Conditional on having observed $n$ events in the interval $(0, t_0]$, $T_i$ are distributed as the ordered statistics from a sample with distribution function $f(t)$ Arkin and Leemis (2000). The kernel density estimator for f(t) from random sample $T_1, T_2, \ldots, T_n$ is

defined by:

$$\widehat{f}_n(t) = \frac{1}{n}\sum_{j=1}^{n}\frac{1}{b(n)}W\left(\frac{t-T_j}{b(n)}\right) \tag{A.1.3}$$

This density estimator does not make any parametric assumptions on $f(t)$. However, it requires a choice of weight function and the bandwidth function. The weight function $W(u)$ is the kernel function which satisfies the following properties:

1. $W(u)$ is integrable
2. $\int W(t)dt = 1$
3. $W(u) \geq 0$ (non-negative)

And the bandwidth function $b(n)$ satisfies the following properties:

1. $\lim_{n\to\infty} b(n) = 0$
2. $b(n) > 0$ ( positive )

**Lemma A.1.1.** The estimator $\widehat{f}_n(t)$ is a non-negative density.

*Proof.*

$$\begin{aligned}
\int \widehat{f}_n(t) &= \int \frac{1}{n}\sum_{j=1}^{n}\frac{1}{b(n)}W\left(\frac{t-T_j}{b(n)}\right)dt \\
&= \frac{1}{n}\sum_{j=1}^{n}\int \frac{1}{b(n)}W\left(\frac{t-T_j}{b(n)}\right)dt \\
&= \frac{1}{n}\sum_{j=1}^{n}\int \frac{1}{b(n)}W(u)du \qquad \text{(A.1.4)} \\
&= \int W(u)du \\
&= 1 \qquad \text{(A.1.5)}
\end{aligned}$$

Equation A.1.4 follows from setting $t = u \cdot b(n) + T_j$ and the last equality follows from the property of the kernel W(u) and this completes the proof. Moreover $\widehat{f}_n(t)$ will have all the differentiability and continuity properties of the choice of kernel W. $\square$

Given that $\Lambda(t)$ is monotone non-decreasing and right continuous, then the number of event occurring in the interval $(0, t_0]$ has Poisson distribution with parameter

$$\Lambda(t_0) - \Lambda(0) = \int_0^{t_0} \lambda(x)dx \tag{A.1.6}$$

So $\Lambda(t_0) - \Lambda(0)$ is the mean of poisson random variable on $(0, t_0]$ which is estimated by $N_{t_0} = n$, the number of events in $(0, t_0]$. Therefore the estimate of the rate function is as follows:

$$\begin{aligned} \widehat{\lambda(t; n, t_0)} &= n \cdot \widehat{f}_n(t) \\ &= \frac{1}{b(n)} \sum_{j=1}^{n} W\left(\frac{t - T_j}{b(n)}\right). \end{aligned} \tag{A.1.7}$$

Next, we need to measure how well the nonparametric estimator $\widehat{f}_n(t)$ estimates $f(t)$. One way is to look at the mean square error (MSE) of the kernel estimate $\widehat{f}_n(t)$ at a point $t$, which is defined as follows:

$$\begin{aligned} \mathrm{MSE}\big(\widehat{f}_n(t)\big) &= \mathbb{E}\big[\widehat{f}_n(t) - f(t)\big]^2 \\ &= \mathbb{E}\big[\widehat{f}_n(t) - \mathbb{E}[\widehat{f}_n(t)]\big]^2 + \big(\mathbb{E}[\widehat{f}_n(t)] - f(t)\big)^2 \\ &= \mathbb{V}[\widehat{f}_n(t)] + \big(\mathrm{Bias}[\widehat{f}_n(t)]\big)^2. \end{aligned} \tag{A.1.8}$$

Observe that the MSE only measures the accuracy at a fixed point $t$. However, it is important to measure the accuracy of the entire function $\widehat{f}_n(t)$ as an estimate of $f(t)$. So instead of MSE, we consider the mean integrated squared error (MISE) defined as

follows:

$$\begin{aligned}
\mathrm{MISE}\big(\widehat{f}_n(t)\big) &= \mathbb{E}\left[\int \big(\widehat{f}_n(t) - f(t)\big)^2 dt\right] \\
&= \int \mathbb{E}\Big[\widehat{f}_n(t) - f(t)\Big]^2 dt \\
&= \int \mathrm{MSE}\Big(\widehat{f}_n(t)\Big) dt \\
&= \int \mathbb{V}\Big[\widehat{f}_n(t)\Big] dt + \int \Big(\mathrm{Bias}[\widehat{f}_n(t)]\Big)^2 dt. \qquad \text{(A.1.9)}
\end{aligned}$$

The second equality follows from the fact that $\big(\widehat{f}_n(t) - f(t)\big)^2$ is non negative, and the fourth equality follows from linearity of integral.

**Lemma A.1.2.** The kernel estimate $\widehat{f}_n(t)$ has the following properties:

$$\mathbb{E}[\widehat{f}_n(t)] = \int \frac{1}{b(n)} W\left(\frac{t-y}{b(n)}\right) f(y) dy,$$

$$\mathrm{Bias}[\widehat{f}_n(t)] = \int \frac{1}{b(n)} W\left(\frac{t-y}{b(n)}\right) [f(y) - f(t)] dy,$$

and

$$\mathbb{V}[\widehat{f}_n(t)] = \frac{1}{n}\left[\int \left[\frac{1}{b(n)^2} W^2\left(\frac{t-y}{b(n)}\right)\right] f(y) dy + \int \left[\frac{1}{b(n)} W\left(\frac{t-y}{b(n)}\right) f(y) dy\right]^2\right]$$

*Proof.* Since the $T_j$'s are random variables and $\widehat{f}_n(t)$ depends on the observations $T_1, T_2, \ldots, T_n$, for each t,$\widehat{f}_n(t)$ is continuous random variable. Hence we can compute

its expectation and variance.

$$\begin{aligned}
\mathbb{E}[\widehat{f}_n(t)] &= \mathbb{E}\left[\frac{1}{nb(n)}\sum_{j=1}^{n} W\left(\frac{t-T_j}{b(n)}\right)\right] \\
&= \frac{1}{n}\sum_{j=1}^{n}\mathbb{E}\left[\frac{1}{b(n)}W\left(\frac{t-T_j}{b(n)}\right)\right] \\
&= \frac{1}{n}\sum_{j=1}^{n}\int \frac{1}{b(n)}W\left(\frac{t-y}{b(n)}\right) f(y)dy \\
&= \int \frac{1}{b(n)}W\left(\frac{t-y}{b(n)}\right) f(y)dy. \qquad \text{(A.1.10)}
\end{aligned}$$

$$\begin{aligned}
\text{Bias}[\widehat{f}_n(t)] &= \mathbb{E}[\widehat{f}_n(t)] - f(t) \\
&= \int \frac{1}{b(n)}W\left(\frac{t-y}{b(n)}\right) f(y)dy - f(t) \\
&= \int \frac{1}{b(n)}W\left(\frac{t-y}{b(n)}\right) [f(y) - f(t)]dy \qquad \text{(A.1.11)}
\end{aligned}$$

$$\begin{aligned}
\mathbb{V}[\widehat{f}_n(t)] &= \mathbb{V}\left[\frac{1}{nb(n)}\sum_{j=1}^{n} W\left(\frac{t-T_j}{b(n)}\right)\right] \\
&= \frac{1}{n^2}\sum_{j=1}^{n}\mathbb{V}\left[\frac{1}{b(n)}W\left(\frac{t-T_j}{b(n)}\right)\right] \\
&= \frac{1}{n^2}\sum_{j=1}^{n}\left[\mathbb{E}\left[\frac{1}{b(n)^2}W^2\left(\frac{t-T_j}{b(n)}\right)\right] + \mathbb{E}\left[\frac{1}{b(n)}W\left(\frac{t-T_j}{b(n)}\right)\right]^2\right] \\
&= \frac{1}{n^2}\sum_{j=1}^{n}\left[\int\left[\frac{1}{b(n)^2}W^2\left(\frac{t-y}{b(n)}\right)\right] f(y)dy + \int\left[\frac{1}{b(n)}W\left(\frac{t-y}{b(n)}\right) f(y)dy\right]^2\right] \\
&= \frac{1}{n}\left[\int\left[\frac{1}{b(n)^2}W^2\left(\frac{t-y}{b(n)}\right)\right] f(y)dy + \int\left[\frac{1}{b(n)}W\left(\frac{t-y}{b(n)}\right) f(y)dy\right]^2\right].
\end{aligned}$$

□

So with certain regularity condition on the kernel $W$, we can approximate $\text{MISE}\big(\widehat{f}_n(t)\big)$ and the approximation will depend on the sample size n, the kernel function and derivative of the unknown function $f(t)$.

**Sampling from the Non-homogenous Poisson Intensity**

Let $t$ denote time and $I$ denote the number of events that have occurred by time $t$ and $A(I)$ denote the most recent event time. Suppose we have an estimated the Poisson intensity $\widehat{\lambda}(t)$. The idea of sampling from nonhomogeneous Poisson process with intensity $\widehat{\lambda}(t)$, is to first fix $\lambda_0$ such that

$$\widehat{\lambda}(t) \leq \lambda_0 \text{ for all } t \leq T.$$

Then generate the non-homogeneous process by a random selection of event times of Poisson process with rate $\lambda_0$. In other words, by independence, we count the event of $\text{Poi}(\lambda_0)$ with probability $\widehat{\lambda}(t)/\lambda_0$ . Then the counted process follows a $\text{Poi}(\widehat{\lambda}(t))$ Ross (2013).

**Algorithm 6:** Random sampling for Non-homogeneous Poisson Intensity

**Input:** Given the time horizon $T$ and the estimated intensity $\widehat{\lambda}(t)$.

**Output:** $\{A(i) \,|\, i = 1, 2, \ldots, I\}$

$\lambda_0 = \min_{\lambda}\{\lambda | \widehat{\lambda}(t) \leq \lambda,\ t = 1, 2, \ldots, T\}$ and initialize $t = 0$, $I = 0$;

**while** $t \leq T$ **do**

  Generate $U_1 \sim (0, 1)$;

  update $t \leftarrow t + \frac{1}{\lambda_0} log(\frac{1}{U_1})$

  Generate $U_2 \sim (0, 1)$;

  **if** $\lambda_0 \cdot U_2 \leq \widehat{\lambda}(t)$ **then**
    update $I \leftarrow I + 1$ and A(I)= t

  **end**

**end**

## Estimating General Service-time Distribution G

Given a sample $\{X_i\}_{i=i}^{n} \overset{iid}{\sim} G$, where $G$ is the unknown service distribution function. We would like to infer $G$ using the sample $\{X_i\}_{i=i}^{n}$. In what follows, we derive the empirical distribution function $\widehat{G}_n$:

$$\widehat{G}_n(x) = \frac{1}{n} \sum_{i=1}^{n} \big\{X_i \leq x\big\}, \tag{A.1.12}$$

where $\big\{X_i \leq x\big\}$ is an indicator function given by:

$$\big\{X_i \leq x\big\} = \begin{cases} 1 & \text{if} \quad X_i \leq x \\ 0 & \text{otherwise.} \end{cases} \tag{A.1.13}$$

**Lemma A.1.3.** The estimator $\widehat{G}_n(x)$ is unbiased since and $\mathrm{MSE}\big(\widehat{G}_n(x)\big) = \mathbb{V}[\widehat{G}_n(x)]$

*Proof.*

$$\begin{aligned}
\mathbb{E}[\widehat{G}_n(x)] &= \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\big\{X_i \leq x\big\}\right] \\
&= \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\Big[\big\{X_i \leq x\big\}\Big] \\
&= \frac{1}{n}\sum_{i=1}^{n}P\Big\{X_i \leq x\Big\} \\
&= P\Big\{X_i \leq x\Big\} \\
&= G(x). \qquad \text{(A.1.14)}
\end{aligned}$$

Lets look at the mean square error (MSE) of $\widehat{G}_n(x)$.

$$\begin{aligned}
\mathrm{MSE}\big(\widehat{G}_n(x)\big) &= \mathbb{E}\Big[\widehat{G}_n(x) - G(x)\Big]^2 \\
&= \mathbb{E}\Big[\widehat{G}_n(x) - \mathbb{E}[\widehat{G}_n(x)]\Big]^2 + \Big(\mathbb{E}[\widehat{G}_n(x)] - G(x)\Big)^2 \\
&= \mathbb{V}[\widehat{G}_n(x)] + \Big(\mathrm{Bias}[\widehat{G}_n(x)]\Big)^2 \qquad \text{(A.1.15)}
\end{aligned}$$

and

$$\begin{aligned}
\mathrm{Bias}[\widehat{G}_n(x)] &= \mathbb{E}[\widehat{G}_n(x)] - G(x) \\
&= G(x) - G(x) \\
&= 0 \qquad \text{(A.1.16)}
\end{aligned}$$

Since the bias term vanishes, the MSE is just the variance.

$$\begin{aligned}
\mathrm{MSE}\big(\widehat{G}_n(x)\big) &= \mathbb{V}[\widehat{G}_n(x)] \\
&= \mathbb{V}\left[\frac{1}{n}\sum_{i=1}^{n}\big\{X_i \le x\big\}\right] \\
&= \frac{1}{n^2}\sum_{i=1}^{n}\left[\mathbb{E}\Big[\big\{X_i \le x\big\}^2\Big] - \mathbb{E}\Big[\big\{X_i \le x\big\}\Big]^2\right] \\
&= \frac{1}{n^2}\sum_{i=1}^{n}\left[\mathbb{E}\Big[\big\{X_i \le x\big\}\Big] - \mathbb{E}\Big[\big\{X_i \le x\big\}\Big]^2\right] \\
&= \frac{1}{n^2}\sum_{i=1}^{n}\Big[\mathbb{P}\{X_i \le x\} - \mathbb{P}\{X_i \le x\}^2\Big] \\
&= \frac{1}{n}\Big[\mathbb{P}\{X_1 \le x\} - \mathbb{P}\{X_1 \le x\}^2\Big] \\
&= \frac{1}{n}\Big[G(x) - G(x)^2\Big] \qquad \text{(A.1.17)}
\end{aligned}$$

which implies $\widehat{G}_n(x)$ converges to $G(x)$ in probabilistic sense.

$$\begin{aligned}
\mathbb{P}\Big\{|\widehat{G}_n(x) - G(x)| > \epsilon\Big\} &= \mathbb{P}\Big\{|\widehat{G}_n(x) - G(x)|^2 > \epsilon^2\Big\} \\
&\le \frac{\mathbb{E}\Big(\widehat{G}_n(x) - G(x)\Big)^2}{\epsilon^2} \\
&= \frac{\mathrm{MSE}\big(\widehat{G}_n(x)\big)}{\epsilon^2} \\
&= \frac{\Big(G(x) - G(x)^2\Big)}{n\epsilon^2}. \qquad \text{(A.1.18)}
\end{aligned}$$

By Glivenko-Cantelli Theorem, under our assumptions, we can get a much stronger convergence result:

$$\sup_{y\in\mathbb{R}}|\widehat{G}_n(x) - G(x)| \quad \overrightarrow{a.s} \quad 0 \qquad \text{(A.1.19)}$$

as $n \to \infty$. □

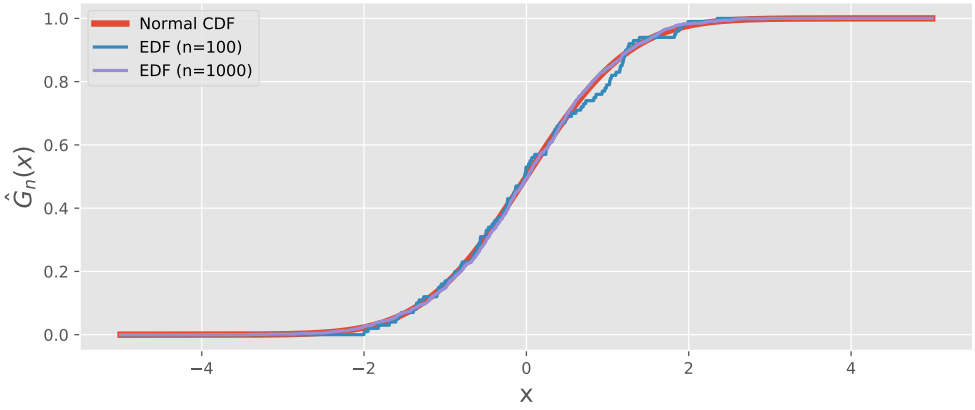



Figure A.2: As the value of $n$ increases, we get a better approximation of the distribution for the normal cumulative distribution (CDF) function. Notice that for $n = 1000$, the approximation is almost indistinguishable from the Normal CDF

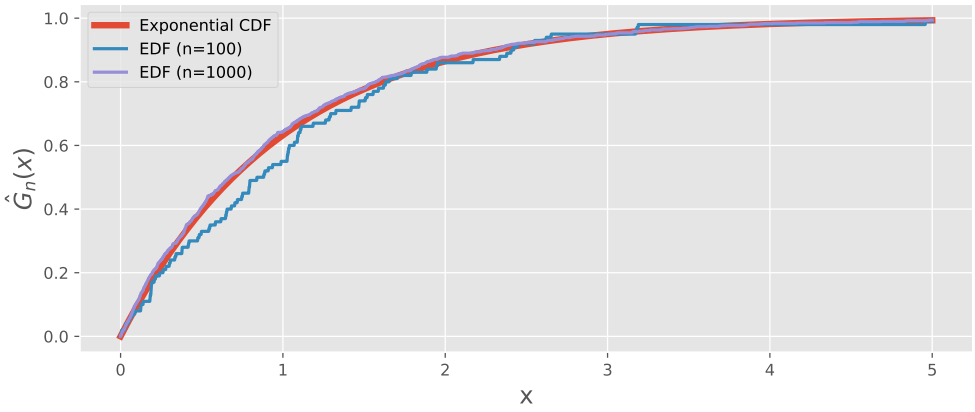



Figure A.3: As the value of $n$ increases, we get a better approximation of the distribution for the exponential cumulative distribution (CDF) function. Notice that for $n = 1000$, the approximation is almost indistinguishable from the Exponential CDF

Although the empirical density estimation is relatively simple, it works well in practice as depicted in Figures A.2 and A.3. Another interesting method of estimating

probability density function is the kernel density estimation method which we will omit.

### Sampling from General Service-time Distributions

**Continuous Inverse Transform method:** Consider a continuous random variable $X$ whose cumulative distribution function is $G$. This method uses the following well-known result.

**Lemma A.1.4.** Let $U \sim uniform(0,1)$ be a uniform random variable. For any continuous distribution $G$ the random variable $X = G^{-1}(U)$ has distribution G Ross (2013).

*Proof.* Note continuity is important for this result to guarantee that $G(x)$ is monotonically increasing.

$$\begin{aligned} G_X(x) &= \mathbb{P}\{X \leq x\} \\ &= \mathbb{P}\{G^{-1}(U) \leq x\} \\ &= \mathbb{P}\{G(G^{-1}(U)) \leq G(x)\} \\ &= \mathbb{P}\{U \leq G(x)\} \\ &= G(x). \end{aligned} \tag{A.1.20}$$

□

This result is very useful, it essentially states that one can generate a random variable $X$, whose continuous distribution is $G$, by simply generating a uniform random variable $U$ and evaluating it at the inverted cumulative distribution of X. So if the distribution $G$ is easily invertible, then the distribution $G$ is easy to sample from.

**Discrete Inverse Transform method:** consider the discrete version of the inverse transform method. Assume that $X$ is a discrete random variable with the following

probability mass function:

$$\mathbb{P}\Big\{X = x_i\Big\}_{i=0}^{\infty} = p_i, \tag{A.1.21}$$

where $\sum_{i=0}^{\infty} p_i = 1$. Notice that we can achieve the above probability using a uniform random variable. Let $U \sim uniform(0,1)$ be a uniform random variable, then

$$\begin{aligned}\mathbb{P}\Big\{X = x_i\Big\}_{i=0}^{\infty} &= \mathbb{P}\left\{\sum_{j=0}^{i-1} p_j \leq U < \sum_{i=0}^{i} p_j\right\}_{i=0}^{\infty} \\ &= p_i. \end{aligned} \tag{A.1.22}$$

## Simulator Validation and Inference

It is not always straight forward how to track the evolution of the probabilistic model via simulation. We essentially generate and observe the stochastic elements of the model over time and track the variables of interest. During simulation, we continually track the time variables and the system state variables summarized in Table A.1. And we update these variables whenever an event occurs.

| Variables | Notation | Description |
|---|---|---|
| Time | $t$ | The discrete time |
| Counter | $(N_A, N_D)$ | The number of arrival and the number of departure at time $t$ |
| System State | $n$ | The number of customer in the system at time $t$ |
| System State | (E, F) | The number of times the system is empty, full respectively at time $t$ |

Table A.1: An example of discrete event simulation variables

### Estimating the Queue Length for $M_{\lambda(t)}/M_{\mu(t)}/1/k$

Let $Q(t)$ be the queue length at time $t$. Use the constructed simulator, we can estimate the mean number $\mathbb{E}[Q)(t)]$ of the system (the number of customers in the queue or buffer plus the customer in service) as a function of time $t$. Suppose we

observe independent and identically distributed samples $\widehat{Q}(t), \widehat{Q}(t), \ldots, \widehat{Q}(T)$, then we can define the empirical estimator:

$$\widehat{EQ}_T = \frac{1}{T} \sum_{t=1}^{T} \widehat{Q}(t) \tag{A.1.23}$$

and the true mean number of the system as $EQ = \mathbb{E}[Q)(t)]$. Then by the law of the large numbers, for $T$ large enough the empirical mean $\widehat{EQ}_T$ will converge to the true mean $EQ$.

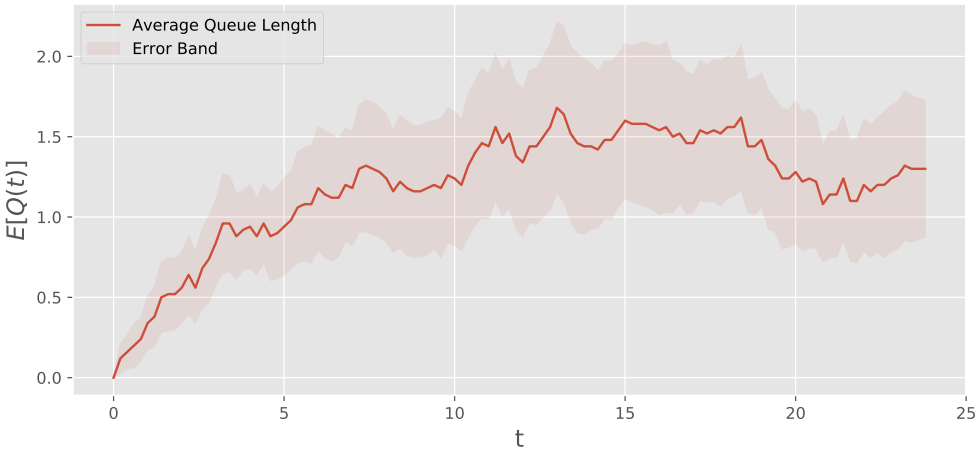



Figure A.4: The queue length as a function of time. Given the queueing model $M_{\lambda(t)}/M_{\mu(t)}/1/k$ with the mean Poisson arrival rate of $\lambda(t) = 0.5$ and exponential service rate of $\mu(t) = 0.8$ for all $t \in [0, T]$. The initial queue length $Q(0) = 0$.

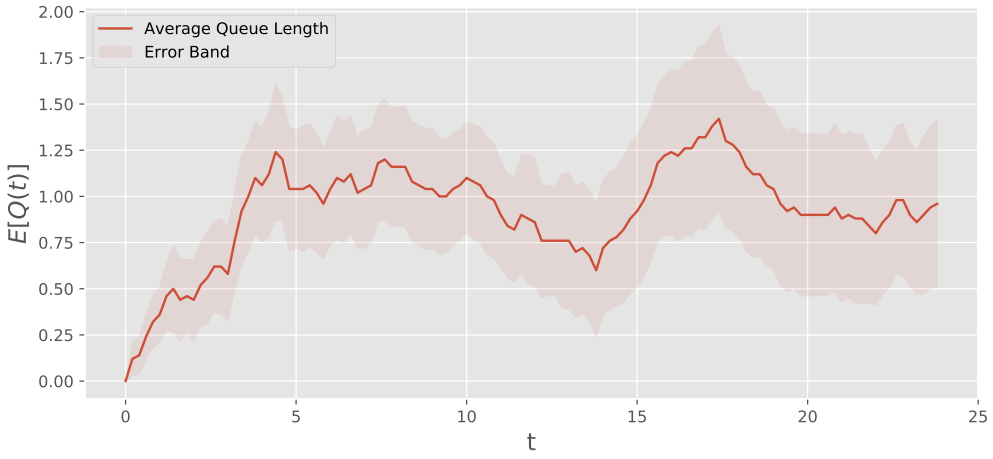
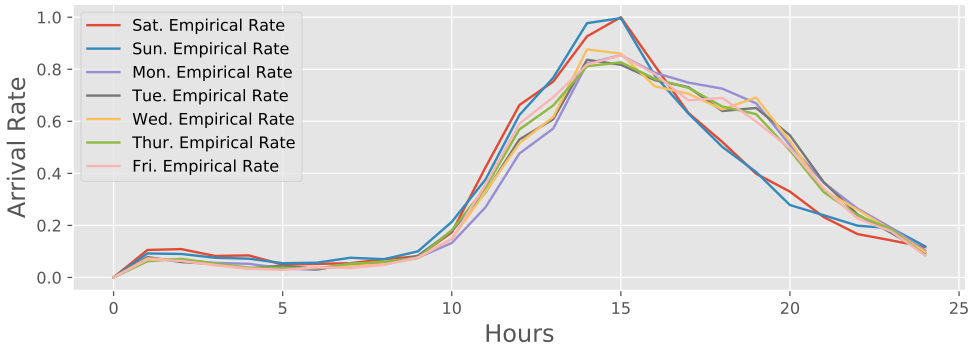


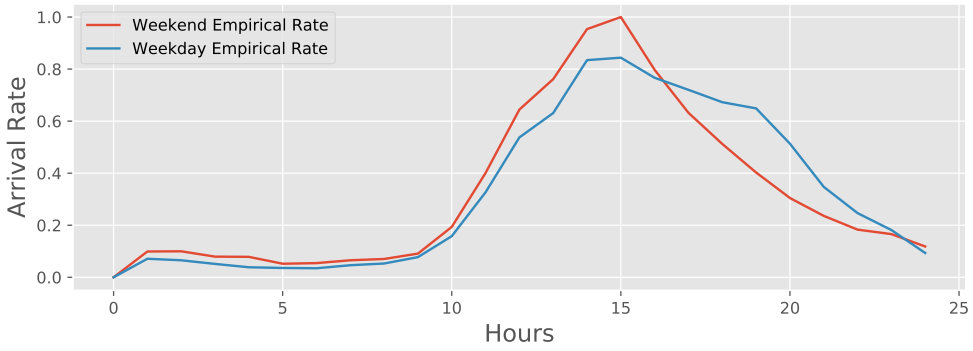

Figure A.5: The queue length as a function of time. Given the queueing model $M_{\lambda(t)}/M_{\mu(t)}/1/k$ with the mean Poisson arrival rate of $\lambda(t) = 0.6 + 0.4 \cdot \sin(0.2\pi t)$, and exponential service rate of $\mu(t) = 0.8 + 0.4 \cdot \sin(0.2\pi t)$ for all $t \in [0, T]$. The initial queue length $Q(0) = 0$.

**Estimating the arrivals and Queue Length for** $M_{\lambda(t)}/G/n/\infty/FCFS$

Understanding arrival rates and expected queue lengths is vital to developing optimal resource allocation. We use real housekeeping data, tracking the bed cleaning process at the University of Pittsburgh Medical Center (UPMC) Harmon, Emily Campbell (2012). The data was collected using a Bed-Tracking system between the years 2010 and 2011, which monitors dirty beds throughout the hospital. The available data contains information such as the arrival of dirty beds, clean time, in progress time, and other relevant information. For simulation, we use the historical arrival rate per hour of each day of the week.

