Aalto University
School of Science

Master's programme in Mathematics and Operations Research

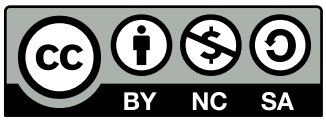
Gaussian process surrogate models in elevator planning

Tuomas Myllymäki

Master's Thesis
2024

© 2024

This work is licensed under a Creative Commons "Attribution-NonCommercial-ShareAlike 4.0 International" license.



Aalto University
School of Science

Author Tuomas Myllymäki

Title Gaussian process surrogate models in elevator planning

Degree programme Mathematics and Operations Research

Major Systems and Operations Research **Code of major** SCI3055

Supervisor Asst. Prof. Philine Schiewe

Advisor Dr. Juho Kokkala

Collaborative partner KONE Corporation

Date 27 April 2024 **Number of pages** 74/2 **Language** English

Abstract

Finding a suitable elevator system for a given building in a complex task. There are many design considerations which must be taken into account, one of main ones being the level of traffic in the building. This thesis investigates how to mathematically define the problem of finding maximum traffic intensity, which is still feasible given certain waiting time and time-to-destination constraints, for a building with a specific elevator system. The problem of finding the maximum intensity is a simplified inverse problem to usual elevator planning. The thesis also provides methods on solving the defined problem efficiently.

A simulator is utilized to simulate waiting time and time-to-destination metrics for a given traffic intensity. However, because of the high computational complexity of the simulator, it cannot be utilized with a simple algorithm for finding the maximum intensity. The thesis suggests the use of Bayesian optimization with Gaussian processes to optimize the search of maximum intensity.

Thesis explores the literature of Bayesian optimization and Gaussian processes and provides an introduction to the topics. From literature, a few candidate Bayesian optimization algorithms are developed to suit the particular traffic optimization problem.

With the help of a toy optimization problem, the optimum of which is known, the performance of the candidate algorithms is analyzed. According to the results of the toy analysis, the list of suitable candidates is narrowed down. The remaining algorithms are then tested with a real traffic simulator and their performance is analyzed from multiple perspectives. At the end, one algorithm is picked as the best. Sensitivity tests are then performed on this algorithm to validate its performance in different scenarios. In the sensitivity analysis we also discover some weaknesses in the chosen algorithm.

The resulting finding is that the chosen algorithm performs well in a variety of circumstances and provides a suitable method estimating the maximum traffic intensity. However, there are some factors which affect the performance of the algorithm, for example the size of the search space. Therefore, the results of the algorithm must be examined with the help of domain knowledge.

Keywords Elevators, Bayesian optimization, Gaussian processes, Optimization

Aalto-yliopisto
Perustieteiden
korkeakoulu

Tekijä Tuomas Myllymäki

Työn nimi Surrogaattimallit gaussisilla prosesseilla hissisuunnittelussa

Koulutusohjelma Mathematics and Operations Research

Pääaine Systems and Operations Research **Pääaineen koodi** SCI3055

Työn valvoja Apulaisprofessori Philine Schiewe

Työn ohjaaja TkT. Juho Kokkala

Yhteistyötaho KONE Oyj

Päivämäärä 27.4.2024 **Sivumäärä** 74/2 **Kieli** englanti

Tiivistelmä

Rakennuksen hissisuunnittelu on moniulotteinen tehtävä, jossa on huomioitava monta muuttujaa. Yksi keskeisimmistä on hissiliikenteen intensiteetti. Tämä diplomityö tarkastelee eräänlaista yksinkertaistettua käänteisongelmaa hissisuunnittelulle, joka on liikenneintensiteetin maksimin löytäminen asetettujen odotus- ja matkustusaikarajoitteiden alla. Liikenneintensiteetin maksimin etsiminen esitetään ensin matemaattisena optimointitehtävänä, jonka jälkeen tutkitaan eri algoritmeja sen ratkaisemiseen.

Liikenneintensiteetin maksimin löytämiseen hyödynnetään simulaattoria, jolla lasketaan odotus- ja matkustusaikoja eri intensiteeteille. Johtuen simulaattorin laskennallisesta raskaudesta, intensiteetin maksimia ei voida löytää käyttämällä yleistä optimointialgoritmia. Tämän vuoksi bayesilaista optimointia ja gaussisia prosesseja hyödynnetään simuloitavien intensiteettiarvojen valitsemiseen.

Diplomityö esittelee bayesilaisen optimoinnin ja gaussisten prosessien teorian. Kirjallisuudesta saatujen taustatietojen avulla kehitetään muutama algoritmiehdokas liikenneintensiteetin optimointia varten.

Algoritmiehdokkaita testataan ensin leluongelmalla, jonka optimi on tiedossa. Algoritmien suorituskykyä tarkastellaan monesta eri näkökulmasta ja tämän analyysin tuloksena rajataan epäsopivia ehdokkaita pois. Parhaat algoritmit otetaan seuraavaan vaiheeseen jossa tarkastellaan niiden suorituskykyä oikean liikennesimulaattorin yhteydessä. Paras algoritmi valitaan eri metriikoiden perusteella. Parhaalle algoritmille suoritetaan herkkyysanalyysi, joka kertoo kuinka algoritmi suoriutuu erilaisissa tilanteissa. Herkkyysanalyysin avulla löydetään myös muutama heikkous valitussa algoritmissa.

Diplomityön johtopäätöksenä on että valittu algoritmi suoriutuu hyvin erilaisissa tilanteissa ja soveltuu ongelman ratkaisualgoritmiksi. Kuitenkin huomataan että algoritmi ei suoriudu yhtä hyvin, jos tietyt ongelman muuttujat ovat asetettu väärin. Esimerkiksi algoritmin hakualueen rajat pitää määrittää oikein jotta algoritmi löytää optimaalisen intensiteetin. Näiden herkkyyksien vuoksi algoritmia käytettäessä on hyödynnettävä asiantuntemusta ja sen tuottamia tuloksia on syytä tarkastella kriittisesti.

Avainsanat Hissit, Bayesilainen optimointi, Gaussiset prosessit, Optimointi

Preface

First of all, I would like to thank Aalto University for their excellent teaching and culture. I would not be the person I am today without my experiences in Otaniemi. In addition to the university, I would like to thank the Aalto University student union AYY for everything they have done for myself and the wider Aalto community during my studies. Regarding this particular thesis, I would like to thank KONE Corporation for their excellent cooperation during this thesis.

I would like to thank Dr. Juho Kokkala for his excellent advise and expertise as my advisor for this thesis. Regarding Aalto University, I want to thank the whole Department of Mathematics and Systems Analysis. Especially, I want to thank Assistant Professor Philine Schiewe for her excellent advice and for supervising my thesis. I would like to also thank both Dr. Juho Kokkala and Prof. Philine Schiewe for their willingness to work with flexible time schedules, which came in handy multiple times during this work.

As studies are only a part of my university experience, I thank both Fyysikkokilta and Tietokilta, as they have granted me very fond memories which I am sure to cherish for the rest of my life. My gratitude also extends to Aalto Strength Society and Krokettikonklaavi as they have also provided me excellent memories and friendships which I am sure will last for the rest of my life. I would also like to thank some other less official clubs, which I will not name here but needless to say have provided me with endless entertainment during my studies and lifelong friends.

Finally, but most importantly, I would like to thank my parents for supporting me throughout my studies and for always having my back. I would not be here currently finishing my master's thesis if I did not have such great parents.

To end this preface, I would like to state that it snowed yesterday, in the middle of April. Inspired by that quite unusual weather condition for this time of the year, I would like to encourage the reader of this thesis to continue on, regardless of the perhaps less optimal circumstances you might find yourself in.

Espoo, 24 April 2024

Tuomas S. Myllymäki

Contents

Symbols and abbreviations

Symbols

$\mathbb{R}^+$	The positive real values
ϕ	Probability density function for the standard normal distribution
Φ	Cumulative density function for the standard normal distribution
$\mathcal{D}_f$	A dataset containing samples of function f
$f\|\mathcal{D}_f$	The posterior Gaussian process for function f computed using $\mathcal{D}_f$
$\mathcal{M}_f$	A Gaussian process model for function f
T_{AWT}	The average waiting time function
T_{ATTD}	The average time-to-destination function
C_{AWT}	Constraint value for average waiting time
C_{ATTD}	Constraint value for average time-to-destination

Abbreviations

WT	Elevator waiting time
TTD	Elevator time-to-destination
AWT	Average waiting time
ATTD	Average time-to-destination

1 Introduction

1.1 Background

The Shanghai Tower has the fastest elevator in the world. The elevator reaches speeds of 73 kilometers per hour on its approximately 580 meter continuous run [21]. The increasing prevalence of high-rise buildings requires innovations on the design and production of elevator systems.

An elevator group consists of a couple of basic elements. The elevators themselves, the elevator shafts, and the control systems. The control system is responsible for managing the elevator call allocations. In other words, after a person calls an elevator, the control system assigns an elevator to handle that call.

When designing an elevator group for a particular building, one must consider many different aspects. For example, the number of floors, energy efficiency, tenant specific considerations, and elevator cost. When performing this design analysis, one must also consider the capacity of the elevators. The capacity must be such that the elevator group is capable of performing at a suitable efficiency and such that the elevator group system does not get overwhelmed by traffic, leading to a traffic blockage.

To find a suitable elevator group capable of handling a specific level of traffic, a simulator can be used to calculate an estimate of the waiting time and time-to-destination for a hypothetical user. By waiting time, we mean the time between the event that the user calls the elevator system and the event that the elevator system to provides them an elevator. Time to destination is defined to be the sum of waiting time and the time it takes for the elevator to move the person to their desired destination.

By using these metrics, one can analyze the performance of specific elevator group configurations and find a suitable elevator for a given building layout and use. However, one major disadvantage in simulating the traffic for a given elevator group is the fact that it takes many computationally intensive simulations to try different elevator configurations. The end result is that it takes a long time to find a good elevator configuration.

1.2 Research questions

This thesis focuses on a simpler inverse problem of elevator design. Specifically, we want to find the maximum traffic intensity a given elevator system can handle under certain performance constraints. Knowledge of the maximum traffic intensity can then be used in the design and planning of future elevator systems.

More concretely, this thesis will provide answers to two research questions. Firstly, how to precisely mathematically define the problem of finding a maximum traffic intensity, under certain waiting time and time-to-destination constraints, for a specific building and elevator system. Secondly, how to solve the aforementioned problem with as few simulations as possible using a simulator in conjunction with statistical methods.

1.3 Structure

In this introduction chapter, we have discussed the context and general idea of elevator planning and defined our research questions. In the next chapter we discuss elevator planning in detail and examine the relevant literature related to it. In Chapter 3 we define mathematically the problem of finding optimal traffic intensity, answering our first research question. The rest of the thesis is dedicated to solving the formulated problem.

In Chapter 4 we discuss the literature related to Gaussian processes, which serve as an important building block for developing the Bayesian optimization algorithms, which we use to solve the problem. The background on Bayesian optimization is given in Chapter 5. Chapter 6 contains implementation details of our candidate algorithms and their pseudo-code.

The candidate algorithms are then tested using a toy problem in Chapter 7 before being used in Chapter 8 to solve the real maximum traffic problem. Finally, we discuss the results in Chapter 9 and conclude the thesis in Chapter 10. The list of references is also provided at the end. Finally, Appendix A contains additional information related to the algorithms.

2 Elevators

This section discusses the main design problem which we are trying to solve. First we will describe the general steps in elevator planning for an office building and discuss the various aspects which we take into consideration. After sufficient background knowledge, we will explore simulation as an instrumental part in elevator planning and define the main problem of this thesis.

2.1 Planning

As discussed in the introduction, the process of designing elevator systems for an office building is a complex task. However, we can rely on the standard ISO 8100-32 [1] as the main source of information.

As the standard [1] describes, when considering an elevator system for a specific building we must choose a variety of different parameters. The parameters include for example, the number of elevators, elevator speed, and size [1]. These parameters in turn affect the service level of our elevator system and are therefore important to get right for a specific building.

On the other hand, if we are given an elevator system for a building, how can we quantify whether or not it is appropriate? The standard [1] defines a set of key metrics which we can use.

2.2 Performance

The standard defines that a suitable elevator system would be capable of transporting enough people such that the waiting time for those passengers would not exceed a defined threshold [1]. We can also intuitively note that as a user of an elevator system, one would want the elevator to transport them to their desired destination in a reasonable time, but safely. Consequently, this thesis will focus on metrics supporting these goals.

The standard defines two different time metrics: the time it takes for the elevator to first arrive to the floor the user currently occupies and the time it takes it to transport them to their desired destination. The time metrics are called *waiting time* (WT) and *time-to-destination* (TTD) respectively [1]. Figure 2.1 illustrates these definitions using a timeline of a person entering the leaving an elevator system.

As a simple example, the case where the user calls the elevator at 13:00:00 and the elevator arrives at 13:01:30, in that case the waiting time would be 90 seconds. Continuing the example, if the elevator transports the user between times 13:01:30 and 13:02:15 to their desired destination, the time-to-destination would then be the total time difference 13:02:15 - 13:00:00 which is 135 seconds.

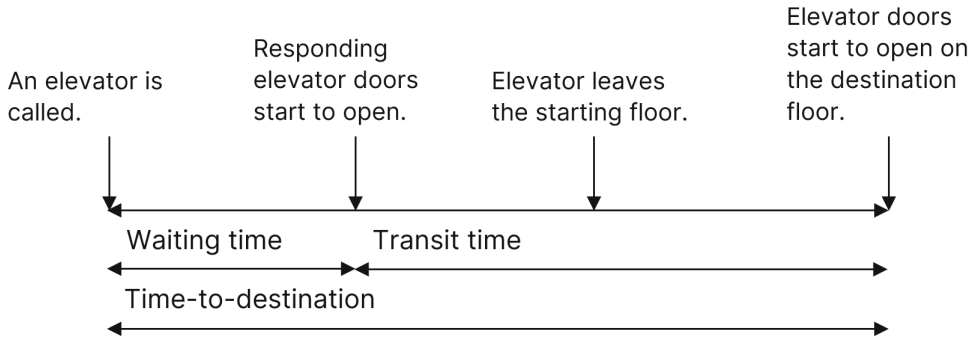


Figure 2.1: Waiting time and time-to-destination depicted on a timeline alongside elevator system events. Adapted from [16].

Furthermore, since we are mainly interested in the overall performance of the elevator system and we intuitively know that the elevator system cannot perform with the same efficiency in all specific traffic situations, the standard considers the averages of these time periods. Namely it considers the average waiting time (AWT) and average time-to-destination (ATTD) [1].

One crucial factor which affects AWT and ATTD is the traffic intensity, the percentage of people in the building entering the elevator system within a 5 minute window. Note that the standard refers to the same quantity as passenger demand [1]. Intuitively, we can think that if the intensity of people arriving to the elevators is high, the more difficult it is for the elevator system to transport them in the same amount of time as with a low traffic intensity. In the next section, we will explore ways of determining whether an elevator system is suitable for a specific building using the concept of traffic intensity.

2.3 Simulation

The standard [1] discusses two different approaches for determining if an elevator system is suitable for a specific building. One approach is an uppeak calculation method where we assume a pure upwards traffic, with a specific intensity, from the entrance floor to other floors with an equal probability. However, as the standard [1] states, there are many constraints when using the uppeak method. The standard deems uppeak method infeasible cases where for example the number of lifts is higher than 8, there are multiple entrance floors, etc. Therefore, instead of the uppeak method we will utilize the second approach proposed in the standard which uses a simulator [1].

Simulators have been used extensively to study elevator systems, for example in [26]. According to the standard, the simulator should use information about the building to estimate AWT values for a specific traffic intensity. The standard [1] defines that the simulations should be run with different traffic mixes, in other words different ratios of inbound, outbound, and interfloor traffic in the building. The standard also gives specific traffic intensities and AWT limits which must be satisfied for the elevator to be determined suitable for the building [1].

The simulation software developed by KONE [27] simulates the passenger traffic

according to a certain traffic profile. The simulator can be used in many different traffic settings and with different control systems. For our purposes we will mainly utilize a 2 hour simulation, however we do also experiment with shorter simulations. Estimates of AWT and ATTD are computed using simulated passengers. The main input to the simulation is the traffic intensity. Other inputs to the simulator include a random seed for reproducibility and data about the building and elevator system such as the number of elevator shafts, speeds of the elevators, elevator door opening and closing times, number of floors in the building, and heights of the floors. The detailed building and elevator data is important for the simulator to create a realistic simulation and therefore accurately estimate the AWT and ATTD values.

2.4 Goal

We have now sufficient background knowledge to discuss the main problem of this thesis. As stated in the introduction, we want to find a suitable method for finding the optimal traffic intensity for a given building.

More specifically, the goal is to use the simulator as a source of AWT and ATTD values to find the maximum traffic intensity for an elevator system which does not violate predetermined AWT and ATTD limits. This is analogous to an inverse problem of elevator planning. Instead of finding a suitable elevator system for a building and which satisfies AWT and ATTD limits for a set traffic intensity, our goal is to find the maximum traffic intensity for a given building and elevator system which satisfies the AWT and ATTD limits. Note that while the standard [1] does not require the use of ATTD values for determining if an elevator system is suitable, we have an ATTD limit in our problem.

Practically, we use the simulator to estimate AWT and ATTD values for different traffic intensities and determine which is the maximum such that AWT and ATTD limits are not violated. However, as the simulator only calculates an estimate of AWT and ATTD values, we must use statistical methods to determine if a given intensity is feasible or not. Furthermore, as the simulation software is computationally intensive, we must be conservative with the number of simulations we run.

The goal is therefore to find the maximum traffic intensity for a given building and elevator system, using as few estimates of the AWT and ATTD values as possible while simultaneously having some guarantee of the validity of the result. We mathematically formalize the problem in the following chapter.

3 Problem definition

To precisely and mathematically define the main problem of this thesis, we utilize the work of Alexandris et al. for defining useful notions.

Definition 3.1 (Intensity)**.** We define traffic intensity (often referred to as just "intensity") p to be the average number of passengers in a 5-minute interval. p is considered a percentage value of the total number of people in the building. For example, $p = 12$ would correspond to the situation where 12% of the total population of the building would be equal to the number of elevator passengers in a 5-minute time interval.

Definition 3.2 (Traffic mix)**.** A traffic mix contains information about how the traffic is split into outbound, inbound, and interfloor traffic. By inbound, we mean traffic flowing into the building from entrance floors, outbound is the traffic flowing out from entrance floors and interfloor is the traffic between the floors. For example one traffic mix could be 40%, 40%, and 20% for inbound, outbound, and interfloor traffic.

Definition 3.3 (Steady-state elevator system, inspired by [2])**.** The steady-state elevator system is a hypothetical elevator system which we define as running continuously and which is subjected to passenger traffic with a certain intensity (Definition 3.1) and a certain traffic mix (Definition 3.2).

Definition 3.4 (Average waiting time, inspired by [1, 2])**.** The average waiting time $T_{\text{AWT}}(p, S, M)$ for an intensity p (Definition 3.1) corresponds to the long-time average waiting time occurring in a steady-state elevator system (Definition 3.3) during traffic with intensity p (Definition 3.1) and mix M (Definition 3.2).

Definition 3.5 (Average time-to-destination, inspired by [1, 2])**.** The average time-to-destination $T_{\text{ATTD}}(p, S, M)$ for an intensity p (Definition 3.1) corresponds to the long-time average time-to-destination in a steady-state elevator system (Definition 3.3) during traffic with intensity p (Definition 3.1) and mix M (Definition 3.2).

The mathematical definitions for AWT and ATTD are inspired by the work of Alexandris et al. [2]. We use the notion of a steady-state system when analyzing the elevator systems. Using a steady-state elevator system, we define the AWT and ATTD values to be computed from that hypothetical elevator system which runs continuously with a specific traffic intensity and traffic mix. Intuitively, we want to our definitions of AWT and ATTD values to correspond to values which would be typical for the elevator system when subjected to a specific traffic intensity and mix.

With these definitions, we can define the main problem of this thesis as

Definition 3.6 (Maximum traffic intensity problem)**.**

$$\begin{aligned} &\max. && p \\ &\text{such that:} && \\ & && T_{\text{AWT}}(p, S, M) \leq C_{AWT} \\ & && T_{\text{ATTD}}(p, S, M) \leq C_{ATTD} \\ & && p \in \mathbb{R}^{+} \cup \{0\} \end{aligned}$$

where $T_{\mathrm{AWT}}(p, S, M)$ and $T_{\mathrm{ATTD}}(p, S, M)$ are the average waiting time and average time-to-destination values for the elevator system S with intensity p and traffic mix M. The bounds for AWT and ATTD are formalized as the quantities C_{AWT} and C_{ATTD} respectively.

Evaluating the objective function p is trivial. The main challenge is in the functions T_{WT} and T_{TTD}. Assuming a fixed elevator system S and traffic mix M, we cannot know the AWT and ATTD values for an intensity p due to the complexity of the elevator system. Intuitively, to know these values precisely, we would have to analyze the elevator system and examine the waiting times and time-to-destination for a very long time before we would have an accurate view on the distributions of AWT and ATTD. This is because for a general traffic mix, the steady-state elevator system is only a hypothetical scenario. Real-world elevator systems behave more chaotically due to the complex nature of their control systems.

However, we use the simulator [27] to sample AWT and ATTD estimates for a specific traffic intensity p. We hope that while the true steady-state AWT and ATTD values are hard to capture, we can still estimate them using a simulation with finite length. Simulating a range of intensity values is however computationally intensive, therefore we cannot for example use a simple grid search along some interval $[a, b]$ to hopefully find a good estimate for optimal intensity p. Luckily, we can use Bayesian optimization with Gaussian processes [23, 8] to create an algorithm for picking an optimal intensity p with as few simulations as possible. We will discuss them in the next Chapters.

4 Gaussian process regression

Gaussian processes have been studied for a long time, the theory of stochastic processes dating back to the times of Kolmogorov and Wiener [25, 30]. Today, they are mostly considered a part of of supervised machine learning and have been used in a variety of different contexts, including stellar astrophysics [6] and space weather [4].

Gaussian processes are commonly applied in regression problems, problems where we have a continuous input variable $\mathbf{x}$ and try to predict some $y = f(\mathbf{x})$ for some unknown function f [23]. The main idea is to model the underlying function by fitting a Gaussian process regression model using some dataset $\mathcal{D}_f$ (Definition. 4.1). Definition 4.3 defines a Gaussian process.

Definition 4.1 (Dataset)**.** We define a dataset $\mathcal{D}_f = (\mathbf{X}, \mathbf{y})$ to be a pair of input and output samples. The input samples $\mathbf{x}_i \in \mathbf{X}$ are members of $\mathbb{R}^M$ and the output samples $\mathbf{y}_i$ are members of $\mathbb{R}$. The input matrix $\mathbf{X}$ is therefore a member of $\mathbb{R}^{N \times M}$ where N is the number of input samples in the dataset $\mathcal{D}_f$. The output vector $\mathbf{y}$ is similarly a vector in $\mathbb{R}^N$.