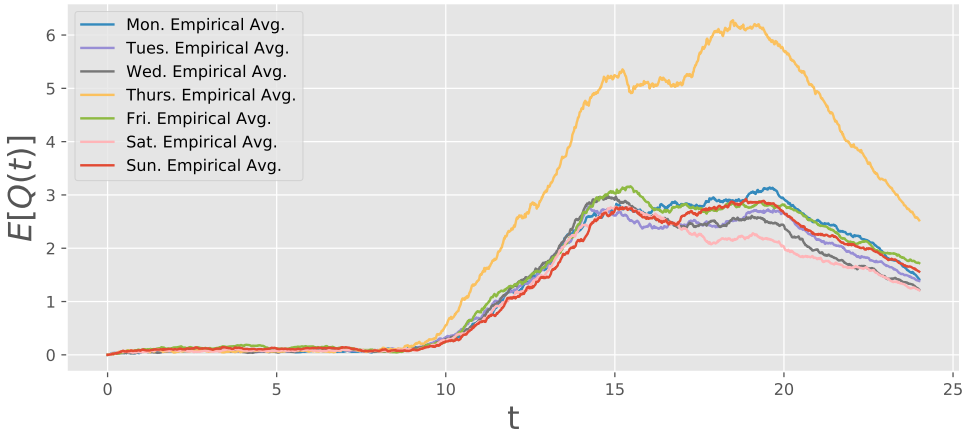



(a)

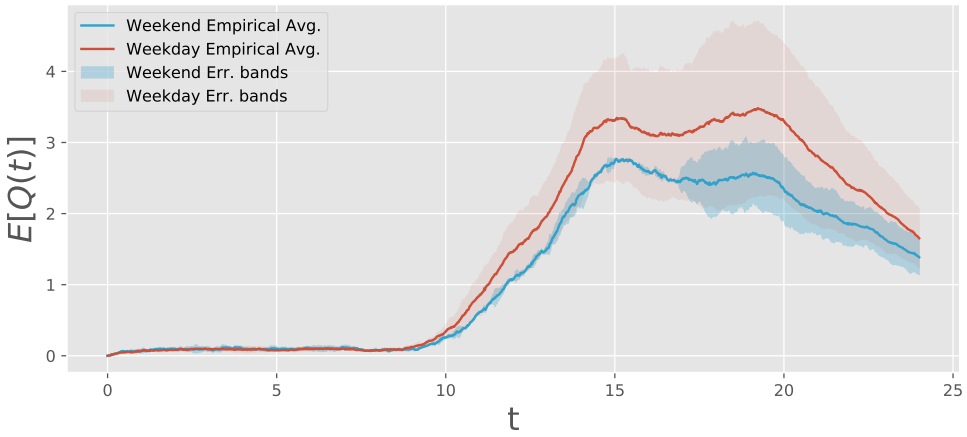



(b)

Figure A.6: Plot of different empirical arrival rates. In (a), we plot the empirical arrival rates by each day of the week. While in (b), we plot the average empirical arrival rates of the weekdays and weekends.

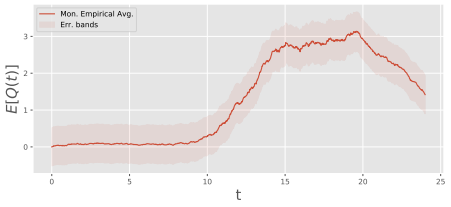



(a)

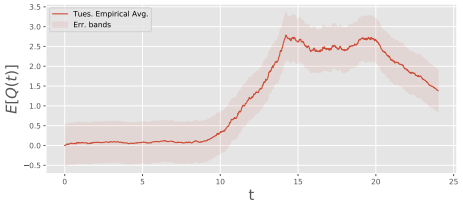



(b)

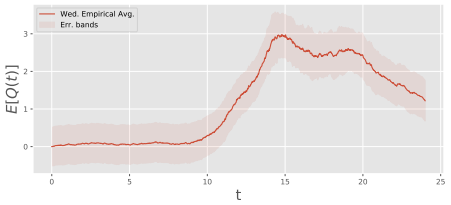
Figure A.7: Plot of different empirical expected queue length. In (a), we plot the empirical expected queue length by each day of the week. In (b), we plot the average empirical expected queue lengths of the weekdays and weekends.



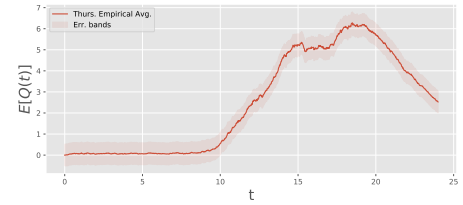

(a) Monday



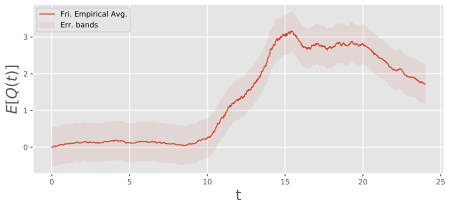

(b) Tuesday



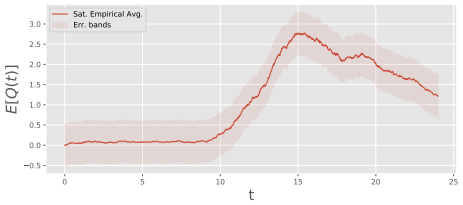

(c) Wednesday



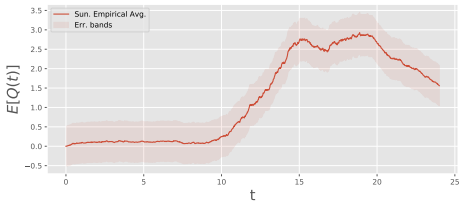

(d) Thursday




(e) Friday




(f) Saturday




(g) Sunday

Figure A.8: Plot of different empirical expected queue length with error bands for each day of the week.

## A.2 The Steady State Queue Analysis

### Background

We first define a few useful lemmas before we give the steady-state queue analysis results.

**Lemma A.2.1.** Let $X$ be an exponential random variable, then $X$ satisfies the memoryless property

$$\mathbb{P}\{X \geq s+t | X \geq s\} = \mathbb{P}\{X \geq t\} \tag{A.2.1}$$

for all positive $x$ and $y$.

*Proof.*

$$\begin{aligned}
\mathbb{P}\{X \geq s+t | X \geq s\} &= \frac{\mathbb{P}\{X \geq s+t, X \geq s\}}{\mathbb{P}\{X \geq s\}} \\
&= \frac{\mathbb{P}\{X \geq s+t\}}{\mathbb{P}\{X \geq s\}} \\
&= \frac{e^{-\lambda(s+t)}}{e^{-\lambda s}} \\
&= e^{-\lambda t} \\
&= \mathbb{P}\{X \geq t\}.
\end{aligned} \tag{A.2.2}$$

□

**Lemma A.2.2.** If $X$ and $Y$ independent exponential random variables with rates $\lambda$ and $\mu$ respectively, then $Z = \min\{X, Y\}$ is an exponential random variable with rate $\lambda + \mu$.

*Proof.*

$$\begin{aligned}\mathbb{P}\{Z > t\} &= \mathbb{P}\{\min(X, Y) > t\} \\ &= \mathbb{P}\{X > t, Y > t\} \\ &= \mathbb{P}\{X > t\} \cdot \mathbb{P}\{Y > t\} \\ &= e^{-\lambda t} \cdot e^{-\mu t} \\ &= e^{-(\lambda+\mu)t}. \end{aligned} \tag{A.2.3}$$

□

**Definition A.2.1** (Markov Property)**.** A continuous time stochastic process $\{X(t) : 0 \geq t < \infty\}$ which has a denumerable state space $\mathcal{S}$, has the Markov property if for every $n \geq 2$, $0 \leq t_1, < t_2, \cdots, < t_n < t_{n+1}$, and any $i_1, i_2, \cdots, i_n, i_{n+1} \in \mathcal{S}$ one has

$$\mathbb{P}\Big\{X(t_{n+1}) = i_{n+1} \,|\, X(t_1) = i_1, \ldots, X(t_n) = i_n\Big\} = \mathbb{P}\Big\{X(t_{n+1}) = i_{n+1} \,|\, X(t_n) = i_n\Big\}. \tag{A.2.4}$$

**Lemma A.2.3.** The following multiple integral identity holds:

$$\int_{0<s_1<\cdots<s_n<t} \cdots \int 1 ds_1 \cdots ds_n = \frac{t^n}{n!} \tag{A.2.5}$$

*Proof.*

$$\begin{aligned} 1 &= \frac{1}{t^n} \cdot \left(\int_0^t 1 ds\right)^n \\ &= \frac{1}{t^n} \cdot \int_0^t \cdots \int_0^t 1 ds_1 \cdots ds_n \\ &= \frac{1}{t^n} \cdot n! \cdot \int_{0<s_1<\cdots<s_n<t} \cdots \int 1 ds_1 \cdots ds_n \end{aligned} \tag{A.2.6}$$

Multiplying both sides of the Equation A.2.6 by $\frac{t^n}{n!}$ gives the identity

$$\frac{t^n}{n!} = \int_{0<s_1<\cdots<s_n<t} \cdots \int 1 ds_1 \cdots ds_n.$$

□

**Lemma A.2.4.** The sum of i.i.d. exponential random variables $Z = X_1 + X_2 + \cdots + X_n$ have $gamma(\lambda, n)$ distributions.

*Proof.*

$$\begin{aligned}
\mathbb{P}\{Z \le t\} &= \mathbb{P}\{X_1 + X_2 + \cdots + X_n \le t\} \\
&= \int_{\substack{0<s_1+\cdots+s_n<t \\ 0\le\min(s_1,\cdots,s_n)}} \cdots \int e^{-s_1} \cdots e^{-s_n} ds_1 \cdots ds_n \\
&= \int_0^t e^{-r} \cdot \left( \int_{\substack{0<s_1+\cdots+s_{n-1}<r \\ 0\le\min(s_1,\ldots,s_{n-1})}} \cdots \int 1 ds_1 \cdots ds_{n-1} \right) dr \quad \text{where } r = s_1 + \cdots + s_n \\
&= \int_0^t e^{-r} \cdot \left( \int_{0<u_1<\cdots<u_{n-1}<r} \cdots \int 1 du_1 \cdots du_{n-1} \right) dr \quad \text{where } \substack{u_k = s_1+\cdots+s_k \\ \text{and } k=1,\ldots,n-1} \\
&= \int_0^t e^{-r} \cdot \frac{r^{n-1}}{(n-1)!} dr \quad \text{(by the multiple integral identity).} \qquad \text{(A.2.7)}
\end{aligned}$$

□

**Theorem A.2.5** (Central Limit)**.** If $X_1, \ldots, X_n$ are i.i.d. random variables with mean $\mathbb{E}[X_1] = \mu$ and $\mathbb{V}[X_1] = \sigma^2$, then

$$\lim_{n\to\infty} \mathbb{P}\left\{ \frac{\sum_{i=1}^n X_i - \mu}{\sqrt{n \cdot \sigma^2}} \le x \right\} = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt. \qquad \text{(A.2.8)}$$

**Theorem A.2.6** (superposition)**.** if $\{N_1(t)|t \geq 0\}$ and $\{N_2(t)|t \geq 0\}$ are Poisson processes with rates $\mu$ and $\nu$ respectively, then there superposition $\{N_1(t)+N_2(t)|t \geq 0\}$ is also a Poisson process with rate $\mu + \nu$

*Proof.*

$$\begin{aligned}\{N_1(t) + N_2(t) = k\} &= \{N_1(t) = j, N_2(t) = k - j \mid 0 \leq j \leq k\} \\ &= \bigcup_{j=0}^{k} \{N_1(t) = j\} \cap \{N_2(t) = k - j\}. \end{aligned} \tag{A.2.9}$$

So we have

$$\begin{aligned}\mathbb{P}\{N_1(t) + N_2(t) = k\} &= \mathbb{P}\left(\bigcup_{j=0}^{k} \{N_1(t) = j\} \cap \{N_2(t) = k - j\}\right) \\ &= \sum_{j=0}^{k} \mathbb{P}\{N_1(t) = j\} \cdot \mathbb{P}\{N_2(t) = k - j\} \\ &= \sum_{j=0}^{k} \frac{e^{-\mu t} \cdot (\mu t)^j}{j!} \cdot \frac{e^{-\nu t} \cdot (\nu t)^{k-j}}{(k-j)!} \\ &= \frac{e^{-(\mu+\nu)t}}{k!} \cdot \sum_{j=0}^{k} \frac{k!}{j! \cdot (k-j)!} \cdot \big(\mu t\big)^j \cdot \big(\nu t\big)^{k-j} \\ &= \frac{e^{-(\mu+\nu)t}}{k!} \cdot \sum_{j=0}^{k} \binom{k}{j} \cdot \big(\mu t\big)^j \cdot \big(\nu t\big)^{k-j} \\ &= \frac{e^{-(\mu+\nu)t} \cdot \Big((\mu + \nu)t\Big)^k}{k!}. \end{aligned} \tag{A.2.10}$$

□

## Steady State Analysis

Let $Q_{1/K}$ denote the queue length of the $M_{\lambda(t)}/M_{\mu(t)}/1/k$ queueing system in steady state. And let $Q_{1/\infty}$ denote the queue length of the $M_\lambda/M_\mu/1/\infty$ queueing system in steady state. Moreover define $\rho = \frac{\lambda}{\mu}$.

**Lemma A.2.7.** The steady state, number of customers in an $M_\lambda/M_\mu/1/k$ queue is distributed as a truncated Geometric$(\rho, k)$ random variable. Setting $\rho = \frac{\lambda}{\mu}$, we then have

$$\lim_{t\to\infty} \mathbb{P}\left\{Q_{1/k}(t) = n\right\} = \begin{cases} \frac{(1-\rho)\cdot\rho^n}{1-\rho^{k+1}} & \text{if } \rho < 1 \\ \frac{1}{k+1} & \text{if } \rho = 1 \\ 0 & \text{if } \rho > 1. \end{cases} \tag{A.2.11}$$

*Proof.*

$$\begin{aligned} \mathbb{P}\left\{Q_{1/k} = n\right\} &= \mathbb{P}\left\{Q_{1/\infty} = n | Q_{1/\infty} \leq k\right\} \\ &= \frac{\mathbb{P}\left\{Q_{1/\infty} = n, Q_{1/\infty} \leq k\right\}}{\mathbb{P}\left\{Q_{1/\infty} \leq k\right\}} \\ &= \frac{\mathbb{P}\left\{Q_{1/\infty} = n\right\}}{\mathbb{P}\left\{Q_{1/\infty} \leq k\right\}} \\ &= \frac{(1-\rho)\cdot\rho^n}{\sum_{\ell=0}^{k}(1-\rho)\cdot\rho^\ell} \\ &= \frac{\rho^n}{\sum_{\ell=0}^{k}\rho^\ell} \\ &= \begin{cases} \frac{(1-\rho)\cdot\rho^n}{1-\rho^{k+1}} & \text{if } \rho < 1 \\ \frac{1}{k+1} & \text{if } \rho = 1. \end{cases} \end{aligned} \tag{A.2.12}$$

□

**Lemma A.2.8.** The steady state of $M_\lambda/M_\mu/1/\infty$ and $M_\lambda/M_\mu/1/k$ queues all have a steady state distributions of the form:

$$\pi(m) = \frac{\alpha_m \cdot \rho^m}{G(\rho)} \quad \text{where} \quad G(\rho) = \sum_{\ell=0}^{\infty} \alpha_\ell \cdot \rho^\ell \quad \text{and} \quad \alpha_0 = 1. \tag{A.2.13}$$

*Proof.* For $M_\lambda/M_\mu/1/\infty$ queue, the steady state distribution is given by:

$$\pi(m) = (1-\rho)\rho^m \cdot \mathbf{1}_{\{\rho<1\}}. \tag{A.2.14}$$

If we let $\alpha_m = \mathbf{1}_{\{\rho<1\}}$, then the steady state can be rewritten as follows:

$$\begin{aligned}
\pi(m) &= (1-\rho)\rho^m \cdot \mathbf{1}_{\{\rho<1\}} \\
&= \frac{\rho^m \cdot \mathbf{1}_{\{\rho<1\}}}{\frac{1}{(1-\rho)}} \\
&= \frac{\rho^m \cdot \mathbf{1}_{\{\rho<1\}}}{\sum_{k=0}^{\infty} \mathbf{1}_{\{\rho<1\}} \cdot \rho^k} \\
&= \frac{\alpha_m \cdot \rho^m}{G(\rho)}.
\end{aligned} \tag{A.2.15}$$

Similarly, for $M_\lambda/M_\mu/1/k$ queue, the steady state distribution is given by:

$$\pi(m) = \frac{(1-\rho)\cdot\rho^m \cdot \mathbf{1}_{\{0\le m\le k\}}}{1-\rho^{k+1}}. \tag{A.2.16}$$

If we let $\alpha_m = \mathbf{1}_{\{0\le m\le k\}}$, then the steady state can be rewritten as follows:

$$\begin{aligned}
\pi(m) &= \frac{(1-\rho)\cdot\rho^m \cdot \mathbf{1}_{\{0\le m\le k\}}}{1-\rho^{k+1}} \\
&= \frac{\rho^m \cdot \mathbf{1}_{\{0\le m\le k\}}}{\sum_{\ell=0}^{k} \rho^\ell} \\
&= \frac{\rho^m \cdot \mathbf{1}_{\{0\le m\le k\}}}{\sum_{\ell=0}^{\infty} \mathbf{1}_{\{0\le m\le k\}} \cdot \rho^\ell} \\
&= \frac{\alpha_m \cdot \rho^m}{G(\rho)}.
\end{aligned} \tag{A.2.17}$$

□

**Lemma A.2.9.** For the $M_\lambda/M_\mu/1/k$ and $M_\lambda/M_\mu/1/\infty$ queueing system, the steady state queue length denoted by $Q$, we always have

$$\mathbb{E}[Q] = \mathcal{E}G(\rho),\ \mathbb{P}\{Q=0\} = \frac{1}{G(\rho)},\ \text{and}\ \mathrm{Var}[Q] = \rho\frac{d}{d\rho}\mathbb{E}[Q]. \tag{A.2.18}$$

where we define the elasticity $\mathcal{E}f(x) = \frac{xf'(x)}{f(x)}$ for a given differentiable, strictly positive function $f$.

*Proof.*

$$\begin{aligned}
\mathbb{E}[Q] &= \sum_{m=0}^{\infty} m \cdot \mathbb{P}\{Q=m\} \\
&= \sum_{m=0}^{\infty} m \cdot \pi(m) \\
&= \sum_{m=0}^{\infty} \frac{m \cdot \alpha_m \cdot \rho^m}{G(\rho)} \\
&= \frac{\sum_{m=1}^{\infty} m \cdot \alpha_m \cdot \rho^m}{G(\rho)} \\
&= \frac{\rho \cdot \sum_{m=1}^{\infty} m \cdot \alpha_m \cdot \rho^{m-1}}{G(\rho)} \\
&= \frac{\rho \cdot \frac{d}{d\rho}\left(\sum_{m=1}^{\infty} \alpha_m \cdot \rho^m\right)}{G(\rho)} \\
&= \frac{\rho \cdot G'(\rho)}{G(\rho)} \\
&= \mathcal{E}G(\rho)
\end{aligned} \tag{A.2.19}$$

Since the steady state distribution is given by

$$\mathbb{P}\{Q=m\} = \frac{\alpha_m \cdot \rho^m}{G(\rho)}, \tag{A.2.20}$$

and $\alpha_0 = 1$, we have

$$\begin{aligned}\mathbb{P}\{Q = 0\} &= \frac{\alpha_0 \cdot \rho^0}{G(\rho)} \\ &= \frac{1}{G(\rho)}.\end{aligned} \tag{A.2.21}$$

$$\begin{aligned}\mathbb{E}[Q^2] &= \sum_{m=0}^{\infty} m^2 \cdot \mathbb{P}\{Q = m\} \\ &= \sum_{m=0}^{\infty} m^2 \cdot \pi(m) \\ &= \sum_{m=0}^{\infty} \frac{m^2 \cdot \alpha_m \cdot \rho^m}{G(\rho)} \\ &= \frac{1}{G(\rho)} \sum_{m=0}^{\infty} m^2 \cdot \alpha_m \cdot \rho^m \\ &= \frac{\rho}{G(\rho)} \sum_{m=1}^{\infty} m^2 \cdot \alpha_m \cdot \rho^{m-1} \\ &= \frac{\rho}{G(\rho)} \sum_{m=1}^{\infty} m \cdot \alpha_m \cdot \frac{d}{d\rho}\rho^m \\ &= \frac{\rho}{G(\rho)} \cdot \frac{d}{d\rho}\left(\sum_{m=1}^{\infty} m \cdot \frac{\alpha_m \cdot \rho^m}{G(\rho)} \cdot G(\rho)\right) \\ &= \frac{\rho}{G(\rho)} \cdot \frac{d}{d\rho}\Big(\mathbb{E}[Q] \cdot G(\rho)\Big) \\ &= \frac{\rho}{G(\rho)} \cdot \left(\frac{d}{d\rho}\mathbb{E}[Q] \cdot G(\rho) + \mathbb{E}[Q] \cdot G'(\rho)\right) \\ &= \rho \cdot \frac{d}{d\rho}\mathbb{E}[Q] + \mathbb{E}[Q]\frac{\rho \cdot G'(\rho)}{G(\rho)} \\ &= \rho \cdot \frac{d}{d\rho}\mathbb{E}[Q] + \mathbb{E}[Q]^2\end{aligned} \tag{A.2.22}$$

and

$$\begin{aligned}\mathrm{Var}[Q] &= \mathbb{E}[Q^2] - \mathbb{E}[Q]^2 \\ &= \rho \cdot \frac{d}{d\rho}\mathbb{E}[Q] + \mathbb{E}[Q]^2 - \mathbb{E}[Q]^2 \\ &= \rho \cdot \frac{d}{d\rho}\mathbb{E}[Q]. \end{aligned} \tag{A.2.23}$$

□

**Lemma A.2.10.** The steady state mean and variance for the number of customers in the $M_\lambda/M_\mu/1/k$ queueing system are

$$\mathbb{E}[Q_{1/k}] = \frac{\rho}{1-\rho} - (k+1) \cdot \frac{\rho^{k+1}}{1-\rho^{k+1}} \tag{A.2.24}$$

and

$$\mathrm{Var}[Q_{1/k}] = \frac{\rho}{(1-\rho)^2} - (k+1)^2 \cdot \frac{\rho^{k+1}}{(1-\rho^{k+1})^2}. \tag{A.2.25}$$

Moreover, we also have

$$\mathbb{P}\{Q_{1/k} > 0\} = \frac{\mathbb{E}[Q_{1/k}]}{1 + \mathbb{E}[Q_{1/k-1}]} \tag{A.2.26}$$

and the mean response time for a customers who is successfully admitted into the queue is

$$\mathbb{E}[R_{1/k}] = \frac{1}{\mu} \cdot \left( \frac{1-\rho^{k+1}}{(1-\rho)(1-\rho^k)} - \frac{(k+1)\cdot\rho^k}{1-\rho^k} \right) \tag{A.2.27}$$

*Proof.* Using the result from Lemma A.2.9,

$$\begin{aligned}
\mathbb{E}[Q_{1/k}] &= \mathcal{E}G(\rho) \\
&= \frac{\rho \cdot G'(\rho)}{G(\rho)} \\
&= \frac{\rho \cdot \left(\frac{1-\rho^{k+1}}{1-\rho}\right)'}{\frac{1-\rho^{k+1}}{1-\rho}} \\
&= \rho \cdot \left(\frac{-(1-\rho)(k+1)\rho^k + 1 - \rho^{k+1}}{(1-\rho)^2}\right) \cdot \frac{1-\rho}{1-\rho^{k+1}} \\
&= \rho \cdot \left(\frac{-(1-\rho)(k+1)\rho^k + 1 - \rho^{k+1}}{(1-\rho)(1-\rho^{k+1})}\right) \\
&= \rho \cdot \left(\frac{1-\rho^{k+1}}{(1-\rho)(1-\rho^{k+1})} - (k+1) \cdot \frac{(1-\rho)\rho^k}{(1-\rho)(1-\rho^{k+1})}\right) \\
&= \rho \cdot \left(\frac{1}{1-\rho} - (k+1) \cdot \frac{\rho^k}{1-\rho^{k+1}}\right) \\
&= \frac{\rho}{1-\rho} - (k+1) \cdot \frac{\rho^{k+1}}{1-\rho^{k+1}} \qquad \text{(A.2.28)}
\end{aligned}$$

$$\begin{aligned}
\mathrm{Var}[Q_{1/k}] &= \rho \cdot \frac{d}{d\rho}\mathbb{E}[Q_{1/k}] \\
&= \rho \cdot \frac{d}{d\rho}\left(\frac{\rho}{1-\rho} - (k+1) \cdot \frac{\rho^{k+1}}{1-\rho^{k+1}}\right) \\
&= \rho \cdot \left(\frac{1-\rho+\rho}{(1-\rho)^2} - (k+1) \cdot \frac{(1-\rho^{k+1})(k+1)\rho^k + \rho^{k+1}(k+1)\rho^k}{(1-\rho^{k+1})^2}\right) \\
&= \rho \cdot \left(\frac{1}{(1-\rho)^2} - (k+1)^2 \cdot \frac{\rho^k}{(1-\rho^{k+1})^2}\right) \\
&= \frac{\rho}{(1-\rho)^2} - (k+1)^2 \cdot \frac{\rho^{k+1}}{(1-\rho^{k+1})^2} \qquad \text{(A.2.29)}
\end{aligned}$$

Using the fact that

$$Q_{1/k} = Q_{1/k} \cdot \{Q_{1/k} > 0\} + Q_{1/k} \cdot \{Q_{1/k} = 0\}, \qquad \text{(A.2.30)}$$

and the fact that Markovian queueing models are time reversible

$$\begin{aligned}
\mathbb{E}[Q_{1/k}] &= \mathbb{E}[Q_{1/k} \cdot \{Q_{1/k} > 0\}] + \mathbb{E}[Q_{1/k} \cdot \{Q_{1/k} = 0\}] \\
&= \mathbb{E}[Q_{1/k} | Q_{1/k} > 0] \cdot \mathbb{P}\{Q_{1/k} > 0\} + \mathbb{E}[Q_{1/k} | Q_{1/k} = 0] \cdot \mathbb{P}\{Q_{1/k} = 0\} \\
&= \mathbb{E}[Q_{1/k} | Q_{1/k} > 0] \cdot \mathbb{P}\{Q_{1/k} > 0\} + 0 \cdot \mathbb{P}\{Q_{1/k} = 0\} \\
&= \mathbb{E}[Q_{1/k} | Q_{1/k} \geq 1] \cdot \mathbb{P}\{Q_{1/k} \geq 1\} \\
&= \Big(\mathbb{E}[Q_{1/k} - 1 | Q_{1/k} \geq 1] + 1\Big) \cdot \mathbb{P}\{Q_{1/k} > 0\} \\
&= \Big(\mathbb{E}[Q_{1/k-1}] + 1\Big) \cdot \mathbb{P}\{Q_{1/k} > 0\}
\end{aligned} \tag{A.2.31}$$

therefore, we have

$$\mathbb{P}\{Q_{1/k} > 0\} = \frac{\mathbb{E}[Q_{1/k}]}{1 + \mathbb{E}[Q_{1/k-1}]} \tag{A.2.32}$$

According to Little's law Morse (1958) for steady state queueing, for any queueing system in steady state, we have that the expected number of customers in the system is the same as the effective input rate times the expected response time. In other words we have the following averaged relation:

$$\mathbb{E}[Q_{1/k}] = \alpha \cdot \mathbb{E}[R_{1/k}] \tag{A.2.33}$$

So we have

$$\begin{aligned}
\mathbb{E}[R_{1/k}] &= \frac{\mathbb{E}[Q_{1/k}]}{\alpha} \\
&= \frac{\mathbb{E}[Q_{1/k}]}{\lambda \cdot \mathbb{P}\{Q_{1/k} \leq k\}} \\
&= \frac{\mathbb{E}[Q_{1/k}]}{\lambda \cdot \Big(1 - \mathbb{P}\{Q_{1/k} = k\}\Big)} \\
&= \frac{\mathbb{E}[Q_{1/k}]}{\lambda \cdot \left(\frac{1-\rho^k}{1-\rho^{k+1}}\right)} \\
&= \frac{\mathbb{E}[Q_{1/k}]}{\lambda} \cdot \frac{1-\rho^{k+1}}{1-\rho^k} \\
&= \frac{1}{\lambda} \cdot \left(\frac{\rho}{1-\rho} - (k+1) \cdot \frac{\rho^{k+1}}{1-\rho^{k+1}}\right) \cdot \frac{1-\rho^{k+1}}{1-\rho^k} \\
&= \frac{1}{\lambda} \cdot \left(\frac{\rho \cdot (1-\rho^{k+1})}{(1-\rho)(1-\rho^k)} - \frac{(k+1) \cdot \rho^{k+1}}{1-\rho^k}\right) \\
&= \frac{1}{\mu} \cdot \left(\frac{1-\rho^{k+1}}{(1-\rho)(1-\rho^k)} - \frac{(k+1) \cdot \rho^k}{1-\rho^k}\right). \qquad \text{(A.2.34)}
\end{aligned}$$