A dataset is also related to a function $f : \mathbb{R}^M \to \mathbb{R}$. We define the outputs by requiring that $\mathbf{y}$ is the image of function applied on input samples $\mathbf{X}$. Formally, we define

$$\mathbf{y}_i = f(\mathbf{x}_i)$$

Note that we denote by $\mathbf{x}_i \in \mathbf{X}$ a sample $\mathbf{x}_i$ being a row at index i of the matrix $\mathbf{X}$. We will use similar notation for other matrices which contain samples.

Definition 4.2 (Noisy dataset)**.** A noisy dataset $\mathcal{D}_f = (\mathbf{X}, \mathbf{y})$ is similar to a regular dataset as defined in Definition 4.1 with one key difference. Specifically, we allow some noise in the evaluation of the function at the input sample points $\mathbf{X}$. Formally, we define

$$\mathbf{y}_i = f(\mathbf{x}_i) + \epsilon$$

where $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon)$ is the error. We assume the error to be independently and normally distributed with mean of 0 and non-zero standard deviation of σ_ϵ for each input sample $\mathbf{x}_i \in \mathbf{X}$.

Definition 4.3 (Gaussian process, from [23])**.** We define a *Gaussian process* to be a collection of real-valued random variables, any finite number of which having a joint normal distribution. The mean and covariance functions of a Gaussian process $f : \mathcal{X} \to \mathbb{R}$ are defined to be

$$\begin{aligned} \mu(\mathbf{x}) &:= \mathbb{E}[f(\mathbf{x})] \\ \Sigma(\mathbf{x}, \mathbf{x'}) &:= \mathbb{E}[(f(\mathbf{x}) - \mu(\mathbf{x})(f(\mathbf{x'}) - \mu(\mathbf{x'})] \end{aligned}$$

We shall also denote the Gaussian process as $f(\mathbf{x}) \sim \mathcal{GP}(\mu(\mathbf{x}), \Sigma(\mathbf{x}, \mathbf{x'}))$. The set $\mathcal{X}$ contains the points $\mathbf{x}$ for which the random variable $f(\mathbf{x})$ is defined. The distribution of a Gaussian process f is (almost) uniquely characterized by the specific mean and covariance functions, μ and Σ.

There are two ways of looking at Gaussian processes, the weight space and function space views [23]. A connection between them can be made, however to introduce the topic a weight space approach would be a more friendly introduction [23]. We will therefore repeat the main points in the introduction to Gaussian processes using the weight space approach as described in [23].

To begin, let us introduce a basic linear function regression

$$f(\mathbf{x}) = \phi(\mathbf{x})^T \mathbf{w}$$

where $\phi : \mathcal{X} \to \mathbb{R}^D$ is a feature map, mapping the input $\mathbf{x}$ into some feature space of dimension D. For example, for a scalar input $\mathbf{x} = x \in \mathbb{R}$, one possible feature map would be to consider the exponentiation of the original input, $\phi : x \mapsto [1, x, x^2, x^3, \ldots x^{D-1}]^T$. Choosing a suitable feature map ϕ allows us to perform a linear regression without worrying if the original data fits a linear function [23]. The vector $\mathbf{w} \in \mathbb{R}^D$ contains the weights for each feature. We place a prior distribution on the weight vector $\mathbf{w} \sim \mathcal{N}(0, \Sigma_p)$, a multivariate normal distribution with mean of 0 and Σ_p covariance matrix.

We define $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \Sigma_p \phi(\mathbf{x}')$ as the covariance function, also known as the kernel. The connection to a Gaussian process can be made evident by noting that $f(\mathbf{x}) = \phi(\mathbf{x})^T \mathbf{w}$ as a random variable has an expectation of 0

$$\mathbb{E}[f(\mathbf{x})] = \phi(\mathbf{x})^T \mathbb{E}[\mathbf{w}] = 0$$

and the covariance of $f(\mathbf{x})$ and $f(\mathbf{x}')$ is the kernel function value $k(\mathbf{x}, \mathbf{x}')$

$$\operatorname{cov}(f(\mathbf{x}) f(\mathbf{x}')) = \mathbb{E}[f(\mathbf{x}) f(\mathbf{x}')] = \phi(\mathbf{x})^T \mathbb{E}[\mathbf{w}\mathbf{w}^T] \phi(\mathbf{x}') = \phi(\mathbf{x})^T \Sigma_p \phi(\mathbf{x}') = k(\mathbf{x}, \mathbf{x}')$$

We note that f is a Gaussian process $f \sim \mathcal{GP}(0, \Sigma_k)$. Σ_k is the Gaussian process covariance matrix computed by using the kernel k, the elements of Σ_k are $k(\mathbf{x}, \mathbf{x}'), \mathbf{x}, \mathbf{x}' \in \mathcal{X}$ [23].

If we have a dataset $\mathcal{D}_f$ of function f samples, we can compute the posterior Gaussian process using Bayesian formalism. In Bayesian formalism, we not only consider some point estimates of the weights $\mathbf{w}$, but a distribution over the parameters [23]. As $\mathbf{w} \sim \mathcal{N}(0, \Sigma_p)$ is the prior distribution on the weights, we can use Bayes' rule to compute the posterior distribution density p of the weights $\mathbf{w}$

$$p[\mathbf{w}|\mathbf{X}, \mathbf{y}] = \frac{p[\mathbf{y}|\mathbf{X}, \mathbf{w}] p[\mathbf{w}]}{p[\mathbf{y}|\mathbf{X}]}$$

where $p[\mathbf{y}|\mathbf{X}]$ is a normalizing constant [23]. The vector $\mathbf{y}$ is the outputs of dataset $\mathcal{D}_f$ with inputs $\mathbf{X}$, $\mathcal{D}_f = (\mathbf{X}, \mathbf{y})$ as per Definition 4.1. Assuming we have the feature map ϕ, we can calculate the posterior distribution of the weights using matrix $\Phi = \phi(\mathbf{X})$, the feature map computed for each input sample $\mathbf{x} \in \mathbf{X}$. The posterior distribution density for $\mathbf{w}$ is,

$$p[\mathbf{w}|\mathbf{X}, \mathbf{y}] \sim \mathcal{N}(\bar{\mathbf{w}}, \bar{\Sigma})$$

where $\bar{\mathbf{w}} = \frac{1}{\sigma_\epsilon^2}(\sigma_\epsilon^{-2}\mathbf{\Phi}\mathbf{\Phi}^T + \Sigma_p^{-1})^{-1}\mathbf{\Phi}\mathbf{y}$ and $\bar{\Sigma} = (\sigma_\epsilon^{-2}\mathbf{\Phi}\mathbf{\Phi}^T + \Sigma_p^{-1})^{-1}$ [23]. Now that we have the formula for the posterior distribution of the weights $\mathbf{w}$, we can predict the function value at a new point $\mathbf{x}$,

$$\mathbb{P}[f(\mathbf{x}) = y|\mathbf{x}, \mathbf{X}, \mathbf{y}] = \int \mathbb{P}[f(\mathbf{x}) = y|\mathbf{x}, \mathbf{w}]p[\mathbf{w}|\mathbf{X}, \mathbf{y}]\mathrm{d}\mathbf{w}$$

The resulting posterior is again a normal distribution

$$f(\mathbf{x})|\mathcal{D}_f \sim \mathcal{N}(\sigma_\epsilon^{-2}\phi(\mathbf{x})^T A^{-1}\mathbf{\Phi}\mathbf{y}, \phi(\mathbf{x})^T A^{-1}\phi(\mathbf{x}))$$

for which $A = \sigma_\epsilon^{-2}\mathbf{\Phi}\mathbf{\Phi}^T + \Sigma_p^{-1}$ [23]. We can rewrite the posterior as having mean of

$$\phi(\mathbf{x})^T\Sigma_p\mathbf{\Phi}(K + \sigma_\epsilon^2 I)^{-1}\mathbf{y}$$

and covariance matrix

$$\phi(\mathbf{x})^T\Sigma_p\phi(\mathbf{x}) - \phi(\mathbf{x})^T\Sigma_p\mathbf{\Phi}(K + \sigma_\epsilon^2 I)^{-1}\mathbf{\Phi}^T\Sigma_p\phi(\mathbf{x})$$

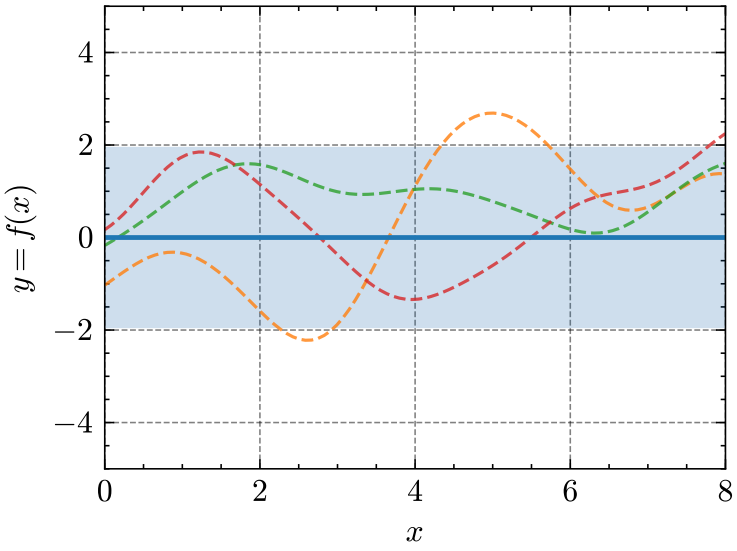
when setting $K = \mathbf{\Phi}^T\Sigma_p\mathbf{\Phi}$. The vector $\mathbf{k}(\mathbf{x})$ is defined to contain values $k(\mathbf{x}, \mathbf{x}_i)$ for samples $\mathbf{x}_i \in \mathbf{X}$. Theorems 4.1 and 4.2 encapsulate the formalism of computing a posterior Gaussian process using a dataset.

Theorem 4.1 (Gaussian process posterior, from [23])**.** *Given a Gaussian process $f_p \sim \mathcal{GP}(0, \Sigma_k)$ and a noiseless dataset $\mathcal{D}_f = (\mathbf{X}, \mathbf{y})$ (Definition 4.1). Then, the posterior process $f|\mathcal{D}_f \sim \mathcal{GP}(\mu, \Sigma)$, which is also a Gaussian process, has the following mean and variance functions:*

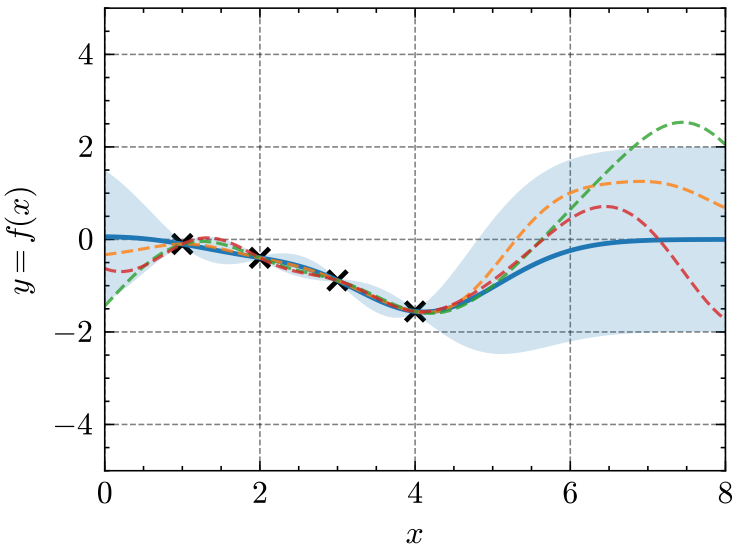
$$\mu(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T K^{-1}\mathbf{y}$$
$$\mathrm{var}(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) - \mathbf{k}(\mathbf{x})^T K\mathbf{k}(\mathbf{x}) = \Sigma(\mathbf{x}, \mathbf{x})$$

where K is the covariance matrix. K contains the values of the kernel k evaluated at training samples $\mathbf{X}$. The vector $\mathbf{k}(\mathbf{x})$ contains the covariances of the input point $\mathbf{x}$ and the points in the input sample set $\mathbf{X}$.

According to Theorem 4.1, after conditioning on the sampled points in the dataset $\mathcal{D}_f$, the corresponding posterior process is a Gaussian process with specific mean and covariance structures. Figure 4.1 demonstrates the prior and posterior distributions for an unknown function f when conditioning on five samples.



(a) Example of a prior distribution. The shaded area corresponds to a 95% confidence region. The lines correspond to three random functions (Gaussian processes) drawn from the prior distribution of f.



(b) Example of a posterior distribution. The random samples and confidence intervals behave similarly to the prior distribution case. However, now we draw from the posterior distribution of f given 4 noiseless samples of the function f, marked in black. The specific function used is $f(x) = -x \sin x$. The input points are $x \in \{1, 2, 3, 4\}$.

Figure 4.1: Prior and posterior distributions. Adapted from [23].

Theorem 4.2 (Noisy GP posterior, from [23])**.** *According to [23], with the basic setup as in Theorem 4.1, and a noisy dataset $\mathcal{D}_f$ (Definition 4.2), the posterior Gaussian process $f|\mathcal{D}_f \sim \mathcal{GP}(\mu, \Sigma)$ has the following mean and variance functions:*

$$\mu(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T (K + \sigma_\epsilon^2 I)^{-1} \mathbf{y}$$
$$\text{var}(\mathbf{x}) = \Sigma_p(\mathbf{x}, \mathbf{x}) - \mathbf{k}(\mathbf{x})^T (K + \sigma_\epsilon^2 I)\mathbf{k}(\mathbf{x}) = \Sigma(\mathbf{x}, \mathbf{x})$$

Theorem 4.1 assumes that the training dataset is non-noisy, with a noisy dataset $\mathcal{D}_f = (\mathbf{X}, \mathbf{y})$ (Definition 4.1), we need to simply add the noise to the covariance matrix K. K corresponds to the original covariance matrix evaluated at the samples $\mathbf{x}_i \in \mathbf{X}$ and $\sigma_\epsilon^2 I$ forms the additive noise by increasing the diagonal entries of K by the noise level σ_ϵ^2. Theorem 4.2 encapsulates the case of using a noisy dataset $\mathcal{D}_f$ when computing the posterior distribution $f|\mathcal{D}_f$. Note that to use Theorem 4.2 we must know the noise level σ_ϵ in our samples. However, in practice we estimate the noise σ_ϵ as an additional hyperparameter alongside the kernel hyperparameters, Section 4.2 discusses the optimization of kernel hyperparameters.

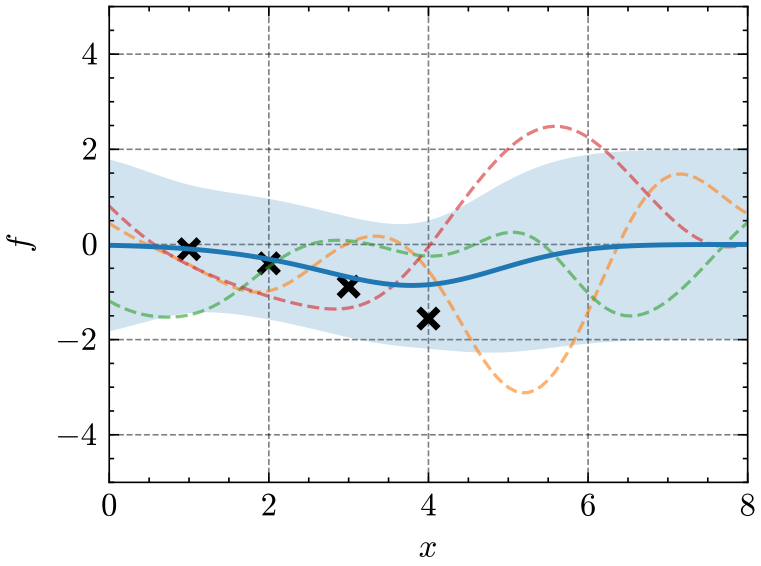
We have to be precise when talking about the posterior process $f|\mathcal{D}_f$ versus the predictive distribution of a new sample y. Using a noisy dataset $\mathcal{D}_f$ (Definition 4.2), predicting the true function f is achieved by using the posterior process $f|\mathcal{D}_f$ computed using Theorem 4.2. The prediction of a new sample y is made by adding the noise value ϵ to the value of f and treating the result as a random variable y, formally

$$y = f(\mathbf{x}) + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma_\epsilon)$$

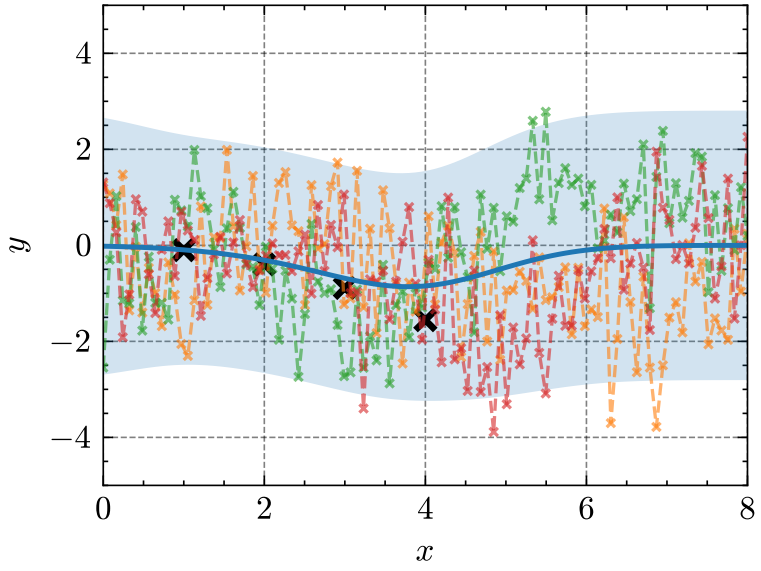
Conditioning on the dataset $\mathcal{D}_f$ which we use to model $f \sim \mathcal{GP}(\mu, \Sigma)$ as a Gaussian process, the probability distribution of random variable y is a normal distribution with specific mean and variance. Formally the conditional distribution of y is,

$$y|\mathcal{D}_f \sim \mathcal{N}(\mu(\mathbf{x}), \Sigma(\mathbf{x}, \mathbf{x}) + \sigma_\epsilon^2)$$

where σ_ϵ is the additive noise observed in the samples [23]. Figure 4.2 indicates the difference between the distributions $f|\mathcal{D}_f$ and $y|\mathcal{D}_f$. We note that with a noiseless $\mathcal{D}_f$ dataset, the function prediction no uncertainty at sample locations (at inputs $\mathbf{x}_i \in \mathbf{X}$). Intuitively, we can think that since the samples are assumed to be non-noisy, we have all of the information required to predict the true function f value at those input points. This is in contrast to the case where we have noisy samples, the variance of the posterior $f|\mathcal{D}_f$ is greater than zero even at points $\mathbf{x}_i \in \mathbf{X}$. This reflects the fact that sampling at a point does not give us enough information to perfectly predict the true function f at those points.



(a) Latent function f prediction of a GP with a noisy kernel.



(b) Predicting a noisy sample y, $y = f(x) + \epsilon$ with a Gaussian process model. The smaller crosses indicate possible values for the noisy sample y.

Figure 4.2: The posterior of the latent function f and the posterior of a noisy sample y modelled with Gaussian processes. The 95% confidence region is shown in the shaded area. Blue solid line indicates the mean of the posterior process, training samples marked in black. Orange, red, and green dashed lines are samples from the posterior distributions. Dataset used is the same as in Figure 4.1

For ease of notation and to make following algorithms easier to understand, we define a *model* $\mathcal{M}_f$ to encapsulate the posterior and the kernel in Definition 4.4.

Definition 4.4 (Model)**.** A model $\mathcal{M}_f$ is the encapsulation of the posterior process with the kernel used in the creation of the posterior. Formally a model is a collection $\mathcal{M}_f = (\mu, \Sigma, k)$, where μ and Σ are the posterior process mean and covariance functions and k is the kernel used. For notation, we define $\mathcal{M}_f.\mu = \mu$, $\mathcal{M}_f.\Sigma = \Sigma$, and $\mathcal{M}_f.k = k$.

4.1 Kernels & Covariance matrices

For the kernel functions of the Gaussian processes, we will describe a few commonly used kernel functions which include the squared exponential kernel, also known as the Gaussian kernel, and a subset of the Matérn kernels. The definitions for these kernels are adopted from the book by Rasmussen and Williams [23], with additions from [7].

Definition 4.5 (Squared exponential kernel, adapted from [23, 7])**.** The Gaussian kernel or the squared exponential kernel is defined to be the following.

$$k(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2l^2})$$

where $\mathbf{x} \in \mathbb{R}^M$. The parameter l is the characteristic length-scale of the kernel. The parameter σ is the scale parameter in the output space. This kernel is also known as the Radial Basis Function kernel or RBF kernel.

Definition 4.6 (An isotropic kernel, from [23])**.** A kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is isotropic, if for all pairs $(\mathbf{x}, \mathbf{x}') \in \mathcal{X} \times \mathcal{X}$ the value of k depends only on the distance between the points. Formally $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{y}, \mathbf{y}') \iff \|\mathbf{x} - \mathbf{x}'\| = \|\mathbf{y} - \mathbf{y}'\|$, where $\|\cdot\|$ denotes some norm. In this thesis, the norm $\|\cdot\|$ is the standard Euclidean norm.

The Figure 4.3 illustrates the form of the squared exponential kernel (Def. 4.5), we can plot the value of the kernel as a graph of a scalar input r by defining $r = \|\mathbf{x} - \mathbf{x}'\|$ since the value of the kernel depends only on the distance between the input points. Kernels with this property are called isotropic [23]. Due to their simplicity and robustness, we will only explore some of most common isotropic kernels, Definition 4.6 defines the notion of an isotropic kernel.

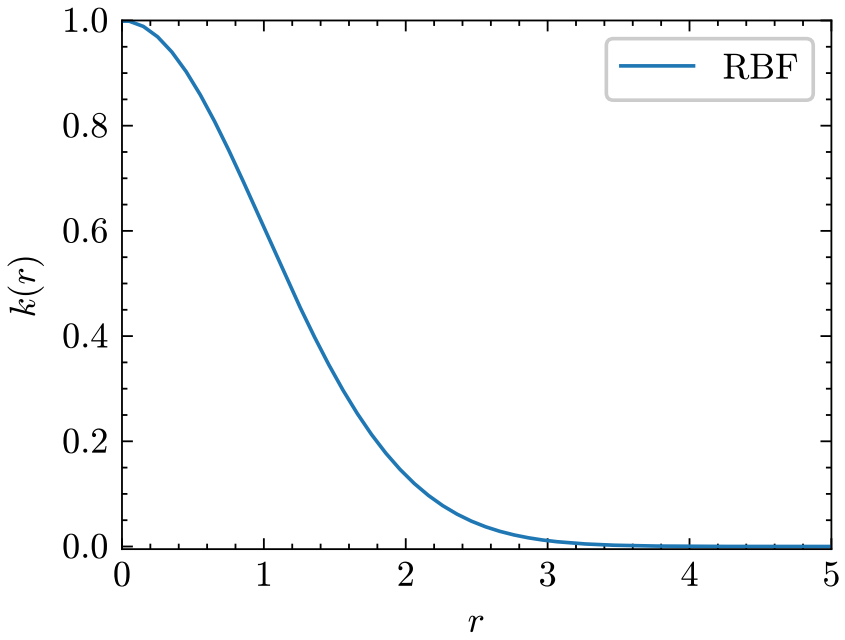


Figure 4.3: Squared exponential kernel value $k(\mathbf{x}, \mathbf{x}')$ as a function of the distance $r = \|\mathbf{x} - \mathbf{x}'\|$ between input points, since RBF is isotropic. Hyperparameters are set to $\sigma = 1, l = 1$.