□

**Lemma A.2.11.** The steady state, number of customers in an $M/M/1/\infty$ queue is a Geometric($\rho$) random variable. Setting $\rho = \frac{\lambda}{\mu}$, we then have

$$\lim_{t\to\infty} \mathbb{P}\left\{Q_{1/\infty}(t) = m\right\} = \begin{cases} (1-\rho) \cdot \rho^m & \text{if } \rho < 1 \\ 0 & \text{if } \rho \geq 1 \end{cases} \qquad \text{(A.2.35)}$$

**Lemma A.2.12.** The steady state mean and variance for the number of customers in the $M_\lambda/M_\mu/1/\infty$ queueing system are

$$\mathbb{E}\left[Q_{1/\infty}\right] = \frac{\rho}{1-\rho} \qquad \text{(A.2.36)}$$

and

$$\mathbb{V}\left[Q_{1/\infty}\right] = \frac{\rho}{(1-\rho)^2}. \tag{A.2.37}$$

*Proof.*

$$\begin{aligned}
\mathbb{E}\left[Q_{1/\infty}\right] &= \sum_{n=0}^{\infty} n \cdot \mathbb{P}\left\{Q_{1/\infty} = n\right\} \\
&= \sum_{n=0}^{\infty} n \cdot (1-\rho) \cdot \rho^n \\
&= \rho \cdot (1-\rho) \cdot \sum_{n=1}^{\infty} n \cdot \rho^{n-1} \\
&= \rho \cdot (1-\rho) \cdot \sum_{n=0}^{\infty} \frac{d}{d\rho} \rho^n \\
&= \rho \cdot (1-\rho) \cdot \frac{d}{d\rho} \left( \sum_{n=0}^{\infty} \rho^n \right) \\
&= \rho \cdot (1-\rho) \cdot \frac{d}{d\rho} \left( \frac{1}{1-\rho} \right) \\
&= \rho \cdot (1-\rho) \cdot \frac{1}{(1-\rho)^2} \\
&= \frac{\rho}{1-\rho}
\end{aligned} \tag{A.2.38}$$

$$
\begin{aligned}
\mathbb{E}\left[Q_{1/\infty}^2\right] &= \sum_{n=0}^{\infty} n^2 \cdot \mathbb{P}\left\{Q_{1/\infty} = n\right\} \\
&= \sum_{n=0}^{\infty} n^2 \cdot (1-\rho) \cdot \rho^n \\
&= \sum_{n=0}^{\infty} n(n+1) \cdot (1-\rho) \cdot \rho^n - \sum_{n=0}^{\infty} n \cdot (1-\rho) \cdot \rho^n \\
&= \rho \cdot (1-\rho) \cdot \sum_{n=1}^{\infty} n(n+1) \cdot \rho^{n-1} - \frac{\rho}{1-\rho} \\
&= \rho \cdot (1-\rho) \cdot \frac{d}{d\rho}\left(\sum_{n=1}^{\infty} (n+1) \cdot \rho^n\right) - \frac{\rho}{1-\rho} \\
&= \rho \cdot (1-\rho) \cdot \frac{d}{d\rho}\left(\sum_{m=2}^{\infty} m \cdot \rho^{m-1}\right) - \frac{\rho}{1-\rho} \\
&= \rho \cdot (1-\rho) \cdot \frac{d}{d\rho}\left(\sum_{m=1}^{\infty} m \cdot \rho^{m-1} - 1\right) - \frac{\rho}{1-\rho} \\
&= \rho \cdot (1-\rho) \cdot \frac{d}{d\rho}\left(\frac{1}{(1-\rho)^2} - 1\right) - \frac{\rho}{1-\rho} \\
&= \rho \cdot (1-\rho) \cdot \left(\frac{2}{(1-\rho)^3}\right) - \frac{\rho}{1-\rho} \\
&= \frac{2\rho}{(1-\rho)^2} - \frac{\rho}{1-\rho} \qquad \text{(A.2.39)}
\end{aligned}
$$

$$
\begin{aligned}
\mathbb{V}\left[Q_{1/\infty}\right] &= \mathbb{E}\left[Q_{1/\infty}^2\right] - \mathbb{E}\left[Q_{1/\infty}\right]^2 \\
&= \frac{2\rho}{(1-\rho)^2} - \frac{\rho}{1-\rho} - \left(\frac{\rho}{1-\rho}\right)^2 \\
&= \frac{\rho}{(1-\rho)^2}. \qquad \text{(A.2.40)}
\end{aligned}
$$

□

**Lemma A.2.13.** For $M_\lambda/M_\mu/1/\infty$ in steady state, we have the following geometric distribution identity:

$$\mathbb{E}[Q^{(n)}_{1/\infty}] = n! \cdot \left(\frac{\rho}{1-\rho}\right)^n \tag{A.2.41}$$

*Proof.* By the definition of descending factorial, we have

$$\begin{aligned} x^{(n)} &= x(x-1)(x-2)\cdots(x-n+1) \\ &= \prod_{k=0}^{n-1}(x-k) \\ &= \frac{x!}{(x-n)!} \\ &= n!\binom{x}{n}. \end{aligned} \tag{A.2.42}$$

We begin by showing

$$(x+1)^{(n)} = x^{(n)} + nx^{(n-1)} \tag{A.2.43}$$

$$
\begin{aligned}
(x+1)^{(n)} &= n!\binom{x+1}{n} \\
&= \frac{(x+1)!}{(x+1-n)!} \\
&= \frac{(x+1)x!}{(x+1-n)!} \\
&= \frac{(x+1-n+n)x!}{(x+1-n)!} \\
&= \frac{(x+1-n)x! + nx!}{(x+1-n)!} \\
&= \frac{(x+1-n)x!}{(x+1-n)(x-n)!} + \frac{nx!}{(x+1-n)!} \\
&= \frac{x!}{(x-n)!} + \frac{nx!}{(x+1-n)!} \\
&= n!\binom{x}{n} + n\left[(n-1)!\binom{x}{n-1}\right] \\
&= x^{(n)} + nx^{(n-1)}. \qquad \text{(A.2.44)}
\end{aligned}
$$

Hence,

$$
Q_{1/\infty}^{(n)} = \left(\left(Q_{1/\infty} - 1\right)^{(n)} + n\left(Q_{1/\infty} - 1\right)^{(n-1)}\right) \cdot \mathbf{1}_{\{Q_{1/\infty} \geq 1\}} \qquad \text{(A.2.45)}
$$

Taking expectation over both sides, we obtain the following equation:

$$
\begin{aligned}
\mathbb{E}[Q_{1/\infty}^{(n)}] &= \mathbb{E}\left[\left(\left(Q_{1/\infty} - 1\right)^{(n)} + n\left(Q_{1/\infty} - 1\right)^{(n-1)}\right) \cdot \{Q_{1/\infty} \geq 1\}\right] \\
&= \left(\mathbb{E}\Big[\left(Q_{1/\infty} - 1\right)^{(n)} | Q_{1/\infty} \geq 1\Big] + \mathbb{E}\Big[n\left(Q_{1/\infty} - 1\right)^{(n-1)} | Q_{1/\infty} \geq 1\Big]\right) \cdot \mathbb{P}\{Q_{1/\infty} \geq 1\} \\
&= \left(\mathbb{E}\Big[\left(Q_{1/\infty} - 1\right)^{(n)} | Q_{1/\infty} \geq 1\Big] + \mathbb{E}\Big[n\left(Q_{1/\infty} - 1\right)^{(n-1)} | Q_{1/\infty} \geq 1\Big]\right) \cdot \rho \\
&= \left(\mathbb{E}\Big[Q_{1/\infty}^{(n)}\Big] + n\mathbb{E}\Big[Q_{1/\infty}^{(n-1)}\Big]\right) \cdot \rho \qquad \text{(A.2.46)}
\end{aligned}
$$

which implies that

$$\mathbb{E}[Q_{1/\infty}^{(n)}] = \left(n\frac{\rho}{1-\rho}\right) \cdot \mathbb{E}[Q_{1/\infty}^{(n-1)}]. \tag{A.2.47}$$

In Equation A.2.46, we use the following memoryless property of the geometric distribution:

$$\begin{aligned}\mathbb{P}\Big\{(Q_{1/\infty}-1)^+ = n | Q_{1/\infty} \geq 1\Big\} &= \mathbb{P}\{Q_{1/\infty} - 1 = n | Q_{1/\infty} \geq 1\} \\ &= \mathbb{P}\{Q_{1/\infty} = n\}\end{aligned} \tag{A.2.48}$$

hence, we can write

$$\begin{aligned}\mathbb{E}\Big[\big(Q_{1/\infty}-1\big)^{(n)} | Q_{1/\infty} \geq 1\Big] &= \sum_{k=0}^{\infty} k^{(n)} \cdot \mathbb{P}\Big\{Q_{1/\infty} - 1 = k | Q_{1/\infty} \geq 1\Big\} \\ &= \sum_{k=0}^{\infty} k^{(n)} \cdot \mathbb{P}\Big\{Q_{1/\infty} = k\Big\} \\ &= \mathbb{E}\Big[Q_{1/\infty}^{(n)}\Big]\end{aligned} \tag{A.2.49}$$

Finally, using recursion and the fact that in steady state $Q_{1/\infty}$ has geometric distribution with rate $\rho$, we have

$$\begin{aligned}\mathbb{E}[Q_{1/\infty}^{(1)}] &= \mathbb{E}[Q_{1/\infty}] \\ &= \frac{\rho}{1-\rho},\end{aligned} \tag{A.2.50}$$

and

$$\begin{aligned}
\mathbb{E}[Q_{1/\infty}^{(n)}] &= \left(n\frac{\rho}{1-\rho}\right)\cdot\mathbb{E}[Q_{1/\infty}^{(n-1)}] \\
&= \left(n\frac{\rho}{1-\rho}\right)\cdot\left((n-1)\frac{\rho}{1-\rho}\right)\cdot\mathbb{E}[Q_{1/\infty}^{(n-2)}] \\
&= \left(n\frac{\rho}{1-\rho}\right)\cdot\left((n-1)\frac{\rho}{1-\rho}\right)\cdots\left(2\frac{\rho}{1-\rho}\right)\cdot\mathbb{E}[Q_{1/\infty}^{(1)}] \\
&= n!\cdot\left(\frac{\rho}{1-\rho}\right)^{n-1}\cdot\mathbb{E}[Q_{1/\infty}] \\
&= n!\cdot\left(\frac{\rho}{1-\rho}\right)^{n-1}\cdot\frac{\rho}{1-\rho} \\
&= n!\cdot\left(\frac{\rho}{1-\rho}\right)^{n}. \qquad \text{(A.2.51)}
\end{aligned}$$

□

**Lemma A.2.14.** A closed-form expression for $p_k(t,m)$, the transient probabilities for the $M_{\lambda(t)}/M_{\mu(t)}/1/k$ queue, exists. This is a solution to the Kolmogorov forward equations and its antiderivative exists.

*Proof.* The transient solution as shown in (Lajos, 1962; Morse, 1958) for the $M_{\lambda(t)}/M_{\mu(t)}/1/k$ queue is as follows:

$$\begin{aligned}
p_m(t,n) \;=\; & \frac{(1-\rho)\cdot\rho^n}{1-\rho^{k+1}} + \frac{2\rho^{\frac{n-m}{2}}}{k+1}\sum_{j=1}^{k}\frac{e^{-(\lambda+\mu)t+2t\sqrt{\lambda\mu}\cos\left(\frac{\pi j}{k+1}\right)}}{1-2\cdot\sqrt{\rho}\cos\left(\frac{\pi j}{k+1}\right)+\rho} \\
& \cdot \left(\sin\left(\frac{mj\pi}{k+1}\right)-\sqrt{\rho}\sin\left(\frac{(m+1)j\pi}{k+1}\right)\right) \\
& \cdot \left(\sin\left(\frac{nj\pi}{k+1}\right)-\sqrt{\rho}\sin\left(\frac{(n+1)j\pi}{k+1}\right)\right) \qquad \text{(A.2.52)}
\end{aligned}$$

for when $\lambda \neq \mu$ and

$$\begin{aligned} p_m(t,n) &= \frac{1}{k+1} + \frac{1}{k+1}\sum_{j=1}^{k} \frac{e^{-(2\lambda)t+2t\lambda\cos\left(\frac{\pi j}{k+1}\right)}}{1-\cos\left(\frac{\pi j}{k+1}\right)} \\ &\cdot \left(\sin\left(\frac{mj\pi}{k+1}\right) - \sin\left(\frac{(m+1)j\pi}{k+1}\right)\right) \\ &\cdot \left(\sin\left(\frac{nj\pi}{k+1}\right) - \sin\left(\frac{(n+1)j\pi}{k+1}\right)\right) \end{aligned} \tag{A.2.53}$$

for when $\lambda = \mu$. This expression satisfies the standard Chapman-Kolmogorov equations

$$\begin{aligned} p_m(t+s,n) &= \mathbb{P}\left(Q(t+s)=n \middle| Q(0)=m\right) \\ &= \sum_{\ell=0}^{N} \mathbb{P}\left(Q(t+s)=n \middle| Q(t)=\ell, Q(0)=m\right) \cdot \mathbb{P}\left(Q(t)=\ell \middle| Q(0)=m\right) \\ &= \sum_{\ell=0}^{N} \mathbb{P}\left(Q(s)=n \middle| Q(t)=\ell\right) \cdot p_m(t,\ell) \\ &= \sum_{\ell=0}^{N} p_m(t,\ell)\cdot p_\ell(s,n) \qquad (t>0,\ s>0). \end{aligned} \tag{A.2.54}$$

Hence $p_m(t,n)$ is a solution to the Kolmogorov forward equations. By integrating $p_m(t,n)$ on the interval $[0,T]$, one obtains a closed-form expression of the integral, i.e.

$$\begin{aligned} \int_0^T p_m(t,n)dt &= \frac{T\cdot(1-\rho)\cdot\rho^n}{1-\rho^{k+1}} \\ &+ \frac{2\rho^{\frac{n-m}{2}}}{k+1}\sum_{j=1}^{k} \frac{e^{-(\lambda+\mu)t+2t\sqrt{\lambda\mu}\cos\left(\frac{\pi j}{k+1}\right)}-1}{\left[1-2\cdot\sqrt{\rho}\cos\left(\frac{\pi j}{k+1}\right)+\rho\right]\cdot\left[2\sqrt{\lambda\mu}\cos\left(\frac{\pi j}{k+1}\right)-(\lambda+\mu)\right]} \\ &\cdot \left(\sin\left(\frac{mj\pi}{k+1}\right) - \sqrt{\rho}\sin\left(\frac{(m+1)j\pi}{k+1}\right)\right) \\ &\cdot \left(\sin\left(\frac{nj\pi}{k+1}\right) - \sqrt{\rho}\sin\left(\frac{(n+1)j\pi}{k+1}\right)\right) \end{aligned} \tag{A.2.55}$$

for when $\lambda \neq \mu$ and

$$\int_0^T p_m(t,n)dt = \frac{T}{k+1} + \frac{1}{k+1}\sum_{j=1}^{k} \frac{e^{-(2\lambda)t+2t\lambda\cos\left(\frac{\pi j}{k+1}\right)} - 1}{\left[1-\cos\left(\frac{\pi j}{k+1}\right)\right]\cdot\left[2\lambda\cos\left(\frac{\pi j}{K+1}\right)-2\lambda\right]} \cdot\left(\sin\left(\frac{mj\pi}{k+1}\right)-\sin\left(\frac{(m+1)j\pi}{k+1}\right)\right) \cdot\left(\sin\left(\frac{nj\pi}{k+1}\right)-\sin\left(\frac{(m+1)j\pi}{k+1}\right)\right) \tag{A.2.56}$$

for when $\lambda = \mu$. □

Next, we derive the functional forward equations for the reflecting queue length process and compute the expected time spent in a state. For convenience, we denote the initial queue length by $Q(0) = Q_0$.The *functional version* of the Kolmogorov forward equations for the $M_{\lambda(t)}/M_{\mu(t)}/1/k$ queue length process for a bike-sharing system has the following form:

$$\begin{aligned}
\dot{E}\left[f(Q)\;\middle|\; Q(0)=Q_0\right] &= \mu(t)\cdot E\left[(f(Q+1)-f(Q))\cdot\{Q<k\}\;\middle|\; Q(0)=Q_0\right] \\
&- \lambda(t)\cdot E\left[(f(Q-1)-f(Q))\cdot\{Q>0\}\;\middle|\; Q(0)=Q_0\right] \\
&= \mu(t)\cdot E_{Q_0}\left[(f(Q+1)-f(Q))\cdot\{Q<k\}\right] \\
&- \lambda(t)\cdot E_{Q_0}\left[(f(Q-1)-f(Q))\cdot\{Q>0\}\right], \qquad \text{(A.2.57)}
\end{aligned}$$

for all appropriate functions $f$. For special cases of $f$ such as the mean and variance, we can then obtain the following set of differential equations:

$$\begin{aligned}
\dot{E}_{Q_0}[Q] &= \mu(t)\cdot E_{Q_0}[\{Q<k\}] - \lambda(t)\cdot E_{Q_0}[\{Q>0\}] \\
\dot{\mathrm{Var}}_{Q_0}[Q] &= \mu(t)\cdot E_{Q_0}[\{Q<k\}] + \lambda(t)\cdot E_{Q_0}[\{Q>0\}] \\
&+ 2\cdot\mu(t)\cdot \mathrm{Cov}_{Q_0}\left[Q,\{Q<k\}\right] - 2\cdot\lambda(t)\mathrm{Cov}_{Q_0}\left[Q,\{Q>0\}\right].
\end{aligned}$$

Recall we define the transient probabilities of the Markov chain by

$$p_m(t,n) \equiv \mathbb{P}\left(Q(t)=n \middle| Q(0)=m\right). \tag{A.2.58}$$

Moreover, the transient probabilities of the non-stationary model solve the following Kolmogorov forward equations:

$$\begin{aligned}
\frac{d}{dt}p_m(t,0) &= \mu(t)\cdot p_m(t,1) - \lambda(t)\cdot p_m(t,0)\\
\frac{d}{dt}p_m(t,\ell) &= \mu(t)\cdot p_m(t,\ell+1) + \lambda(t)\cdot p_m(t,\ell-1) - \big(\lambda(t)+\mu(t)\big)\cdot p_m(t,\ell)\\
&\qquad \forall \ell \in \{1,\ldots,k-1\}\\
\frac{d}{dt}p_m(t,k) &= \lambda(t)\cdot p_m(t,k-1) - \mu(t)\cdot p_m(t,k).
\end{aligned}$$

We now calculate the expected time spent in a state over a time interval $[0,T]$ given that the queue length process started in the state $Q_0$. We will show that this is straight forward if we have an explicit solution to the transient probabilities.

**Theorem A.2.15.** The expected time spent in a state over the time interval $[0,T]$ given that the queue length process started in the state $Q_0$ is given by the following formula

$$\mathbb{E}\left[\int_0^T \left\{Q(t)=n \;\middle|\; Q(0)=m\right\} dt\right] = \int_0^T p_m(t,n)dt. \tag{A.2.59}$$

*Proof.*

$$\begin{aligned}
\mathbb{E}\left[\int_0^T \left\{Q(t)=n \;\middle|\; Q(0)=m\right\} dt\right] &= \int_0^T \mathbb{E}\left[\left\{Q(t)=n \;\middle|\; Q(0)=m\right\}\right] dt\\
&= \int_0^T \mathbb{P}\left(Q(t)=n \;\middle|\; Q(0)=m\right) dt\\
&= \int_0^T p_m(t,n)dt.
\end{aligned} \tag{A.2.60}$$

This completes the proof. □

**Theorem A.2.16.** The expected time spent in a state over the time interval $[0, T]$ given that the queue length process started in the state $Q_0$ is given by the following formula

$$\int_0^T \vec{p}_m(t)dt = \mathcal{A}^{-1}\vec{p}_m(0)\left(e^{\mathcal{A}T} - \mathcal{I}\right). \tag{A.2.61}$$

*Proof.*

$$\begin{aligned}\int_0^T \vec{p}_m(t)dt &= \int_0^T \vec{p}_m(0)e^{\mathcal{A}t}dt \\ &= \mathcal{A}^{-1}\vec{p}_m(0)\left(e^{\mathcal{A}T} - e^{\mathcal{A}0}\right) \\ &= \mathcal{A}^{-1}\vec{p}_m(0)\left(e^{\mathcal{A}T} - \mathcal{I}.\right)\end{aligned} \tag{A.2.62}$$

This completes the proof. □

**Theorem A.2.17.** The variance of the time spent in a state over the time interval $[0, T]$ given that the queue length process started in the state $Q_0$ is given by the following formula

$$\begin{aligned}\mathbb{V}_m\left[\int_0^T \{Q(t) = n\ \}\, dt\right] &\equiv \mathrm{Var}\left[\int_0^T \left\{Q(t) = n \;\middle|\; Q(0) = m\right\} dt\right] \\ &= 2\int_0^T \int_0^t p_m(t-s, n)\cdot p_m(s, n)dsdt - \mathbf{E}^2_{m,n}(T).\end{aligned} \tag{A.2.63}$$

where

$$\begin{aligned}\mathbf{E}_{m,n}(T) &\triangleq \mathbb{E}_m\left[\int_0^T \{Q(t) = n\}\, dt\right] \\ &\equiv \mathbb{E}\left[\int_0^T \left\{Q(t) = n \;\middle|\; Q(0) = m\right\} dt\right]\end{aligned} \tag{A.2.64}$$

*Proof.*

$$\begin{aligned}
& \mathbb{V}_k\left[\int_0^T \{Q(t)=n\ \}\,dt\right] && \text{(A.2.65)}\\
= \;& \mathrm{Cov}\left[\int_0^T \left\{Q(t)=n \;\middle|\; Q(0)=m\right\} dt, \int_0^T \left\{Q(s)=n \;\middle|\; Q(0)=m\right\} ds\right]\\
= \;& \mathbb{E}_m\left[\int_0^T \{Q(t)=n\}\,dt \cdot \int_0^T \{Q(s)=n\}\,ds\right]\\
- \;& \mathbb{E}_m\left[\int_0^T \{Q(t)=n\}\,dt\right] \cdot \mathbb{E}_m\left[\int_0^T \{Q(s)=n\}\,ds\right]\\
= \;& \mathbb{E}\left[\int_0^T \int_0^T \{Q(t)=n\} \cdot \{Q(s)=n\}\,dsdt\right] - \mathbf{E}^2_{m,n}(T)\\
= \;& \mathbb{E}_m\left(\int_0^T \int_0^T \left\{Q(t)=n \;\middle|\; Q(s)=n\right\} \cdot \left\{Q(s)=n\right\}\right) - \mathbf{E}^2_{m,n}(T)\\
= \;& \int_0^T \int_0^T \mathbb{E}_m\left[\{Q(t)=n\}\right] \cdot \mathbb{E}_k\left[\{Q(s)=n\}\right] dsdt - \mathbf{E}^2_{m,n}(T)\\
= \;& 2\int_0^T \int_0^t p_m(t-s,n) \cdot p_m(s,n)dsdt - \mathbf{E}^2_{m,n}(T). && \text{(A.2.66)}
\end{aligned}$$

This completes the proof. □

Next, we will define the exponential of a matrix as the tailor series and derive some useful properties of series representation.

**Definition A.2.2.** Define $\exp(t\boldsymbol{A})$ by the power series for all $t \geq 0$

$$\exp(t\boldsymbol{A}) = \sum_{n=0}^{\infty} \frac{t^n}{n!}\boldsymbol{A}^n. \tag{A.2.67}$$

For $N$ large enough, $\exp(t\boldsymbol{A})$ can be approximated as follows:

$$\begin{aligned}
\exp(t\boldsymbol{A}) &= \sum_{n=0}^{\infty} \frac{t^n}{n!}\boldsymbol{A}^n\\
&= \lim_{n\to\infty}\left[\boldsymbol{I} + \frac{\boldsymbol{A}t}{n}\right]^n\\
&\approx \left[\boldsymbol{I} + \frac{\boldsymbol{A}t}{N}\right]^N. && \text{(A.2.68)}
\end{aligned}$$

**Lemma A.2.18.** The power series representation has the following properties:

$$\exp((s+t)\boldsymbol{A}) = \exp(s\boldsymbol{A}) \cdot \exp(t\boldsymbol{A})$$

and

$$\frac{d}{dt}\exp(t\boldsymbol{A}) = \boldsymbol{A}\exp(t\boldsymbol{A}).$$

*Proof.*

$$\begin{aligned}
\exp((s+t)\boldsymbol{A}) &= \sum_{n=0}^{\infty} \frac{(s+t)^n}{n!}\boldsymbol{A}^n \\
&= \sum_{n=0}^{\infty} \frac{(s\boldsymbol{A}+t\boldsymbol{A})^n}{n!} \\
&= \sum_{n=0}^{\infty} \frac{1}{n!}\sum_{k=0}^{n} \binom{n}{k}(t\boldsymbol{A})^{n-k}(s\boldsymbol{A})^k \\
&= \sum_{n=0}^{\infty} \frac{1}{n!}\sum_{k=0}^{n} \frac{n!}{k!(n-k)!}(t\boldsymbol{A})^{n-k}(s\boldsymbol{A})^k \\
&= \sum_{n=0}^{\infty}\sum_{k=0}^{n} \frac{s^k}{k!}\boldsymbol{A}^k\frac{t^{n-k}}{(n-k)!}\boldsymbol{A}^{n-k} \\
&= \sum_{k=0}^{\infty}\sum_{n=k}^{\infty} \frac{s^k}{k!}\boldsymbol{A}^k\frac{t^{n-k}}{(n-k)!}\boldsymbol{A}^{n-k} \\
&= \sum_{k=0}^{\infty} \frac{s^k}{k!}\boldsymbol{A}^k\sum_{n=k}^{\infty}\frac{t^{n-k}}{(n-k)!}\boldsymbol{A}^{n-k} \\
&= \exp(s\boldsymbol{A}) \cdot \exp(t\boldsymbol{A}). \qquad \text{(A.2.69)}
\end{aligned}$$

$$
\begin{aligned}
\frac{d}{dt}\exp(t\boldsymbol{A}) &= \frac{d}{dt}\sum_{n=0}^{\infty}\frac{t^n}{n!}\boldsymbol{A}^n \\
&= \sum_{n=0}^{\infty}\frac{d}{dt}\frac{t^n}{n!}\boldsymbol{A}^n \\
&= \sum_{n=1}^{\infty}\frac{t^{n-1}}{(n-1)!}\boldsymbol{A}^n \\
&= \boldsymbol{A}\cdot\sum_{n=1}^{\infty}\frac{t^{n-1}}{(n-1)!}\boldsymbol{A}^{n-1} \\
&= \boldsymbol{A}\cdot\sum_{m=0}^{\infty}\frac{t^m}{m!}\boldsymbol{A}^m \\
&= \boldsymbol{A}\cdot\exp(t\boldsymbol{A}). \qquad \text{(A.2.70)}
\end{aligned}
$$

□

## A.3 Inventory Repositioning Problem: The Routing Subroutine

The routing problen associated to inventory repositioning problem plans to minimize the repositioning cost by planning cost-effective truck routes. This is equivalent to solving a capacitated multi-vehicle routing problem with Pickups and Deliveries (CVRPPD). The CVRPPD is known to be an NP-hard problem. The cost-effective truck routes could be achieved by setting up a large scale multi-vehicle VRPPD. However, for large scale problems, this has high combinatorial complexity. Instead of solving multi-vehicle VRPPD, we will solve a series of single-vehicle VRPPD for computational convenience. To accomplish this, we split the routing problem into two phases: (1) In this phase, all the stations are spatially clustered into different clusters. A cluster is a collection of data items that are similar between them and

dissimilar to data items in other clusters; (2) In the second phase, we solve a single single-vehicle VRPPD within each cluster.

**Spatial clustering of stations:** Spatial clustering is the process of grouping a set of stations into clusters such that stations within a cluster have high similarity but dissimilar to stations in other clusters. We use distance-based cluster analysis to partition different stations. In particular, we will focus on the Euclidean distance and great circle distance metric. The great-circle distance also known as geodesics distance is the shortest distance between two points on the surface of a sphere. In spaces with curvature, straight lines are replaced by geodesics. Geodesics on the sphere are circles on the sphere whose centers coincide with the center of the sphere and are called great circles. Whereas in Euclidean space, the distance between two points is the length of a straight line between them.