We also experiment with Matérn kernels corresponding to once and twice differentiable processes. These kernels are defined in Definition 4.7. These kernels are also isotropic. Figure 4.4 illustrates the values for the two Matérn kernels.

Definition 4.7 (Matérn kernels, adapted from [23])**.** The general Matérn kernel is of form

$$k(\mathbf{x}, \mathbf{x}') := \sigma^2 \frac{2^{\nu-1}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}}{l} \|\mathbf{x} - \mathbf{x}'\| \right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu}}{l} \|\mathbf{x} - \mathbf{x}'\| \right)$$

where ν is a positive parameter and l is the associated length scale parameter. Γ is the Gamma function and K_ν is a modified Bessel function. The parameter σ is the scale parameter in the output space.

From the general form, we obtain kernels for once and twice differentiable processes by setting $\nu = 3/2$ and $\nu = 5/2$ respectively. For half-integer values of ν the general form becomes simpler to work with [23]. The aforementioned kernels are,

$$k_{\nu=3/2}(\mathbf{x}, \mathbf{x}') = \sigma^2 \left(1 + \frac{\sqrt{3}}{l} \|\mathbf{x} - \mathbf{x}'\| \right) \exp \left(\frac{-\sqrt{3}}{l} \|\mathbf{x} - \mathbf{x}'\| \right)$$

$$k_{\nu=5/2}(\mathbf{x}, \mathbf{x}') = \sigma^2 \left(1 + \frac{\sqrt{5}}{l} \|\mathbf{x} - \mathbf{x}'\| + \frac{5}{3l^2} \|\mathbf{x} - \mathbf{x}'\|^2 \right) \exp \left(\frac{-\sqrt{5}}{l} \|\mathbf{x} - \mathbf{x}'\| \right)$$

where again the parameter σ is a output scaling factor and l is the characteristic length-scale.

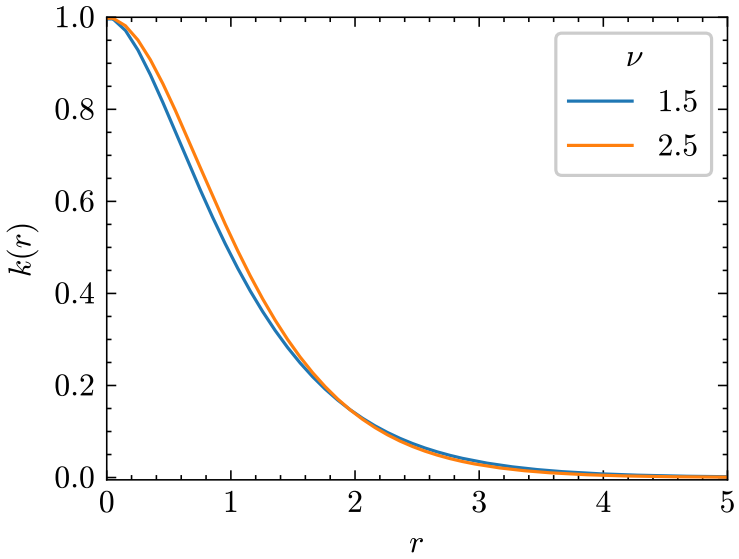


Figure 4.4: Matérn kernel value $k(r)$, for $\nu = 3/2$ and $\nu = 5/2$ values.

4.2 Hyperparameter optimization

The common hyperparameters are the output scaling factor σ and the characteristic length-scale l. Given a dataset $\mathcal{D}_f = (\mathbf{X}, \mathbf{y})$ with N samples, to find the best hyperparameters for describing the data we utilize marginal likelihood estimation (MLE). In this section, we will describe the process of finding optimal parameters with MLE.

To find appropriate models using these kernels we need to find suitable hyperparameter values θ. We find optimal values θ^* for kernel k using log marginal likelihood estimation. From [23] we note the log marginal likelihood of a Gaussian process model as being

$$\log \mathbb{P}[\mathbf{y}|\mathbf{X}, \theta, \sigma_\epsilon] = -\frac{1}{2}\mathbf{y}^T K_y(\theta, \sigma_\epsilon)^{-1}\mathbf{y} - \frac{1}{2}\log|K_y(\theta, \sigma_\epsilon)| - \frac{N \log 2\pi}{2}$$

where $K_y(\theta, \sigma_\epsilon) = K(\theta) + \sigma_\epsilon^2 I$ is the covariance matrix for noisy function predictions $\mathbf{y}$ computed from the kernel matrix $K(\theta), K(\theta)_{ij} = k_\theta(\mathbf{x}_i, \mathbf{x}_j)$ by adding the specified noise σ_ϵ level to the diagonal entries. Figure 4.5 illustrates the log marginal likelihood values for different values of two hyperparameters. We have formalized the finding of optimal hyperparameters as fitting the model using data in Algorithm 4.1.

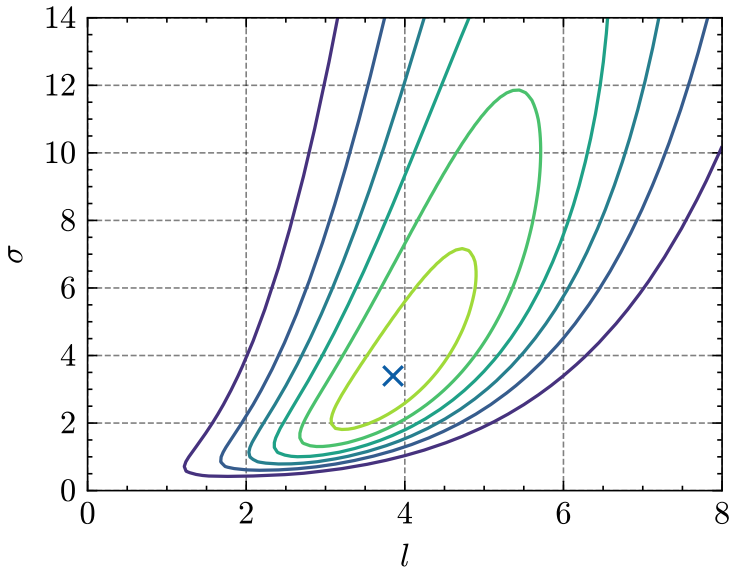


Figure 4.5: A contour plot of log marginal likelihood when optimizing hyperparameters for a RBF kernel. Optimizing the σ scaling factor and length-scale parameter l. Log marginal likelihood increases towards the green-yellow color with the optimum marked as a blue cross. Using the same dataset as in Figure 4.1

Algorithm 4.1 Model fitting algorithm

Input: Dataset $\mathcal{D}_f$ and kernel k_θ parameterized by hyperparameter values θ.
Output: Model $\mathcal{M}_f$.

Dataset $\mathcal{D}_f = (\mathbf{X}, \mathbf{y})$.
if Dataset $\mathcal{D}_f$ is noisy **then**
    Optimize hyperparameters and noise $\theta^*, \sigma_\epsilon^* = \arg\max_{\theta,\sigma_\epsilon} \log \mathbb{P}\left[\mathbf{y}|\mathbf{X}, \theta, \sigma_\epsilon\right]$ using kernel k_θ.
    Compute $f|\mathcal{D}_f \sim \mathcal{GP}(\mu, \Sigma)$ using Theorem 4.2 with kernel k_{θ^*} and noise σ_ϵ^*.
else
    Optimize hyperparameters $\theta^* = \arg\max_{\theta,\sigma_\epsilon} \log \mathbb{P}\left[\mathbf{y}|\mathbf{X}, \theta\right]$ using kernel k_θ.
    Compute $f|\mathcal{D}_f \sim \mathcal{GP}(\mu, \Sigma)$ using Theorem 4.1 with kernel k_{θ^*}.
end if
Create a model $\mathcal{M}_f = (\mu, \Sigma, k_f)$ using Definition 4.4.
return $\mathcal{M}_f$

In practical terms, to optimize the difference hyperparameters, we will utilize a common optimization technique L-BFGS-B which is a quasi-Newton method for solving constrained nonlinear optimization problems [3], implemented in the GPy-library [14]. For numerical stability reasons, we add a small noise level to the covariance matrix K even in the case of not having noisy datasets and using Theorem 4.1. The exact value of this stability noise is in Appendix A.

5 Bayesian optimization

Gaussian processes provide a great framework for modeling functions based on few samples. However, as our original problem is to optimize the intensity p, we need to shift our focus to optimization.

Bayesian optimization provides a toolkit for optimizing black-box functions [8]. By a black-box, we mean a function $f : \mathcal{X} \to \mathbb{R}$ which is computationally intensive to evaluate for a given input $\mathbf{x} \in \mathcal{X}$, for example our elevator traffic simulation computation for AWT at an intensity p. Bayesian optimization, like Gaussian processes has been studied for a relatively long time and earliest related papers circulated in the 1960s and 1970s [19, 31]. It has been used in many different applications, for example in chemistry [10] and drug development [24].

In this chapter, we will first introduce the general algorithm of Bayesian optimization, which we will then augment to handle constraints and noisy datasets. From literature, we will then introduce two algorithms which we will use in our own algorithms which we will implement in Chapter 6.

5.1 Basics

The basic framework of Bayesian optimization is displayed in Algorithm 5.1. The Algorithm takes as input the objective function f, the kernel k for Gaussian process and an initial dataset $\mathcal{D}_f$. This algorithm can be used to solve optimization problems of the form

$$\min_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x})$$

where $\mathcal{X}$ is some simple set for which membership of a point is cheap to evaluate, for example a rectangle in $\mathbb{R}^M$, $\mathbf{c}_1 \leq \mathbf{x} \leq \mathbf{c}_2$ [8]. In our thesis the set $\mathcal{X}$ is a subset of the real line $\mathbb{R}$.

Algorithm 5.1 High-level algorithm of Bayesian optimization, adapted from [8].

Input: The objective function f, kernel k with set of hyperparameters θ_{k_f}, and set of initial dataset $\mathcal{D}_f = (\mathbf{X}, \mathbf{y}), \mathbf{X} \in \mathbb{R}^{N_0 \times M}, \mathbf{y} \in \mathbb{R}^{N_0}$. Acquisition function AF, optimal point discovery function OP, and stopping criterion function CR. The underlying input set $\mathcal{X}$.

Output: The pair of optimal input and output values $(\mathbf{x}^*, f^*)$.

Set $n = N_0$.
while $\mathrm{CR}(\mathrm{AF}_{\max}, n) = 0$ **do**
    $\mathcal{M}_f = \mathrm{Fit}(\mathcal{D}_f, k_f, \theta_k)$ ▷ Model fitting using Algorithm 4.1.
    $\mathrm{AF}_{\max} = \max_{x \in \mathcal{X}} \mathrm{AF}(\mathbf{x}, \mathcal{D}_f, \mathcal{M}_f)$
    $\mathbf{x}_{n+1} = \arg\max_{\mathbf{x} \in \mathcal{X}} \mathrm{AF}(\mathbf{x}, \mathcal{D}_f, \mathcal{M}_f)$
    Create a new sample $y_{n+1} = f(\mathbf{x}_{n+1})$ of objective function f at $\mathbf{x}_n$
    Add the new input-output pair $(\mathbf{x}_{n+1}, y_{n+1})$ into dataset $\mathcal{D}_f$
    Increment $n = n + 1$
end while
$(\mathbf{x}^*, f^*) = \mathrm{OP}(\mathcal{D}_f, \mathcal{M}_f)$
return $(\mathbf{x}^*, f^*)$

One of the most important components of the Bayesian optimization method is to choose an *acquisition function* (AF). The acquisition function is responsible for choosing which input point $\mathbf{x} \in \mathcal{X}$ is sampled next in the optimization process. In other words, the acquisition function should model how much it is worth for us to sample a specific point next in our optimization. The acquisition function should take a candidate point $\mathbf{x}$, one or more datasets, and one or more Gaussian process models as inputs. The function should then return some value for each point in $\mathcal{X}$ and the point to be sampled next should be the point where the acquisition function attains its global maximum. Although in a general setting, finding the global optimum for the acquisition function could be challenging. One of the commonly used acquisition functions is *expected improvement* which is defined in Definition 5.1 [8, 11, 20, 18]. Other commonly used acquisition functions include for example Knowledge Gradient and Entropy Search [9, 17].

Definition 5.1 (Expected improvement, from [8])**.** Using a dataset $\mathcal{D}_f$, and model $\mathcal{M}_f$, the expected improvement (EI) at candidate point $\mathbf{x} \in \mathcal{X}$ is defined as

$$\mathrm{EI}(\mathbf{x}) = \mathbb{E}[\max(f^* - f(\mathbf{x}), 0) | \mathcal{M}_f]$$

The improvement between the Gaussian process value at $\mathbf{x}$ and the currently optimal value $f^* = \min_{\mathbf{x} \in \mathbf{X}} f(\mathbf{x}), \mathcal{D}_f = (\mathbf{X}, \mathbf{y})$. Note that we define the expectation as being conditioned on the model $\mathcal{M}_f$ which contains the posterior distribution of $f(\mathbf{x})$.

Intuitively, expected improvement, as defined in Definition 5.1, measures the expected improvement between the currently optimum value f^* and some candidate value which is predicted using a Gaussian process model $\mathcal{M}_f$. Essentially, EI answers

the question: "How much my objective function f improves at input point $\mathbf{x}$ compared to the currently best value f^*?".

To compute expected improvement as defined in Definition 5.1 we can use integration by parts, as described in [18]. Using integration by parts, we can express the expectation in an analytical form, this result is shown in Theorem 5.1.

Theorem 5.1 (from [8])**.**

$$EI(\mathbf{x}) = \Delta(x)\Phi\left(\frac{\Delta(x)}{\sigma(\mathbf{x})}\right) + \sigma(\mathbf{x})\phi\left(\frac{\Delta(x)}{\sigma(\mathbf{x})}\right) \tag{1}$$

where $\Delta(\mathbf{x}) = f^* - \mu(\mathbf{x})$ *is the difference between the best observed function value and the mean* $\mathcal{M}_f.\mu$ *of the posterior distribution at the candidate point* $\mathbf{x}$*.* $\sigma(\mathbf{x})^2 = \mathcal{M}_f.\Sigma(\mathbf{x}, \mathbf{x})$ *denotes the posterior variance at* $\mathbf{x}$*.* $\sigma(\mathbf{x}) = \sqrt{\sigma(\mathbf{x})^2}$ *is the posterior standard deviation.* ϕ *is here the probability density function for a standard normal and* Φ *is the corresponding cumulative density function.*

Other aspects of the Algorithm 5.1 include the choice of stopping criterion $\mathrm{CR}(\mathrm{AF}_{\max}, n)$ and optimal point discovery function $\mathrm{OP}(\mathcal{D}_f, \mathcal{M}_f)$. The stopping criterion function takes as input the current maximum acquisition function value and the iteration number n. We utilize the acquisition function value as our stopping criterion,

$$\mathrm{CR}(\mathrm{AF}_{\max}, n) = \begin{cases} 1 & \mathrm{AF}_{\max} \leq C_{\mathrm{AF}} \\ 1 & n \geq C_N \\ 0 & \text{otherwise} \end{cases}$$

in other words, we stop if the acquisition function maximum value is below some threshold C_{AF} or we have reached maximum iterations $n \geq C_N$.

In a setting with no constraints and no noise, the result of the optimization algorithm is determined from the dataset $\mathcal{D}_f = (\mathbf{X}, \mathbf{y})$ by function OP after the stopping criterion has been met. The optimal point is either an already sampled point $\mathbf{x}^* \in \mathbf{X}$ or the point with the minimum posterior mean [8]. Formally the optimal input point $\mathbf{x}^*$ is computed

$$\mathbf{x}^* = \min\left(\min_{\mathbf{x}_i \in \mathbf{X}} \mathbf{y}_i, \min_{\mathbf{x}_i \in \mathbf{X}} \mu(\mathbf{x})\right)$$

The optimal function value f^* is then obtained by sampling the function at the optimal point $f^* = f(\mathbf{x}^*)$.

To add constraints into the optimization problem, we modify many of the fundamental building blocks of Algorithm 5.1 to take feasibility into account. The next section will describe this augmentation in detail.

5.2 Constrained optimization

To handle constraints in Bayesian optimization, we must first define commonly used notions of feasibility.

Definition 5.2 (Feasibility)**.** We consider a point $\mathbf{x} \in \mathcal{X}$ to be feasible if for all $j \in J$, $c_j(\mathbf{x}) \leq 0$. The index set $J = \{ 1, 2, \ldots, |J| \}$ indexes our constraint functions $c_1, c_2, \ldots, c_J$. A constraint function $c : \mathcal{X} \to \mathbb{R}$ limits the domain of our problem to only consider solutions $\mathbf{x} \in \mathcal{X}$ which satisfy $c(\mathbf{x}) \leq 0$.

Definition 5.3 (Independence)**.** We consider Gaussian processes $(f_i)_{i=1}^N$ to be independent if for any $\mathbf{x} \in \mathcal{X}$ we have the equality

$$\mathbb{P}\left[f_1(\mathbf{x}) = y_1, f_2(\mathbf{x}) = y_2, \ldots, f_N(\mathbf{x}) = y_N\right] = \prod_{i=1}^{N} \mathbb{P}\left[f_i(\mathbf{x}) = y_i\right]$$

for any values $y_1, y_2, \ldots, y_N$.

Our aim is to extend the optimization framework to solve problems of form

$$\begin{aligned} &\min_{\mathbf{x} \in \mathcal{X}} && f(\mathbf{x}) \\ &\text{such that:} && \\ & && c_1(\mathbf{x}) \leq 0 \\ & && c_2(\mathbf{x}) \leq 0 \\ & && \vdots \\ & && c_J(\mathbf{x}) \leq 0 \end{aligned}$$

In many constrained Bayesian optimization methods the constraint functions $c_j : \mathcal{X} \to \mathbb{R}$ are modelled by mutually independent (Definition 5.3) Gaussian processes which are also independent of the Gaussian process used to model the objective function [11, 20]. In Definition 5.2, we have defined the feasibility of input points. One possible way of implementing constraints to our optimization algorithm is to modify the expected improvement defined in Definition 5.1 by a factor related to the probability of $\mathbf{x}$ being feasible [11, 20].

Algorithm 5.2 High-level algorithm of constrained Bayesian optimization

Input: The objective function f, constraint functions $c_1, c_2, \ldots, c_J$, objective function kernel k_f alongside a set of kernel hyperparameters θ_f, collection of constraint function kernels $k_{c_1}, k_{c_2}, \ldots, k_{c_J}$ each with kernel hyperparameter set $\theta_{k_{c_1}}, \theta_{k_{c_2}}, \ldots, \theta_{k_{c_J}}$, initial objective function dataset $\mathcal{D}_f$ with N_0 samples, collection of initial constraint function datasets $\mathcal{D}_{c_1}, \mathcal{D}_{c_2}, \ldots, \mathcal{D}_{c_J}$ with N_0 samples. Acquisition function AF, optimal point discovery function OP, and stopping criterion function CR. The underlying input set $\mathcal{X}$.

Output: The pair of optimal and feasible input and output values $(\mathbf{x}^*, f^*)$.

Set $n = N_0$.
while $\mathrm{CR}(\mathrm{AF}_{\max}, n) = 0$ **do** ▷ Stopping criterion.
  $\mathcal{M}_f = \mathrm{Fit}(\mathcal{D}_f, k_f, \theta_{k_f})$
  for $j \in J$ **do**
    $\mathcal{M}_{c_j} = \mathrm{Fit}(\mathcal{D}_{c_j}, k_{c_j}, \theta_{k_{c_j}})$
  end for
  $\mathrm{AF}_{\max} = \max_{\mathbf{x}\in\mathcal{X}} \mathrm{AF}(\mathbf{x}, \mathcal{D}_f, \mathcal{D}_{c_1}, \ldots, \mathcal{D}_{c_J}, \mathcal{M}_f, \mathcal{M}_{c_1}, \ldots, \mathcal{M}_{c_J})$
  $\mathbf{x}_{n+1} = \arg\max_{\mathbf{x}\in\mathcal{X}} \mathrm{AF}(\mathbf{x}, \mathcal{D}_f, \mathcal{D}_{c_1}, \ldots, \mathcal{D}_{c_J}, \mathcal{M}_f, \mathcal{M}_{c_1}, \ldots, \mathcal{M}_{c_J})$
  Sample the objective function f at $\mathbf{x}_{n+1}$: $y_{n+1} = f(\mathbf{x}_{n+1})$
  Add the new sample $(\mathbf{x}_{n+1}, y_{n+1})$ to $\mathcal{D}_f$
  for $j \in J$ **do**
    Sample the constraint function c_j at $\mathbf{x}_{n+1}$: $c_{j,n+1} = c_j(\mathbf{x}_{n+1})$
    Add the new sample $(\mathbf{x}_{n+1}, c_{j,n+1})$ to $\mathcal{D}_{c_j}$
  end for
  Increment $n = n + 1$
end while
$(\mathbf{x}^*, f^*) = \mathrm{OP}(\mathcal{D}_f, \mathcal{D}_{c_1}, \ldots, \mathcal{D}_{c_J}, \mathcal{M}_f, \mathcal{M}_{c_1}, \ldots, \mathcal{M}_{c_J})$ ▷ Optimal point.
return $(\mathbf{x}^*, f^*)$

Note that Algorithm 5.2 is a high-level overview of the constrained Bayesian optimization setting. Most of the details regarding modeling the objective function and the constraints are hidden in the acquisition function AF and optimal point discovery OP function. Also the convergence criterion CR is modified to take constraints into account.

More specifically, we want the optimal point $\mathbf{x}^*$ to satisfy $f(\mathbf{x}^*) \leq f(\mathbf{x}), \forall \mathbf{x} \in \mathcal{X}^{\text{feasible}}$ and $\forall j \in J : c_j(\mathbf{x}^*) \leq 0$. $\mathcal{X}^{\text{feasible}}$ is the set of feasible input points, $\mathcal{X}^{\text{feasible}} = \{\mathbf{x} \in \mathcal{X} : c_j(\mathbf{x}) \leq 0, j \in J\}$. In other words, the optimal point should minimize f among feasible points while being itself feasible (Definition 5.2).

As a helpful definition, let us define the dataset of feasible objective function observations, $\mathcal{D}_f^{\text{feasible}}$ in Definition 5.4. We will use this in a later section when discussing practical implementations of constrained optimization, see Section 5.4.

Definition 5.4. We define the feasible objective function dataset computed from

$\mathcal{D}_f = (\mathbf{X}, \mathbf{y})$ as being $\mathcal{D}_f^{\text{feasible}} = (\mathbf{X}^{\text{feasible}}, \mathbf{y}^{\text{feasible}})$ where

$$\mathbf{X}^{\text{feasible}} = \{\, \mathbf{x} \in \mathbf{X} : \mathbf{x} \in \bigcap_{j \in J} \mathbf{X}_j, \mathcal{D}_{c_j}^{\text{feasible}} = (\mathbf{X}_j, \mathbf{c}_j) \,\}$$
$$\mathbf{y}^{\text{feasible}} = \{\, y_i \in \mathbf{y} : \mathbf{x}_i \in \mathbf{X}^{\text{feasible}} \,\}$$

$\mathcal{D}_{c_j}^{\text{feasible}}$ is the dataset of feasible samples, formally $\mathcal{D}_{c_j}^{\text{feasible}} = \{\, \mathbf{x}_i \in \mathbf{X}_j : \mathbf{c}_{j_i} \leq 0 \,\}$. $\mathbf{y}^{\text{feasible}}$ is filtered from $\mathbf{y}$ using the indices of $\mathbf{X}$ and $\mathbf{X}^{\text{feasible}}$. In essence, we want to save those observations y where the corresponding $\mathbf{x} \in \mathbf{X}$ is feasible.

5.3 Noisy constrained optimization

After adding constraints we must handle the possible noise in our objective function and constraint function samples. In other words, we do not observe them directly, rather we have access to noisy samples which we assume to be of the form

$$\mathbf{y}_i = f(\mathbf{x}_i) + \epsilon_{fi}$$
$$\mathbf{C}_{j_i} = c_j(\mathbf{x}_i) + \epsilon_{c_j i}$$

where $\epsilon_{fi} \sim \mathcal{N}(0, \sigma_f), \epsilon_{c_j i} \sim \mathcal{N}(0, \sigma_{c_j}), \mathbf{x}_i \in \mathbf{X}$. In other words, we assume that for each sample, the noise values in the objective function sample $\mathbf{y}_i$ and constraint function samples $\mathbf{C}_{j_i}$ are normally distributed random variables with means of 0 and some standard variances which are same for all samples $\mathbf{x}_i \in \mathbf{X}$.

Note that the addition of noise does not affect the structure of our optimization problem, it only affects the optimization algorithm and the method we use to find the optimal input-output pair. In other words, we are still optimizing a problem of the form

$$\begin{aligned} \min_{\mathbf{x} \in \mathcal{X}} \quad & f(\mathbf{x}) \\ \text{such that:} \quad & \\ & c_1(\mathbf{x}) \leq 0 \\ & c_2(\mathbf{x}) \leq 0 \\ & \vdots \\ & c_J(\mathbf{x}) \leq 0 \end{aligned}$$

we just do not have visibility into the objective function f or constraints $c_1, c_2, \ldots, c_J$ directly.

Algorithm 5.3 contains the pseudo-code used in the noisy constrained Bayesian optimization. The only difference between Algorithm 5.2 and Algorithm 5.3 is that we now assume that the samples contain some additive noise.

Algorithm 5.3 High-level algorithm of noisy constrained Bayesian optimization

Input: The objective function f, constraint functions $c_1, c_2, \ldots, c_J$, objective function kernel k_f alongside a set of kernel hyperparameters θ_{k_f}, collection of constraint function kernels $k_{c_1}, k_{c_2}, \ldots, k_{c_J}$ each with kernel hyperparameter set $\theta_{k_{c_1}}, \theta_{k_{c_2}}, \ldots, \theta_{k_{c_J}}$, initial objective function dataset $\mathcal{D}_f$ with N_0 samples, collection of initial constraint function datasets $\mathcal{D}_{c_1}, \mathcal{D}_{c_2}, \ldots, \mathcal{D}_{c_J}$ with N_0 samples each. Acquisition function AF, optimal point discovery function OP, and stopping criterion function CR. The underlying input set $\mathcal{X}$.

Output: The pair of optimal input and output values $(\mathbf{x}^*, f^*)$.

Set $n = N_0$.
while $\mathrm{CR}(\mathrm{AF}_{\max}, n) = 0$ **do**
  $\mathcal{M}_f = \mathrm{Fit}(\mathcal{D}_f, k_f, \theta_{k_f})$
  for $j \in J$ **do**
    $\mathcal{M}_{c_j} = \mathrm{Fit}(\mathcal{D}_{c_j}, k_{c_j}, \theta_{k_{c_j}})$
  end for
  $\mathrm{AF}_{\max} = \max_{\mathbf{x}\in\mathcal{X}} \mathrm{AF}(\mathbf{x}, \mathcal{D}_f, \mathcal{D}_{c_1}, \ldots, \mathcal{D}_{c_J}, \mathcal{M}_f, \mathcal{M}_{c_1}, \ldots, \mathcal{M}_{c_J})$
  $\mathbf{x}_{n+1} = \arg\max_{\mathbf{x}\in\mathcal{X}} \mathrm{AF}(\mathbf{x}, \mathcal{D}_f, \mathcal{D}_{c_1}, \ldots, \mathcal{D}_{c_J}, \mathcal{M}_f, \mathcal{M}_{c_1}, \ldots, \mathcal{M}_{c_J})$
  Sample the objective function f at $\mathbf{x}_{n+1}$: y_{n+1} ▷ Noisy sample
  Add the new sample $(\mathbf{x}_{n+1}, y_{n+1})$ to $\mathcal{D}_f$
  for $j \in J$ **do**
    Sample the constraint function c_j at $\mathbf{x}_{n+1}$: $c_{j,n+1}$ ▷ Noisy sample
    Add the new sample $(\mathbf{x}_{n+1}, c_{j,n+1})$ to $\mathcal{D}_{c_j}$
  end for
  Increment $n = n + 1$
end while
$(\mathbf{x}^*, f^*) = \mathrm{OP}(\mathcal{D}_f, \mathcal{D}_{c_1}, \ldots, \mathcal{D}_{c_J}, \mathcal{M}_f, \mathcal{M}_{c_1}, \ldots, \mathcal{M}_{c_J})$
return $(\mathbf{x}^*, f^*)$

5.4 Solution methods

Looking back to our original maximum traffic intensity problem in Chapter 3, we assume that the samples for the AWT and ATTD values are inherently noisy. The simulator gives us some samples for those averages, however they are noisy estimates of the true average values. Therefore, we currently have the correct layers to solve our original problem. Next, we will explore literature and describe a couple of proposed methods for solving these kinds of optimization problems.

More specifically, we have hidden the most important detail of choosing the next sample point behind the acquisition function AF. We will next explore ways to modify and extend the traditional expected improvement of Definition 5.1 to behave well with constraints and noisy samples.

5.4.1 Constrained expected improvement algorithm (CEI)

The paper [11] describes the first approach which we will examine. The main idea is to use constrained expected improvement (Theorem 5.2) as the acquisition function. Assuming that our samples are not noisy, according to the paper, the best point to sample next would be the point $\mathbf{x}^*$ which is feasible and maximizes expected improvement (Definitions 5.1, 5.2). Definition 5.5 formalizes this idea.

Definition 5.5 (Constrained improvement, from [11])**.** We define the constrained improvement at $\mathbf{x}$ as being

$$\mathrm{I_C}(\mathbf{x}) = \mathrm{F}(\mathbf{x}) \max(f^* - f(\mathbf{x}), 0)$$

where $F(\mathbf{x})$ is an indicator variable for $\mathbf{x}$ being a feasible point or not. f^* is the best value seen from already sampled points.

Since we do not know if $\mathbf{x}$ is feasible, we must treat $\mathrm{F}(\mathbf{x})$ as a random variable. When taking the expectation of $\mathrm{I_C}(\mathbf{x})$ and modeling the constraint functions c_j with models $\mathcal{M}_{c_j}$, we can compute the expectation of $\mathbb{E}\left[\mathrm{I_C}|\mathcal{M}_f, \mathcal{M}_{c_1}, \mathcal{M}_{c_2}, \ldots, \mathcal{M}_{c_J}\right]$ [11]. Theorem 5.2 is the main result which we use to evaluate the aforementioned expectation.

Theorem 5.2 (Constrained expected improvement, from [11])**.** *The constrained expected improvement is simply the original expected improvement scaled with the probability of the candidate point being feasible $PF(\mathbf{x})$,*

$$\mathbb{E}\left[I_C|\mathcal{M}_f, \mathcal{M}_{c_1}, \mathcal{M}_{c_2}, \ldots, \mathcal{M}_{c_J}\right] = EI_C(\mathbf{x}) = PF(\mathbf{x})EI(\mathbf{x})$$

where $PF(\mathbf{x}) = \prod_{j=1}^{J} \mathbb{P}[\mathcal{M}_{c_j}(\mathbf{x}) \leq 0]$ by assumption of having independent (Def. 5.3) constraint models $\mathcal{M}_{c_1}, \mathcal{M}_{c_2}, \ldots, \mathcal{M}_{c_J}$. The expected improvement $EI(\mathbf{x})$ is defined in Definition 5.1. However, we should augment Definition 5.1 to consider the best value f^* as being computed from feasible objective values $\mathbf{y}^{feasible}$, where $\mathcal{D}_f^{feasible} = (\mathbf{X}^{feasible}, \mathbf{y}^{feasible})$ as defined in Definition 5.4.

The expected constrained improvement of Theorem 5.2 is used as the acquisition function when iterating the Bayesian optimization Algorithm 5.3. The stopping criterion and optimal point discovery are discussed in Chapter 6 since we use different than those proposed in the paper [11].

5.4.2 Noisy expected improvement algorithm (NEI)

The other method, described in paper [20], extends on the idea of constrained improvement. Combining feasibility with expected improvement, defined in Definition 5.1, allows us to handle constraints. Again, the main concept is to multiply the improvement by a probability that the sample point is feasible. The resulting Theorem 5.3 also allows us to handle cases where there does not exist a sample in the feasible dataset $\mathcal{D}_f^{\text{feasible}}$ [20].

The paper [20] first defines an utility function $u(\mathcal{D}_f)$

$$u(\mathcal{D}_f) = \begin{cases} -\min_{\mathbf{x}_i \in \mathbf{X}} f(x_i) & \mathcal{D}_f^{\text{feasible}} \neq (\emptyset, \emptyset) \\ -M & \mathcal{D}_f^{\text{feasible}} = (\emptyset, \emptyset) \end{cases}$$

which assumes no noise in our dataset. Assuming we have otherwise identical datasets $\mathcal{D}_f^n$ and $\mathcal{D}_f^{n+1}$, containing n and $n+1$ samples respectively with $\mathbf{x}_{n+1}$ being the new sample only contained in the latter dataset. We can then compute the improvement in utility $I(\mathbf{x}_{n+1}) = u(\mathcal{D}_f^{n+1}) - u(\mathcal{D}_f^n)$ between the datasets as

$$I(\mathbf{x}_{n+1}) = \begin{cases} 0 & \mathbf{x}_{n+1} \text{ is infeasible.} \\ M - f(\mathbf{x}_{n+1}) & \mathbf{x}_{n+1} \text{ is feasible and } \mathcal{D}_f^{n,\text{feasible}} = (\emptyset, \emptyset) \\ \max(f^* - f(\mathbf{x}_{n+1}), 0) & \text{otherwise} \end{cases}$$

where f^* is the best feasible objective function value obtained in dataset $\mathcal{D}_f^n$ [20]. Now taking the expectation of the improvement for a candidate point $\mathbf{x}$, we obtain Theorem 5.3.

Theorem 5.3 (Expected improvement with feasibility, from [20])**.** *Assuming noiseless datasets $\mathcal{D}_f, \mathcal{D}_{c_1}, \mathcal{D}_{c_2}, \ldots, \mathcal{D}_{c_J}$, let the number of feasible objective function observations be $F = \left|\mathcal{D}_f^{feasible}\right|$. Then*

$$EI_{Inf.}(\mathbf{x}|\mathcal{D}_f, \mathcal{D}_{c_1}, \mathcal{D}_{c_2}, \ldots, \mathcal{D}_{c_J}) = \begin{cases} EI(\mathbf{x})PF(\mathbf{x}) & F > 1 \\ (M - \mu_f(\mathbf{x}))PF(\mathbf{x}) & F = 0 \end{cases}$$

where $PF(\mathbf{x}) = \prod_{j=1}^{J} \mathbb{P}\left[\mathcal{M}_{c_j}(\mathbf{x}) \leq 0\right]$ is again the probability of being feasible, computed using the models $\mathcal{M}_{c_1}, \mathcal{M}_{c_2}, \ldots, \mathcal{M}_{c_J}$. $\mathcal{M}_f.\mu = \mu_f$ is the mean of the Gaussian process $\mathcal{M}_f$ modeling the objective function f. The dependence on the datasets is through the models and the computation of the expected improvement using the best previously seen feasible objective function value.

The parameter $M \in \mathbb{R}$ is a hyperparameter which affects the trade-off between exploration and exploitation in the case of not having feasible samples. In other words, with high M we prefer to sample in those regions of the input space where there are more likely feasible points.

Since in Theorem 5.3 we require noiseless datasets of function and constraint samples, we first build Gaussian process models using noisy datasets and use those models to sample noiseless values. In other words, using noisy datasets $\mathcal{D}_f, \mathcal{D}_{c_1}, \mathcal{D}_{c_2}, \ldots, \mathcal{D}_{c_J}$ we build models $\mathcal{M}_f, \mathcal{M}_{c_1}, \mathcal{M}_{c_2}, \ldots, \mathcal{M}_{c_J}$ which we in turn use to build noiseless datasets $\tilde{\mathcal{D}}_f, \tilde{\mathcal{D}}_{c_1}, \tilde{\mathcal{D}}_{c_2}, \ldots, \tilde{\mathcal{D}}_{c_J}$ which we use in Theorem 5.3. We can therefore define the noisy expected improvement as an expectation in Definition 5.6.

Definition 5.6 (Noisy expected improvement, from [20])**.** Given noisy dataset $\mathcal{D}_f$ of objective function values and a set of noisy datasets $(\mathcal{D}_{c_j})_{j=1}^J$ of constraint samples for constraint functions c_j, $j \in J$, we define the noisy expected improvement as the expectation

$$\mathrm{EI_N}(\mathbf{x}|\mathcal{D}_f, \mathcal{D}_{c_1}, \mathcal{D}_{c_2}, \ldots, \mathcal{D}_{c_J}) = \mathbb{E}[\mathrm{EI_{Inf.}}(\mathbf{x}|\tilde{\mathcal{D}}_f, \tilde{\mathcal{D}}_{c_1}, \tilde{\mathcal{D}}_{c_2}, \ldots, \tilde{\mathcal{D}}_{c_J})]$$

where we take the expectation using the noiseless datasets $\tilde{\mathcal{D}}_f, \tilde{\mathcal{D}}_{c_1}, \tilde{\mathcal{D}}_{c_2}, \ldots, \tilde{\mathcal{D}}_{c_J}$ which we generate using models $\mathcal{M}_f, \mathcal{M}_{c_1}, \mathcal{M}_{c_2}, \ldots, \mathcal{M}_{c_J}$ created using the noisy datasets $\mathcal{D}_f, \mathcal{D}_{c_1}, \mathcal{D}_{c_2}, \ldots, \mathcal{D}_{c_J}$. In other words, we consider the generated datasets as random variables drawn from the models.

Theorem 5.4 (from [20])**.** *With the context of Definition 5.6, we express the expectation as an integral*

$$EI_N(\mathbf{x}|\mathcal{D}_f, \mathcal{D}_{c_1}, \mathcal{D}_{c_2}, \ldots, \mathcal{D}_{c_J}) = \int_{\mathbf{y}} \int_{\mathbf{c}} EI_{Inf.}(\mathbf{x}|\tilde{\mathcal{D}}) \mathbb{P}\left[\mathbf{y}|\mathcal{M}_f\right] \prod_{j=1}^{J} \mathbb{P}\left[\mathbf{c}_j|\mathcal{M}_{c_j}\right] \mathrm{d}\mathbf{c}\mathrm{d}\mathbf{y}$$

where $\tilde{\mathcal{D}} = \{\tilde{\mathcal{D}}_f = (\mathbf{X}, \mathbf{y}), (\tilde{\mathcal{D}}_{c_j} = (\mathbf{X}, \mathbf{c}_j))_{j=1}^{J}\}$*. We compute the probabilities of sampling noiseless objective function values* $\mathbf{y}$ *using model* $\mathcal{M}_f$ *and similarly for the noiseless constraint samples* $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_J$ *we use models* $\mathcal{M}_{c_1}, \mathcal{M}_{c_2}, \ldots, \mathcal{M}_{c_J}$*. In essence, we are integrating over all possible noiseless samples using the Gaussian process models. The models themselves are created using the noisy datasets* $\mathcal{D}_f, \mathcal{D}_{c_1}, \mathcal{D}_{c_2}, \ldots, \mathcal{D}_{c_J}$*.*

Expressing the expectation of Definition 5.6 using an integral over the generated noiseless samples we have Theorem 5.4. The integral is intractable, however as the paper describes there is a simple quasi Monte Carlo simulation method to approximate this integral [20]. However, to utilize the proposed sampling method, the integral must be transformed as an integral over an unit cube. Applying Theorem 5.5 to the integral of Theorem 5.4, we can form the main result in Theorem 5.6.

Theorem 5.5 (Changing domain of integration, from [20])**.** *The integral of a function* $f : \mathbb{R}^d \to \mathbb{R}$ *of a multivariate normal random variable* $\mathbf{y}$*, with probability density function* $p(\mathbf{y}|\mu, \Sigma)$*, over the domain* $\mathbb{R}^d$ *can be transformed into a integral over an unit cube* $[0, 1]^d \subset \mathbb{R}^d$*,*

$$\int_{\mathbb{R}^d} f(\mathbf{y}) p(\mathbf{y}|\mu, \Sigma) d\mathbf{y} = \int_{[0,1]^d} f(A\Phi^{-1}(\mathbf{u}) + \mu) \mathrm{d}\mathbf{u}$$

where μ *and* Σ *are the parameters of the multivariate normal random variable,* A *is a matrix such that* $AA^T = \Sigma$*, for example the Cholesky decomposition of* Σ *[5].*

Theorem 5.6 (Noisy expected improvement estimation, from [20])**.** *Estimating the integral in Theorem 5.4 can be achieved by having a Sobol sequence [28]* $(\mathbf{t}_k)_{k=1}^{N}$ *and then using Theorem 5.5 to transform the integral. The end result is that*

$$EI_N(\mathbf{x}|\mathcal{D}_f, \mathcal{D}_{c_1}, \mathcal{D}_{c_2}, \ldots, \mathcal{D}_{c_J}) \approx \frac{1}{N} \sum_{k=1}^{N} EI_{Inf.}(\mathbf{x}|\tilde{\mathbf{c}}(\boldsymbol{t}_k), \tilde{\mathbf{f}}(\mathbf{t}_k))$$

where

$$\begin{bmatrix} \tilde{\mathbf{f}}(\boldsymbol{t}_k) \\ \tilde{\mathbf{c}}(\boldsymbol{t}_k) \end{bmatrix} = A\Phi^{-1}(\mathbf{t}_k) + \boldsymbol{\mu}$$
$$\boldsymbol{\mu} = [\mu_f, \mu_{c_1}, \mu_{c_2}, \ldots, \mu_{c_J}]^T$$
$$\Sigma = diag(\Sigma_f, \Sigma_{c_1}, \Sigma_{c_2}, \ldots, \Sigma_{c_J})$$
$$\Sigma = AA^T \text{ (Cholesky decomposition.)}$$

The vector $\boldsymbol{\mu}$ contains the posterior means of the objective function and constraint functions, fitted using Gaussian processes, at the sample points. The matrix Σ is a block-diagonal matrix containing the covariance matrices of the posterior distributions for the objective functions and constraint functions.

Using Theorem 5.6, we can estimate the value of the noisy expected improvement, defined in Definition 5.6. Note that in Theorem 5.6 we estimate the noisy expected improvement using a Sobol sequence, it provides us with a better convergence for the estimate when compared to uniform random sampling [20, 28].

Finally, the estimate of Theorem 5.6 is used as our acquisition function value in Algorithm 5.3.

6 Implementation

In Chapter 5 we described methods used to solve a general optimization problem with noisy objective function and constraints. However, there are certain things which make our elevator problem a special case of the general Bayesian optimization problem.

First, the objective function is simply the input p, indicating the traffic intensity. we have no further objective than to maximize the input p, given certain expensive to evaluate constraints which depend on p. Second, for our underlying set we simply have $\mathcal{X} = [0, p_{\max}] \subset \mathbb{R}$. In other words, we only consider an interval of the real line with non-negative elements. This corresponds to the intensity p being a non-negative real-value.

Finally, we have a reason to believe that the constraint functions behave monotonically after a certain point, in the sense that there exists some p such that constraint functions at p are non-positive and at $p + \delta, \delta > 0$ they are positive, indicating that $p + \delta$ is infeasible. In other words, there cannot be some larger $p' > p$ which would also be feasible. The monotone behaviour of constraints is domain knowledge related to the original elevator problem. This behaviour is used as motivation for the toy problem in Chapter 7.

Since the objective function for the elevator problem is the intensity p, we do not need to model it using a Gaussian process since it is trivial to compute. In general, assuming the objective function of our problem is simply $x \in \mathbb{R}$, we can simplify many of the equations used in the general settings. We will describe those simplifications next and then we describe the optimization algorithms which we implement.

The expected improvement in Definition 5.1 can be computed simply as,

$$\mathrm{EI}(x) = \max(x - x^*, 0)$$

where x is the input value and x^* the maximum observed value in objective function dataset $\mathcal{D}_f = (\mathbf{X}, \mathbf{y})$. This is because the objective function of our problem is simply $f(x) = x$ and we have a scalar input $\mathbf{x}_i = x_i$, in other words $\mathcal{D}_f = (\mathbf{x}, \mathbf{x})$ where $\mathbf{x}_i = x_i \in \mathbb{R}$.

Note that [20] and [11] have defined the optimization as a minimization problem and we have a maximization problem. Therefore we have the opposite sign in some of the equations used by our implementation, an alternate way of thinking is to consider our objective function to be $f(x) = -x$ in a minimization problem. However, changing the sign does not affect any of the theory described in previous chapters, it only affects the specific implementation for our elevator problem.

Next, we will describe how we can augment and modify the two algorithms of papers [20] and [11], introduced in Chapter 5 to fit our problem to suit a trivial maximization problem $\max_{x \in \mathcal{X}} x$ with expensive-to-evaluate constraints $c_1, c_2, \ldots, c_J$. In total, we have 5 different algorithms which could suit our purposes. The NEI algorithm, described in the next section based on [20], and 4 different variants on the constrained expected improvement idea of paper [11]. The different variants are CEI, NCEI, CEI-Mean, NCEI-Mean which we describe in section 6.2.

6.1 NEI Algorithm

In this section we implement an algorithm inspired by paper [20] and discuss some of the details of our implementation and differences compared to the original in [20].

Algorithm 6.1 NEI Algorithm, based on [20].