**Vehicle Routing within each Cluster:** We now describe a mathematical model for the VRPPD. VRPPD refers to problems where goods are transported from pickup to delivery points. There are two classes: paired and unpaired. The paired subclass considers transportation requests, each associated with an origin and a destination, resulting in paired pickup and delivery points. Whereas in unpaired subclass, an identical good is considered and each unit picked up could be used to fulfill the demand of every delivery customer. The natural subclass we consider is the unpaired and the mixed-integer linear programming (MILP) model formulations below adapted from (Parragh et al., 2008; Kallehauge et al., 2006; Cordeau et al., 2002; Desaulniers et al., 2002).

| Parameters | Meaning |
|---|---|
| $n$ | The number of pickup vertices |
| $\widetilde{n}$ | The number of delivery vertices |
| $P$ | The set of pickup vertices |
| $D$ | The set of delivery vertices |
| $K$ | The set of vehicles |
| $q_i$ | The demand/supply at vertex i: pickup vertices are associated with positive value, delivery vertice with a negative value, excluding the deport node which is always zero. |
| $t_{ij}$ | The travel time from vertex i to vertex j |
| $B_i$ | The load of vehicle at the begining of service at vertex i |
| $Q_i$ | The load of vehicle when leaving vertex i |
| $c_{ij}$ | The cost to traverse arc $(i, j)$ |
| $C^k$ | The capacity of vehicle $k$. In the single vehicle problem, the superscript $k$ is omitted |

Table A.2: The notation table for the vehicle routing problem

Define $P = \{1, 2, \ldots, n\}$ and $D = \{n+1, 2, \ldots, n+\widetilde{n}\}$ the set of pickup vertices and delivery vertices respectively. Then VRPPD can be modeled on a complete graph $G = (V, A)$, where $V = \{0, n+\widetilde{n}+1\} \cup P \cup D$ is the set of all vertices and $A = \{(i, j) | i, j \in V; i \neq j, i \neq n+\widetilde{n}+1, j \neq 0\}$ the set of arcs. Define the decision variables:

$$x_{ij} = \begin{cases} 1 & \text{if arc } (i, j) \in A \text{ is traversed} \\ 0 & \text{otherwise.} \end{cases} \tag{A.3.1}$$

Let c be a positive semi-definited cost matrix, where the $i^{th}$ row and and $i^{th}$ column of the matrix $c[i][j] = c_{ij}$ represent the cost of going from node $i$ to node $j$. So a non-negative cost is associated with each arc $(i, j) \in A$. It is often convenient to

assume the cost matrix satisfies the triangle inequality

$$c_{ij} \leq c_{ik} + c_{kj} \quad \forall i, k, j \in V. \tag{A.3.2}$$

This assumption discourages deviation from direct link between two vertices $i$ and $j$. The single-vehicle VRPPD model is as follows:

$$\text{(S-VRPPD)} \quad \min_{x_{ij}} \quad \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{A.3.3}$$

$$\text{s.t.} \quad \sum_{i:(i,j)\in A} x_{ij} = 1 \quad \forall j \in V\backslash\{0\}, \tag{A.3.4}$$

$$\sum_{j:(i,j)\in A} x_{ij} = 1 \quad \forall i \in V\backslash\{n+\widetilde{n}+1\}, \tag{A.3.5}$$

$$\sum_{(i,j)\in A(S,\bar{S})} x_{ij} \geq 1 \quad \forall S \subseteq V\backslash\{n+\widetilde{n}+1\}, S \neq \emptyset, \tag{A.3.6}$$

$$Q_j \geq (Q_i + q_i)x_{ij} \quad i \in V, j \in V\backslash\{0\}, \tag{A.3.7}$$

$$Q_j \geq (Q_i - q_i)x_{ij} \quad i \in V\backslash\{0\}, j \in V, \tag{A.3.8}$$

$$\max\{0, q_i\} \leq Q_i \leq \min\{C, C + q_i\} \quad \forall i \in V, \tag{A.3.9}$$

$$x_{ij} \in \{0, 1\} \quad \forall (i,j) \in A. \tag{A.3.10}$$

The objective function (A.3.3) minimizes the total routing cost. The constraints (A.3.4) and (A.3.5) impose that exactly one arc enters and leaves each vertex associated with a customer, respectively except the deport. Constraints (A.3.6) ensures route-connectivity and avoides subtours. subtour is also a round tour that returns back to the origin without visiting all the vertices. This constraint ensures there is no subtour by considering all subset of vertices and make sure that there is an arc leaving a vertex in the subset and entering a vertex that is not in that subset. Constraints (A.3.7) and (A.3.8) ensure that the flow conservation is modeled independent of the sign of $q_i$. Here it is implicitly assumed that every unit picked up can be used to

satisfy every delivery customer's demand. Constraints (A.3.9) simply give lower and upper bounds on the loads. Lastly, constraint (A.3.10) are the integrality constraints.

Similarly, for the multi-vehicle pickup and delivery problem formulations, define the decision variables:

$$x_{ij}^k = \begin{cases} 1 & \text{if arc } (i,j) \in A \text{ is traversed by vehicle k} \\ 0 & \text{otherwise} \end{cases} \tag{A.3.11}$$

$$\text{(M-VRPPD)} \quad \min \quad \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ij}^k \tag{A.3.12}$$

$$\text{s.t.} \quad \sum_{k \in K} \sum_{j:(i,j) \in A} x_{ij}^k = 1 \quad \forall i \in P \cup D, \tag{A.3.13}$$

$$\sum_{j:(0,j) \in A} x_{0j}^k = 1 \quad \forall k \in K, \tag{A.3.14}$$

$$\sum_{i:(i,n+\widetilde{n}+1) \in A} x_{ij}^k = 1 \quad \forall k \in K, \tag{A.3.15}$$

$$\sum_{i:(i,j) \in A} x_{ij}^k - \sum_{i:(i,j) \in A} x_{ji}^k = 0 \quad \forall j \in P \cup D, k \in K, \tag{A.3.16}$$

$$(B_j^k - B_i^k - d_i - t_{ij}^k) x_{ij}^k > 0 \quad \forall (i,j) \in A, k \in K, \tag{A.3.17}$$

$$Q_j^k \geq (Q_i^k + q_i) x_{ij}^k \quad i \in V, j \in V \backslash \{0\}, k \in K, \tag{A.3.18}$$

$$Q_j^k \geq (Q_i^k - q_i) x_{ij}^k \quad i \in V \backslash \{0\}, j \in V, k \in K, \tag{A.3.19}$$

$$\max\{0, q_i\} \leq Q_i^k \leq \min\{C^k, C^k + q_i\} \quad \forall i \in V, \tag{A.3.20}$$

$$x_{ij}^k \in \{0,1\} \quad \forall (i,j) \in A. \tag{A.3.21}$$

The objective function (A.3.12) minimizes the total routing cost. The constraints (A.3.13) impose that every vertex has to be served exactly once. The constraints (A.3.14) and (A.3.15) ensure that every vehicle starts at the deport in the begining of the route and every vehicle ends at the deport at the end of the route. Constraint

(A.3.16) provides flow conservation. Constraint (A.3.17) eliminates subtours by introducing time variables and ensures route connectivity. Constraints (A.3.18) and (A.3.19) ensure that the flow conservation is modeled independent of the sign of $q_i$. Constraint (A.3.18), (A.3.19), and (A.3.20) impose capacity restrain; a vehicle capacity is not exceed thoughout it's tours. Lastly, constraint (A.3.21) is the integrality constraint.

### Solution Techniques to The Routing Problem

One solution is to split the routing problem into two phases: clustering and routing phase. In the clustering phase, all the stations are spatially clustered into different clusters. Whereas in the routing phase, we solve a single single-vehicle VRPPD within each cluster.

**Spatial Clustering** Clustering is an unsupervised learning technique used to find structure, in this case, clusters in the unlabeled dataset. We perform spacial clustering of stations using two popular clustering techniques. The first clustering technique is popularly known as the $k$-means clustering algorithm The $k$-means algorithm partitions the given data into $k$ clusters, where $k$ is prespecified by the user (Lloyd, 1957; 1982; MacQueen, 1967). And the second clustering technique is known as the density-based spatial clustering of applications with noise (DBSCAN). DBSCAN is a non-parametric algorithm that works as follows: given a set of points, it groups together points that are closely packed together into a cluster, marking as outliers points that lie alone in low-density regions (Singh and Meshram, 2017; Ester et al., 1996).

Distance metrics play a very important role in the clustering process. The $k$-means clustering algorithm uses the Euclidean distance metric. The Euclidean distance assumes a flat geometry surface and simply takes the straight-line distance between two points represented as (latitude, longitude) in the Euclidean space. For the DB-

SCAN clustering, we use the great circle distance metric. The great-circle distance also known as the orthodromic distance is the shortest distance between two points on the surface of a sphere, measured along the surface of the sphere. Since the bicycle stations are on Earth, It is also useful to try a distance metric that doesn't assume flat geometry. Below are the two distance metrics in two dimensions: The Euclidean distance is given by

$$\mathbf{d}(x_i, x_j) = \sqrt{\left(x_i^{(1)} - x_j^{(1)}\right)^2 + \left(x_i^{(2)} - x_j^{(2)}\right)^2} \tag{A.3.22}$$

and the great circle distance is given by

$$\mathbf{d}(x_i, x_j) = R \cdot \cos^{-1}\left[\cos\left(x_i^{(1)}\right)\cos\left(x_j^{(1)}\right)\cos\left(x_i^{(2)} - x_j^{(2)}\right) + \sin\left(x_i^{(1)}\right)\sin\left(x_j^{(1)}\right)\right] \tag{A.3.23}$$

where $R$ is the radius of the sphere (or equatorial radius).

**In-cluster Routing** A popular algorithm for the vehicle routing problem is the local search heuristic. The local search heuristic algorithms search the space of candidate solutions by applying local changes until a solution deemed optimal is found or a time budget is exhausted Angel (2006). Some of the advantages of Local search heuristics include running as many iterations as the budget allows. Moreover, the algorithm could be stopped at any point and always have a complete solution but not necessarily optimal. However, the downside of local search heuristics is that it is very sensitive to the starting point. If the starting point is very bad, it might take a large number of iterations to get to a good solution. In practice, one approach to remedy this limitation is to start with a greedy solution rather than just a random solution. For a survey on the local search heuristics and other heuristics for the vehicle routing problems see (Angel, 2006; Laporte et al., 2000). Next, we look at clustering results using real data from two bike-sharing systems: CitiBike and DivvyBike.

**DivvyBike Description.** The DivvyBike is Chicagoland's bike share system and the second largest in the United States of America. The DivvyBike launched in June 2013 and has become an essential part of the transportation network in Chicago and Evanston. DivvyBike, like other bike share systems, consists of a fleet of specially designed, sturdy and durable bikes that are locked into a network of docking stations throughout the region. The DivvyBike clients can unlock a bike from one station and returned to any other station in the system. The DivvyBike provides residents and visitors with a convenient, fun and affordable transportation option for commuting and exploring Chicago. Also, data provided by Chicago Divybike, for 585 stations that were active in the month of July 2017 Motivate International (b).

**CitiBike Description.** The CitiBike is the New York City's bike share system, and the largest in the United States of America. The CitiBike launched in May 2013 and has become an essential part of the transportation network in New York. CitiBike, like other bike share systems, consists of a fleet of specially designed, sturdy and durable bikes that are locked into a network of docking stations throughout the city. The CitiBike's clients can unlock a bike from one station and returned to any other station in the system. The CitiBike provides residents and visitors with a convenient, fun and affordable transportation option for commuting and exploring New York City. We use station data provided by NYC Citibike, for 623 stations that were active in the month of July 2017 Motivate International (a).

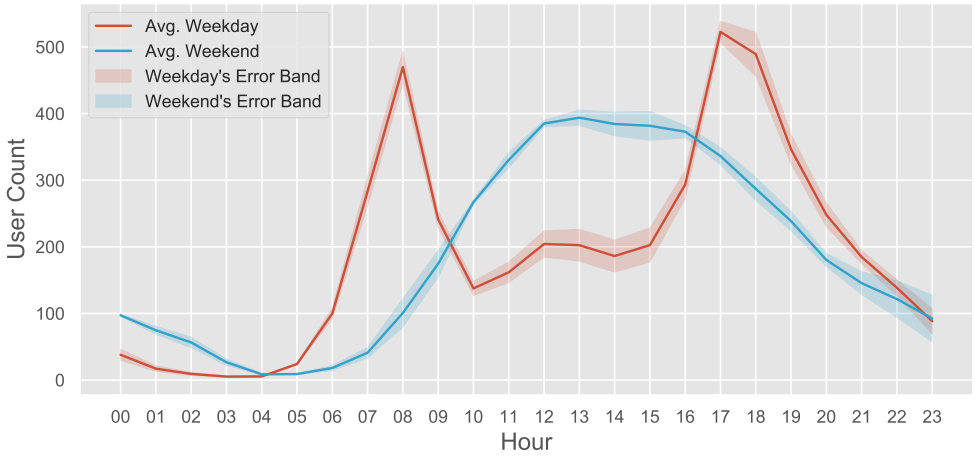
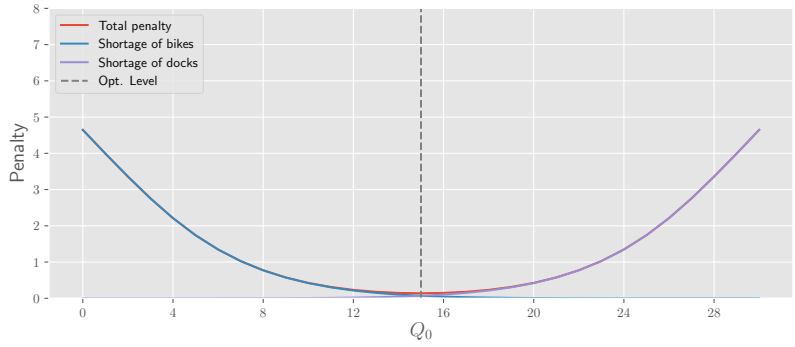
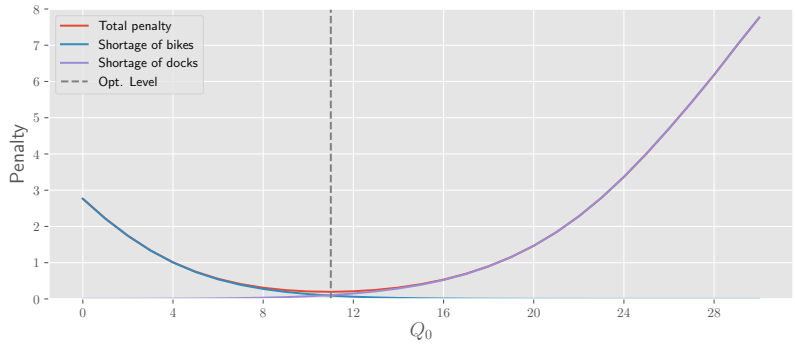



Figure A.9: The user count by hour of the day (CitiBike)

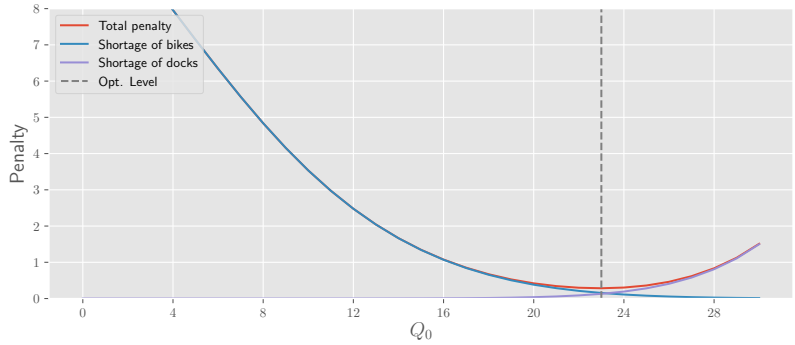
We show the plots for the performance metric objective function $J(m)$, in the non-constant rate setting in Figure A.10 using approximated rate from a realization of an actual bike-sharing station data. We estimated the arrival rates $\lambda(t)$ from the average weekdays' demand rate from the NYC Citibike data. Figure A.9, shows the arrival distribution for both the weekdays and the weekend using the averaged data from the CitiBike Dataset. Since the raw $\lambda(t)$ values are very high, we further normalized them by the average value for all $t \in [0, 24]$ and use one hour increment for the time step. Moreover, the station capacity $k = 30$; also the penalty charge for each customer lost due to a shortage of bicycles and docks is the same $\pi_e = \pi_f = 1$.




(a) Symmetric rates: $\lambda(t) \equiv \mu(t)$

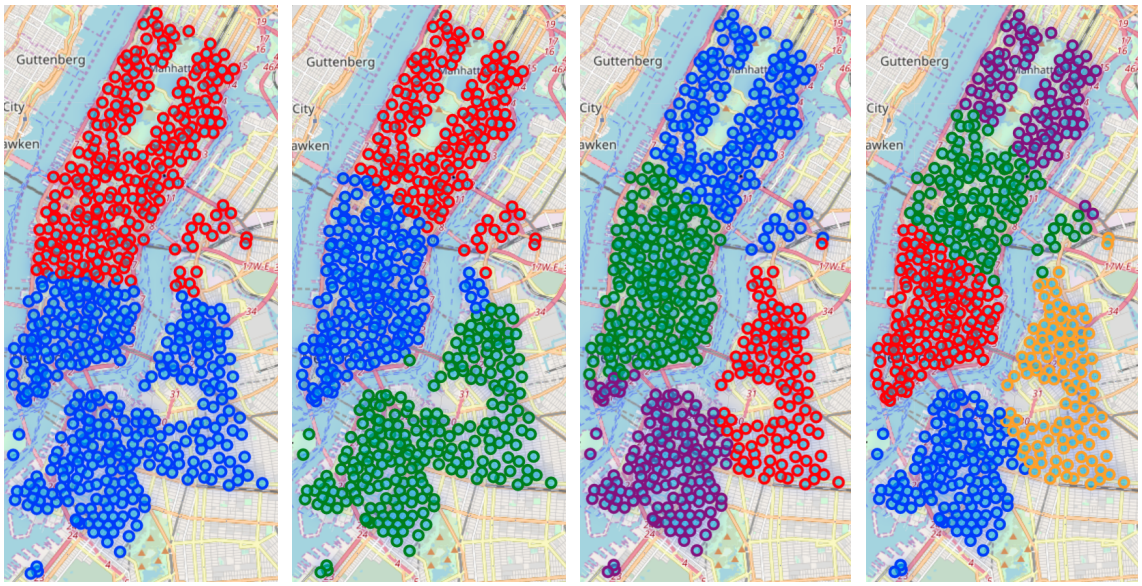



(b) Non-symmetric rates: $\lambda(t) = \mu(t) + 0.2$



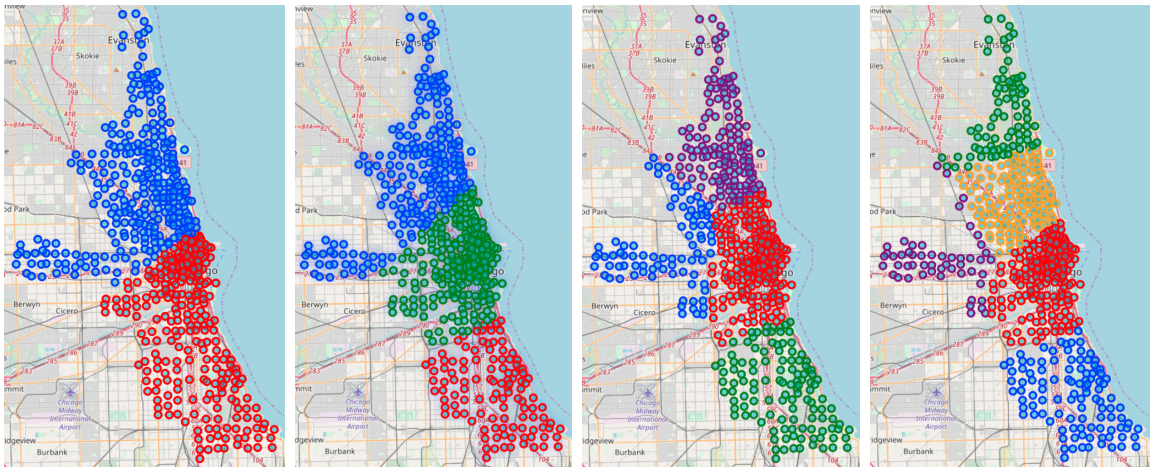

(c) Non-symmetric rates: $\lambda(t) = \mu(t) - 0.4$

Figure A.10: The plot of different penalties incurred due to lost demands at a single station. In (a), the optimal objective value is 0.139, which was achieved at $Q_0^* = 15$. In (b), the optimal objective value is 0.196, which was achieved at $Q_0^* = 11$. In (c), the optimal objective value is 0.284, which was achieved at $Q_0^* = 23$.




(a) (b) (c) (d)

Figure A.11: Spatial clusters of the Citibike (New York City) stations using non-parametric $k$-means cluster with Euclidean distance metric. In (a), there are $k = 2$ clusters. In (b), there are $k = 3$ clusters. In (c), there are $k = 4$ clusters. In (d), there are $k = 5$ clusters.



(a) (b) (c) (d)

Figure A.12: Spatial clusters of the Divvybike (Chicago) stations using non-parametric $k$-means cluster with Euclidean distance metric. In (a), there are $k = 2$ clusters. In (b), there are $k = 3$ clusters. In (c), there are $k = 4$ clusters. In (d), there are $k = 5$ clusters.

Figures A.11 and A.12 show the spatial cluster of stations using the non-parametric $k$-means clustering with Euclidean distance metric with different cluster sizes $k$. The $k$-means does not appear to properly cluster the Citibike stations. For example, when $k = 2$, there are two natural clusters for the stations given by the river line, but $k$-means did not capture this. However, the $k$-means did a great job clustering the Divvybike stations into different clusters. The differences in the performance of the $k$-means for both Citibike stations and Divvybike stations could be explained by the geographical difference between New York and Chicago. Moreover, the Euclidean distance used in $k$-means assumes a flat geometry surface and simply takes the straight-line distance between two points represented as (latitude, longitude) in the Euclidean space.

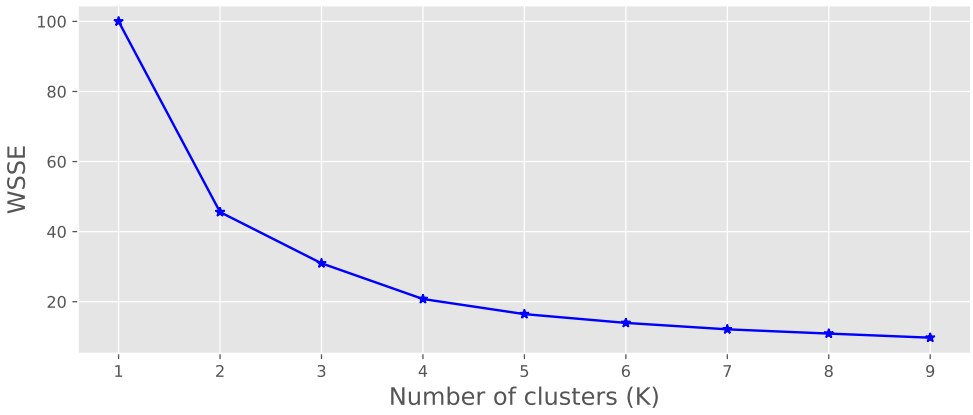



Figure A.13: The Elbow curve for $k$-means clustering Citibike (New York City): calculates the within-cluster sum of squared Error (WSSE) for different values of $k$.

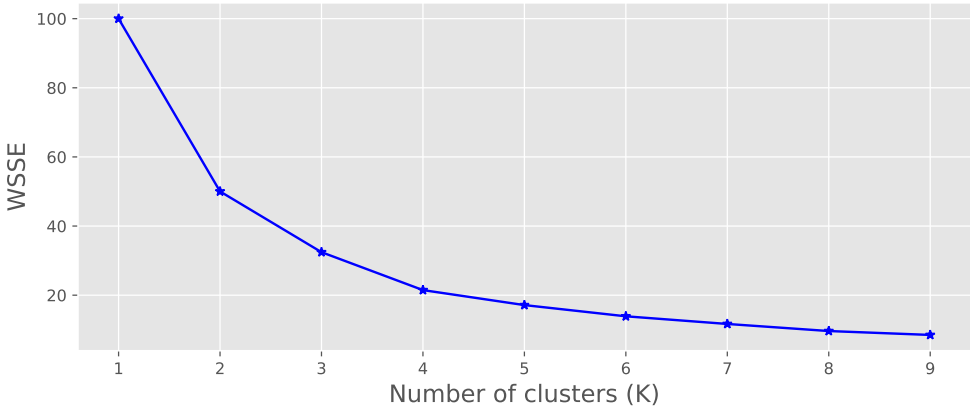
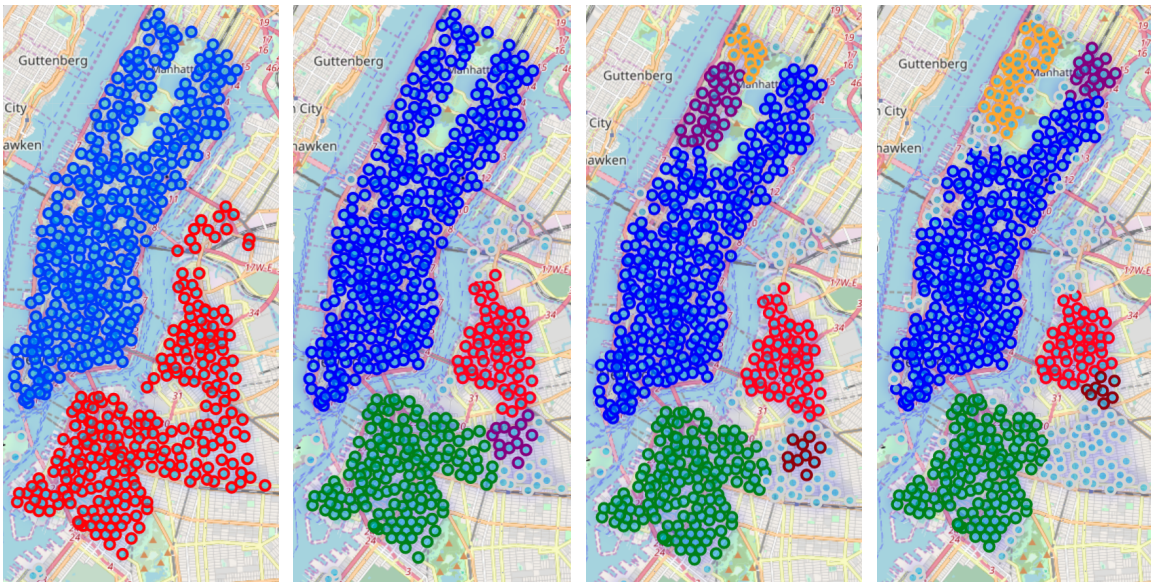



Figure A.14: The Elbow curve for $k$-means clustering Divvybike (Chicago): calculates the within-cluster sum of squared Error (WSSE) for different values of $k$.

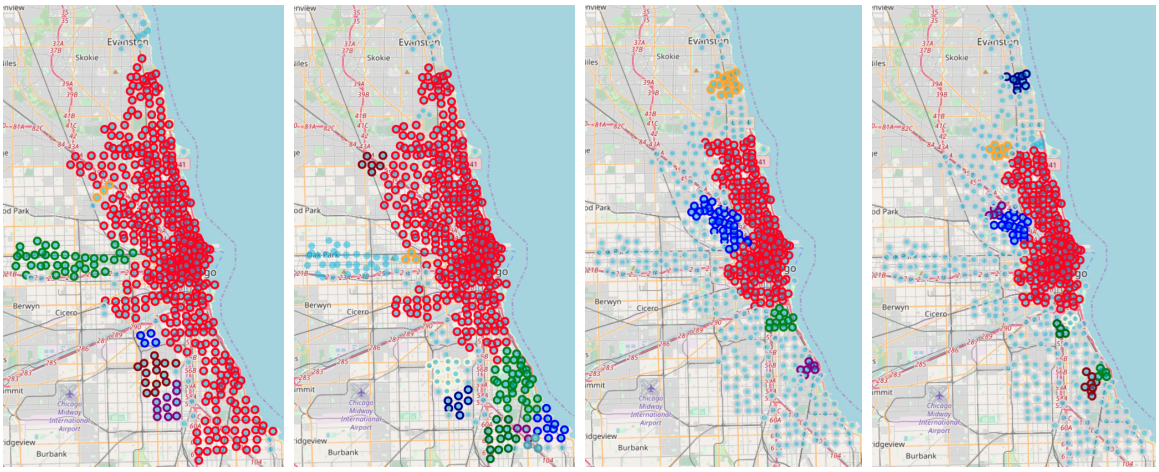
In Figures A.13 and A.14, we show the Elbow method. The Elbow method is one of the most popular methods to determine this optimal value of $k$. This is a fundamental step for the unsupervised $k$-means algorithm to help determine the optimal number of clusters into which the data may be clustered. The Elbow method interprets and validates the consistency of the within-cluster analysis by looking at the percentage of variance explained as a function of the number of clusters (the within-cluster sum

of Squared Error (WSSE) for different values of $k$). Typically one chooses a number of clusters so that adding another cluster doesnt give much better modeling of the data.