Input: Objective function f, constraint functions $c_1, c_2, \ldots, c_J$, constraint sample datasets $\mathcal{D}_{c_1}, \mathcal{D}_{c_2}, \ldots, \mathcal{D}_{c_J}$, objective function sample dataset $\mathcal{D}_f$, kernels $k_f, k_{c_1}, k_{c_2}, k_{c_3}, \ldots, k_{c_J}$, candidate point x, and the underlying input set $\mathcal{X}$. The datasets have N samples.

Output: Acquisition function value at x.

for $j \in J$ **do**
  $\mathcal{M}_{c_j} = \text{Fit}(\mathcal{D}_{c_j}, k_{c_j}, \theta_{k_{c_j}})$
  Initialize empty vector $\mu_j \in \mathbb{R}^N$ and empty matrix $\Sigma_j \in \mathbb{R}^N$
  Let $\mu_{ji} = \mathcal{M}_{c_j}.\mu(x_i), x_i \in \mathbf{X}$ ▷ Mean values at the sample points.
  Collect $\mu_j = \left[\mu_{j1}, \mu_{j2}, \ldots, \mu_{jN}\right]$
  Let $(\Sigma_j)_{i,k} = \mathcal{M}_{c_j}.\Sigma(x_i, x_k)$ ▷ Covariance values at sample points
end for
Construct a block-diagonal matrix $\Sigma = \text{diag}\left((\Sigma_j)_{j=1}^{J}\right)$
Let μ be a vector created by concatenating the vectors $\mu_1, \mu_2, \ldots, \mu_j$
Cholesky decomposition of the covariance matrix, $\Sigma = AA^T$
Generate a Sobol sequence, $(\mathbf{t}_i)_{i=1}^{i=K}$
for $k = 1, 2, \ldots, K$ **do**
  Using Theorem 5.5, we can estimate the noisy expected constrained improvement, by first creating samples $\tilde{\mathbf{C}}_k$

$$\tilde{\mathbf{C}}_k = A\Phi^{-1}(\mathbf{t}_k) + \boldsymbol{\mu} \in \mathbb{R}^{JN}$$

  for $j \in J$ **do**
    From $\tilde{\mathbf{C}}_k$ extract samples for constraint j, $\tilde{\mathbf{c}}_{kj}$
    Create a new non-noisy dataset $\tilde{\mathcal{D}}_{c_j}$ for constraints using samples $\tilde{\mathbf{c}}_{kj}$
    Fit a new model $\tilde{\mathcal{M}}_{c_j}$ using $\tilde{\mathcal{D}}_{c_j}$, k_{c_j}, and $\theta_{k_{c_j}}$ without optimizing hyperparameters
  end for
end for
Initialize $\text{EI}_\text{N} = 0$
for $k = 1, 2, \ldots, K$ **do**
  Compute $\text{EI}_{\text{Inf.}}\left(\mathbf{x} | (\tilde{\mathcal{M}}_{c_j})_{j=1}^{J}\right)$
  Update $\text{EI}_\text{N} = \text{EI}_\text{N} + \frac{1}{K}\text{EI}_{\text{Inf.}}(\mathbf{x})$
end for
return EI_N

The Algorithm 6.1 is adapted from the original paper [20], however since the objective function is trivial, some of the definitions of Section 5.4.2 can be simplified.

The quantity in Theorem 5.3 can be simply computed as

$$\mathrm{EI}_{\mathrm{Inf.}}(x) = \begin{cases} \max(x - x^*, 0)PF(x) & F > 1 \\ (M + x)PF(x) & F = 0 \end{cases} \tag{2}$$

Note the changed sign in both branches of the augmented definition. The probability of feasibility $PF(x)$ is computed as in Theorem 5.3.

We should note that practical implementation runs the loop on lines $1 - 19$ once per iteration of the Bayesian optimization. The lines $20 - 25$, are run for each point $x \in \mathcal{X}$ used to find the maximum of the acquisition function as NEI is utilized as the acquisition function in Algorithm 5.3.

6.2 CEI Algorithm and variants

Another alternative acquisition function algorithm can be created by implementing the ideas of paper [11]. The paper does not mention noisy evaluations at all. However, since we do not need to model the objective function, we can ignore the noise errors of objective function evaluations. In this section we will describe a basic version of the algorithm using paper [11], which we call CEI. We will then explore augmentations and modifications of this algorithm to handle noisy constraint samples. These modified versions will be called NCEI, CEI-Mean, and NCEI-Mean algorithms.

We begin by first describing a close implementation of the algorithm in paper [11]. This will serve as our baseline algorithm for further developments.

Algorithm 6.2 Constrained expected improvement (CEI) algorithm, based on [11].

Input: Objective function f, constraint functions $c_1, c_2, \ldots, c_J$, constraint sample datasets $\mathcal{D}_{c_1}, \mathcal{D}_{c_2}, \ldots, \mathcal{D}_{c_J}$, objective function sample dataset $\mathcal{D}_f$, kernels $k_f, k_{c_1}, k_{c_2}, k_{c_3}, \ldots, k_{c_J}$, candidate point x, underlying input set $\mathcal{X}$. The datasets have N samples.

Output: Acquisition function value at x.

1: **for** $j \in J$ **do**
2: $\quad \mathcal{M}_{c_j} = \mathrm{Fit}(\mathcal{D}_{c_j}, k_{c_j}, \theta_{k_{c_j}})$
3: **end for**
4: Build feasible dataset $\mathcal{D}_f^{feasible}$ from $\mathcal{D}_f$ by filtering infeasible samples out.
5: **if** $\left|\mathcal{D}_f^{feasible}\right| > 0$ **then**
6: $\quad$ Let $\mathrm{PF}(x) = \mathbb{P}[x \text{ is feasible}] = \prod_{j \in J} \mathbb{P}[\mathcal{M}_{c_j}(x) \leq 0]$.
7: $\quad$ Compute $\mathrm{EI}(x)$ using $\mathcal{D}_f^{feasible}$ as per Theorem 5.1.
8: $\quad \mathrm{EI}_{\mathrm{C}}(x) = \mathrm{EI}(x)\mathrm{PF}(x)$
9: $\quad$ **return** $\mathrm{EI}_{\mathrm{C}}(x)$
10: **else**
11: $\quad$ **return** $\mathbf{1}(x = \hat{x})$
12: **end if**

Algorithm 6.2 simply first fits independent Gaussian process models for each of the constraints $c_j, j \in J$. The models are then used on line 6 to compute the probability of the candidate point $x \in \mathcal{X}$ being feasible. The improvement of objective function value is then weighed by the probability of being feasible on lines $7-8$, calculating the expectation of constrained improvement, as defined in Definition 5.5. The creation of the feasible dataset $\mathcal{D}_f^{\text{feasible}}$ on line 4 is done by filtering out infeasible samples from $\mathcal{D}_f$. Formally the feasible dataset is $\mathcal{D}_f^{\text{feasible}} = (\mathbf{X}^{\text{feasible}}, \mathbf{y}^{\text{feasible}})$ where

$$\mathbf{X}^{\text{feasible}} = \{ x_i \in \mathbf{X} : \mathbf{c}_{j_i} \leq 0, \forall j \in J \}$$
$$\mathbf{y}^{\text{feasible}} = \{ y_i \in \mathbf{y} : x_i \in \mathbf{X}^{\text{feasible}} \}$$

and $\mathbf{c}_{j_i} \in \mathcal{D}_{c_j}$ is the constraint sample for input sample i of constraint c_j.

In the case of $\mathcal{D}_f^{\text{feasible}} = (\emptyset, \emptyset)$ we will simply propose to sample a random point $\hat{x} \in \mathcal{X}$. We assume that this situation is quite rare in our experiments, since we set the range of searchable traffic intensity values to contain both low and high values. However, this is an important fail safe to guarantee the Bayesian optimization algorithm continues to iterate when there is no feasible samples. This is an addition which is not discussed in the original paper [11].

In the original version of the algorithm (CEI), we consider the constraint models $\mathcal{M}_{c_j}, j \in J$ to not incorporate noise. Therefore, the posterior variance $\sigma^2(x) = \mathcal{M}_{c_j}.\Sigma(x, x)$ is 0 for all points x in the training sample set $\mathbf{X}$.

When considering noisy constraint samples, we can extend the base algorithm simply by fitting (Algorithm 4.1) a noise value σ_ϵ alongside other kernel parameters for models $\mathcal{M}_{c_j}, j \in J$. However, when computing the probability of candidate point x being feasible, we use model $\mathcal{M}_{c_j}, j \in J$ to model the value the real constraint function $c_j(x)$. In other words, we try to predict the underlying constraint function, rather than the noisy samples. We will call this algorithm variant NCEI for noisy constrained expected improvement. This variant does not change the computation of expected improvement (EI), we only change the fitting process of models $\mathcal{M}_{c_j}, j \in J$ to include a noise level.

One problem with using CEI and NCEI algorithms is the fact that we have to compute $\text{EI}(x)$ using the previously observed feasible samples. However, if our datasets $\mathcal{D}_{c_1}, \mathcal{D}_{c_2}, \ldots, \mathcal{D}_{c_J}$ are noisy, we might have a feasible sample x_i in $\mathcal{D}_f^{\text{feasible}}$ for which the input x_i is not truly in the feasible range. In other words $c_j(x_i) > 0$ for some constraint function $c_j \in J$ while the constraint samples are all non-positive, $\mathbf{c}_{j_i} \leq 0, \forall j \in J$.

To handle this, we should filter the feasible input samples using the Gaussian process models for the constraints, to make sure that feasible input samples in $\mathcal{D}_f^{\text{feasible}}$ are truly feasible. We therefore modify the filtering step of line 4 of Algorithm 6.2 to build the feasible dataset using the models, formally the feasible dataset is then composed of

$$\mathbf{X}^{\text{feasible}} = \{ x_i \in \mathbf{X} : \mathbb{P}\left[\tilde{y}_i \leq 0\right] \geq 1 - \delta, \forall j \in J \}$$
$$\mathbf{y}^{\text{feasible}} = \{ y_i \in \mathbf{y} : x_i \in \mathbf{X}^{\text{feasible}} \}$$

where $\tilde{y}_i$ is a noisy sample drawn from the model $\mathcal{M}_{c_j}$ at input point x_i. δ is some hyperparameter corresponding to the sensitivity of our filter [12]. We shall use $\delta = 0.05$ in all empirical experiments. In other words, the filtering step on line 4 should remove all samples for which some constraint model $\mathcal{M}_{c_j}, j \in J$ predicts a low probability of being feasible. We call algorithms with this filtering CEI-Mean and NCEI-Mean, corresponding to the CEI and NCEI algorithms.

In the CEI-Mean and NCEI-Mean algorithms, we also augment the case where there are no feasible samples (lines 10 – 12 in Algorithm 6.2). In those cases, we use the point with highest chance of being feasible from the dataset, $x^* = \arg\max_{x \in \mathbf{X}} \mathrm{PF}(x)$ and return the value $\mathrm{EI}_C(x)$ computed with respect to that point x^*. Essentially, we take constrained expected improvement with respect to a point which has highest likelihood of being feasible from the dataset $\mathcal{D}_f$.

Table 1 illustrates the differences between the different variants of the CEI algorithm. In total, we have 4 different variants CEI, NCEI, CEI-Mean and NCEI-Mean, depending on whether we filter based on raw constraint function samples or constraint function models, whether we fit a noise model alongside other hyperparameters for models $\mathcal{M}_{c_j}, j \in J$, and how we handle the case of $\mathcal{D}_f^{\text{feasible}} = (\emptyset, \emptyset)$.

Algorithm	Fit noise level	Filtering	$\mathcal{D}_f^{\text{feasible}} = (\emptyset, \emptyset)$
CEI	No	Samples $\mathbf{c}_{j_i}$	Random point
NCEI	Yes	Samples $\mathbf{c}_{j_i}$	Random point
CEI-Mean	No	Models $\mathcal{M}_{c_j}, j \in J$	$\mathrm{EI}_C(x \mid \arg\max_{x' \in \mathcal{X}} \mathrm{PF}(x'))$
NCEI-Mean	Yes	Models $\mathcal{M}_{c_j}, j \in J$	$\mathrm{EI}_C(x \mid \arg\max_{x' \in \mathcal{X}} \mathrm{PF}(x'))$

Table 1: Different variants of the CEI algorithm.

6.3 Other implementation details

We have now explored the different acquisition function variants and in Chapter 5 we defined the stopping criterion. Therefore, the optimal point discovery function is the last piece which we will need to complete the general Algorithm 5.3. In this section we will also discuss the normalization of constraint function values and the discretization of our domain.

6.3.1 Optimal point

After we have reached convergence criterion in Algorithm 5.3, we return some best estimate x^* of the true optimum input point. The rule proposed in the paper [20] considers the trade-off between feasibility and the objective function value. We define the estimate formally in Definition 6.1.

Definition 6.1 (Optimal point in noisy setting, adapted from [20])**.** The optimal point $\mathbf{x}^*$ is defined as

$$\mathbf{x}^* = \arg\max_{\mathbf{x} \in \mathcal{X}} \mathbf{x} \prod_{j \in J} \mathbb{P}[\mathcal{M}_{c_j}(\mathbf{x}) \leq 0]$$

where the constraints are modelled using Gaussian processes. Therefore the required probabilities can be computed using models $\mathcal{M}_{c_j}$, see Definition 4.4 for the definition of a model.

6.3.2 Normalization of samples

To improve the performance of the Gaussian process models $\mathcal{M}_j$, $j \in J$, we normalize the constraint function samples. For any constraint function value $c_{ji} = c_j(\mathbf{x}_i), \mathbf{x}_i \in \mathbf{X}$, we normalize it using a simple centering and scaling normalization according to Definition 6.2. Normalization of training samples is a common way to help model fit in supervised machine learning [23].

Definition 6.2 (Normalization of constraint values)**.** Given a set of constraint function values $\mathcal{D}_{c_j} = (\mathbf{X}, \mathbf{c}_j)$ of size N_j for constraint function c_j, $j \in J$, normalization is done by first computing the empirical mean

$$\mathrm{m}_j = \frac{1}{N_j} \sum_{i=1}^{N_j} (\mathbf{c}_j)_i$$

and empirical standard deviation

$$\mathrm{std}_j = \sqrt{\frac{1}{N_j} \sum_{i=1}^{N_j} ((\mathbf{c}_j)_i - m_j)^2}$$

Using the aforementioned values, we scale the constraint value vector $\mathbf{c}_j$ values

$$\bar{c}_{j_i} = \frac{c_{j_i} - \mathrm{m}_j}{\mathrm{std}_j}$$

where c_{j_i} is the element at index i of constraint vector $\mathbf{c}_j$.

6.3.3 Discretization of domain

The interval $[0, p_{\max}]$ is discretized for the implementation. The discretization is done uniformly, the step size in the toy problem (Chapter 7) is approximately 0.785 and in the elevator traffic problem (Chapter 8) it is exactly 0.01. The maximum acquisition function value $\mathrm{AF}_{\max}$ is obtained by simply computing the value for each candidate point $x \in [0, p_{\max}]$.

6.4 Software

For practical implementation of the algorithms, we utilize Python [29] alongside common libraries NumPy [15] and scikit-learn [22]. These provide us with excellent tools for implementing the algorithms and computing the necessary matrices and other mathematical structures. The Gaussian process models themselves are implemented using the GPy library [14], which contains a robust implementation of Gaussian process modeling and hyperparameter search.

7 Toy problem experimentation

To find the best model for the original elevator problem, in this chapter we perform an analysis on the performance of each candidate algorithm when solving an optimization problem with known constraint and objective functions and whose optimum point is known. This toy problem, as defined Definition 7.1 should suffice for our first analysis.

Definition 7.1 (Toy problem)**.**

$$\begin{aligned} \max. \quad & x \\ \text{such that:} \quad & \\ & c_1(x) \leq 0 \\ & c_2(x) \leq 0 \\ & x \in [0, 50/2\pi] \end{aligned}$$

Where $c(x) = \frac{x}{10}\sin\left(\frac{x}{10}\right) + 5$, and the constraints are $c_1(x) = c(x) - 8$ and $c_2(x) = -c(x) + 2$. The problem domain is the set $\mathcal{X} = [0, 50/2\pi]$ which is discretized as described in Section 6.3.3. For the toy problem, we pretend that c_1 and c_2 are expensive to evaluate and we therefore utilize the framework of Bayesian optimization to solve the toy problem. In Figure 7.1, we can see geometrically that the constraint function c_1 restricts us to consider input values whose constraint function value $c(x)$ is 8, similarly the constraint c_2 restricts us to consider input values where $c(x)$ is larger than 2. The constraint function c behaves monotonically in the region where $x \geq 50$, this is motivated by the earlier discussion in Chapter 6 regarding the monotonicity of AWT and ATTD values.

From Figure 7.1 we can clearly see that an optimal solution to the toy problem is at $x^* \approx 67.4$. We can also solve numerically the feasible region for x from the constraints c_1, c_2. The approximate feasible regions are

$$\begin{aligned} x &\in [0, 40.0] \\ x &\in [57.3, 67.4] \end{aligned}$$

The endpoint of the later feasible region is also the global optimum point x^*.

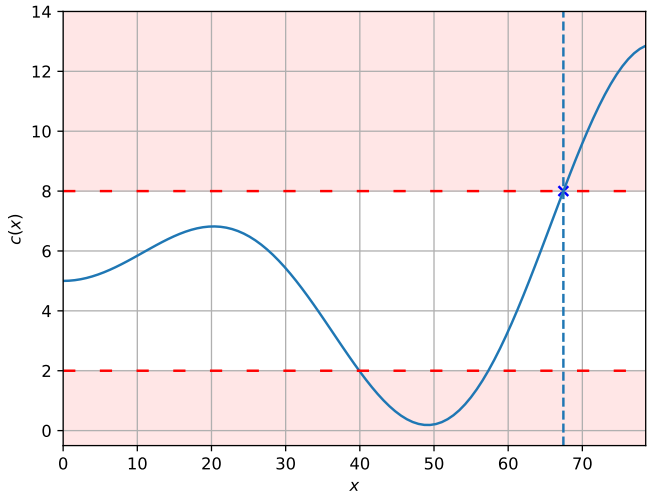


Figure 7.1: The function $c(x) = \frac{x}{10}\sin(\frac{x}{10}) + 5$ shown in blue, boundary of feasibility for constraints c_1 and c_2 indicated by dashed red lines and shaded areas. The vertical dashed blue line indicates the position of the optimal point x^*.

7.1 Toy experiment setup

To correspond to our original problem, we experiment with having noise in our constraint function observations. The noise levels are varied to examine how the magnitude of noise affects the performance of our optimization methods. The constraint samples c_{j1} and c_{j1} are generated according to

$$c_{1i} = \frac{x_i}{10}\sin(\frac{x_i}{10}) - 3 + \epsilon_{1i}$$
$$c_{2i} = -\frac{x_i}{10}\sin(\frac{x_i}{10}) - 3 + \epsilon_{2i}$$

for samples $x_i \in \mathbf{X}$. The errors $\epsilon_{1i}, \epsilon_{2i} \sim \mathcal{N}(0, \sigma_\epsilon)$ are identically distributed for all input samples $x_i \in \mathbf{X}$ and the noise standard deviation σ_ϵ is the same for both c_1 and c_2. We experiment with noise standard deviations in the set $\{0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0\}$.

As for the kernel used, for most experiments we utilize a basic squared exponential kernel (RBF kernel, Definition 4.5). However, we do experiment with Matérn kernels (Definition 4.7) as well.

For each algorithm, NEI, CEI, NCEI, CEI-Mean, NCEI-Mean (Chapter 6), we perform 50 experiments. A single experiment consists of finding the maximum $x \in \mathcal{X}$ such that the constraints c_1 and c_2 are satisfied. The initial input samples $\mathbf{X}$ are exactly the same each time, namely $\{25, 50, 75\} \subset \mathcal{X}$. However, the initial constraint datasets are subject to noise, therefore we have different initial constraint samples in each experiment. The exact hyperparameters for each candidate algorithm are described in the Appendix A.1.

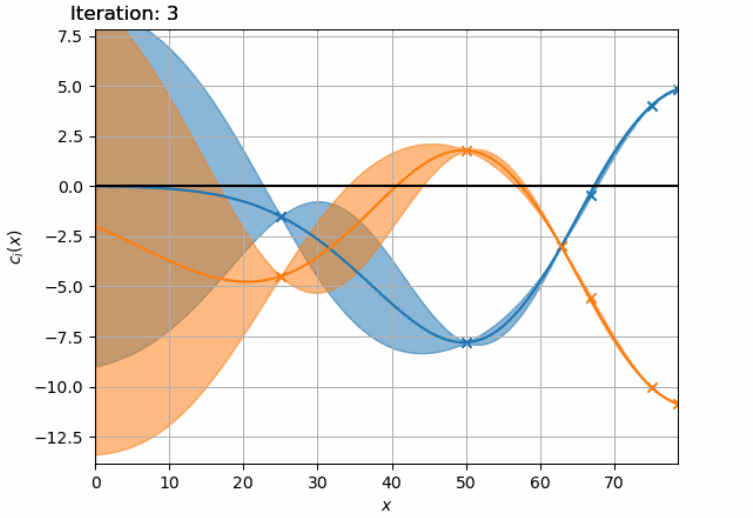


Figure 7.2: The state of the NEI algorithm for a noiseless version of the toy problem. The blue and orange lines indicate mean of the constraint $j, j = 1, 2$ posterior distribution, shaded areas are 95% confidence bands for each constraint. Blue and orange crosses indicate training samples for that iteration of constraint GP models. Feasible constraint region is below the black line of $c_i(x) = 0$.

Figure 7.2 illustrates the NEI optimization algorithm in the noiseless toy problem setting. We can note that the blue Gaussian process posterior distribution is similar in shape to the real constraint c_1, at least near the training sample points. This indicates that the NEI algorithm, in conjunction with the Gaussian process kernel hyperparameter search, has found suitable Gaussian process models for the constraint functions c_1 and c_2.

7.2 Toy experiment results

To analyze the results of the toy problem, we use a couple of different metrics. We will use the difference to optimum, as defined in Definition 7.2, as our most important metric in analyzing the performance of the different optimization methods. We will also examine how difference to optimum, root-mean-square error, iteration count, and optimization time change depending on the level of noise in our samples.

Definition 7.2 (Difference to optimum)**.** The difference to true optimum point x^* by an estimated optimum $\hat{x}^*$ is defined to be $x^* - \hat{x}^*$.

Definition 7.3 (Root-mean-square error)**.** The root-mean-square error of a set of differences to optimum $\mathbf{d} \in \mathbb{R}^N$ is

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}\mathbf{d}_i^2}$$

7.2.1 Non-noisy observations

We will first examine how the algorithms perform under non-noisy samples. In other words, in the case of $\sigma_\epsilon = 0$.

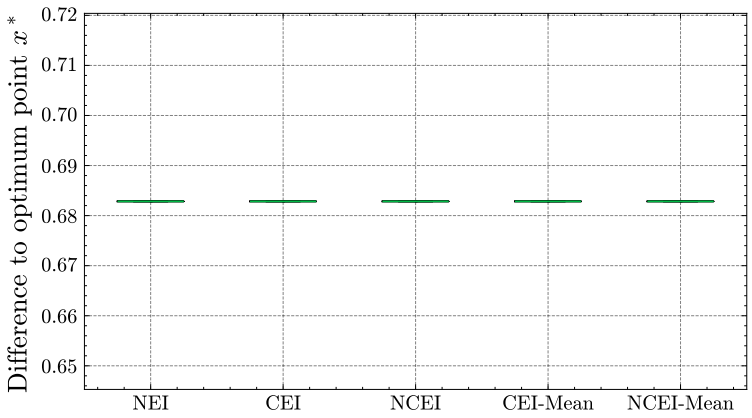


Figure 7.3: Boxplot of the difference of estimated optimum $\hat{x}^*$ to correct optimum point x^* for different algorithms in a noiseless toy problem. Note that due to not having any deviation, the boxes are displayed as just the median lines in green. The distributions are computed from 50 experiment repetitions.

As we can see from Figure 7.3, the difference to the optimal point (Definition. 7.2) is the same with every algorithm across the 50 experiments. Regarding difference to optimum, they behave identically in a noiseless setting. The positivity of difference to optimum means that they all underestimate the optimum point x^* by approximately 0.682. Furthermore, the estimate of the optimum is the best possible since the next point from the estimate, when taking discretization into account, is already an infeasible point.

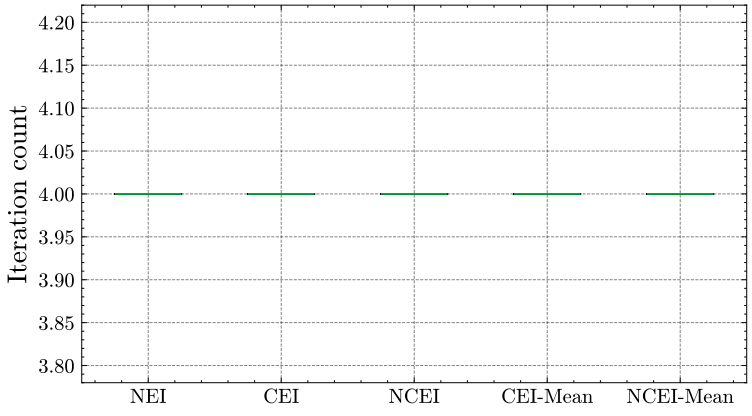


Figure 7.4: Boxplot of the number of iterations in a noiseless toy problem setting. Note that since the standard deviation of the results is 0, we simply have the median line for each algorithm displayed. In other words, the boxes in our boxplots have heights of 0. The distributions are computed from 50 experiment repetitions.

From Figure 7.4, we note that the number of iterations is the same for all algorithms in the noiseless setting. Therefore, the computation time it takes for the algorithms to complete is the only metric which we can use to separate them from each other.

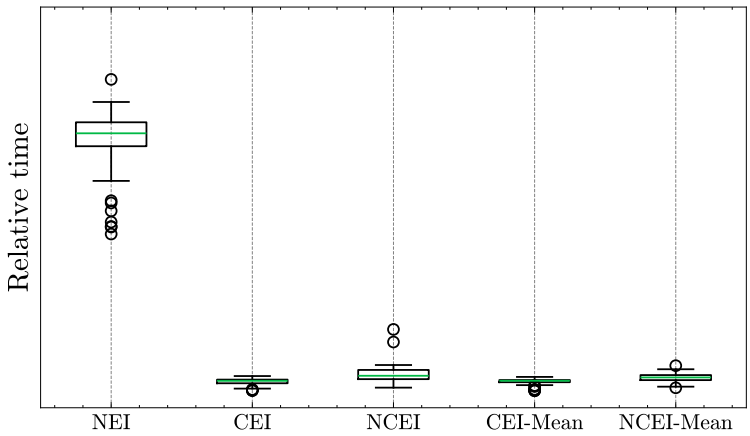


Figure 7.5: Relative time taken for algorithms to terminate in a noiseless setting. The distributions are from 50 repetitions. Median is displayed with the green line. Boxes correspond to lower and upper quartiles. Whiskers are displayed to the datapoints at most 1.5 times the interquartile range (distance between 3rd and 1st quartile) from the edges of the box. Outliers are datapoints to be over 1.5 times the interquartile range from the edges of the box.

Figure 7.5 displays the number of seconds taken to solve the noiseless toy problem. We can see that NEI algorithm takes the longest to complete, this can be explained by the fact that NEI is computationally heavy when compared to the other algorithms [11, 20].

7.2.2 All noise levels

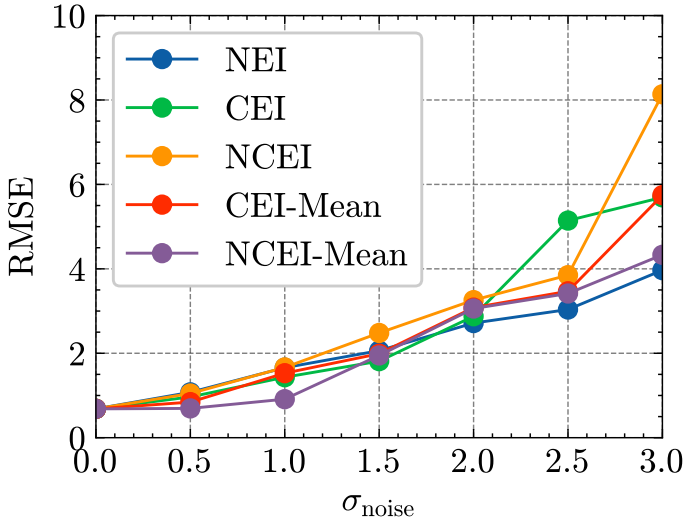


Figure 7.6: RMSE (Definition 7.3) as function of sample noise standard deviation σ_{noise}. 50 experiments for each algorithm and noise value.

When moving to the noisy observations, things become more interesting. The difference to optimum, while still close to 0, begins to fluctuate as we increase the level of noise. The root-mean-square error depicted in Figure 7.6 indicates that the NEI algorithm performs the best in terms of the root-square-mean error when there is a high level of noise. NCEI-Mean algorithm also works comparatively well under high noise.

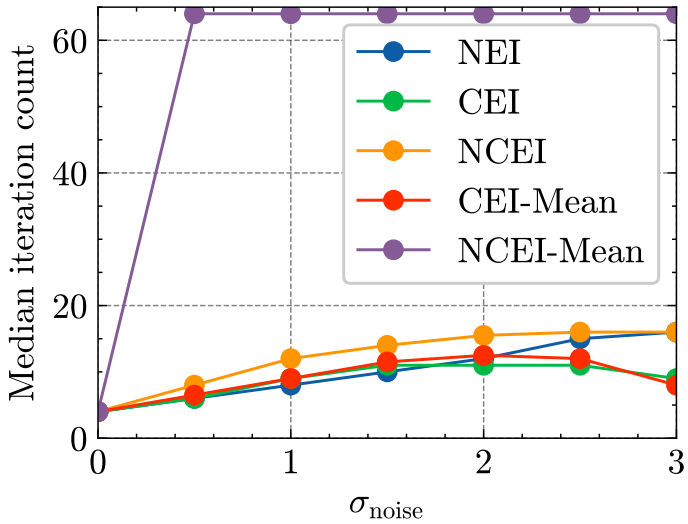


Figure 7.7: Median number of iterations needed for convergence for different noise levels σ_{noise}. $C_N = 64$ is the upper limit for number of iterations. 50 experiments for each algorithm and noise value.

However, things are not as straightforward with the number of iterations. For example the NCEI-Mean algorithm exhibits bad behaviour when it comes to terminating, the iteration count is often the maximum allowed iteration count (64 iterations). All of the other algorithms except NCEI-Mean perform similarly well in this regard. Figure 7.7 illustrates this phenomenon.

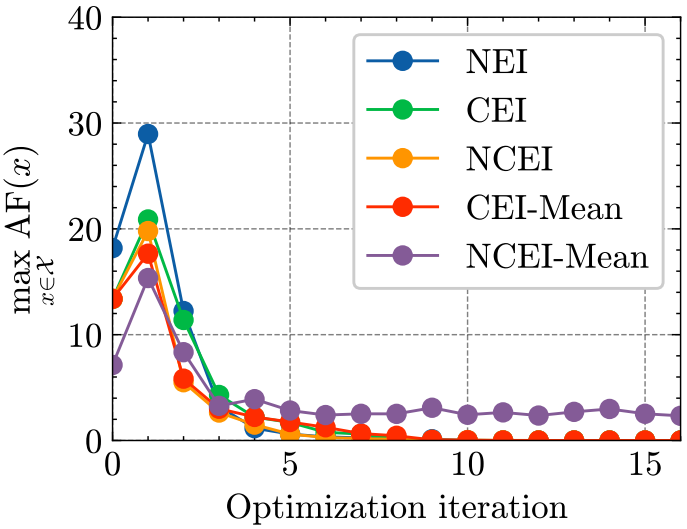


Figure 7.8: From the 50 samples, the median of the maximum acquisition function value for different algorithms as a function of the iteration progress, $\sigma_{\text{noise}} = 2.0$. Note that for already terminated algorithms, we place 0 as the AF value in the figure.

The bad termination behavior of NCEI-Mean algorithm is also illustrated in Figures 7.8 and 7.9. The acquisition function value does not converge for NCEI-Mean very well. However, other algorithms converge quite well, even in a noisy setting.

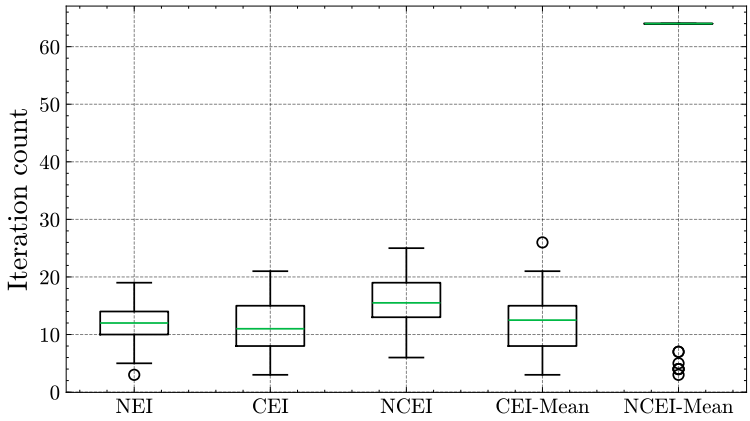


Figure 7.9: Boxplot of iteration count for different algorithms computed from the 50 repetitions. Noise level $\sigma_{\text{noise}} = 2.0$. Median is displayed with the green line. Boxes correspond to lower and upper quartiles. Whiskers are displayed to the datapoints at most 1.5 times the interquartile range (distance between 3rd and 1st quartile) from the edges of the box. Outliers are datapoints to be over 1.5 times the interquartile range from the edges of the box.

From this analysis, we can clearly see that the best performing algorithms is NEI. NCEI, and CEI are quite error prone in high noise cases. The mean corrected NCEI (NCEI-Mean) is quite accurate, but struggles with convergence as stated previously. In this light, it is best to use the NEI algorithm as the main subject of study when moving to the actual elevator problem.

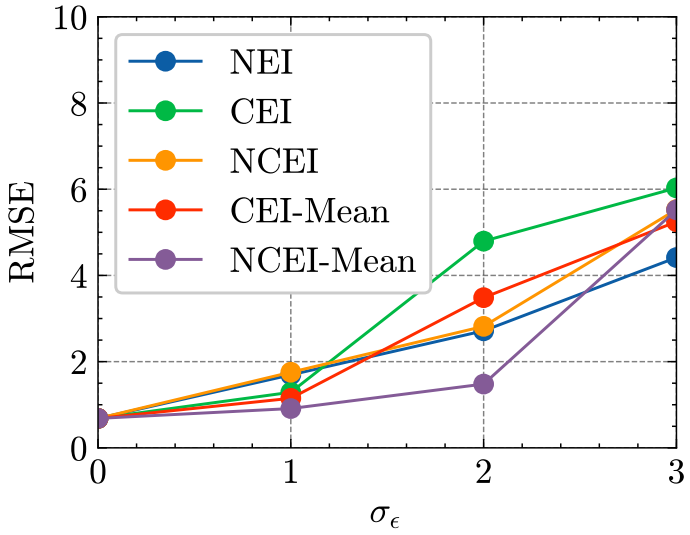


Figure 7.10: Root-mean-square error for Matérn kernel with $\nu = 3/2$. 50 experiments for each algorithm and noise value.

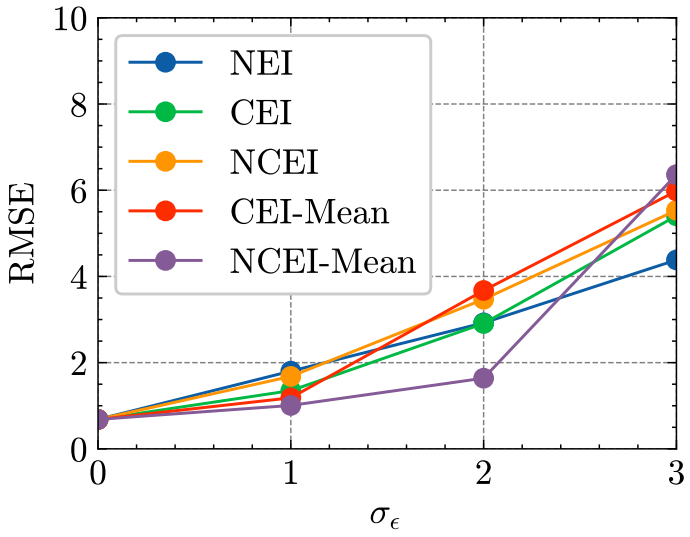


Figure 7.11: Root-mean-square error for Matérn kernel with $\nu = 5/2$. 50 experiments for each algorithm and noise value.

As an additional study, we replace the previously used RBF kernel with a once or twice differentiable Matérn kernel (Definition 4.7). Figures 7.10 and 7.11 illustrate the fact that we can get comparable results with a Matérn kernel instead of RBF. We also note that the performances of the optimization methods were comparable to ones with RBF kernel in all relevant aspects.

8 Elevator problem results

As we saw with our toy problem there are many factors to consider when comparing the different optimization algorithms. However, the performance of the NEI algorithm was overall good and the algorithm could handle high noise scenarios as well. Therefore, in this chapter we will mainly use the NEI algorithm to optimize the real traffic intensity using the simulator [27].

Recalling the original problem, we have the following optimization problem for the intensity p

$$\begin{aligned} &\max . && p \\ &\text{such that:} \\ &&& T_{\mathrm{AWT}}(p, S, M) \leq C_{AWT} \\ &&& T_{\mathrm{ATTD}}(p, S, M) \leq C_{ATTD} \\ &&& p \in \mathbb{R}^{+} \cup \{0\} \end{aligned}$$

fixing the elevator system S and traffic mix M, the simulator is used to estimate AWT and ATTD for the different intensities p.

For the main study in this Chapter, we utilize a 2-hour simulation with a traffic profile consisting of 40% outgoing, 40% ingoing, and 20% interfloor traffic. From the 2-hour simulation time, we will cut a 15-minute interval from the beginning and a 5-minute interval from the simulation data, as described in the standard [1].

The simulated building has 13 floors (12 tenant floors and 1 entrance floor) and 4 elevator shafts and elevators. The simulated elevator control system is a conventional control system, which allows users to call elevators based on direction of travel and then indicate the destination floor from inside the elevator car. The exact building parameters and elevator parameters are described in [13] under the L4 configuration.

After obtaining the main results using the NEI algorithm, we will examine how different variables affect the optimization performance and results. The characteristics which we will change are the length of the simulation, maximum intensity $p_{\max}$, the ATTD limit C_{ATTD}, and the Gaussian process kernel used in modeling the constraints. More information about the case studies are in the relevant sections. The case studies will provide us valuable information about the sensitivity of the NEI algorithm in our particular elevator setting when it comes to certain key variables.

8.1 Comparison of selected algorithms

We compare how NEI, NCEI, CEI, and CEI-Mean algorithms performed in terms of their optimal point estimate, iteration count and computation time taken. The main experiments were performed by tasking each algorithm to optimize intensity p with limits 35 and 50 for AWT and ATTD respectively, $C_{\mathrm{AWT}} = 35$, $C_{\mathrm{ATTD}} = 50$. The range of intensity percentage values considered was $p \in [0, 20]$ with discretization step size of 0.01 as described in Section 6.3.3. Stopping criterion of each algorithm was $\mathrm{AF}_{\max} \leq C_{\mathrm{AF}}$ with $C_{\mathrm{AF}} = 0.05$. Exact hyperparameter ranges for different algorithms are in the Appendix A.2. For each algorithm, we performed the main experiment 5 times and the results of those experiments are referred to as baseline results.

Figure 8.1 shows the number of iterations needed for algorithm stopping. We can see that the NEI and NCEI algorithms have the lowest iteration count. Looking at the optimization problem solutions in Figure 8.2, we can see that NCEI algorithm does not agree with the rest of the algorithms as to what is the best possible intensity p.

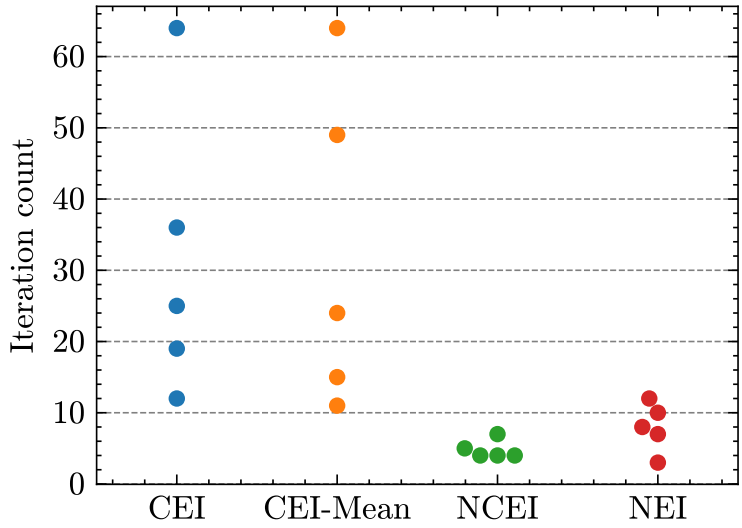


Figure 8.1: Number of iterations for different algorithms, 5 optimization runs per each.

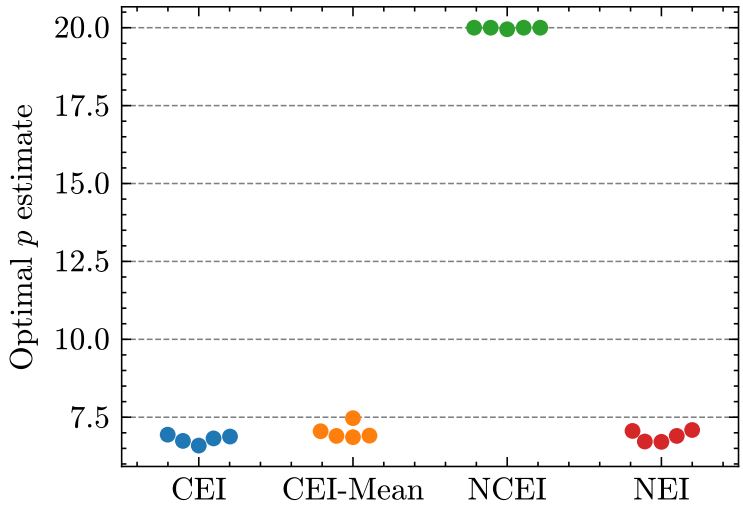


Figure 8.2: Plot of the estimated optimal intensity p in the 5 repetitions of the baseline experiments.

Because we do not know the perfect optimum intensity for the L4 building, we must rely on statistical methods for determining whether our algorithms find a suitable intensity value. This is done by running the simulator at some intensity $p + \delta$ where $\delta \in \{0, 0.1, 0.5, 1.0, 2.0\}$ and p is the estimated optimum intensity obtained using

one of the algorithms. In other words, we increase the traffic intensity p by a small or large amount relative to the estimate of the optimal value and examine the AWT and ATTD values obtained from running the simulator at those new intensities. From those samples we compute a validation percentage as defined in Definition 8.1.

Definition 8.1 (Validation percentage)**.** From validation dataset $\mathcal{D}_V^\delta$ containing N samples of AWT and ATTD,

$$\mathcal{D}_V^\delta = (\mathbf{y}_{\text{AWT}}, \mathbf{y}_{\text{ATTD}})$$

computed using a simulation at intensity $p + \delta$, we can define the validation percentage as being the ratio

$$\frac{1}{N} \sum_{i=1}^{N} \mathbf{1}((\mathbf{y}_{\text{AWT}})_i \leq C_{\text{AWT}} \text{ and } (\mathbf{y}_{\text{ATTD}})_i \leq C_{\text{ATTD}})$$

In other words, we empirically estimate if $p + \delta$ is also a feasible intensity value.

In the baseline experiments, for each of the 5 optimization runs, after estimating the optimal intensity value p the validation was run for 10 iterations for each of the δ values. Using those $N = 10$ AWT and ATTD values, we compute a validation percentage for each δ.

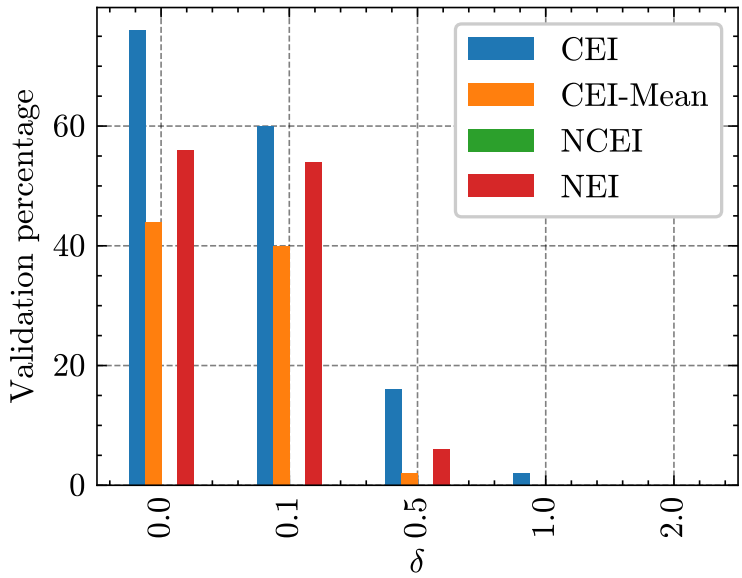


Figure 8.3: Validation percentage as a function of δ for different algorithms. 50 samples per δ-value; 5 optimization runs with 10 validation samples each equals 50 samples for single algorithm and δ value.

Figure 8.3 shows the validation percentage for each of the algorithms. One immediate observation is that for the NCEI algorithm, the validation percentage is

0 for all $p + \delta$ values and even for $\delta = 0$. Therefore, there is significant statistical evidence to support the conclusion that NCEI does not find a suitable intensity.