(a) (b) (c) (d)

Figure A.15: Spatial clusters of the Citibike (New York City) stations using Density-based spatial clustering of applications with noise (DBSCAN). Along with the great-circle metric distance metric. In (a) all clusters are stations that have at least 5 neighbors in 800 meters. In (b) all clusters are stations that have at least 10 neighbors in 650 meters. In (c) all clusters are stations that have at least 10 neighbors in 600 meters. And in (d) all clusters are stations that have at least 8 neighbors in 500 meters.



(a) (b) (c) (d)

Figure A.16: Spatial clusters of the Divvybike (Chicago) stations using Density-based spatial clustering of applications with noise (DBSCAN). Along with the great-circle metric distance metric. In (a) all clusters are stations that have at least 4 neighbors in 950 meters. In (b) all clusters are stations that have at least 5 neighbors in 1000 meters. In (c) all clusters are stations that have at least 7 neighbors in 800 meters. And in (d) all clusters are stations that have at least 5 neighbors in 650 meters.

Figures A.15 and A.16 show the spatial cluster of stations using the using Density-based spatial clustering of applications with noise (DBSCAN). With the great-circle metric distance metric. The DBSCAN does not appear to properly cluster the Divvybike stations. For example, DBSCAN tends to find one huge cluster in the Divvybike stations dataset. But on the other hand, The DBSCAN did a great job clustering the Citibike stations into different clusters. For example, when $k = 2$, the two natural clusters in the Citibike stations dataset given by the river, were properly captured. Again the differences in the performance of the DBSCAN algorithm for both Citibike stations and Divvybike stations could be explained by the geographical difference between New York and Chicago. Moreover, the great-circle metric distance metric used in the DBSCAN algorithm does not assume a flat geometry surface. It takes the

shortest distance between two points represented as (latitude, longitude) along the surface of the sphere, which makes perfect sense for the bicycle stations application measured along the Earth surface.

# Appendix B

# Appendix to Chapter 3

## B.1 Technical Lemmas

**Lemma B.1.1** (Markov's Inequality)**.** Suppose that $Z$ has a finite mean and that $\mathbb{P}(Z \geq 0) = 1$. Then for any $\epsilon > 0$,

$$\mathbb{P}(Z > \epsilon) \leq \frac{\mathbb{E}(Z)}{\epsilon}$$

*Proof.*

$$\begin{aligned}
\mathbb{E}(Z) &= \int_0^\infty z \, dP(z) \\
&\geq \int_\epsilon^\infty z \, dP(z) \\
&\geq \epsilon \int_\epsilon^\infty dP(z) \\
&\geq \mathbb{P}(z > \epsilon)
\end{aligned}$$

□

**Lemma B.1.2** (Chebyshev's Inequality)**.** Suppose that $Z$ has a finite mean $\mu = \mathbb{E}(Z)$ and variance $\sigma^2 = \mathbb{V}(Z)$. Then for any $\epsilon > 0$,

$$\mathbb{P}\Big\{|Z-\mu|>\epsilon\Big\} \leq \frac{\sigma^2}{\epsilon^2}$$

*Proof.*

$$\begin{aligned}\mathbb{P}\Big\{|Z-\mu|>\epsilon\Big\} &= \mathbb{P}\Big\{|Z-\mu|^2>\epsilon^2\Big\} \\ &\leq \frac{\mathbb{E}(Z-u)^2}{\epsilon^2} \quad \text{(By Markov's Inequality)} \\ &= \frac{\sigma^2}{\epsilon^2}\end{aligned}$$

□

**Lemma B.1.3.** Suppose that a random variable $Z$ has mean $\mu$ such that $a \leq Z \leq b$. Then for any $t$

$$\mathbb{E}(e^{tZ}) \leq e^{\frac{t\mu+t^2(b-a)^2}{8}}$$

*Proof.* Assume $\mu = 0$. Pick $\alpha = \frac{Z-a}{b-a}$ then we can write $Z$ as a convex combination of $a$ and $b$. $Z = \alpha b + (1-\alpha)a$

$$\begin{aligned}e^{tZ} &= e^{t\Big(\alpha b+(1-\alpha)a\Big)} \\ &\leq \alpha e^{tb} + (1-\alpha)e^{ta} \quad \text{(By convexity)} \\ &= \frac{Z-a}{b-a}e^{tb} + \frac{b-Z}{b-a}e^{ta}\end{aligned}$$

Taking expectation of both sides implies

$$\begin{aligned}\mathbb{E}\Big[e^{tZ}\Big] &\leq \mathbb{E}\left[\frac{Z-a}{b-a}e^{tb}+\frac{b-Z}{b-a}e^{ta}\right]\\ &= -\frac{a}{b-a}e^{tb}+\frac{b}{b-a}e^{ta}\\ &= e^{h(w)}\end{aligned}$$

where $h(w) = -\xi w + \log(1-\xi+\xi e^w)$, $w = t(b-a)$ and $\xi = -\frac{a}{b-a}$. There exist $\gamma \in (0,w)$ such that

$$\begin{aligned}g(w) &= g(0) + wg'(0) + \frac{w^2}{2}g''(\gamma)\\ &= \frac{w^2}{2}g''(\gamma) \quad \big(g(0) = g'(0) = 0\big)\\ &\leq \frac{w^2}{8} \quad \big(g''(w) \leq \frac{1}{4},\, \forall w > 0\big)\\ &= \frac{t^2(b-a)^2}{8}\end{aligned}$$

The conclusion follows from taking expectation of the last equation. □

**Theorem B.1.4** (Hoeffding)**.** If $Z_1, Z_2, \ldots, Z_n$ are independent with $\mathbb{P}(a \leq Z_i \leq b) = 1$ and common mean $\mu$ then for any $t > 0$

$$\mathbb{P}\Big\{|\bar{Z}_n - \mu| > \epsilon\Big\} \leq 2e^{\frac{-2n\epsilon^2}{(b-a)^2}}$$

where $\bar{Z}_n = \frac{1}{n}\sum_{i=1}^{n} Z_i$.

*Proof.* Assume $\mathbb{E}(Z_i) = 0$. For any $t > 0$, we have

$$
\begin{aligned}
\mathbb{P}\Big\{\frac{1}{n}\sum_{i=1}^{n} Z_i \geq \epsilon\Big\} &= \mathbb{P}\Big\{\frac{t}{n}\sum_{i=1}^{n} Z_i \geq t\epsilon\Big\} \\
&= \mathbb{P}\Big\{e^{\frac{t}{n}\sum_{i=1}^{n} Z_i} \geq e^{t\epsilon}\Big\} \\
&= e^{-t\epsilon}\mathbb{E}\Big[e^{\frac{t}{n}\sum\limits_{i=1}^{n} Z_i}\Big] \quad \text{(By Markov's Inequality)} \\
&= e^{-t\epsilon}\prod_{i=1}^{n}\mathbb{E}\Big[e^{\frac{t}{n}Z_i}\Big] \\
&\leq e^{-t\epsilon}e^{\frac{t^2}{n^2}\frac{\sum\limits_{i=1}^{n}(b_i-a_i)^2}{8}} \quad \text{(By Lemma B.1.3 )} \\
&= e^{\frac{-2n\epsilon^2}{c}} \quad \Big(\text{setting} \quad t = \frac{4\epsilon n^2}{\sum\limits_{i=1}^{n}(b_i-a_i)^2}\Big)
\end{aligned}
$$

Also, by similar argument, $\mathbb{P}\Big\{-\frac{1}{n}\sum_{i=1}^{n} Z_i \geq \epsilon\Big\} \leq e^{\frac{-2n\epsilon^2}{c}}$ □

**Lemma B.1.5.** Positive linear combination of $n$ convex functions is convex.

*Proof.* Given n convex functions $\Big\{f_i : \Omega \to \mathbb{R}\Big\}_{i=1}^{n}$, then for all x,y in $\Omega$ and $\lambda \in (0,1)$, we have

$$
\sum_{i=1}^{n} f_i\big(\lambda y + (1-\lambda)x\big) \leq \lambda\sum_{i=1}^{n} f_i(x) + (1-\lambda)\sum_{i=1}^{n} f_i(y)
$$

□

**Lemma B.1.6.** Let $J : \Omega \to \mathbb{R}$ be continuously differentiable. $J$ convex implies $J(y) - J(x) \geq \nabla J(x)^\top (y-x)$ for all x,y in $\Omega$.

*Proof.* $J$ convex for all x,y in $\Omega$ and $\lambda \in (0,1)$

$$
\begin{aligned}
J\big(\lambda y + (1-\lambda)x\big) &\leq \lambda J(y) + (1-\lambda)J(x) \\
&\leq \lambda\big[J(y) - J(x)\big] + J(x)
\end{aligned}
$$

Which implies

$$J(x+\lambda(y-x))-J(x) \leq \lambda J(y)+(1-\lambda)J(x)$$
$$\leq \lambda\big[J(y)-J(x)\big]$$

Which implies

$$\frac{J(x+\lambda(y-x))-J(x)}{\lambda} \leq \big[J(y)-J(x)\big]$$

which implies

$$\lim_{\lambda(y-x)\to 0}\Big[\frac{J(x+\lambda(y-x))-J(x)}{\lambda(y-x)}\Big](y-x) \leq \big[J(y)-J(x)\big]$$

Which implies

$$\nabla J(x)^{\top}(y-x) \leq \big[J(y)-J(x)\big].$$

Hence $J(y) \geq J(x)\nabla J(x)^{\top}(y-x)$ □

**Lemma B.1.7.** If the function $J(x)$ has L-Lipschitz continuous gradient then $||\nabla J(x)|| \leq L$ for all x

*Proof.* Since J has lipschitz continuous gradient,

$$||\nabla J(y)-\nabla J(x)|| \leq L||y-x|| \quad \forall\ x,y \in \mathbb{R}^n.$$

This implies

$$||\frac{\nabla J(y)-\nabla J(x)}{y-x}|| \leq L \quad \forall\ x,y \in \mathbb{R}^n,$$

and

$$\lim_{y \to x} ||\frac{\nabla J(y) - \nabla J(x)}{y - x}|| \leq L \quad \forall\ x, y \in \mathbb{R}^n.$$

Hence

$$||\nabla^2 J(x)|| \leq L \quad \forall\ x \in \mathbb{R}^n.$$

□

**Theorem B.1.8.** Gradient descent with fixed step size $\alpha \geq \frac{1}{L}$ satisfies

$$J^\pi(\theta^{(k)}) - J^\pi(\theta^\star) \leq \frac{||\theta^{(k)} - \theta^\star||}{2\alpha k}$$

*Proof.* Since $J$ convex then Let $J : \Omega \to \mathbb{R}$ be continuously differentiable and convex implies $J(y) - J(x) \geq \nabla J(x)^\top (y - x)$ for all x,y in $\Omega$. Since the gradient is Lipschitz.

By tailor expansion, we have

$$
\begin{aligned}
J(\boldsymbol{\theta}^{(k+1)}) &\approx J(\boldsymbol{\theta}^{(k)}) + \nabla J(\boldsymbol{\theta}^{(k)})^\top(\boldsymbol{\theta}^{(k+1)} - \boldsymbol{\theta}^{(k)}) + \frac{\nabla^2 J(\boldsymbol{\theta}^{(k)})}{2}||\boldsymbol{\theta}^{(k+1)} - \boldsymbol{\theta}^{(k)}||^2 \\
&\leq J(\boldsymbol{\theta}^{(k)}) + \nabla J(\boldsymbol{\theta}^{(k)})^\top(\boldsymbol{\theta}^{(k+1)} - \boldsymbol{\theta}^{(k)}) + \frac{L}{2}||\boldsymbol{\theta}^{(k+1)} - \boldsymbol{\theta}^{(k)}||^2 \\
&\leq J(\boldsymbol{\theta}^{(k)}) - \alpha(1 + \frac{\alpha L}{2})||\nabla J(\boldsymbol{\theta}^{(k)})||^2 \quad \text{(plug in } \boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \alpha\nabla J(\boldsymbol{\theta}^{(k)})) \\
&\leq J(\boldsymbol{\theta}^\star) - \nabla J(\boldsymbol{\theta}^{(k)})^\top(\boldsymbol{\theta}^\star - \boldsymbol{\theta}^{(k)}) - \alpha(1 + \frac{\alpha L}{2})||\nabla J(\boldsymbol{\theta}^{(k)})||^2 \quad \text{(J convex)} \\
&\leq J(\boldsymbol{\theta}^\star) + \nabla J(\boldsymbol{\theta}^{(k)})^\top(\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^\star) - \frac{\alpha}{2}||\nabla J(\boldsymbol{\theta}^{(k)})||^2, \quad (1 + \frac{\alpha L}{2}) > \frac{1}{2} \\
&\leq J(\boldsymbol{\theta}^\star) + \nabla J(\boldsymbol{\theta}^{(k)})^\top(\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^\star) - \frac{\alpha}{2}||\nabla J(\boldsymbol{\theta}^{(k)})||^2 + \frac{1}{2\alpha}(||\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^\star||^2 - ||\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^\star||^2) \\
&\leq J(\boldsymbol{\theta}^\star) + \frac{1}{2\alpha}||\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^\star||^2 - \frac{1}{2\alpha}\Big[ - 2\alpha\nabla J(\boldsymbol{\theta}^{(k)})^\top(\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^\star) + \alpha^2||\nabla J(\boldsymbol{\theta}^{(k)})||^2 \\
&\qquad\qquad + \ ||\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^\star||^2\Big] \\
&\leq J(\boldsymbol{\theta}^\star) + \frac{1}{2\alpha}||\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^\star||^2 - \frac{1}{2\alpha}\Big[||-\alpha\nabla J(\boldsymbol{\theta}^{(k)}) + (\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^\star)||^2\Big] \\
&\leq J(\boldsymbol{\theta}^\star) + \frac{1}{2\alpha}||\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^\star||^2 - \frac{1}{2\alpha}\Big[||(\boldsymbol{\theta}^{(k)} - \alpha\nabla J(\boldsymbol{\theta}^{(k)})) - \boldsymbol{\theta}^\star||^2\Big] \\
&\leq J(\boldsymbol{\theta}^\star) + \frac{1}{2\alpha}\Big[||\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^\star||^2 - ||\boldsymbol{\theta}^{(k+1)} - \boldsymbol{\theta}^\star||^2\Big]
\end{aligned}
$$

Summing over iterations

$$
\begin{aligned}
\sum_{i=1}^{k}\Big(J(\boldsymbol{\theta}^{(i)}) - J(\boldsymbol{\theta}^\star)\Big) &\leq \frac{1}{2\alpha}\Big(||x^{(0)} - \boldsymbol{\theta}^\star||^2 - ||\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^\star||^2\Big) \\
&\leq \frac{1}{2\alpha}||\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^\star||^2
\end{aligned}
$$

Since $J(\boldsymbol{\theta}^{(k)})$ is nonincreasing, we have

$$
\begin{aligned}
J(\boldsymbol{\theta}^{(k)}) - J(\boldsymbol{\theta}^{\star}) &\leq \frac{1}{k}\sum_{i=1}^{k}\Big(J(\boldsymbol{\theta}^{(i)}) - J(\boldsymbol{\theta}^{\star})\Big) \\
&\leq \frac{1}{2\alpha k}\Big(||\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^{\star}||^2 - ||\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^{\star}||^2\Big) \\
&\leq \frac{1}{2\alpha k}||\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^{\star}||^2
\end{aligned}
$$

□

Suppose that $\mathcal{H}$ is a finite hypothesis space. Let $\Big\{(x^{(i)}, y^{(i)})\Big\}_{i=1}^{m}$ denote the training sample set, define the training error as $\widehat{R}_n(h) = \frac{1}{n}\sum_{i=1}^{n}\mathbf{1}\{y^{(i)} \neq h(x^{(i)})\}$ and the true classification error as $R(h) = \mathbb{P}(\boldsymbol{Y} \neq h(\boldsymbol{X}))$. Assume that through empirical risk minimization, there exist $\widehat{h} \in \mathcal{H}$ that attains $\widehat{R}(\widehat{h}) = \inf_{h\in\mathcal{H}}\widehat{R}(h)$, since $|\mathcal{Y}| < \infty$. In addition, let $R^{\star} = \inf_{h\in\mathcal{M}} R(h)$ be the Bayes risk, where $\mathcal{M} = \{h \mid h : \mathcal{X} \to \mathcal{Y} \text{ (measurable function)}\}$. And let $R_{\mathcal{H}} = \inf_{h\in\mathcal{H}} R(h)$ be the optimal performance within the class $\mathcal{H}$. Further assume that a hypothesis $\widehat{h}$ minimizes the empirical risk $\widehat{R}(\widehat{h}) \leq \widehat{R}(h),\ \forall h \in \mathcal{H}$. Then the difference between a hypothesis risk and the Bayes risk is as follows:

$$
R(h) - R^{\star} = \underbrace{R(h) - \widehat{R}(h)}_{I} + \underbrace{R(\widehat{h}) - R_{\mathcal{H}}}_{II} + \underbrace{R_{\mathcal{H}} - R^{\star}}_{III},
$$

where

- Optimization error $(I)$: measures how good the optimization that led to the hypothesis $h$ is compared to the empirical risk minimization.
- Estimation error $(II)$: measures how well the empirical risk minimizer $\widehat{h}$ performs compare to the true risk minimizer in the hypothesis class.
- Approximation error $(III)$: measures how well the hypothesis class $\mathcal{H}$ is suited for the problem under consideration.

We are interested in bounding the estimation error, which is often referred to as the generalization bound. The generalization bound quantifies how well the chosen hypothesis generalizes from the observed dataset to the unobserved dataset.

$$\begin{aligned} R(\widehat{h}) - R_{\mathcal{H}} &= R(\widehat{h}) - \inf_{h\in\mathcal{H}} R(h) - \widehat{R}(\widehat{h}) + \widehat{R}(\widehat{h}) \\ &= R(\widehat{h}) - \widehat{R}(\widehat{h}) - \inf_{h\in\mathcal{H}} \Big[R(h) + \widehat{R}(\widehat{h})\Big] \\ &= R(\widehat{h}) - \widehat{R}(\widehat{h}) + \sup_{h\in\mathcal{H}} \Big[\widehat{R}(\widehat{h}) - R(h)\Big] \\ &\leq 2\sup_{h\in\mathcal{H}} \Big[\widehat{R}(h) - R(h)\Big] \end{aligned}$$

$$\begin{aligned} \mathbb{P}\Big\{\sup_{h\in\mathcal{H}} |R(h) - \widehat{R}(h)| > \epsilon\Big\} &= \mathbb{P}\Big\{\bigcup_{h\in\mathcal{H}} |R(h) - \widehat{R}(h)| > \epsilon\Big\} \\ &\leq \sum_{h\in\mathcal{H}} \mathbb{P}\Big\{|R(h) - \widehat{R}(h)| > \epsilon\Big\} \quad \text{(Union Bound)} \\ &\leq 2|\mathcal{H}| \exp(-2m\epsilon^2) \quad \text{(Hoeffding's Inequality)} \end{aligned}$$

$|\mathcal{H}|$ is the size of the hypothesis space. Set $\delta = 2|\mathcal{H}| \exp(-2m\epsilon^2)$, then

$$\begin{aligned} \epsilon &= \sqrt{-\frac{\log(\frac{\delta}{2|\mathcal{H}|})}{2m}} \\ &= \sqrt{\frac{\log(\frac{2|\mathcal{H}|}{\delta})}{2m}} \\ &= \sqrt{\frac{\log(\frac{2}{\delta}) - \log(|\mathcal{H}|)}{2m}} \end{aligned}$$

So the generalization error is bounded by the training error plus a log function of the size of the hypothesis class and the size of the dataset. From this bound, we see that big hypothesis space will lead to big generalization error.

**Lemma B.1.9.** The logistic regression cost function $J(\theta)$ is convex,

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\Big[y^{(i)}\log h_\theta(x^{(i)}) + (1-y^{(i)})\log[1-h_\theta(x^{(i)})]\Big] \tag{B.1.1}$$

*Proof.* The convexity of the logistic cost function can also be justified mathematically. To show that the objective function $J(\theta)$ is a convex function of $\theta$, it suffices to show that $-\log\big(h_\theta(x)\big)$ and $-\log[1-h_\theta(x)]$ are convex. Then conclude with the fact that linear combination of convex functions is convex. We begin by first rewriting $-\log\big(h_\theta(x)\big)$,

$$\begin{aligned} -\log\big(h_\theta(x)\big) &= -\log\frac{1}{1+e^{-\theta^\top x}} \\ &= \log\big(1+e^{-\theta^\top x}\big). \end{aligned}$$

Taking the gradient

$$\begin{aligned} \nabla_\theta\Big[-\log\big(h_\theta(x)\big)\Big] &= \nabla_\theta\Big[\log\big(1+e^{-\theta^\top x}\big)\Big] \\ &= \Big(\frac{-e^{-\theta^\top x}}{1+e^{-\theta^\top x}}\Big)x \\ &= \Big(\frac{1}{1+e^{-\theta^\top x}}-1\Big)x \\ &= (h_\theta(x)-1)x, \end{aligned} \tag{B.1.2}$$

and the Hessian

$$\begin{aligned}
\nabla_\theta^2\Big[-\log\big(h_\theta(x)\big)\Big] &= \nabla_\theta\Big[\nabla_\theta\big[-\log\big(h_\theta(x)\big)\big]\Big] \\
&= \nabla_\theta\Big[(h_\theta(x)-1)x\Big] \\
&= \Big[\frac{e^{-\theta^\top x}}{(1+e^{-\theta^\top x})^2}\Big]xx^\top \\
&= \Big[\frac{1+e^{-\theta^\top x}-1}{(1+e^{-\theta^\top x})^2}\Big]xx^\top \\
&= \Big[\frac{1+e^{-\theta^\top x}}{(1+e^{-\theta^\top x})^2}-\frac{1}{(1+e^{-\theta^\top x})^2}\Big]xx^\top \\
&= \Big[\frac{1}{(1+e^{-\theta^\top x})}-\frac{1}{(1+e^{-\theta^\top x})^2}\Big]xx^\top \\
&= \Big[\frac{1}{(1+e^{-\theta^\top x})}-\big(\frac{1}{1+e^{-\theta^\top x}}\big)^2\Big]xx^\top \\
&= \Big[h_\theta(x)-\big(h_\theta(x)\big)^2\Big]xx^\top \\
&= h_\theta(x)\Big[1-h_\theta(x)\Big]xx^\top. \qquad \text{(B.1.3)}
\end{aligned}$$

Similarly, we rewrite $-\log[1-h_\theta(x)]$,

$$\begin{aligned}
-\log\big(1-h_\theta(x)\big) &= -\log\Big[1-\frac{1}{1+e^{-\theta^\top x}}\Big] \\
&= -\log\Big[\frac{e^{-\theta^\top x}}{1+e^{-\theta^\top x}}\Big] \\
&= \theta^\top x+\log\big(1+e^{-\theta^\top x}\big).
\end{aligned}$$

So the gradient is given by

$$\begin{aligned}
\nabla_\theta\Big[-\log\big(1-h_\theta(x)\big)\Big] &= \nabla_\theta\Big[\theta^\top x+\log\big(1+e^{-\theta^\top x}\big)\Big] \\
&= x+\nabla_\theta\Big[\log\big(1+e^{-\theta^\top x}\big)\Big] \\
&= x+(h_\theta(x)-1)x \quad \text{(by } B.1.2\text{)},
\end{aligned}$$

and the Hessian

$$\begin{aligned}\nabla_\theta^2\Big[-\log\big(1-h_\theta(x)\big)\Big] &= \nabla_\theta\Big[\nabla_\theta\big[-\log\big(1-h_\theta(x)\big)\big]\Big]\\ &= \nabla_\theta\Big[x+(h_\theta(x)-1)x\Big]\\ &= \nabla_\theta\Big[(h_\theta(x)-1)x\Big]\\ &= h_\theta(x)\Big[1-h_\theta(x)\Big]xx^\top \quad \text{(by } B.1.3\text{)}.\end{aligned} \tag{B.1.4}$$

Next we show that is Hessian is positive semi-definite. For all $y$ we have

$$\begin{aligned}y^\top\nabla_\theta^2\Big[-\log\big(h_\theta(x)\big)\Big]y &= y^\top\Big[h_\theta(x)\big[1-h_\theta(x)\big]xx^\top\Big]y\\ &= h_\theta(x)\big[1-h_\theta(x)\big]y^\top xx^\top y\\ &= h_\theta(x)\big[1-h_\theta(x)\big](y^\top x)^2\\ &\geq 0.\end{aligned} \tag{B.1.5}$$

Clearly equations B.1.3 and B.1.4 are positive semi-definite as shown in equation B.1.5. Therefore result follows by the second-order condition for convexity and the fact that positive linear combination of convex functions is convex (see Lemma B.1.5 in the Appendix). □

## B.2 Overview of Supervised Learning

In this section, we briefly introduce regression and classification which are two major components of supervised machine learning.

**Regression:** We briefly introduce a popular supervised learning algorithm for linear regression. Let $\mathcal{D} = \left\{(x^{(i)}, y^{(i)})\right\}_{i=1}^{m}$ denote the training sample set. Where

$$m : \text{is the number of training samples.}$$
$$x^{(i)} : \text{ is the input variable (feature).}$$
$$y^{(i)} : \text{is the output variable (target).}$$

We call the learning problem a regression problem, when the target variable is continuous. Given the training sample, we wish to design a learning algorithm that takes as an input the training samples and outputs a hypothesis function $h_{\boldsymbol{\theta}}(\cdot)$. In other words we want to learn a function $h_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ such that $h_{\boldsymbol{\theta}}\left(x^{(i)}\right) \approx y^{(i)}$. For example, when the relationship between the feature and the target is assumed to be linear, then the hypothesis function is given by $h_{\boldsymbol{\theta}}(x) = \boldsymbol{\theta}^{\top} x$. Where $\boldsymbol{\theta}$ is the parameter vector of the hypothesis function. We denote the loss by $\mathcal{L}\Big(h_{\boldsymbol{\theta}}\left(x^{(i)}\right), y^{(i)}\Big)$, which measures the cost incurred for incorrect prediction. The total cost for lost over our entire training dataset $\mathcal{D} = \left\{(x^{(i)}, y^{(i)})\right\}_{i=1}^{m}$ is given by the cost function

$$J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}\Big(h_{\boldsymbol{\theta}}\left(x^{(i)}\right), y^{(i)}\Big) \tag{B.2.1}$$

For regression problem it is common to use the mean squared error cost function

$$\mathcal{L}\Big(h_{\boldsymbol{\theta}}\left(x^{(i)}\right), y^{(i)}\Big) = \Big(h_{\boldsymbol{\theta}}\left(x^{(i)}\right) - y^{(i)}\Big)^2 \tag{B.2.2}$$

The goal is to find the value of $\boldsymbol{\theta}^\star \in \boldsymbol{\Theta}$ that minimizes the regression cost function.

$$\begin{aligned}\boldsymbol{\theta}^\star &\in \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\operatorname{argmin}}\, J(\boldsymbol{\theta}) \\ &= \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\operatorname{argmin}}\, \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}\Big(h_{\boldsymbol{\theta}}\left(x^{(i)}\right), y^{(i)}\Big) \\ &= \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\operatorname{argmin}}\, \frac{1}{2m}\sum_{i=1}^{m}\Big(h_{\boldsymbol{\theta}}\left(x^{(i)}\right) - y^{(i)}\Big)^2. \end{aligned} \tag{B.2.3}$$

This problem then become least square problem, where we need to optimally choose the hypothesis parameter $\boldsymbol{\theta} \in \boldsymbol{\Theta}$. The idea is to choose hypothesis parameter $\boldsymbol{\theta}$ such the $h_{\boldsymbol{\theta}}(x)$ is close to $y$ in our training example without either overfitting or underfitting. One way to choose $\boldsymbol{\theta}$ that minimizes the cost function is via gradient descent (GD) as shown in Algorithm 7.

**Algorithm 7:** Steepest Gradient Descent Algorithm for Linear Regression

**Input:** Arbitrarily initialize $\boldsymbol{\theta}^k$ for k=0, step size $\alpha > 0$, integer $T > 0$ and sample $(\boldsymbol{x}, \boldsymbol{y}) \sim \mathcal{D}$

**Output:** Weight vector $\bar{\boldsymbol{\theta}}$ that performs best on the validation set.

**for** $k = 1, 2, \ldots, T$ **do**

1 Update the weight: $\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \alpha\nabla J(\boldsymbol{\theta}^{(k)})$

2 Set $k = k + 1$ and return to step 1.

**end**

3 $\bar{\boldsymbol{\theta}} = \sum_{k=1}^{T} \boldsymbol{\theta}^{(k)}$

It is important to note that Algorithm 7 automatically takes smaller steps as it approaches local minimum. In practice, GD algorithm requires hyperparameter tuning for better performance. Often feature scaling and mean normalization are employed for better performance of GD algorithm. The idea of feature scaling is to make features look similar to each other by appropriately rescaling them. Whereas mean normalization corrects the values by subtracting the mean and normalizing by

the range or the standard deviation. Practically, given large enough dataset, GD tends to work well although it is susceptible to local minimum. The typical complexity of the a standard GD algorithm is $O(km^2)$.

Normal equations method is another common way of choosing the parameter $\boldsymbol{\theta}$ that minimizes the cost function. This method does not require choosing a learning rate. In addition, feature scaling is not necessary. For this method, it is convenient to rewrite cost function in vector notation:

$$\begin{aligned} J(\boldsymbol{\theta}) &= \frac{1}{2m}\sum_{i=1}^{m}\left(h_{\boldsymbol{\theta}}\left(x^{(i)}\right) - y^{(i)}\right)^2 \\ &= \frac{1}{2m}\sum_{i=1}^{m}\left(\boldsymbol{\theta}^\top x^{(i)} - y^{(i)}\right)^2 \\ &= \frac{1}{2m}(y - X^\top\boldsymbol{\theta})^\top(y - X^\top\boldsymbol{\theta}), \end{aligned} \tag{B.2.4}$$

where

$$X = \begin{bmatrix} -\left(x^{(1)}\right)^\top - \\ \vdots \\ -\left(x^{(m)}\right)^\top - \end{bmatrix} \in \mathbb{R}^{m\times n}, \text{ and } y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^m.$$

Then by the first order conditions of optimality, we have that

$$\begin{aligned} \widehat{\boldsymbol{\theta}} &= \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\operatorname{argmin}}\, J(\boldsymbol{\theta}) \\ &= (X^\top X)^{-I}X^\top y. \end{aligned} \tag{B.2.5}$$

The normal equation approach works well for small data set and it's complexity is $O(m^3)$, which is incurred when calculating the inverse $(X^\top X)^{-I}$.

**Classification:** We briefly introduce a popular supervised learning algorithm for classification. Let $\mathcal{D} = \left\{(x^{(i)}, y^{(i)})\right\}_{i=1}^{m}$ denote the training sample set. Where

$$m : \text{is the number of training samples}$$
$$x^{(i)} : \text{ is the input variable (feature)}$$
$$y^{(i)} : \text{is the output variable (target)}$$

We call the learning problem a classification problem, when the target variable, that we trying to predict, can only take small number of discrete values. For simplicity we will start with binary classification problem. Similar to regression, we want to find a mapping $h_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$, such that $h(x)$ is close to $y$. We modify the representation of our regression hypothesis function as follows $h_{\boldsymbol{\theta}}(x) = g(\boldsymbol{\theta}^T x)$ where $g(z) = \frac{1}{1+e^{-z}}$ is the logistic function. It is clear that $h_{\boldsymbol{\theta}}(x) \leq$ is bounded in the unit interval, but we want just the binary values. So we further define the decision boundary. This is how to translate the continuous output of our hypothesis to strictly binary values. We design our classifier to predict:

$$y = \begin{cases} 1 & \text{if } h_{\boldsymbol{\theta}}(x) \geq 0.5 \\ \\ 0 & \text{if } h_{\boldsymbol{\theta}}(x) < 0.5 \end{cases} . \tag{B.2.6}$$

Since $h_{\boldsymbol{\theta}}(x) = g(\boldsymbol{\theta}^T x)$ implies that $h_{\boldsymbol{\theta}}(x) \geq 0.5$ if $\boldsymbol{\theta}^T x \geq 0$. The decision boundary basically separates the area where $y = 0$ from the area where $y = 1$. So we can interpret the output of our hypothesis as a probability $h_{\boldsymbol{\theta}}(x) := \mathbb{P}(y = 1|x; \boldsymbol{\theta})$, estimate of the probability that $y = 1$. Thus the conditional likelihood function for this logistic regression has Bernoulli distribution:

$$
\begin{aligned}
L(\boldsymbol{\theta}) &= \prod_{i=1}^{m} \mathbb{P}\Big\{ Y = y^{(i)} | X = x^{(i)} \Big\} \\
&= \prod_{i=1}^{m} h_{\boldsymbol{\theta}} \left( x^{(i)} \right)^{y^{(i)}} \left[ 1 - h_{\boldsymbol{\theta}} \left( x^{(i)} \right) \right]^{1 - y^{(i)}}. \qquad \text{(B.2.7)}
\end{aligned}
$$

And the log likelihood then becomes:

$$
\begin{aligned}
\ell(\boldsymbol{\theta}) &= \log L(\boldsymbol{\theta}) \\
&= \log \prod_{i=1}^{m} h_{\boldsymbol{\theta}} \left( x^{(i)} \right)^{y^{(i)}} \left[ 1 - h_{\boldsymbol{\theta}} \left( x^{(i)} \right) \right]^{1 - y^{(i)}} \\
&= \sum_{i=1}^{m} y^{(i)} \log h_{\boldsymbol{\theta}} \left( x^{(i)} \right) + (1 - y^{(i)}) \log \left[ 1 - h_{\boldsymbol{\theta}} \left( x^{(i)} \right) \right]. \qquad \text{(B.2.8)}
\end{aligned}
$$

Next we define the logistic regression cost function, also known as the cross entropy loss, from the log likelihood function above:

$$
\begin{aligned}
J(\boldsymbol{\theta}) &= \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}\Big( h_{\boldsymbol{\theta}} \left( x^{(i)} \right), y^{(i)} \Big) \\
&= -\frac{1}{m} \sum_{i=1}^{m} \Big[ y^{(i)} \log h_{\boldsymbol{\theta}} \left( x^{(i)} \right) + (1 - y^{(i)}) \log \left[ 1 - h_{\boldsymbol{\theta}} \left( x^{(i)} \right) \right] \Big], \qquad \text{(B.2.9)}
\end{aligned}
$$

where

$$
\begin{aligned}
\mathcal{L}(h_{\boldsymbol{\theta}}(x), y) &= \begin{cases} -\log \big( h_{\boldsymbol{\theta}}(x) \big) & \text{if } y = 1 \\ -\log[1 - h_{\boldsymbol{\theta}}(x)] & \text{if } y = 0 \end{cases} \\
&= -y \log \big( h_{\boldsymbol{\theta}}(x) \big) - (1 - y) \log \big[ 1 - h_{\boldsymbol{\theta}}(x) \big]. \qquad \text{(B.2.10)}
\end{aligned}
$$

This cost function maintains convexity and penalizes our algorithm for wrong prediction. So that during optimization GD will be guaranteed to converge to the

global optimal due to the presence of convexity. Figure B.1 pictorially shows the convexity of the logistic cost function.

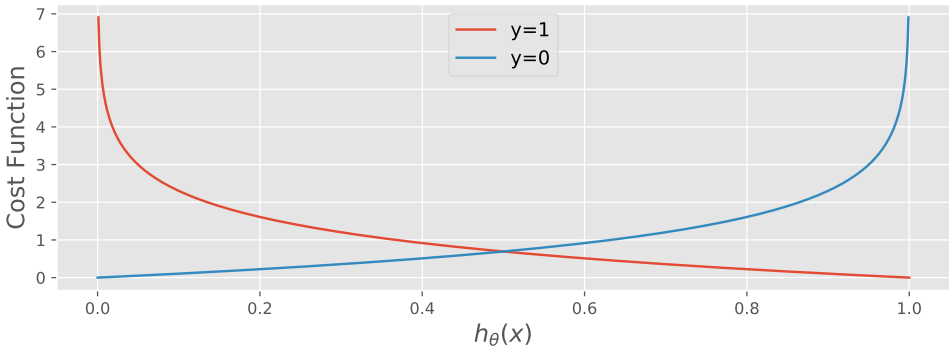



Figure B.1: The Logistic Regression Penalty Function

**Lemma B.2.1.** The logistic regression cost function $J(\boldsymbol{\theta})$ is convex,

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log h_{\boldsymbol{\theta}}\left(x^{(i)}\right) + (1-y^{(i)})\log\left[1-h_{\boldsymbol{\theta}}\left(x^{(i)}\right)\right]\right] \tag{B.2.11}$$

The proof Lemma B.2.1 is given in the Appendix B.1. Next, we need to find the value of $\boldsymbol{\theta}$ that minimizes the cost function.

$$\begin{aligned}
\boldsymbol{\theta}^{\star} &\in \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\operatorname{argmin}}\, J(\boldsymbol{\theta}) \\
&= \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\operatorname{argmin}}\, \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}\left(h_{\boldsymbol{\theta}}\left(x^{(i)}\right), y^{(i)}\right) \\
&= \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\operatorname{argmin}} -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log h_{\boldsymbol{\theta}}\left(x^{(i)}\right) + (1-y^{(i)})\log\left[1-h_{\boldsymbol{\theta}}\left(x^{(i)}\right)\right]\right]
\end{aligned} \tag{B.2.12}$$

Algorithm 8 uses gradient descent to find $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ that minimizes the cost function $J(\boldsymbol{\theta})$.

**Remark B.2.1.** The partial derivative of the cost function with respect to $\boldsymbol{\theta}^k$ is given by $\frac{\partial}{\partial\boldsymbol{\theta}^k}J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}(h_{\boldsymbol{\theta}}\left(x^{(i)}\right) - y^{(i)})x_k^{(i)}$.

**Algorithm 8:** Gradient Descent Algorithm for Logistic Regression

**Input:** Arbitrarily initialize $\boldsymbol{\theta}^k$ for k=0, step size $\alpha > 0$, integer $T > 0$ and sample $(\boldsymbol{x}, \boldsymbol{y}) \sim \mathcal{D}$

**Output:** Weight vector $\bar{\boldsymbol{\theta}}$ that performs best on the validation set.

**for** $k = 1, 2, \ldots, T$ **do**

1 Update weight: $\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \alpha \nabla J(\boldsymbol{\theta}^{(k)})$

2 Set $k = k + 1$ and return to step 1.

**end**

3 $\bar{\boldsymbol{\theta}} = \sum_{k=1}^{T} \boldsymbol{\theta}^{(k)}$

Although Algorithms 7 and 8 look identical, the difference comes from the cost function $J(\boldsymbol{\theta})$. Recall that the regression used $h_{\boldsymbol{\theta}}(x) = \boldsymbol{\theta}^\top x$ hypothesis function, but the classification used $h_{\boldsymbol{\theta}}(x) = \frac{1}{1+e^{-\boldsymbol{\theta}^\top x}}$ hypothesis function. Just as in the regression case, Gradient decent works well in practice for differentiable convex function $J(\boldsymbol{\theta})$. For example, suppose $J(\boldsymbol{\theta})$ is a differentiable convex function with Lipschitz gradient then for fixed step size $\alpha \geq \frac{1}{L}$, we can get the following convergence rate $J^\pi(\boldsymbol{\theta}^{(k)}) - J^\pi(\boldsymbol{\theta}^\star) \leq \frac{||\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}^\star||}{2\alpha k}$ (see Appendix B.1.8).

For multiclass classification problem where $y \in \{1, \ldots, n\}$, Algorithm 9 is used to reduce it to the binary case. In other words, we will divide the multiclass problem into n binary classification problems. Let $\mathcal{S} = \left\{(x^{(i)}, y^{(i)})\right\}_{i=1}^{m}$ denote the training sample set. We split the sample $\mathcal{S}$ into $n$ binary sets $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n$, where $\mathcal{S}_k = \left\{\left(x^{(i)}, 0 \cdot \mathbf{1}_{(y^{(i)} \neq k)}\right)\right\}_{i=1}^{m}$ for each $k \in [n]$. Moreover, for each problem we will predict the probability that $y$ is a member of one of our classes. We train $n$ classifiers,

$h_{\boldsymbol{\theta}}^{(k)} : \mathcal{X} \to \{0, 1\}$ based on $\mathcal{S}_k$ for each $k \in [n]$.

$$\begin{aligned} h_{\boldsymbol{\theta}}^{(1)}(x) &= \mathbb{P}(y = 1|x; \boldsymbol{\theta}) \\ h_{\boldsymbol{\theta}}^{(2)}(x) &= \mathbb{P}(y = 2|x; \boldsymbol{\theta}) \\ &\vdots \\ h_{\boldsymbol{\theta}}^{(n-1)}(x) &= \mathbb{P}(y = n - 1|x; \boldsymbol{\theta}) \\ h_{\boldsymbol{\theta}}^{(n)}(x) &= \mathbb{P}(y = n|x; \boldsymbol{\theta}) \end{aligned} \tag{B.2.13}$$

Thus the rule for the multiclass predictor is given by

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \max_{k \in [n]} \left( h_{\boldsymbol{\theta}}^{(k)}(x) \right) \tag{B.2.14}$$

---

**Algorithm 9:** One-vs-All Classification

---

**Input:** Sample training set $\mathcal{S} = \left\{ (x^{(i)}, y^{(i)}) \right\}_{i=1}^{m}$ and a binary classification algorithm $\mathcal{A}$.

**Output:** Multiclass hypothesis function $h_{\boldsymbol{\theta}}(\boldsymbol{x})$

**for** $k = 1, 2, \ldots, |\mathcal{Y}|$ **do**

1 $\quad \mathcal{S}_k = \left\{ \left( x^{(i)}, 0 \cdot \mathbf{1}_{(y^{(i)} \neq k)} \right) \right\}_{i=1}^{m}$

2 $\quad h_{\boldsymbol{\theta}}^{(k)} = \mathcal{A}(\mathcal{S}_k)$.

**end**

3 $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \max_{k \in [n]} \left( h_{\boldsymbol{\theta}}^{(k)}(x) \right)$

---

At each stage, Algorithm 9 chooses one class and combine the rest into a single second class, resulting to a binary classification problem. .

## Generalization and Overfitting

Generalization of a machine learning algorithm refers to your model's ability to adapt properly to new, previously unseen data, drawn from the same distribution as the one used to create the model. There are two main concerns: underfitting and overfitting. Underfitting usually occurs when the learning model has a high bias. In other words, the hypothesis function maps poorly to the trend of the data set. This problem is sometimes caused by an overly simplified function that uses very few features. So the oversimplification may impact the performance of the model both on training data set and new data sets. On the other hand, overfitting occurs when the model has a high variance. In other words, the hypothesis function fits the training data perfectly but does not generalize well to new data sets. This problem is usually caused by an overly complicated function that uses too many features. Figure B.2 shows the trade-off between bias and variance for learning algorithms.

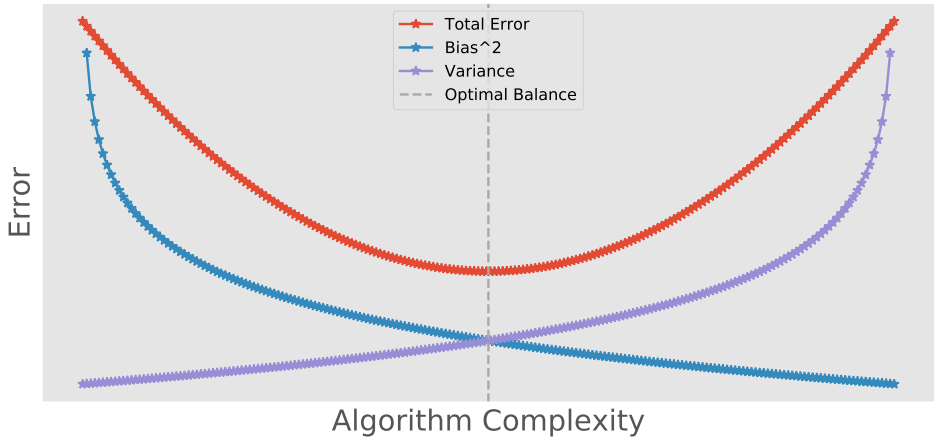



Figure B.2: Bias-variance Trade-off Complexity Illustration

In practice, these fitting problems are often solved via regularization. The idea is to penalize the loss function by adding a complexity term that would give a bigger loss for more complex models. $l_2-$norm and $l_1-$norm are two commonly used penalization

functions. Bellow we give an example of these regularization using the regression objective function.

- $l_2$ Regularization for regression:

$$J(\boldsymbol{\theta}) = \frac{1}{2m}\Big[\sum_{i=1}^{m}\Big(h_{\boldsymbol{\theta}}\left(x^{(i)}\right) - y^{(i)}\Big)^2 + \lambda||\boldsymbol{\theta}||_2^2\Big], \tag{B.2.15}$$

- $l_1$ Regularization for regression:

$$J(\boldsymbol{\theta}) = \frac{1}{2m}\Big[\sum_{i=1}^{m}\Big(h_{\boldsymbol{\theta}}\left(x^{(i)}\right) - y^{(i)}\Big)^2 + \lambda||\boldsymbol{\theta}||_1\Big], \tag{B.2.16}$$

where $\lambda > 0$ is the regularization parameter. The regularization parameter essentially controls the trade off between fitting the data well and keeping the parameters of the model relatively small.

The regularization for the normal equation is also similar. Let's define an $(n+1)$ by $(n+1)$ matrix as

$$\bar{I} = \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}$$

where all the off diagonal elements are zeros. So the optimizer for the normal equation to the regularized regression is $\widehat{\boldsymbol{\theta}} = (X^\top X + \lambda\bar{I})^{-I}X^\top y$. It is easy to see that $(X^\top X + \lambda\bar{I})^{-I}$ always exists. For the proof, it suffices to show that $(X^\top X + \lambda\bar{I})$ is positive

definite. Clearly $X^\top X \succeq 0$, since for all $z \in \mathbb{R}^n$

$$\begin{aligned} z^\top X^\top X z &= ||Xz||^2 \\ &\geq 0 \end{aligned}$$

So we have

$$\begin{aligned} z^\top (X^\top X + \lambda \bar{I}) z &= z^\top X^\top X z + \lambda z^\top z \\ &= ||Xz||^2 + \lambda ||z||^2 \\ &\geq 0 \end{aligned}$$

Thus $(X^\top X + \lambda \bar{I}) \succeq 0$ and $z^\top (X^\top X + \lambda \bar{I}) z = 0$ only if $z = 0$.

For the classification objective function, we have the following.

- $l_2$ Regularization for classification:

$$J(\boldsymbol{\theta}) = -\frac{1}{m} \sum_{i=1}^{m} \left[ y^{(i)} \log h_{\boldsymbol{\theta}} \left( x^{(i)} \right) + (1 - y^{(i)}) \log \left[ 1 - h_{\boldsymbol{\theta}} \left( x^{(i)} \right) \right] \right] + \frac{\lambda}{2m} ||\boldsymbol{\theta}||_2^2, \tag{B.2.17}$$

- $l_1$ Regularization for classification:

$$J(\boldsymbol{\theta}) = -\frac{1}{m} \sum_{i=1}^{m} \left[ y^{(i)} \log h_{\boldsymbol{\theta}} \left( x^{(i)} \right) + (1 - y^{(i)}) \log \left[ 1 - h_{\boldsymbol{\theta}} \left( x^{(i)} \right) \right] \right] + \frac{\lambda}{m} ||\boldsymbol{\theta}||_1. \tag{B.2.18}$$

The regularized cost functions can then be optimized using gradient decent as described previously. It is expected that the new hypothesis function, obtained from optimizing the regularized cost function, would generalize to new data sets. Furthermore, it is common in machine learning to try multiple models and find one that

works best for a particular problem. There is no one model that works best for every problem (see the No-Free-Lunch theorem in Shalev-Shwartz and Ben-David (2014)).

## B.3 The Structural Building Block for Deep Learning

We briefly introduce the fundamental building block of a neural network, which is a single neuron, also known as a perceptron. Figure B.3 depicts the forward propagation of information through a Perceptron (also known as the neuron). Given a set of inputs $\{x_1, \ldots, x_m\}$ with the corresponding weights $\{w_1, \ldots, w_m\}$. The final output of a perceptron $\widehat{y}$ is obtained by multiplying each input with the corresponding weight, sum them to get a single number and then pass the sum through a non-linear activation function. We also have what is called a bias term in this neuron, which you can see here in green. The purpose of the bias term is to allow you to shift your activation function to the left and the right, regardless of your input.

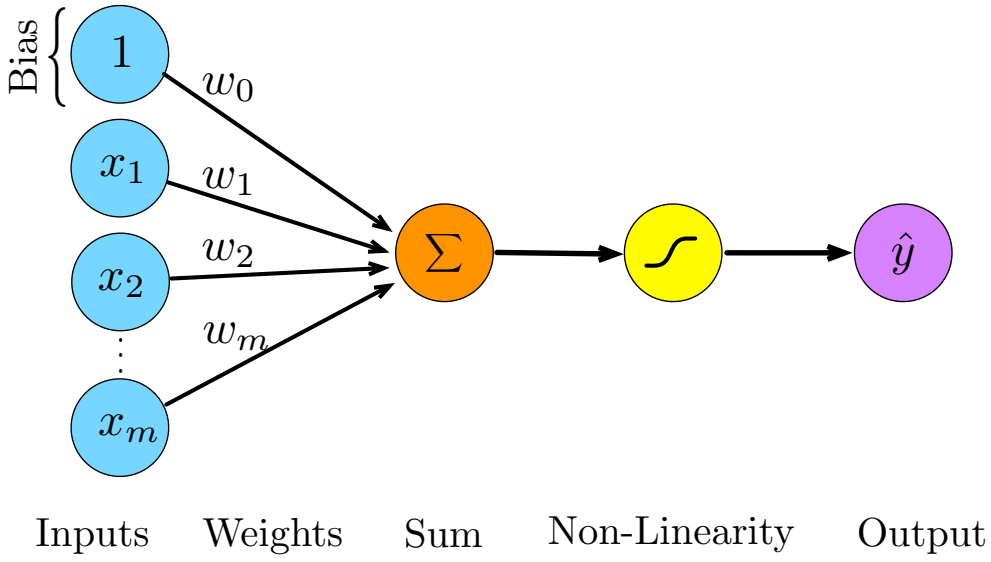



Figure B.3: The forward propagation of information through a neuron.

For convenience, Figure B.4 gives a simplified version of the perceptron by removing the bias and weight labels. This implicitly assumes that every line has a weight associated with it. And Denote $z$ as the dot product (element-wise multiplication of input and weights). So the final output is activation function applied on $z$.

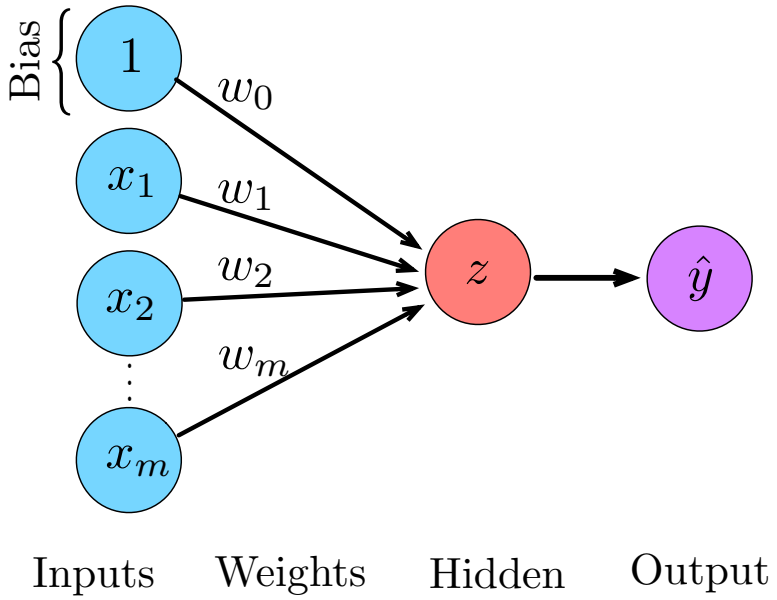



Figure B.4: The simplified perceptron

$$z = w_0 + \sum_{j=1}^{m} x_j w_j = w_0 + \boldsymbol{X}^T \boldsymbol{W} \tag{B.3.1}$$

and

$$\widehat{y} = g(z) = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W}\right) \tag{B.3.2}$$

where:

$$\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \quad \text{and} \quad \boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$$

To define a multi-output neural network, we simply just add another one of these neurons. Now, let's take a look at a single layer Neural Network. We have a single hidden layer that feeds into a two output layer as depicted in Figure B.5. The middle layer is often called the hidden layer because unlike our inputs and outputs, the states of the hidden layer are not directly observable. Since we have a transformation between the input and hidden layer and the hidden layer and the output, each of those two transformations would have their own weight matrices denoted by $W^{(1)}$ and $W^{(2)}$.

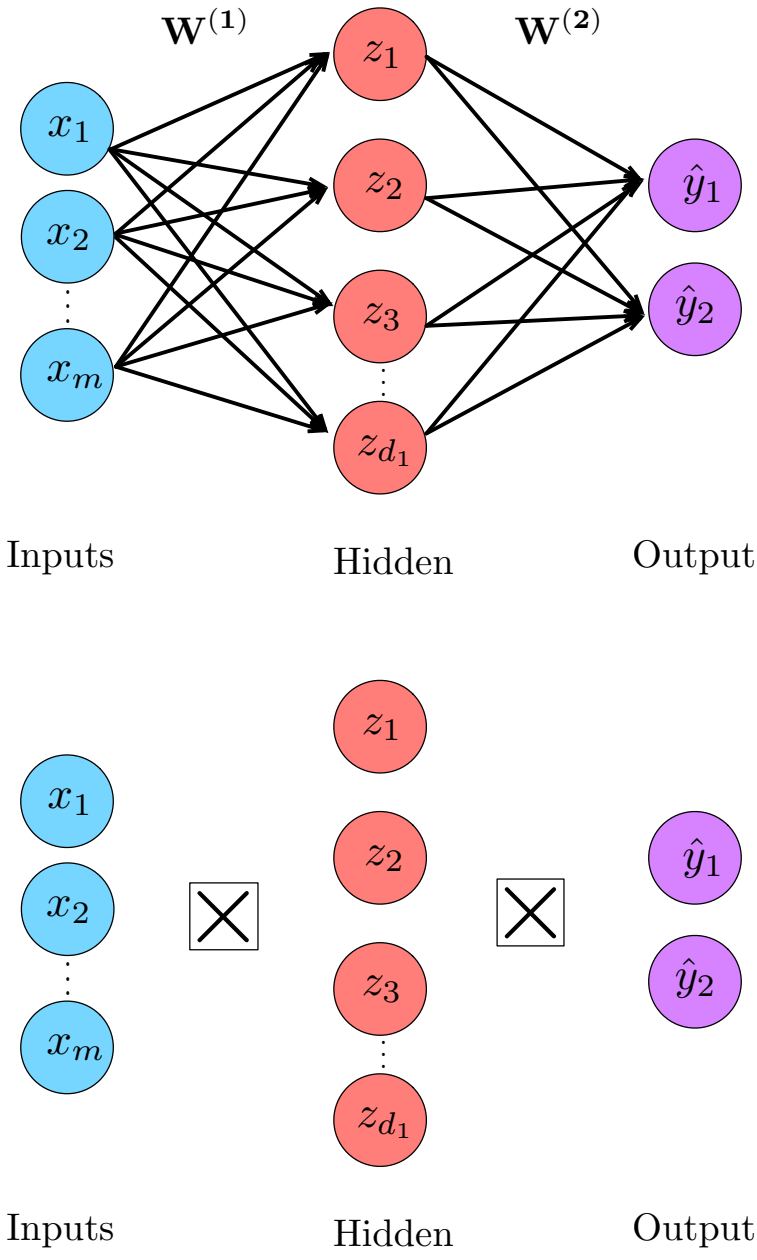



Figure B.5: Two output single layer neural network. In (a), a fully connected densed layer is denoted by forward directed arrows and in (b), a special symbol is used to denote fully connected densed layer.

$$z_i = \boldsymbol{W}_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \boldsymbol{W}_{j,i}^{(1)} \tag{B.3.3}$$

and

$$\widehat{y}_i = g\left(\boldsymbol{W}_{0,i}^{(2)} + \sum_{j=1}^{d_1} g(z_j)\boldsymbol{W}_{j,i}^{(2)}\right) \tag{B.3.4}$$

Lastly, if we want to create a deep Neural network, the idea is basically the same thing except you just keep stacking on more of these hidden layers as shown in Figure B.6.