From the same figure we can clearly see that the other algorithms performed quite well. The validation percentage for the estimated optimal p is at least 44% and when increasing p by 1 percentage-point, the validation percentage decreases. This shows that we can quite clearly estimate the optimal p value quite adequately, because the immediate neighbourhood in the positive direction of estimated optimal p is infeasible.

However, Figure 8.3 also shows that increasing p by 0.1 percentage-point, also gives us a good estimate for optimal p. This indicates that while we do find the correct neighbourhood for optimal p, we do not have a fully accurate estimate of the true optimal intensity. This is nonetheless expected since it would be impossible to determine the absolutely optimal value using noisy estimates for AWT and ATTD values.

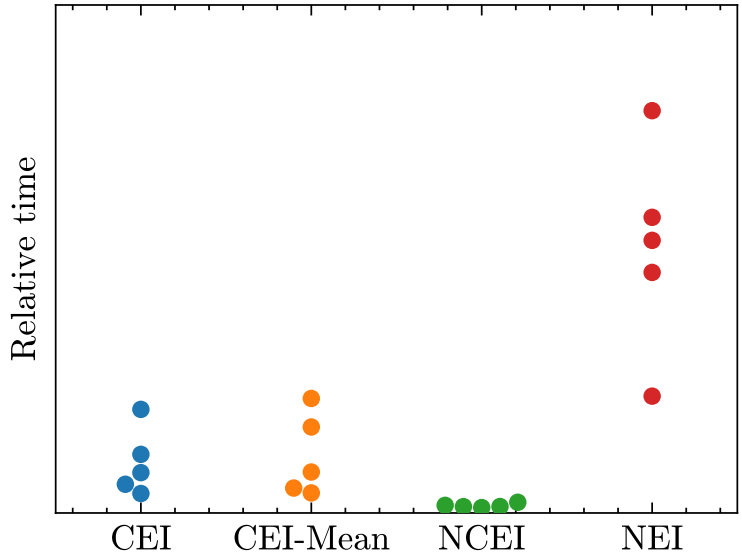


Figure 8.4: Plot of the relative time taken to find optimal sampling points for different algorithms. 5 repetitions for each of the algorithms.

Finally, to get a better understanding of the different algorithms, we examine runtime taken for each algorithm to arrive at the estimate of optimal p. Figure 8.4 illustrates how NEI algorithm takes the longest time to find next sampling point. However, as we can see from 8.5, NEI is still competitive in terms of the total optimization time taken. The total time is the time taken to find each suitable sampling point combined with the time it takes for the simulator to simulate the traffic for the proposed intensities.

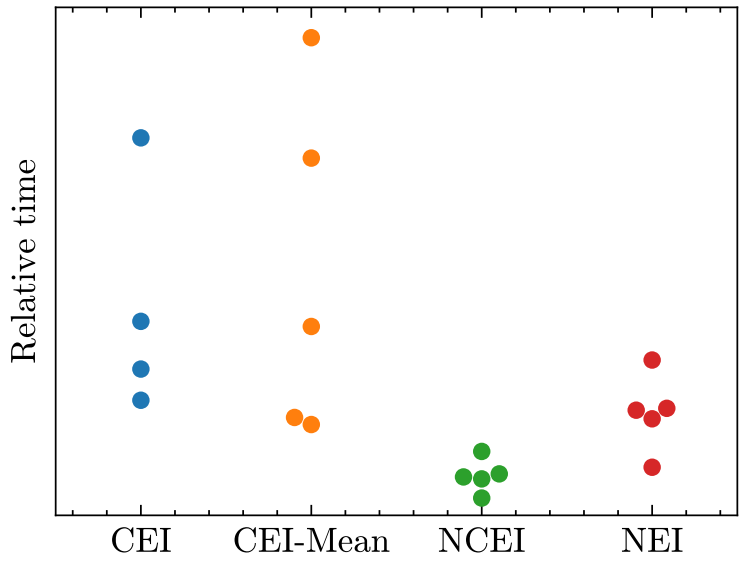


Figure 8.5: Plot of the relative total time taken in for different algorithms. 5 independent repetitions for each algorithm.

In conclusion, from the baseline results we can see that the NEI algorithm performs adequately for our purposes. NEI is computationally most expensive but in terms of the total time taken it is still competitive. NEI also has one of the lowest iteration counts which is important since we want to find the optimum with as few samples as possible. The solution NEI algorithm finds is also often a good approximation of the optimal intensity and the variance of the results between experiments is low. We therefore focus our next case studies to the NEI algorithm and disregard the other examined algorithms CEI, NCEI, and CEI-Mean.

8.2 Case studies

After examining the baseline results, we move onto a series of case studies using the NEI algorithm. These case studies primarily include changing some aspect of the algorithm or simulation setup and we will see how sensitive the NEI optimization algorithm is to various changes.

More specifically, we first examine the effect that shortening the simulation time has on the optimization. This is done to see whether we can decrease the simulation time and still have comparably good optimization results when compared to longer simulations. After that we change the upper bound $p_{\max}$ of the domain $[0, p_{\max}]$ and see how sensitive the NEI algorithm is to changes in the optimization region.

We also examine the effect of changing the ATTD limit has on the optimization. Changing of the ATTD limit also has the effect of changing the active constraint to be the one set in the standard [1] for AWT (C_{AWT}).

Finally, we examine whether changing the RBF kernel to a once or twice differentiable Matérn process kernel has any effect on the optimization performance. Our experiments with the toy model revealed that it might not affect the results significantly,

however examining that result in the traffic intensity context is an interesting exercise nonetheless.

8.2.1 Shorter simulations

For the baseline results, we utilized a 2-hour simulation time, of which we again removed 15 minutes from the start and 5 minutes from the end as specified in [1]. This case study examines the effect of reducing the simulation time on the optimization result and performance. In addition to the baseline 2-hour simulation, we experiment with 1-hour and 30-minute simulations.

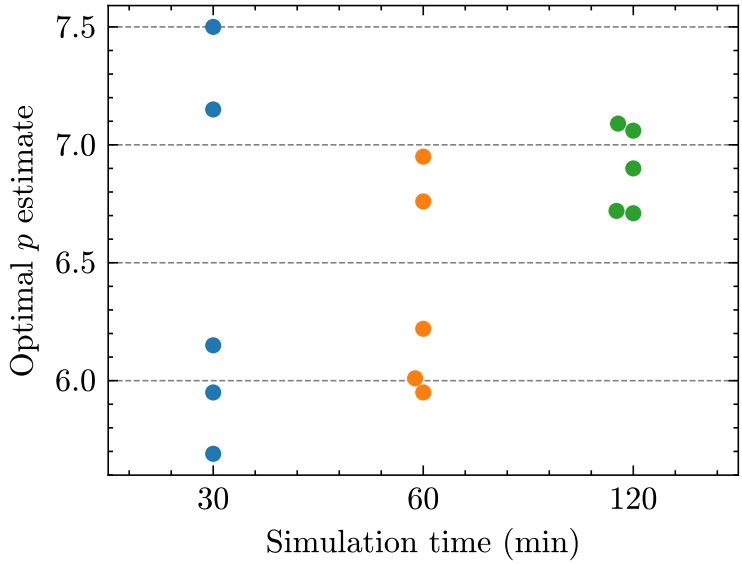


Figure 8.6: Plot of the estimate of optimal intensity p as a function of the simulation time. Sampled from 5 independent experiments.

Figure 8.6 illustrates the effect of shortening the simulation time on the estimated optimum intensity p. From the figure we can clearly deduce that lowering the simulation time increases the standard deviation of the optimization estimate. Intuitively, when simulating for a longer time we can obtain a better estimate of the AWT and ATTD for a given p and this decreases the variance in our optimization results. Figure 8.6 also illustrates another phenomenon, namely the mean of the estimates for the optimal p is lower when simulating for a shorter time. This could be because when having more noisy samples for the ATTD and AWT values, the NEI algorithm is more conservative in the optimal point discovery as defined in Definition 6.1.

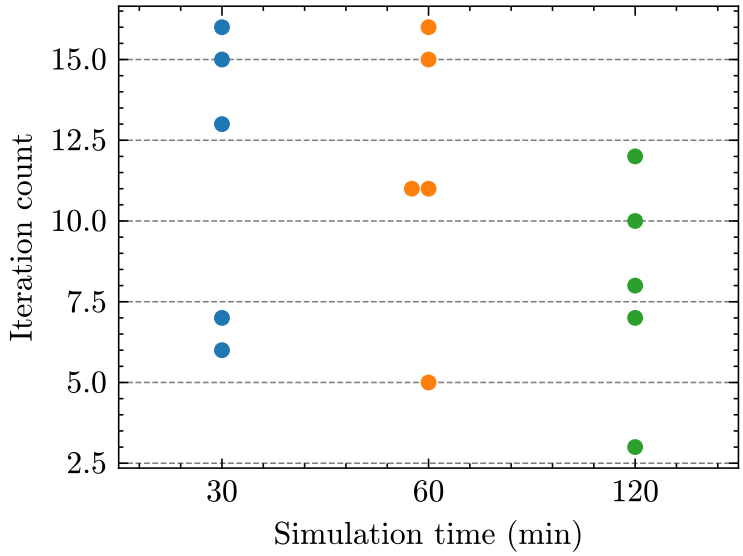


Figure 8.7: Plot of the iteration count for different simulation times. Sampled from 5 independent experiments.

From Figure 8.7 we can see that decreasing the simulation time increases the mean iteration count. Again, this is intuitively clear as decreasing the simulation time will increase the noise in our samples for ATTD and AWT. Therefore, the NEI algorithm has to compensate by examining more of the search space and with more samples and this increases the iteration count.

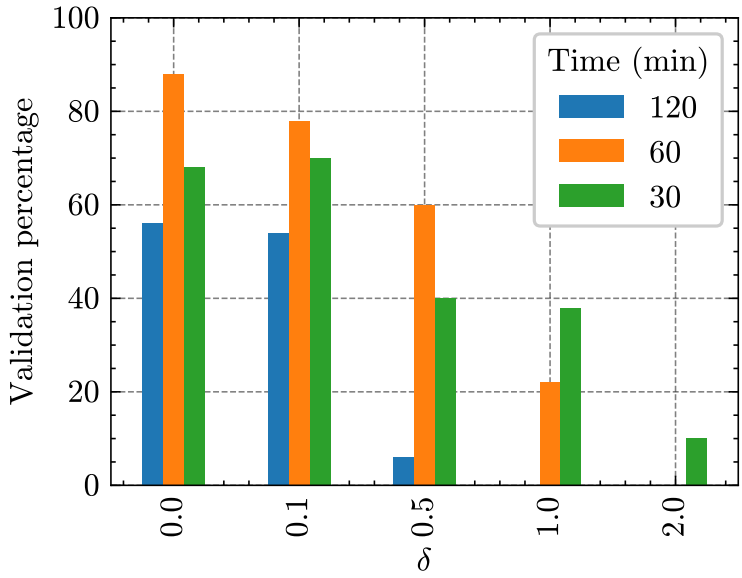


Figure 8.8: Validation percentage for different simulation times. Computed across 5 experiments with 10 validation samples for each δ in each experiment.

Figure 8.8 illustrates the effect of having shorter simulations on the validity of the estimated optimum intensity p. The figure indicates that for shorter simulation times,

the NEI algorithm tends to be more conservative. We suspect that this might be the result of having more noise in the Gaussian process models. Referring to optimal point definition (Definition 6.1), we can say that since the samples generated by shorter simulations are more noisy, the models $\mathcal{M}_{c_{\text{AWT}}}$, $\mathcal{M}_{c_{\text{ATTD}}}$ have a larger uncertainty for the value of the constraint functions T_{AWT} and T_{ATTD}. In other words, the posterior processes $T_{\text{AWT}}|\mathcal{D}_{\text{AWT}}$ and $T_{\text{ATTD}}|\mathcal{D}_{\text{ATTD}}$ have a variance function which predicts larger variances for each input point $x \in \mathcal{X}$.

An alternative explanation is that for short simulations, the simulation has not reached an approximate steady state. Removing the first 15 minutes from the calculations might not be enough to start obtaining steady-state estimates for AWT and ATTD from the simulation. This is not a problem for a longer simulation, because even if 15 minutes is not enough, we still obtain a reasonable number of steady-state estimates. Those samples are then averaged to estimate the AWT and ATTD values which are good enough approximations of the true steady-state values.

8.2.2 Changing upper bound for optimization domain

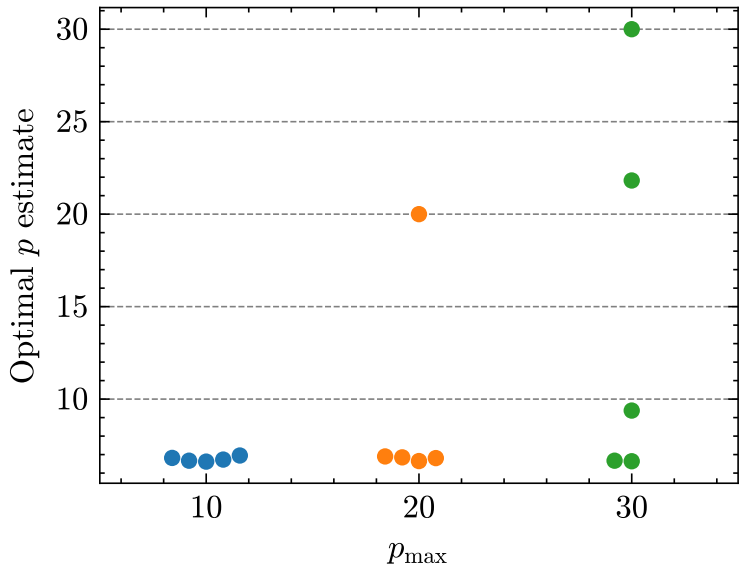


Figure 8.9: Boxplot of optimal p estimate according to NEI algorithm for different upper bounds for range of intensity p. The ranges are $[0, 20]$ and $[0, 30]$. Computed across 5 experiments.

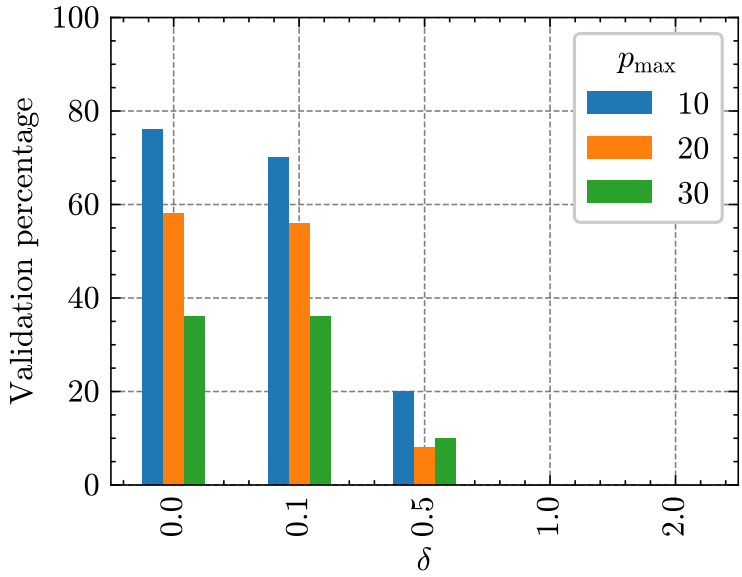


Figure 8.10: Validation percentages for instances of NEI algorithm with different upper bounds for p, $p \in [0, p_{\max}]$. Computed across 5 experiments with 10 validation samples for each δ in each experiment.

Increasing the higher bound allows us to see how well the NEI algorithm performs when the searchable space is bigger. More specifically, in this case study we replicate the same setup as with baseline results beside changing the percentage range upper bound to either 10, 20 or 30. We also change the inital grid of $\{4, 16\}$ to only contain a single element $\{4\}$, this is done to make sure that the initial grid is contained in the optimization domain.

Figure 8.9 illustrates that compared to the baseline results, we obtain a higher value for the estimated optimal intensity p. Figure 8.10 shows the validation percentage as a function of the domain upper bound. From that figure, we find that the higher values for the optimal intensity are infeasible intensities since the percentage of AWT and ATTD samples being under the limits is under 25% for the high estimated optimal intensity values.

We also note that the iteration count median is 8 for the baseline result using $[0, 20]$ and 2 for the range $[0, 30]$. This indicates that the NEI algorithm does not optimize well for the higher bound case. The worse convergence is explained by the phenomenon of the AWT and ATTD samples increasing fast as we increase p. This corresponds to the phenomenon of the elevator system not being able to handle the traffic as traffic intensity p increases enough, causing very high AWT and ATTD values.

8.2.3 Changing ATTD limit

In this case study, we change the average time-to-destination limit. We experiment with $C_{\text{ATTD}} = 70$, $C_{\text{ATTD}} = 90$, and $C_{\text{ATTD}} = 110$ in addition to the baseline result made with $C_{\text{ATTD}} = 50$.

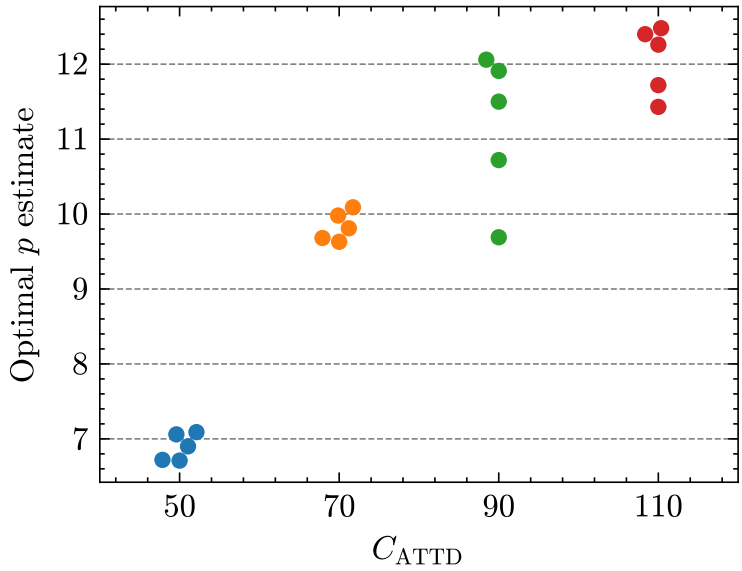


Figure 8.11: Effect of changing ATTD limit on the optimal intensity p estimate. 5 independent repetitions for each limit value.

From Figure 8.11, we can see that the optimal intensity increases as we increase the ATTD constraint C_{ATTD}. Also from Figure 8.12 we can see the transfer of active constraint from ATTD to AWT. By active constraint we mean the constraint which limits the intensity value p from increasing further. Mathematically, an active constraint c satisfies $c(x) = 0$ for the optimal value x.

With $C_{\mathrm{ATTD}} = 50, C_{\mathrm{ATTD}} = 70$, the active constraint is ATTD according to Figure 8.12. The increase of optimal p can be explained by the fact that increasing ATTD limit changes the active constraint from ATTD to AWT as displayed in the figure.

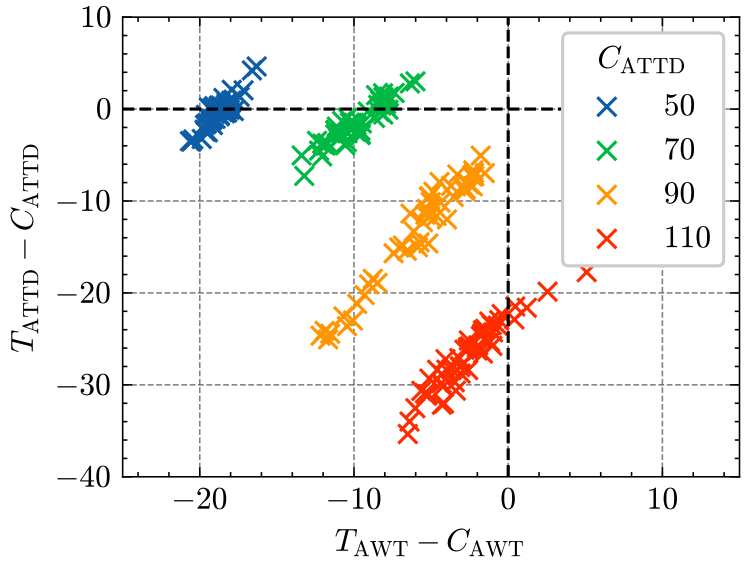


Figure 8.12: Effect of changing C_{ATTD} on which constraints are active. The datapoints are AWT and ATTD values sampled using estimated optimal intensity p. The points are gathered from 5 separate experiments with 10 repetitions each for a single ATTD limit. x-axis represents the difference of C_{AWT} and the AWT value at estimated optimal intensity p. y-axis shows the same quantity for ATTD.

From Figure 8.12 we can however note that the NEI algorithm is more conservative when both AWT and ATTD constraints are close to being active. The dataset corresponding to $C_{\mathrm{ATTD}} = 90$ illustrates this phenomenon by not being close to either of the axis.

8.2.4 Kernel comparison

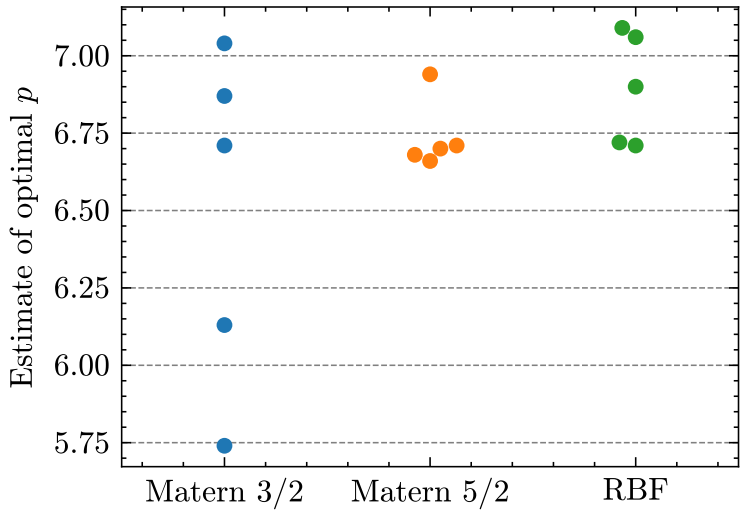


Figure 8.13: Plot of the optimal p estimate with different kernels. 5 experiments with each kernel.

We also examine the effect changing the kernel function has on the result of the NEI optimization algorithm. Previously we utilized the RBF kernel (Definition 4.5), this case study examines optimization performance when utilizing a Matérn kernel with $\nu = 3/2$ and $\nu = 5/2$, Definition 4.7. Note that we keep the bounds on the specific hyperparameters the same as with RBF kernel, since the Matérn kernels have similar hyperparameters as the RBF kernel [23, 7].

Figure 8.13 illustrates the effect of changing the kernel used in the NEI algorithm. We can see that the Matérn kernel corresponding to a once differentiable process causes the estimate for optimum p to have a larger variation than the RBF and the twice differentiable Matérn kernels.

8.3 Validation building

As a final analysis tool, we analyze a similar building but with 8 lifts and see how our NEI optimization algorithm performs under new building data. The exact building and elevator parameters are described in [13] as L8. This serves as validation of our NEI algorithm since most of the development and previous testing was done using the L4 case including the baseline results.