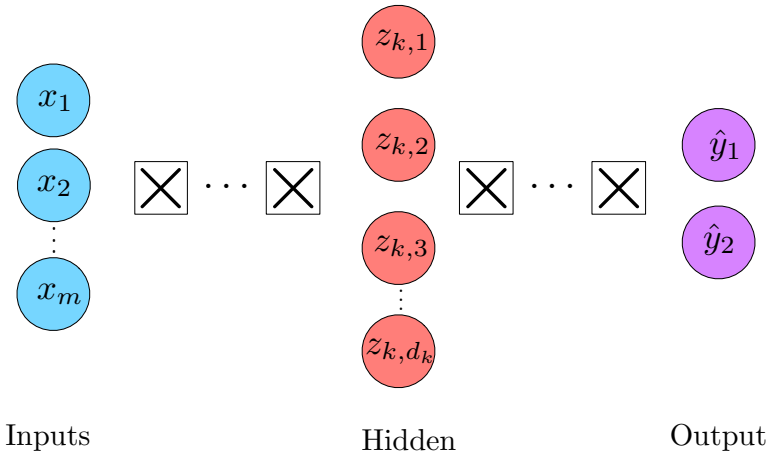



Figure B.6: Two output deep neural network

$$z_{k,i} = \boldsymbol{W}_{0,i}^{(k)} + \sum_{j=1}^{d_{k-1}} g(z_{k-1,j})\boldsymbol{W}_{j,i}^{(k)} \tag{B.3.5}$$

Activation functions are used to determine the firing of neurons in a neural network. Given a linear combination of inputs and weights from the previous layer, the activation function controls how we'll pass that information on to the next layer. An ideal activation function is both nonlinear and differentiable. The nonlinear behavior of an activation function allows the neural network to learn nonlinear relationships in the data; it essentially introduces non-linearities into the network. Differentiability of an activation function is important because it allows us to backpropagate the model's

error during training and the process of optimizing the network weights. Below are some common activation functions.

**Sigmoid (Logistic) Function:** An activation function of the form given in Equation B.3.6 . Its Range is between 0 and 1 and has an Sshape as depicted in Figure B.7.

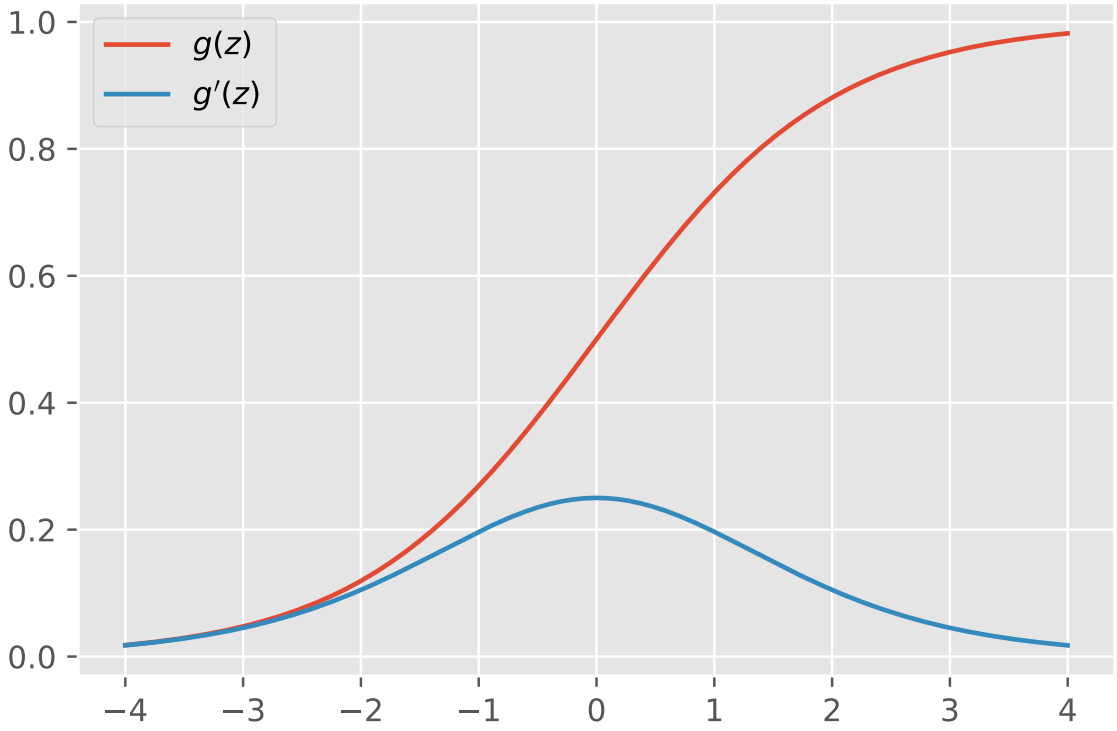



Figure B.7: Sigmoid (Logistic) Function

$$g(z) = \frac{1}{1 + e^{-z}} \tag{B.3.6}$$

and

$$g'(z) = g(z) \cdot (1 - g(z)) \tag{B.3.7}$$

**Hyperbolic Tangent:** An activation function of the form given Equation B.3.8. Now its output is zero centered because its range in between $-1$ to 1 as shown in Figure B.8.

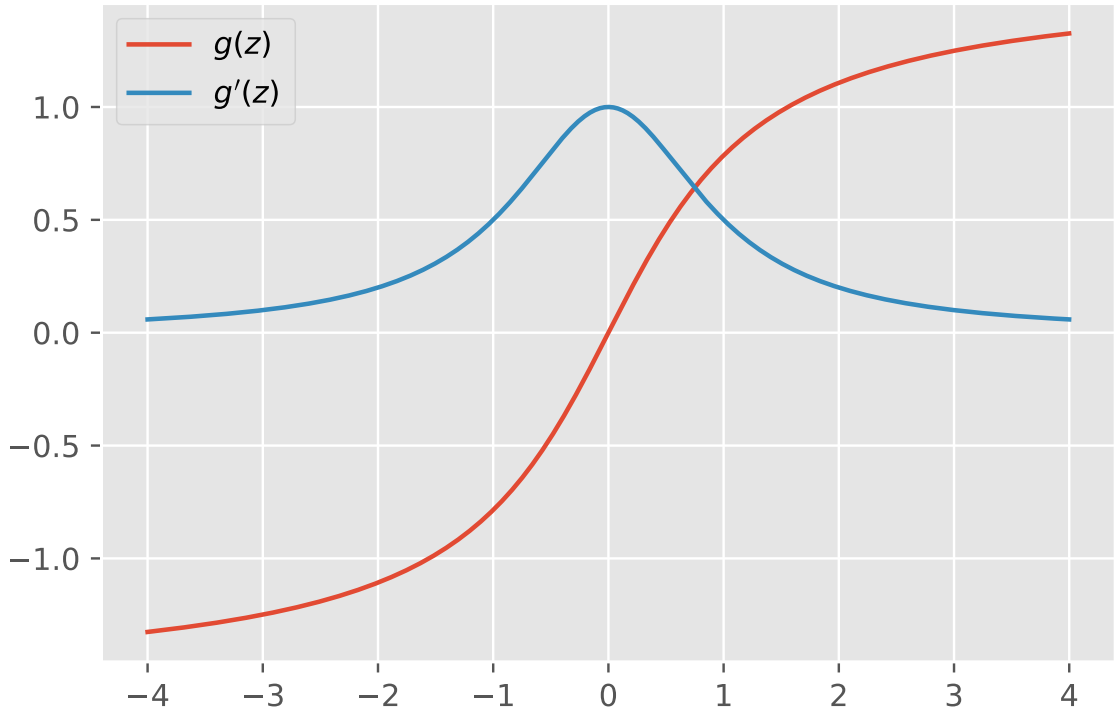



Figure B.8: Hyperbolic Tangent

$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{B.3.8}$$

and

$$g'(z) = 1 - g(z)^2 \tag{B.3.9}$$

**Rectified Linear Unit (ReLu):** An activation function of the form given Equation B.3.10. Now its output is a piecewise linear function as shown in Figure B.9.

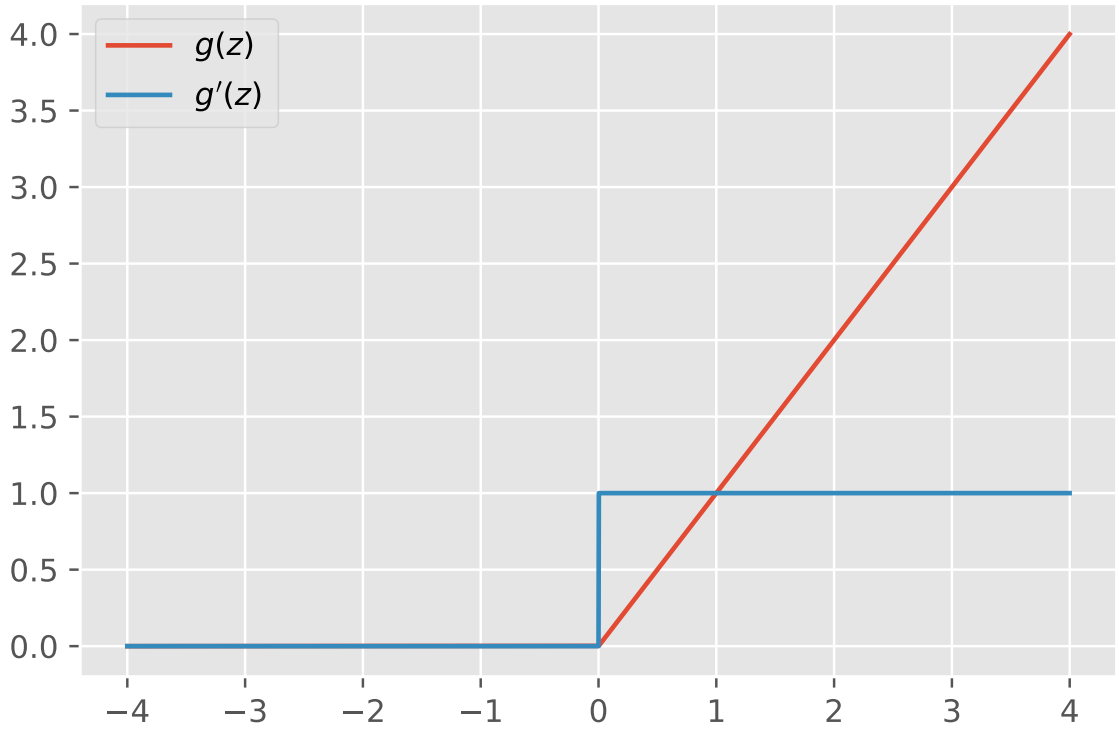



Figure B.9: Rectified Linear Unit (ReLu)

$$g(z) = \max\{0, z\} \tag{B.3.10}$$

and

$$g'(z) = \begin{cases} 1 & \text{if} \quad z > 0 \\ 0 & \text{otherwise} \end{cases} \tag{B.3.11}$$

**Scaled-exponential Linear Units (SeLU):** An activation function of the form given Equation B.3.12. Figure B.10 shows SeLU. The right side (for z larger than zero) resembles ReLUs. However, the left side (for z smaller than zero) seems to approach a gradient of zero.

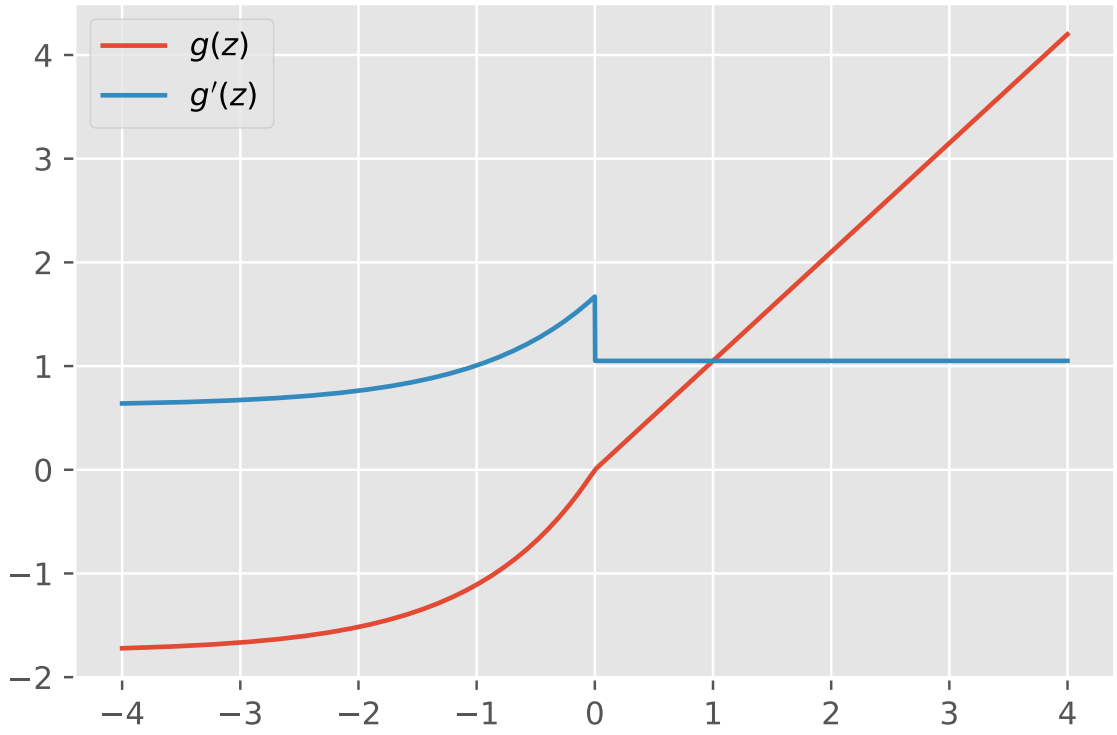



Figure B.10: Scaled-exponential Linear Units (SeLU)

$$g(z) = \lambda \begin{cases} \alpha(e^z - 1) & \text{for} \quad z < 0 \\ z & \text{for} \quad z \geq 0 \end{cases} \tag{B.3.12}$$

and

$$g'(z) = \lambda \begin{cases} g(z) + \alpha & \text{for} \quad z < 0 \\ 1 & \text{for} \quad z \geq 0 \end{cases} \tag{B.3.13}$$

# Appendix C

# Appendix to Chapter 4

## C.1 Additional Background on Reinforcement Learning

### Definition

**Definition C.1.1** (Contraction Mapping)**.** An operator B is a contraction mapping if $\forall$ F, G and $\gamma \in [0, 1)$, we have $||BF - BG||_\infty \leq \gamma ||F - B||_\infty$.

**Properties of Contraction Mapping:** If B is a contraction mapping, then

1. $F^\star = BF^\star$ has a solution and it is unique,
2. $F_t = BF_{t+1}$ and $F_t$ converges to the $F^\star$.

**Definition C.1.2** (Bellman Operator)**.** Let B be an operator from a value functions defined as follows:

$$[BQ](s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(s', a')$$

## Theorem

**Theorem C.1.1.** Let $T$ be a bellman operator. Let $Q_1$ and $Q_2$ be two value functions. Then

$$||TQ_1 - TQ_2||_\infty \leq \gamma ||Q_1 - Q_2||_\infty$$

*Proof.* Given two value functions $Q_1 and Q_2$, then we have

$$\begin{aligned}
||TQ_1 - TQ_2||_\infty &= \max_{a,s} \Big| [TQ_1](s,a) - [TQ_2](s,a) \Big| \\
&= \max_{a,s} \Big| \big(R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q_1(s',a')\big) - \Big(R(s,a) \\
&\qquad\qquad + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q_2(s',a')\Big) \Big| \\
&= \max_{a,s} \Big| \gamma \sum_{s'} T(s,a,s') \Big( \max_{a'} Q_1(s',a') - \max_{a'} Q_2(s',a') \Big) \Big| \\
&\leq \max_{a,s} \Big| \gamma \max_{s'} \Big( \max_{a'} Q_1(s',a') - \max_{a'} Q_2(s',a') \Big) \Big| \\
&= \gamma \max_{s'} \Big| \max_{a'} Q_1(s',a') - \max_{a'} Q_2(s',a') \Big| \\
&\leq \gamma \max_{s'} \max_{a'} \Big| Q_1(s',a') - Q_2(s',a') \Big| \text{(max is non-expansive)} \\
&= \gamma \max_{a,s} \Big| Q_1(s,a) - Q_2(s,a) \Big| \\
&= \gamma ||Q_1 - Q_2||_\infty
\end{aligned}$$

□

**Theorem C.1.2.** Let $\mathcal{F}$ be a complete vector space equipped with the norm $||\cdot||$ and let $T : \mathcal{F} \to \mathcal{F}$ be a $\gamma$-contraction mapping. Then $T$ admits a unique fixed point $V$.

*Proof.* We show that the bellman operator admits a unique fixed point. For any $U, V \in \mathcal{F}$:

$$\begin{aligned} ||U - V|| &= ||U - TU + TU - TV + TV - V|| \\ &\leq ||U - TU|| + ||TU - TV|| + ||TV - V|| \quad \text{(triangle inequality)} \\ &\leq ||U - TU|| + \gamma||U - V|| + ||TV - V|| \quad \text{(contraction property).} \end{aligned}$$

So we have

$$||U - V|| \leq \frac{||U - TU|| + ||TV - V||}{1 - \gamma}. \tag{C.1.1}$$

If U and V are both fixed points then $||U - TU|| = 0$ and $||TV - V|| = 0$. From equation C.1.1, $||U - V|| \leq 0$, which implies $U = V$. Therefore $T$ admits at most one fixed point.

□

**Theorem C.1.3** (Value Improvement)**.** Let $\pi$ and $\pi'$ be any two deterministic policies. If $Q^{\pi}(s, \pi'(s)) \geq V^{\pi}(s)$ for all $s \in \mathcal{S} \implies V^{\pi'}(s) \geq V^{\pi}(s)$ for all $s \in \mathcal{S}$

*Proof.*

$$
\begin{aligned}
V^{\pi}(s) &\leq Q^{\pi}(s, \pi'(s)) \\
&= \mathbb{E}^{\pi'}\Big[R_{t+1} + \gamma V^{\pi}(S_{t+1})\Big|S_t = s\Big] \\
&\leq \mathbb{E}^{\pi'}\Big[R_{t+1} + \gamma Q^{\pi}(S_{t+1}, \pi'(S_{t+1}))\Big|S_t = s\Big] \\
&= \mathbb{E}^{\pi'}\Big[R_{t+1} + \gamma \mathbb{E}^{\pi'}\Big[R_{t+2} + \gamma V^{\pi}(S_{t+2})\Big|S_t = s\Big]\Big|S_t = s\Big] \\
&= \mathbb{E}^{\pi'}\Big[R_{t+1} + \gamma R_{t+2} + \gamma^2 V^{\pi}(S_{t+2})\Big|S_t = s\Big] \\
&\vdots \\
&\leq \mathbb{E}^{\pi'}\Big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \ldots\Big|S_t = s\Big] \\
&= V^{\pi'}(s) \qquad \text{(C.1.2)}
\end{aligned}
$$

□

Next, we briefly review the multi-armed bandit problem, which is useful in understanding how Monte-Carlo tree search balances between exploration and exploitation.

## Multi-armed bandit problem

Multi-armed Bandit problem is a sequential decision problem in which one needs to choose between $K$ actions to maximize the cumulative reward by taking optimal action at each stage. For example, consider a gambler standing at a row of $K$ slot machines. The gambler needs to decide which machines to play and how many times to play each machine. The main objective of the gambler is to maximize the sum of rewards earned through a sequence of lever pulls. Assuming the gambler has no initial knowledge about the underlying reward distributions for each slot machine. Thus the gambler must estimate the rewards from past observation and faces the tradeoff between exploration of new machines, to obtain more information about the

expected payoffs, and exploitation of the observed machines, by using past experience to exploit the machine with the highest expected payoff.

Formally, a multi-bandit problem with $K$ arms is defined by the sequence of random payoff $X_{i,t}$ for $1 \leq i \leq K$, and $t \geq 1$, where each $i$ is the index of a gambling machine. Assume that each random payoff lies in the unit interval. Also assume that successive plays of machine $i$ with random payoffs $X_{i,1}, X_{i,2}, X_{i,3}, \ldots$ are independent and identically distributed according to an unknown distribution $\mathcal{D}_i$ on the unit interval and unknown mean $\mu_i = \mathbb{E}[X_{i,n}]$ for $n \geq 1$.

So the gambler needs a policy that determines which machine to play given the history of payoffs. A policy, in this case, is a mapping that selects the next arm to be played based on the sequence of past selections and payoffs obtained. The goal of a good policy is to minimize the gambler's regret. The regret is defined as the difference between what the gambler would have won by playing solely on the actual best slot machine and the gamblers expected winnings under the strategy that he is using. Formally, the regret $\rho$ after $T$ rounds is defined as the expected difference between the reward sum associated with an optimal strategy and the sum of the collected rewards:

$$\rho_T = T\mu^\star - \sum_{t=1}^{T} \widehat{r}_t$$

where: $\mu^\star = \max_k\{\mu_k\}$ is the maximum expected reward and $\widehat{r}_t = \mathbb{E}[Y_t]$ is the estimated reward at time $t$. More specifically, for each round $t = 1, 2, \ldots, T$, the gambler chooses $I_t \in \{1, 2, \ldots, K\}$ based on his past observation and receives a reward $Y_t \sim \mathcal{D}_{I_t}$. An efficient way of solving the bandit problem, by minimizing the regret, is to choose the move with the highest upper confidence bound. In other words, we need a strategy that will balance the trade-off between exploration and exploitation. The common strategy is to balance between exploration and exploitation using an upper

confidence bound (UCB). The UCB constructs a statistical confidence intervals for each machine.

$$\bar{x}_i \pm \sqrt{\frac{2 \ln n}{n_i}},$$

where:

- $\bar{x}_i$: the mean payout for machine $i$.
- $n_i$: the number of plays of machine $i$.
- $n$: the total number of pays

A simple UCB often know as UCB1 is defined as follows:

$$\text{UCB1} = \bar{x}_i + \sqrt{\frac{2 \ln n}{n_i}} \tag{C.1.3}$$

The first term $\bar{x}_i$ is the exploitation term. It represents the average winning rate discovered so far for the given option $i$. The second term $\sqrt{\frac{2 \ln n}{n_i}}$ is the exploration term. It gives value to those options which have not been explored as much. Using UCB C.1.3 to minimize regret was proposed in Auer et al. (2002). The idea is to choose an action that optimizes the UCB1 value at each round. The strategy is to pick the machine with the highest upper bound during each play. The regret of playing with this strategy grows only as $\mathcal{O}(\ln n)$.

## Monte Carlo Tree Search

We briefly review the Monte-Carlo Tree Search (MCTS), for extensive survey see Browne et al. (2012a). The four main steps of MCTS are summarized below. Each iteration of MCTS involves these four steps.

1. **Selection**: starting from a root node $R$, select successive child down to a leaf node $L$ using some tree policy.

2. **Expansion**: if the leaf node $L$ is not a terminal node, create one or more child nodes and choose node $C$ from them. This step expands the tree by adding one or more child nodes.

3. **Simulation**:play a random rollout from the child node $C$ according to a simulation policy.

4. **Back-Propagation**: use the result of the rollout to update the statistical information in the nodes on the path from $C$ to $R$.

Step 1 and 2 are often grouped as the Tree Policy. and step 3 is sometimes called the default policy. Algorithm 10 summarizes these steps through a generic MCTS procedure.

**Algorithm 10:** Generic MCTS Algorithm

**Input:** Given algorithms TreePolicy, DefaultPolicy, BackPropagate, UpdateStates and BestAction. Let $S$ denote the current root node state and $\tau$ denote the sample time for each MCTS run.

**Output:** Best action $A$ for the current state.

**for** $k = 1, \ldots, \tau$ **do**

1 v $\leftarrow$ TreePolicy(S)

2 $r \leftarrow$ DefaultPolicy($\gamma(v)$);

3 BackPropagate(v, r)

4 UpdateStates(S, r)

**end**

5 A = BestAction(S, 0)

MCTS iteratively builds a partial search tree. How the nodes are selected affect how the MCTS tree is built. The Upper Confidence Bound for Trees (UCT) is the most popular MCTS algorithm. UCT suggests treating the selection of a child node as a multi-armed bandit problem. Just like in the multi-armed bandit problem, UCB1 is used to address the trade-off between exploration and exploitation. Treating the choice of child node as a multi-armed bandit problem, then the value of the child is the expected reward approximated by the Monte Carlo simulations. So the rewards can be treated as random variables with unknown distributions just as the pay-off of the multi-armed bandit problem. Each time a child node is to be selected from the existing tree, the choice may be modeled as an independent multi-armed bandit problem. The choice of the selection of child nodes in the selection and expansion

phases is determined using upper confidence bounds for trees (UCT). Defines as follows:

$$\text{UCT} = \bar{x}_i + 2C_p\sqrt{\frac{2\ln n}{n_i}} \tag{C.1.4}$$

where: $C_p$ is a positive constant. $n$ is the number of times the current parent node has been visited and $n_i$ is the number of times child $i$ has been visited. Equation C.1.4 is very similar to the $UCB1$ equation except for the added constant. A child node $i$ is selected to maximize the value of UCT C.1.4. When $n_i = 0$ then UCT value is infinity. The previously unvisited nodes are assigned the largest possible value. This ensures that the previously unvisited nodes are considered at least once before any child is expanded further.

## C.2 Markov Decision Process

A Markov Decision Process (MDP) provides a mathematical framework for modeling a sequential decision-making problem. In what follows, we briefly review the observable MDP and the partially observable MDP. Let the tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$ denote the Observable Markov Decision Process, where:

- $\mathcal{S}$ is the finite non-empty set of states.
- $\mathcal{A}$ is the finite non-empty set of actions for each $s \in \mathcal{S}$.
- $P : \mathcal{S} \times \mathcal{A} \to \Pi(\mathcal{S})$ is the state transition function. And $P_a(s, s') = \mathbb{P}(s'|s, a)$ denotes the probability of transitioning from state $s$ to state $s'$ using action $a$.
- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function. We write $R(s, a)$ for the expected reward for taking action $a$ in state $s$.

- $\gamma \in (0, 1)$ is the discount factor.

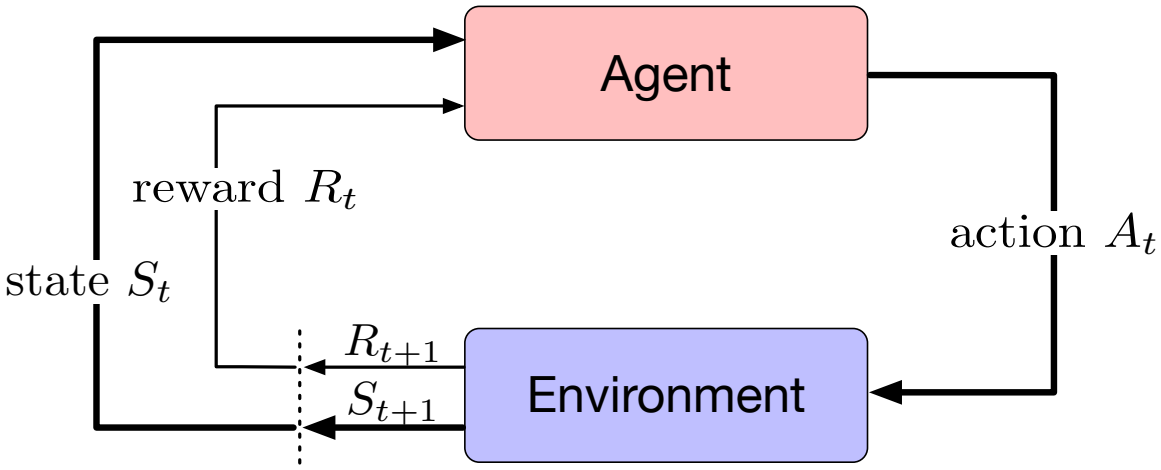



Figure C.1: The Markov Decision Process Environment

Figure C.1 is a diagrammatic representation of an MDP Environment, it shows how the agent interacts with the environment. The agent here refers to the thing that takes actions (neural network) and the environment is simply the world in which the agent operates or acts. Next, define a policy as a mapping from the state space to the action space, $\pi : \mathcal{S} \to \mathcal{A}$. The goal, in this setting, is to discover an optimal policy $\pi^\star$. There are many different approaches to finding a policy with maximum expected return. For illustration purpose, we will use the value function approach. Define the state-value function $V^\pi : \mathcal{S} \to \mathbb{R}$ as follows:

$$V^\pi(s) = \mathbb{E}^\pi\Big\{ \sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) \Big| s_0 = s \Big\}, \quad s \in \mathcal{S}. \tag{C.2.1}$$

The value function (C.2.1) is essentially the expected sum of discounted rewards upon starting in the state $s$ and taking actions according to the policy $\pi$. For a given

policy $\pi$ the value function (C.2.1) satisfies the Bellman equations:

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}^{\pi}\Big[\sum_{t=0}^{\infty}\gamma^{t}R(s_t,\pi(s_t))\Big|s_0=s\Big] \\ &= \mathbb{E}^{\pi}\Big[R(s_0,\pi(s_0))+\sum_{t=1}^{\infty}\gamma^{t}R(s_t,\pi(s_t))\Big|s_0=s\Big] \\ &= \mathbb{E}^{\pi}\Big[R(s,\pi(s))+\gamma\sum_{t=0}^{\infty}\gamma^{t}R(s_t,\pi(s_t))\Big|s_0=s\Big] \\ &= R(s,\pi(s))+\gamma\sum_{s'\in\mathcal{S}}P(s,\pi(s),s')V^{\pi}(s'). \end{aligned} \tag{C.2.2}$$

Equation (C.2.2) decomposes the value function into two terms: First, the immediate reward for starting in state $s$ and the expected sum of future discounted rewards. In a finite-state MDP, the Bellman's equations can be used to efficiently solve for $V^{\pi^\star}(s)$. Since we know how to solve a system of $|\mathcal{S}|$ linear equations in $|\mathcal{S}|$ unknown variables. According to (C.2.1) and (C.2.2), the optimal action-value $V^{\star}(s)$ can be defined as the solution to the simultaneous equations.

$$V^{\star}(s)=\max_{a\in\mathcal{A}}\Big(R(s,a)+\gamma\sum_{s'\in\mathcal{S}}T(s,a,s')V^{\star}(s')\Big),\quad \forall s\in\mathcal{S}, \tag{C.2.3}$$

and the optimal policy

$$\begin{aligned} \pi^{\star}(s) &= \operatorname*{argmax}_{a\in\mathcal{A}} V^{\pi}(s) \\ &= \operatorname*{argmax}_{a\in\mathcal{A}}\Big(R(s,a)+\gamma\sum_{s'\in\mathcal{S}}P(s,a,s')V^{\star}(s')\Big). \end{aligned} \tag{C.2.4}$$

It is often useful to express the value function (C.2.1) in terms of action-value function $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ called Q-function. Define the Q-function as follows:

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}^\pi\Big[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t)\Big|s_0=s,a_0=a\Big] \\ &= \mathbb{E}^\pi\Big[R(s_0,a_0)+\sum_{t=1}^{\infty}\gamma^t R(s_t,a_t)\Big|s_0=s,a_0=a\Big] \\ &= \mathbb{E}^\pi\Big[R(s,a_0)+\gamma\sum_{t=0}^{\infty}\gamma^t R(s_t,\pi(s_t))\Big|s_0=s,a_0=a\Big] \\ &= \sum_{a\in\mathcal{A}}\pi(a|s)\sum_{s'\in\mathcal{S}}P_a(s,s')\Big[R(s_0,a_0)+\gamma V^\pi(s')\Big], \end{aligned} \tag{C.2.5}$$

were $\pi(a|s)$ is the probability of taking action $a$ when in state $s$. Maximizing Equation C.2.5 gives the optimal policy:

$$\begin{aligned} \pi^\star(s) &= \operatorname*{argmax}_{a\in\mathcal{A}} Q^\pi(s,a) \\ &= \operatorname*{argmax}_{a\in\mathcal{A}}\Big(\sum_{a\in\mathcal{A}}\pi(a|s)\sum_{s'\in\mathcal{S}}P_a(s,s')\Big[R(s_0,a_0)+\gamma V^\pi(s')\Big]\Big), \quad \forall s\in\mathcal{S}. \end{aligned} \tag{C.2.6}$$

Similarly, let the tuple $\mathcal{M} =< \mathcal{S}, \mathcal{A}, P, R, \mathcal{O}, Z, \gamma, b_0 >$ denote a Partially Observable Markov Decision Process (POMDP), where

- $\mathcal{S}$ is the non-empty set of states.
- $\mathcal{A}$ is the non-empty set of actions for each $s \in \mathcal{S}$.
- $P : \mathcal{S} \times \mathcal{A} \to \Pi(\mathcal{S})$ is the state transition function. And $P_a(s,s') = \mathbb{P}(s'|s,a)$ denotes the probability of transitioning from state $s$ to state $s'$ using action $a$.
- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function. $R(s,a)$ denotes the expected reward for taking action $a$ in state $s$.
- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function.

- $\mathcal{O}$ is the non-empty observation space
- $Z$ encodes the observation probabilities. Here we write $Z_{a_t}(o_t, s_{t+1}) = \mathbb{P}(o|s, a)$ for the probability of observing $o_t$ by taking action $a_t$ and landing in the state $s_{t+1}$
- $\gamma \in (0, 1)$ is the discount factor.
- $b_0$ is the initial belief state.

POMDP is a generalization of MDP to settings where the states are not fully observable. It provides a natural model for sequential decision making under uncertainty. Although POMDPs extends the MDPs to many realistic settings, the generalization comes with high computational cost. As a result, the application of POMDPs remains limited to small scale problems. In POMDP, it is often assumed that the underlying system dynamics are governed by an MDP but the agent can not directly observe the underlying states. So the agent needs to maintain a belief state. A belief sate $b(s)$ is the probability of being in $s$ according to its history of actions and observations. The belief in a POMDP captures all the information in the history $h_t = \{a_1, o_1, r_1, \ldots, a_t, o_t, r_t\}$ needed to predict future events. When the agent performs an action $a_t \in \mathcal{A}$ based on belief $b$, following a particular policy $\pi$, it receives a reward and transition to state $s_{t+1}$. The quality of the policy $\pi$ is measured by the expected future reward. In this setting, a policy $\pi$ is a mapping from the set of observable histories to the action space $\pi : \mathcal{H}_o \to \mathcal{A}$. The belief update at time

$t+1$ is given by:

$$
\begin{aligned}
b_{t+1}(s_{t+1}) &= \mathbb{P}(s_{t+1}|o_t, a_t, b_t) \\
&= \frac{\mathbb{P}(s_{t+1}, o_t)}{\mathbb{P}(o_t|a_t, b_t)} \\
&= \frac{\mathbb{P}(o_t|s_{t+1}, a_t, b_t)\mathbb{P}(s_{t+1}|a_t, b_t)}{\mathbb{P}(o_t|a_t, b_t)} \\
&= \frac{\mathbb{P}(o_t|s_{t+1}, a_t, b_t)\sum_{s\in\mathcal{S}}\mathbb{P}(s_{t+1}|s, a_t, b_t)\mathbb{P}(s|a_t, b_t)}{\mathbb{P}(o_t|a_t, b_t)} \\
&= \frac{Z_{a_t}(o_t, s_{t+1})\sum_{s\in\mathcal{S}}P_{a_t}(s_t, s_{t+1})b(s)}{\mathbb{P}(o_t|a_t, b_t)} \\
&= K\cdot Z_{a_t}(o_t, s_{t+1})\sum_{s\in\mathcal{S}}P_{a_t}(s_t, s_{t+1})b(s) \\
&\propto Z_{a_t}(o_t, s_{t+1})\sum_{s\in\mathcal{S}}P_{a_t}(s_t, s_{t+1})b(s)
\end{aligned} \tag{C.2.7}
$$

The agent's goal remains to maximize the expected discounted future rewards. It has been shown that finite POMDP can be converted to the framework of fully observable MDP called Belief state MDP Hauskrecht (2000). A belief MDP is a 4-tuple $\mathcal{M} =< \mathcal{B}, \mathcal{A}, T^b, R^b >$, where

- $\mathcal{B}$ is the continuous state space.
- $\mathcal{A}$ is the action space.
- $T^b : \mathcal{B}\times\mathcal{A}\to\mathcal{B}$ is the belief transition function. Here we write $T^b_a(b, b') = \mathbb{P}(b'|b, a)$ for the probability of transitioning from belief $b$ to belief $b'$ using action $a$.

$$
\begin{aligned}
T^b_a(b, b') &= \mathbb{P}(b'|b, a) \\
&= \sum_{o\in\mathcal{O}}\mathbb{P}(b'|b, a, o)\mathbb{P}(o|b, a) \\
&= \sum_{o\in\mathcal{O}}\mathbb{P}(b'|b, a, o)\sum_{s'\in\mathcal{S}}Z_a(o, s')\sum_{s\in\mathcal{S}}P_a(s, s')b(s)
\end{aligned} \tag{C.2.8}
$$

where $\mathbb{P}(b'|b,a,o) = \mathbb{E}[\mathbf{1}_{\{b_o^a = b'\}}]$ and $b_o^a(s') = b_{t+1}(s')$

- $R : \mathcal{B} \times \mathcal{A} \to \mathbb{R}$ is the reward function. We write $R^b(b,a)$ for the expected reward for taking action $a$ under the belief $b$, where

$$R^b(b,a) = \sum_{s \in \mathcal{S}} b(s) R(s,a) \tag{C.2.9}$$

For the infinite horizon, a policy is a mapping from the belief space into actions, $\pi : \mathcal{B} \to \mathcal{A}$. And the $Q$-function is defined by

$$Q^\pi(b,a) = \sum_{s \in \mathcal{S}} b(s) R(s,a) + \gamma \sum_{o \in \mathcal{O}} P(o|a,b) V^\pi(b_o^a) \tag{C.2.10}$$

And the optimal policy is given by

$$\begin{aligned} \pi^\star(s) &= \operatorname*{argmax}_{a \in \mathcal{A}} Q^\pi(b,a) \\ &= \operatorname*{argmax}_{a \in \mathcal{A}} \Big( \sum_{s \in \mathcal{S}} b(s) R(s,a) + \gamma \sum_{o \in \mathcal{O}} P(o|a,b) V^\pi(b_o^a) \Big), \quad \forall s \in \mathcal{S}. \end{aligned} \tag{C.2.11}$$

## C.3 Implementation Details

### Neural Network Architecture

The policy and value function The policy and value function approximations use fully-connected neural networks with five and two hidden layers, respectively, and SELU (scaled exponential linear unit) activation (Klambauer et al., 2017). The policy network contains two sets of outputs: (1) one of seven actions (no action, normal attack, move, skill 1, skill 2, skill 3, and heal) and (2) a two-dimensional direction parameter used for the action. The first two hidden layers are shared and have 120 and 100 hidden units, while each of the two outputs corresponds to a set of three hidden

layers with 80, 70, and 50 hidden units. The value function approximation uses a fully-connected network with 128 hidden units in the first layer and 96 hidden units in the second layer. As mentioned in the main paper, this architecture is consistent across all agents whenever policy and/or value networks are needed.

### Features of the State

As shown in Table C.1, the state of the game is represented by 41-dimensional feature vector, which was constructed using the output from the game engine and API. The features consists of basic attributes of the two heroes, the computer-controlled units, and structures. The feature lists also have information on the relative positions of the other units and structures with respect to the hero controlled by algorithm.

Table C.1: The state feature list for the reinforcement learning agent

| No. | Feature | Dim. |
|---|---|---|
| 1 | Location of Hero 1 | 2 |
| 2 | Location of Hero 2 | 2 |
| 3 | HP of Hero 1 | 1 |
| 4 | HP of Hero 2 | 1 |
| 5 | Hero 1 skill cooldowns | 5 |
| 6 | Hero 2 skill cooldowns | 5 |
| 7 | Direction to enemy hero | 3 |
| 8 | Direction to enemy tower | 4 |
| 9 | Direction to enemy minion | 3 |
| 10 | Enemy tower HP | 1 |
| 11 | Enemy minion HP | 1 |
| 12 | Direction to the spring | 3 |
| 13 | Total HP of allied minions | 1 |
| 14 | Enemy's tower attacking Hero 1 | 3 |
| 15 | Hero 1 in range of enemy towers | 3 |
| 16 | Hero 2 in range of enemy towers | 3 |

**Tree Search Details**

We provide some more information regarding the implementation of feedback-based tree search. A major challenge in implementing in *King of Glory* is that the game engine can only move forward, meaning that our sampled states are not i.i.d. and instead follow the trajectory of the policy induced by `MCTS`. However, to decrease the correlation between visited states, we inject random movements and random switches to the internal AI policy in order to move to a "more random" next state. Rollouts are performed on separate processors to enable tree search in a game engine that cannot

rewind. All experiments use the `c4.2xlarge` instances on Amazon Web Services, and we utilized parallelization across four cores within each call to `MCTS`.

### Computation

For the first two rounds, $k = 1$ and $k = 2$, samples were collected from 60 games each. On the next five rounds, samples were collected from 40 games each. All experiments used the `c4.4xlarge` instance of Amazon Web Services.

# Bibliography


Lina Al-Kanj, Warren B Powell, and Belgacem Bouzaiene-Ayari. The information-collecting vehicle routing problem: stochastic optimization for emergency storm response. *arXiv preprint arXiv:1605.05711*, 2016.

Eric Angel. *A Survey of Approximation Results for Local Search Algorithms*, pages 30–73. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-32213-9. doi: 10.1007/11671541_2.

Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems*, 2017.

András Antos, Csaba Szepesvári, and Rémi Munos. Learning near-optimal policies with bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning*, 71(1):89–129, 2008.

Bradford L. Arkin and Lawrence M. Leemis. Nonparametric estimation of the cumulative intensity function for a nonhomogeneous poisson process from overlapping realizations. *Manage. Sci.*, 46(7):989–998, July 2000. ISSN 0025-1909.

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, May 2002. ISSN 1573-0565. doi: 10.1023/A:1013689704352.

Francois Baccelli and William A. Massey. A sample path analysis of the m/m/1 queue. *Journal of Applied Probability*, 26(2):418–422, 1989. doi: 10.2307/3214049.

François Baccelli, William A. Massey, and Paul E. Wright. Determining the exit time distribution for a closed cyclic network. *Theoretical Computer Science*, 125 (1):149–165, March 1994. ISSN 0304-3975. doi: 10.1016/0304-3975(94)90298-4.

J. Bak and D.J. Newman. *Complex Analysis*. Encyclopaedia of Mathematical Sciences. Springer, 1997. ISBN 9780387947563.

Dimitri P Bertsekas. Convergence of discretization procedures in dynamic programming. *IEEE Transactions on Automatic Control*, 20(3):415–419, 1975.

Dimitri P Bertsekas and John N Tsitsiklis. *Neuro-dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter I. Cowling, Stephen Tavener, Diego Perez, Spyridon Samothrakis, Simon Colton, and et al. A survey of monte carlo tree search methods. *IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI*, 2012a.

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012b.

Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.

Tristan Cazenave. Nested Monte-Carlo search. In *International Joint Conference on Artificial Intelligence*, pages 456–461, 2009.

D. G. Champernowne. An elementary method of solution of the queueing problem with a single server and constant parameters. *Journal of the Royal Statistical Society. Series B (Methodological)*, 18(1):125–128, 1956. ISSN 00359246.

Guillaume Chaslot, Jahn-Takeshi Saito, Jos WHM Uiterwijk, Bruno Bouzy, and H Jaap van den Herik. Monte-Carlo strategies for computer Go. In *18th Belgian-Dutch Conference on Artificial Intelligence*, pages 83–90, 2006.

Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. In *AIIDE*, 2008.

A. B. Clarke. A waiting line process of markov type. *Ann. Math. Statist.*, 27(2): 452–459, 06 1956. doi: 10.1214/aoms/1177728268.

Jean-Franois Cordeau, Guy Desaulniers, Jacques Desrosiers, Marius M. Solomon, and Franois Soumis. *VRP with Time Windows*, chapter 7, pages 157–193. 2002. doi: 10.1137/1.9780898718515.ch7.

Adrien Couëtoux, Jean-Baptiste Hoock, Nataliya Sokolovska, Olivier Teytaud, and Nicolas Bonnard. Continuous upper confidence trees. In *International Conference on Learning and Intelligent Optimization*, pages 433–445. Springer, 2011a.

Adrien Couëtoux, Mario Milone, Mátyás Brendel, Hassan Doghmen, Michele Sebag, and Olivier Teytaud. Continuous rapid action value estimates. In *Asian Conference on Machine Learning*, pages 19–31, 2011b.

Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International Conference on Computers and Games*, pages 72–83, 2006.

D Pucci De Farias and Benjamin Van Roy. On the existence of fixed points for approximate value iteration and temporal-difference learning. *Journal of Optimization theory and Applications*, 105(3):589–608, 2000.

Paul DeMaio and Russel Meddin. The bike-sharing world map, 2019. Assesed: 2019-10-13.

Guy Desaulniers, Jacques Desrosiers, Andreas Erdmann, Marius M. Solomon, and Francois Soumis. *VRP with Pickup and Delivery*, chapter 9, pages 225–242. 2002. doi: 10.1137/1.9780898718515.ch9.

François Dufour and Tomás Prieto-Rumeau. Approximation of markov decision processes with general state space. *Journal of Mathematical Analysis and Applications*, 388(2):1254–1267, 2012.

Markus Enzenberger. Evaluation in go by a neural network using soft segmentation. In *Advances in Computer Games*, pages 97–108. Springer, 2004.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD-96, page 226?231. AAAI Press, 1996.

William Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. Wiley, January 1968. ISBN 0471257087.

Elliot Fishman. Bikeshare: A review of recent literature. *Transport Reviews*, 36(1): 92–113, 2016. doi: 10.1080/01441647.2015.1033036.

Elliot Fishman, Simon Washington, Narelle Haworth, and Armando Mazzei. Barriers to bikesharing: an analysis from melbourne and brisbane. *Journal of Transport Geography*, 41:325 – 337, 2014. ISSN 0966-6923.

Sylvain Gelly and David Silver. Combining online and offline knowledge in UCT. In *Proceedings of the 24th International Conference on Machine learning*, pages 273–280. ACM, 2007.

Sylvain Gelly and David Silver. Monte-carlo tree search and rapid action value estimation in computer Go. *Artificial Intelligence*, 175(11):1856–1875, 2011.

Sylvain Gelly, Levente Kocsis, Marc Schoenauer, Michele Sebag, David Silver, Csaba Szepesvári, and Olivier Teytaud. The grand challenge of computer Go: Monte Carlo tree search and extensions. *Communications of the ACM*, 55(3):106–113, 2012.

Bernard Gorman. Imitation learning through games: theory, implementation and evaluation. 2009.

Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang. Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning. In *Advances in Neural Information Processing Systems*, pages 3338–3346, 2014.

László Györfi, Michael Kohler, Adam Krzyzak, and Harro Walk. *A distribution-free theory of nonparametric regression*. Springer Science & Business Media, 2006.

Harmon, Emily Campbell. Dynamic Rate Queues for Efficient Staffing of Hospital Cleaning Services. http://www.princeton.edu/~mudd/thesis/, May 2012.

William B Haskell, Rahul Jain, and Dileep Kalathil. Empirical dynamic programming. *Mathematics of Operations Research*, 41(2):402–429, 2016.

William B Haskell, Rahul Jain, Hiteshi Sharma, and Pengqian Yu. An empirical dynamic programming algorithm for continuous MDPs. *arXiv preprint arXiv:1709.07506*, 2017.

Matthew J. Hausknecht and Peter Stone. Deep reinforcement learning in parameterized action space. *CoRR*, abs/1511.04143, 2016.

M. Hauskrecht. Value-function approximations for partially observable markov decision processes. *Journal of Artificial Intelligence Research*, 13:3394, Aug 2000. ISSN 1076-9757. doi: 10.1613/jair.678.

David Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation*, 100(1):78–150, 1992.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.

Philip Hingston and Martin Masek. Experiments with Monte Carlo Othello. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 4059–4064. IEEE, 2007.

Junling Hu and Michael P Wellman. Nash Q-learning for general-sum stochastic games. *Journal of Machine Learning Research*, 4:1039–1069, 2003.

Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Comput. Surv.*, 50(2):21:1–21:35, April 2017. ISSN 0360-0300. doi: 10.1145/3054912.

Daniel R Jiang, Lina Al-Kanj, and Warren B Powell. Monte carlo tree search with sampled information relaxation dual bounds. *arXiv preprint arXiv:1704.05963*, 2017.

Daniel R. Jiang, Emmanuel Ekwedike, and Han Liu. Feedback-based tree search for reinforcement learning. *ArXiv*, abs/1805.05935, 2018.

Brian Kallehauge, Jesper Larsen, Oli Madsen, and Marius Solomon. *Vehicle Routing Problem with Time Windows*, pages 67–98. 03 2006. doi: 10.1007/0-387-25486-2_3.

Emilie Kaufmann and Wouter Koolen. Monte-Carlo tree search by best arm identification. In *Advances in Neural Information Processing Systems*, 2017.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, pages 972–981, 2017.

Mykel J. Kochenderfer, Christopher Amato, Girish Chowdhary, Jonathan P. How, Hayley J. Davison Reynolds, Jason R. Thornton, Pedro A. Torres-Carrasquillo, N. Kemal Üre, and John Vian. *Decision Making Under Uncertainty: Theory and Application*. The MIT Press, 1st edition, 2015. ISBN 0262029251.

Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning*, pages 282–293. Springer, 2006.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, page 2012.

Takács Lajos. *Introduction to the Theory of Queues*. Oxford University Press, New York, USA, 1962.

John Langford and Bianca Zadrozny. Relating reinforcement learning performance to classification performance. In *International Conference on Machine Learning*, pages 473–480. ACM, 2005.

G. Laporte, M. Gendreau, J-Y. Potvin, and F. Semet. Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operational Research*, 7(45):285–300, 2000. doi: 10.1111/j.1475-3995.2000.tb00200.x.

Alessandro Lazaric, Mohammad Ghavamzadeh, and Rémi Munos. Analysis of classification-based policy iteration algorithms. *Journal of Machine Learning Research*, 17(1):583–612, 2016.

W. Ledermann, G. E. H. Reuter, and Kurt Mahler. Spectral theory for the differential equations of simple birth and death processes. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 246(914): 321–369, 1954. doi: 10.1098/rsta.1954.0001.

Lihong Li, Vadim Bulitko, and Russell Greiner. Focus of attention in reinforcement learning. *Journal of Universal Computer Science*, 13(9):1246–1269, 2007.

S. Lloyd. Least squares quantization in pcm. *Technical Report*, pages RR–5497, Sept 1957.

S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982. ISSN 1557-9654. doi: 10.1109/TIT.1982.1056489.

James B. MacQueen. Some methods for classification and analysis of multivariate observations. 1967.

Raphaël Maîtrepierre, Jérémie Mary, and Rémi Munos. Adaptative play in Texas hold'em poker. In *European Conference on Artificial Intelligence*, 2008.

William Massey. Calculating exit times for series jackson networks. *Journal of Applied Probability*, 24:226–234, 03 1987. doi: 10.1017/S0021900200030758.

Jean Méhat and Tristan Cazenave. Combining UCT and nested Monte Carlo search for single-player general game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):271–277, 2010.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 00280836.

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2012.

Philip M. Morse. *Queues, inventories, and maintenance ; the analysis of operational systems with variable demand and supply*. Wiley, New York, USA, 1958.

Inc. Motivate International. Citi bike system data. https://www.citibikenyc.com/system-data, a. Accessed: 2020-4-11.

Inc. Motivate International. Divvy system data. https://www.divvybikes.com/system-data, b. Accessed: 2020-4-11.

Rémi Munos. Performance bounds in l_p-norm for approximate value iteration. *SIAM Journal on Control and Optimization*, 46(2):541–561, 2007.

Rémi Munos and Csaba Szepesvári. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9(May):815–857, 2008.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

Yan Pan, Ray Chen Zheng, Jiaxi Zhang, and Xin Yao. Predicting bike sharing demand using recurrent neural networks. *Procedia Computer Science*, 147:562 – 566, 2019. ISSN 1877-0509. 2018 International Conference on Identification, Information and Knowledge in the Internet of Things.

Stephen D. Parkes, Greg Marsden, Susan A. Shaheen, and Adam P. Cohen. Understanding the diffusion of public bikesharing systems: evidence from europe and north america. *Journal of Transport Geography*, 31:94 – 103, 2013. ISSN 0966-6923.

Sophie Parragh, Karl Doerner, and Richard Hartl. A survey on pickup and delivery problems: Part ii: Transportation between pickup and delivery locations. *Journal fr Betriebswirtschaft*, 58:81–117, 06 2008. doi: 10.1007/s11301-008-0036-4.

David Pollard. Empirical processes: Theory and applications. In *NSF-CBMS Regional Conference Series in Probability and Statistics*, pages i–86. JSTOR, 1990.

Martin L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

Tal Raviv and Ofer Kolka. Optimal inventory management of a bike-sharing station. *IIE Transactions*, 45:1077–1093, 10 2013. doi: 10.1080/0740817X.2013.770186.

Tal Raviv, Michal Tzur, and Iris Forma. Static repositioning in a bike-sharing system: Models and solution approaches. *EURO Journal on Transportation and Logistics*, 2, 01 2014. doi: 10.1007/s13676-012-0017-6.

Sidney I Resnick. *A probability path*. Springer Science & Business Media, 2013.

Sheldon Ross. Chapter 11 - statistical validation techniques. In Sheldon Ross, editor, *Simulation (Fifth Edition)*, pages 247 – 270. Academic Press, fifth edition edition, 2013. ISBN 978-0-12-415825-2.

Stephane Ross and Drew Bagnell. Efficient reductions for imitation learning. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 661–668, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

Naci Saldi, Serdar Yüksel, and Tamás Linder. On the asymptotic optimality of finite approximations to markov decision processes with borel spaces. *Mathematics of Operations Research*, 2017.

Stefan Schaal. Is imitation learning the route to humanoid robots?, 1999.

Bruno Scherrer, Mohammad Ghavamzadeh, Victor Gabillon, Boris Lesner, and Matthieu Geist. Approximate modified policy iteration and its application to the game of tetris. *Journal of Machine Learning Research*, 16:1629–1676, 2015.

J. Schuijbroek, Robert C. Hampshire, and W.-J. van Hoeve. Inventory rebalancing and vehicle routing in bike sharing systems. *European Journal of Operational Research*, 257:992–1004, 2017.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014. doi: 10.1017/CBO9781107298019.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016a.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016b. doi: 10.1038/nature16961.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

P. Singh and P. A. Meshram. Survey of density based clustering algorithms and its variants. In *2017 International Conference on Inventive Computing and Informatics (ICICI)*, pages 920–926, Nov 2017.

J.G. Skellam. The frequency distribution of the difference between two poisson variates belonging to different populations. *Journal of the Royal Statistical Society. Series A (General)*, 109(Pt 3):296, 1946. ISSN 0035-9238.

Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

Qiong Tang, Zhuo Fu, and Meng Qiu. A bilevel programming model and algorithm for the static bike repositioning problem. *Journal of advanced transportation*, 2019: 19, 06 2019. doi: 10.1155/2019/8641492.

Shuang Tao and Jamol Pender. A stochastic analysis of bike-sharing systems. *Probability in the Engineering and Informational Sciences*, page 158, 2020. doi: 10.1017/S0269964820000297.

Kazuki Teraoka, Kohei Hatano, and Eiji Takimoto. Efficient sampling method for Monte Carlo tree search problem. *IEICE Transactions on Information and Systems*, 97(3):392–398, 2014.

Christian Thurau, Gerhard Sagerer, and Christian Bauckhage. Imitation learning at all levels of game-ai. 01 2004.

Benjamin Van Roy. Performance loss bounds for approximate value iteration with state aggregation. *Mathematics of Operations Research*, 31(2):234–244, 2006.

Bo Wang and Inhi Kim. Short-term prediction for bike-sharing service using machine learning. *Transportation Research Procedia*, 34:171 – 178, 2018. ISSN 2352-1465.