As before, we optimize 5 times the intensity p with AWT and ATTD targets of 35 and 50 seconds respectively. The total simulation time is 2 hours and we remove 15 minutes from the start and 5 minutes from the end for AWT and ATTD calculations, as previously. The NEI algorithm hyperparameters are kept the same and we optimize the intensity within the percentage range $[0, 20]$. The discretization step is 0.01 and the initial sampling grid is $\{4, 16\}$, as in the baseline experiments for L4.

We examine the same metrics: optimal intensity p estimate, iteration count, total seconds taken, and validation percentage (Definition 8.1). We compare the new L8 results to the baseline results (also referred to as L4) to conclude whether or not the NEI algorithm performs as adequately in the new building environment.

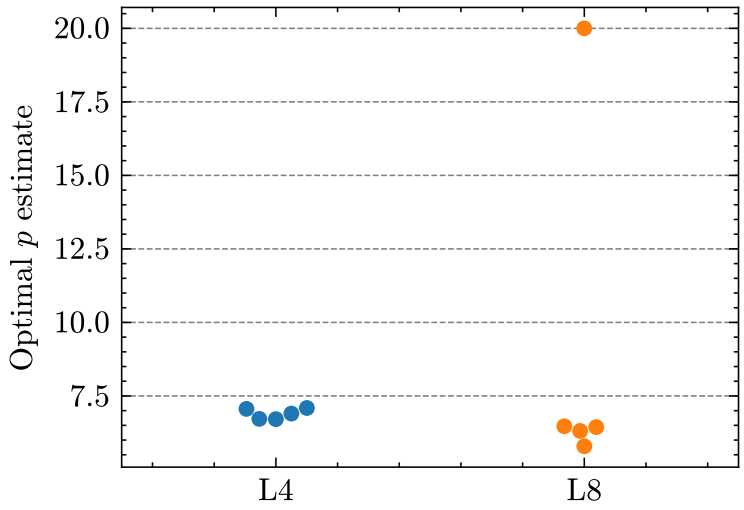


Figure 8.14: Plot of optimal estimated intensity p for baseline (L4) and validation (L8) buildings, 5 experiments in each case.

We begin by examining Figure 8.14 which displays the optimal p estimate for both baseline and validation buildings. The figure illustrates clearly that the NEI algorithm estimated the optimal intensity being quite similar for both buildings. There is a single outlier optimization iteration for L8, which did not converge to a correct estimate. However, the rest of the iterations did converge to a similar estimate for optimal intensity with both building types.

Furthermore, examining the iteration counts in Figure 8.15, we can conclude the NEI algorithm having a very similar performance in that regard. The iteration counts are very similarly located and have a similar spread. L8 has a slightly larger min-max range, however this is probably due to the inherent randomness in the NEI algorithm and the simulation samples.

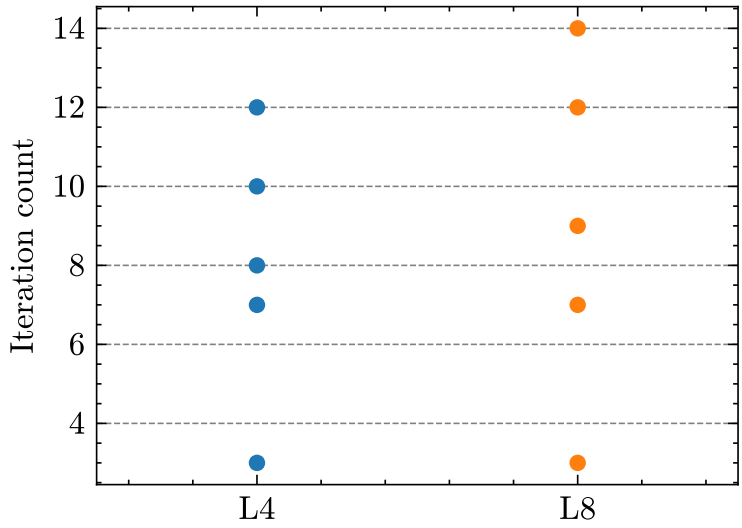


Figure 8.15: Plot of the iteration counts for baseline (L4) and validation (L8) buildings, 5 repetitions for both buildings.

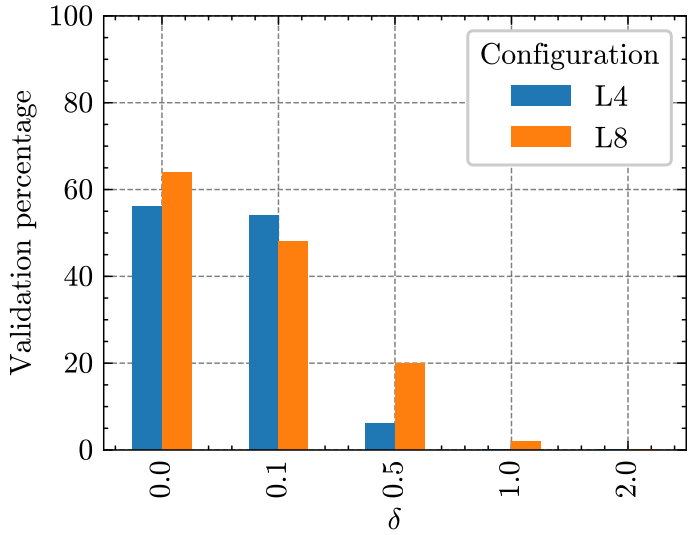


Figure 8.16: Validation percentage (Definition 8.1) for L4 and L8 buildings. Computed across 5 experiments with 10 validation samples for each δ in each experiment.

To validate the estimated optimal p being a good estimate of the optimal intensity, we can use Figure 8.16 which shows the validation percentage for both cases. The figure illustrates the fact that for both buildings, the NEI algorithm can find a good estimation of the optimal intensity. NEI algorithm returns a bit more pessimistic estimate for L8, since for example the validation percentage for $\delta = 0.5$ is 20 per cent. However, the validation percentage for $\delta = 1.0$ is very low. Indicating that while being perhaps a little conservative, the estimate for the L8 building is a good approximation of the true optimal intensity.

9 Discussion of results

In this section, we critically examine the results obtained in Chapters 7 and 8 and propose some solutions to the problems discovered alongside suggestions for further research. The main method for finding interesting details is to compare and combine results from both the toy problem and traffic intensity problem.

From the baseline results, we note that the NEI algorithm can be used as to find the optimal traffic intensity. It performed excellently in terms of the validity of the estimated optimal intensity while providing a competitive total computation time. Furthermore, compared to some of the other algorithms, NEI had a very stable performance in terms of iteration count. The NEI algorithm could also be used with a different building and elevator system, resulting in a very similar performance as seen in the validation case study.

However, the behaviour of the NEI algorithm is affected by the some of the problem parameters. The algorithm seems to struggle when the optimization region contains some traffic intensities which cause the elevator simulation to give very large values for AWT and ATTD, multiple times the limits $C_{\text{AWT}}, C_{\text{ATTD}}$. Therefore, to use the optimizer the domain of the optimization should be set appropriately to not contain such regions. One approach would be to first use some set of initial simulations and then use the NEI algorithm once a suitable region for the optimization has been identified. Alternatively, the use of domain knowledge could be used to set the boundaries of the optimization appropriately.

One other fact can be made evident from the ATTD limit case study. Namely, the NEI algorithm was more conservative in cases where both ATTD and AWT constraints were close to active. Therefore, in the case where both ATTD and AWT for estimated optimal intensity are close to the limit, we might underestimate the optimal intensity value. In these kinds of situations, the estimated value for optimal intensity might be increased manually to a more appropriate level, with the help of further validation simulations. However, this does require some domain knowledge of the problem and more computation time.

The length of the simulation also had an effect on the performance of the NEI algorithm. With a shorter simulation, the algorithm gave a more conservative estimate for the optimal intensity on average, however the results had a larger deviation as well. Therefore, the algorithm can be used with a shorter simulation but could give results which are not as reliable. Also the number of iterations increased, resulting in more computation time. This phenomena can be intuitively explained as when the length of the simulation is lower, the estimates for AWT and ATTD have a larger variation. These samples are then used by the optimizer which has to account for the increase in variation when fitting the Gaussian process models.

An alternative explanation for the worse performance with shorter simulations can be conjured by examining the stopping criterion. It is possible that the chosen cutoff value C_{AF} is too large to allow the NEI algorithm to find a good estimate of the maximum intensity. Further experiments with different values for C_{AF} should be to confirm which of the two proposed explanations is closer to reality.

One other aspect which should be investigated further is the affect of the kernel

on the performance of the algorithm. Specifically, the Matérn kernel for once-differentiable processes did not yield similar performance as the RBF kernel or the Matérn kernel for twice-differentiable processes. This indicates that the smoothness of the models for T_{AWT} and T_{ATTD} is possibly a key attribute when utilizing the NEI algorithm in the elevator traffic optimization setting. However, since this behaviour was not exhibited in the context of the toy problem, further research is needed to identify the real cause of this phenomena.

Another interesting result can be found by analyzing both the toy problem results and the traffic intensity problem results together. Specifically, we note that the CEI algorithm performed very similarly to the NEI algorithm in terms of its iteration count in the context of the toy problem, however the CEI algorithm performed poorly in that aspect when used in the traffic intensity problem. On the other hand, the estimates for the optimal intensity obtained using CEI were very similar to the ones obtained by CEI, even with the high iteration count. This indicates that CEI can provide a comparably good estimate for the optimal traffic intensity, if allowed to iterate longer than NEI.

Noting that CEI and NEI are both good algorithms for the two examined problems, NCEI-Mean exhibited bad performance in the context of the toy problem. This is mainly due to the high iteration count of the algorithm, exemplified by the fact that the iteration count often exceeded the allowed maximum of 64 iterations. However, the resulting estimate after the iteration ended was still a comparably good estimate of the true optimum point. An interesting exercise would be to use NCEI-Mean in the traffic intensity problem.

One final thing of note is the discrepancy between the estimate of the optimal intensity and the validation percentage, Figure 8.2 and Figure 8.3. Aside from NCEI, the other algorithms found similar estimates for the optimal intensity, however there is significant differences in the validation percentages for those algorithms. A simple explanation can concluded from the fact that the validation percentages are computed from noisy AWT and ATTD samples. Therefore, there are some uncertainties with the validation percentage results, especially with the relatively low repetition count of 10 samples.

On the other hand, another explanation could be that even small changes in the estimated maximum intensity cause the validation AWT and ATTD samples to be infeasible. In other words, some algorithms find a slightly more optimistic estimate of the optimum intensity, resulting in more of their validation samples being infeasible. A more careful analysis of the validation results and using longer validation simulations could be used to resolve this issue.

In conclusion, we have discovered some relevant constraints and weaknesses of the NEI algorithm alongside with interesting phenomena to study further. We also examined the performance of other candidate algorithms and found both strengths and weaknesses of the other candidate algorithms. We also expanded on some issues with our methodology which could affect our results and proposed further experiments to alleviate the problems.

10 Conclusion

We began this thesis by discussing the problem of elevator planning for office buildings. Through that discussion and by examining standards related to the area of elevator planning, we selected relevant metrics for designing a building for a given occupancy population. The relevant metrics were the average waiting time (AWT) and average time-to-destination (ATTD).

From the general problem of elevator planning, we formed a simplified inverse problem of finding maximum traffic intensity for a given building and elevator system, which satisfies some AWT and ATTD limits. We also discussed the process of using a simulator to estimate AWT and ATTD values for a given elevator system and building.

Then, we mathematically formalized the problem of finding a maximum traffic intensity. The mathematical formalism of our problem is a Bayesian optimization problem with expensive-to-evaluate constraints. From the theory of Bayesian optimization with Gaussian processes, we picked up crucial elements into our final optimization method, such as the expected improvement acquisition function.

In our implementation of the algorithms, we modified them to suit our particular optimization problem. Since the objective function is trivial, our implementations of the algorithms does not model them using a Gaussian process. We also added special considerations to handle noisy constraint function samples in implementing algorithms which did not take noise into account originally.

After implementing the various candidate algorithms according to literature and our specific problem setting, we tested them using a relevant toy problem with a trivial objective function and constraints which we pretended were expensive-to-evaluate. We also formed the constraints to have hopefully a similar structure to the elevator problem constraints, at least in terms of their monotonicity after a certain point. By analyzing the results from the toy problem, we eliminated some unsuitable algorithm candidates.

Using the remaining candidate algorithms and a suitable virtual test building, we produced optimization results for the maximum traffic intensity. We analyzed various metrics and came to the conclusion that the NEI algorithm performed the best overall. We performed a series of case studies using the NEI algorithm to analyze the sensitivity of the algorithm changes in certain parameters.

The case studies proved that the NEI algorithm is adequately good at adapting to changes in the problem setup. However, we found that in some circumstances the algorithm did not perform as well as in the baseline case. The NEI algorithm struggled in cases where for example, we included traffic intensity values which resulted in high the AWT and ATTD values.

As a final experiment, we used another building and elevator setup with the simulator to see how well the NEI algorithm generalized. From our results, we could note that the NEI algorithm performed well when using a new building. This indicates that the NEI algorithm is a suitable method to find the maximum traffic intensity with different elevator systems and buildings.

Using the results gathered from both the toy problem and the problem of finding the maximum intensity, we discussed some of the problems and constraints with the

NEI algorithm. Moreover, we also found some interesting phenomena, which could be interesting to research further. Some problems with our analysis methodology were also discussed.

Looking back at our research questions, we have discovered a mathematical definition for the problem of finding the maximum traffic intensity. We also found a reasonably robust method for solving the problem using a low number of simulation samples.

References

[1] ISO Standard 8100-32:2020(E). *Lifts for the transportation of persons and goods — Part 32: Planning and selection of passenger lifts to be installed in office, hotel and residential buildings*. 2020.

[2] N.A. Alexandris, G.C. Barney, and C.J. Harris. "Multi-car lift system analysis and design". In: *Applied Mathematical Modelling* 3.4 (1979), pp. 269–274. ISSN: 0307-904X. DOI: 10.1016/S0307-904X(79)80057-8.

[3] R. H. Byrd et al. "A Limited Memory Algorithm for Bound Constrained Optimization". In: *SIAM Journal on Scientific Computing* 16.5 (1995), pp. 1190–1208. DOI: 10.1137/0916069.

[4] M. Chandorkar, E. Camporeale, and S. Wing. "Probabilistic Forecasting of the Disturbance Storm Time Index: An Autoregressive Gaussian Process approach". In: *Space Weather* 15 (July 2017). DOI: 10.1002/2017SW001627.

[5] A. L. Cholesky. "Note Sur Une Méthode de Résolution des équations Normales Provenant de L'Application de la MéThode des Moindres Carrés a un Système D'équations Linéaires en Nombre Inférieur a Celui des Inconnues. — Application de la Méthode a la Résolution D'un Système Defini D'éQuations LinéAires". In: *Bulletin géodésique* 2.1 (Apr. 1924), pp. 67–77. ISSN: 1432-1394. DOI: 10.1007/BF03031308.

[6] I. Czekala et al. "Disentangling Time Series Spectra with Gaussian Processes: Applications to Radial Velocity Analysis". In: *The Astrophysical Journal* 840 (Feb. 2017). DOI: 10.3847/1538-4357/aa6aab.

[7] D. K. Duvenaud. "Automatic Model Construction with Gaussian Processes". PhD thesis. University of Cambridge, 2014.

[8] P. I. Frazier. *A Tutorial on Bayesian Optimization*. 2018. arXiv: 1807.02811 [stat.ML].

[9] P. I. Frazier, W. B. Powell, and S. Dayanik. "A Knowledge-Gradient Policy for Sequential Information Collection". In: *SIAM Journal on Control and Optimization* 47.5 (2008), pp. 2410–2439. DOI: 10.1137/070693424.

[10] S. Gallarati et al. "Reaction-based machine learning representations for predicting the enantioselectivity of organocatalysts". In: *Chem. Sci.* 12 (20 2021), pp. 6879–6889. DOI: 10.1039/D1SC00482D.

[11] J. R. Gardner et al. "Bayesian Optimization with Inequality Constraints". In: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. ICML'14. Beijing, China: JMLR.org, 2014, II–937–II–945.

[12] M. Gelbart, J. Snoek, and R. Adams. "Bayesian Optimization with Unknown Constraints". In: *Uncertainty in Artificial Intelligence - Proceedings of the 30th Conference, UAI 2014* (Mar. 2014).

[13] A. Glad et al. “Reducing Energy Consumption by an Optimization Algorithm in Elevator Group Control”. In: *International Elevator Escalator Symposium 2022* (Dec. 12–13, 2022). 2022, pp. 73–81.

[14] GPy. *GPy: A Gaussian process framework in python*. http://github.com/SheffieldML/GPy. since 2012.

[15] C. R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

[16] T. Hautamäki. “Multiobjective Optimization Model for Elevator Call Allocation”. M.Sc. thesis. Aalto university, 2021.

[17] P. Hennig and C. J. Schuler. “Entropy search for information-efficient global optimization”. In: *Journal of Machine Learning Research.* 13 (June 2012), pp. 1809–1837. ISSN: 1532-4435.

[18] D. Jones, M. Schonlau, and W. Welch. “Efficient Global Optimization of Expensive Black-Box Functions”. In: *Journal of Global Optimization* 13 (Dec. 1998), pp. 455–492. DOI: 10.1023/A:1008306431147.

[19] H. J. Kushner. “A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise”. In: *Journal of Basic Engineering* 86.1 (Mar. 1964), pp. 97–106. ISSN: 0021-9223. DOI: 10.1115/1.3653121. eprint: https://asmedigitalcollection.asme.org/fluidsengineering/article-pdf/86/1/97/5763745/97_1.pdf.

[20] B. Letham et al. “Constrained Bayesian Optimization with Noisy Experiments”. In: *Bayesian Analysis* 14 (June 2017). DOI: 10.1214/18-BA1110.

[21] J. Marsh and J. Sit CNN. *Shanghai Tower picks up 3 Guinness World Records including fastest elevator*. Accessed on 24.4.2024. 2017. URL: https://edition.cnn.com/style/article/worlds-fastest-tower/index.html.

[22] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[23] C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for Machine Learning*. MIT Press, 2006.

[24] S. Sano et al. “Application of Bayesian Optimization for Pharmaceutical Product Development”. In: *Journal of Pharmaceutical Innovation* 15 (Mar. 2019), pp. 1–11. DOI: 10.1007/s12247-019-09382-8.

[25] A. N. Shiryayev. “Interpolation and Extrapolation of Stationary Random Sequences”. In: *Selected Works of A. N. Kolmogorov: Volume II Probability Theory and Mathematical Statistics*. Ed. by A. N. Shiryayev. Dordrecht: Springer Netherlands, 1992, pp. 272–280. ISBN: 978-94-011-2260-3. DOI: 10.1007/978-94-011-2260-3_28.

[26] M. L. Siikonen. “Elevator Traffic Simulation”. In: *Transactions of The Society for Modeling and Simulation International - SIMULATION* 61 (Oct. 1993), pp. 257–267. DOI: 10.1177/003754979306100409.

[27] M. L. Siikonen, T. Susi, and H. Hakonen. “Passenger Traffic Flow Simulation in Tall Buildings”. In: *Elevator World* 49 (Aug. 2001), pp. 117–123.

[28] I. M. Sobol. “On the distribution of points in a cube and the approximate evaluation of integrals”. In: *USSR Computational Mathematics and Mathematical Physics* 7.4 (1967), pp. 86–112. ISSN: 0041-5553. DOI: 10.1016/0041-5553(67)90144-9.

[29] G. Van Rossum and F. L. Drake Jr. *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[30] N. Wiener. *Extrapolation, Interpolation, and Smoothing of Stationary Time Series: With Engineering Applications*. The MIT Press, Aug. 1949. ISBN: 9780262257190. DOI: 10.7551/mitpress/2946.001.0001. eprint: https://direct.mit.edu/book-pdf/2313079/book_9780262257190.pdf.

[31] A. G. Zhilinskas. “Single-step Bayesian search method for an extremum of functions of a single variable”. In: *Cybernetics* 11 (1975), pp. 160–166. URL: https://api.semanticscholar.org/CorpusID:119898868.

A Algorithm hyperparameters

This appendix contains information about the hyperparameters used in the different Bayesian optimization algorithms. See Chapter 6 for the implementation details of the various algorithms.

A.1 Toy problem

For the toy problem, described in Chapter 7, we will utilize the following hyperparameters as described in Table A1.

Algorithm	C_N	K	M	δ	C_{AF}	$\sigma_\epsilon^{\text{stability}}$	$\sigma_\epsilon^{\text{min}}$	$\sigma_\epsilon^{\text{max}}$
NEI	64	32	100	-	0.001	10^{-6}	10^{-6}	100
CEI	64	-	-	-	0.001	10^{-6}	0	0
NCEI	64	-	-	-	0.001	10^{-6}	10^{-6}	100
CEI-Mean	64	-	-	0.05	0.001	10^{-6}	0	0
NCEI-Mean	64	-	-	0.05	0.001	10^{-6}	10^{-6}	100

Table A1: Hyperparameter values for different algorithms.

Table A1 contains 4 different parameter values for the different algorithms. C_N refers to the number of maximum iterations during the Bayesian optimization algorithm. For the NEI algorithm, K number refers to the number of estimates made with the Sobol sequence. Also, M refers to the parameter which describes the explore-exploit ratio in the case with having no feasible samples in the NEI algorithm. For K and M parameters, see Chapter 6 for more information. δ refers to the culling of possibly unfeasible samples made in the CEI-Mean and NCEI-Mean algorithms. The C_{AF} refers to the stopping criterion cutoff point as described in Chapter 6.

Finally, the different σ parameters refer to the process of fitting a Gaussian process model. The $\sigma_\epsilon^{\text{stability}}$ is a value which is added to the covariance matrix of a Gaussian process model as described in Section 4.2. It is used to improve the numerical stability of the posterior distribution calculation. The two remaining parameters, $\sigma_\epsilon^{\text{min}}$ and $\sigma_\epsilon^{\text{max}}$, are the limits of the fitted noise value. In other words, the fitted noise value σ_ϵ should be within those bounds. See Algorithm 4.1 for more details.

A.2 Elevator problem

The hyperparameters used in the elevator problem are displayed in Table A2. They are similar to the ones used in the toy problem (Table A1). One significant difference is the increase of the cutoff value for the acquisition function from 0.001 to 0.05. We also increased the maximum noise bound $\sigma_\epsilon^{\max}$ to include very high noise levels in applying the algorithms to the elevator problem.

Algorithm	C_N	K	M	δ	C_{AF}	$\sigma_\epsilon^{\text{stability}}$	$\sigma_\epsilon^{\min}$	$\sigma_\epsilon^{\max}$
NEI	64	32	100	-	0.05	10^{-6}	10^{-6}	10^{9}
CEI	64	-	-	-	0.05	10^{-6}	0	0
NCEI	64	-	-	-	0.05	10^{-6}	10^{-6}	10^{9}
CEI-Mean	64	-	-	0.05	0.05	10^{-6}	0	0

Table A2: Hyperparameter values for different algorithms used in solving the elevator problem.