# POLITECNICO DI TORINO

**MASTER's Degree in Mathematical Engineering**


**MASTER's Degree Thesis**

# A surrogate-based robust optimization approach for the management of perishable products

**Supervisors**

**Prof. Paolo BRANDIMARTE**

**Prof. Edoardo FADDA**

## Candidate

## Aurona GASHI

**October 2025**

# Summary

An important challenge present in the retail setting is the handling of perishable goods, that is, those products for which there is a limited time to consume them. One example of all is food. It is clear that implementing improved management strategies is crucial since it can greatly decrease food waste by selling food prior to its sell-by date. Perishable retailers face the challenge of maximizing profitability and in addition trying to reduce food waste as well. This thesis focuses on the construction of a method for optimizing replenishment policies and the application of discounts for perishable products in a robust setting with respect to the parameters modeling the customers' preferences. It has already been demonstrated in the literature that the application of discounts has the potential to contribute to reducing waste and increasing profitability. The setting is established within a robust framework that includes a multi-item environment with substitutable products with the objective to maximize the expected daily profit. A major challenge in the management of perishable goods is the uncertainty of demand, which makes planning and scheduling difficult. In our analysis, we consider a vertical differentiation setting for the substitutable products, where price is the determining factor when the products are of the same quality. A linear discrete demand model is used to estimate the utility of each product for each customer. A stochastic parameter that depends on the customer's valuation is employed, while the price and quality of the product are used to model the utility in a linear form. The parameter is assumed to be modeled by a Beta distribution. We consider products with given deterministic prices, qualities and costs, where each product faces a quality degradation as its residual life decreases. A thorough first phase of simulations has been instrumental in providing insight into the environment and the impact of changing parameters and products on profit. Following this phase, the optimization phase is initiated. The approach taken to optimize the policies involves the implementation of a metamodel. In particular, we adopt Kriging, a low-cost interpolation method that uses a stochastic process approach to construct a surrogate model of an expensive function. In our case, we refer to the proposed algorithm as the Efficient Global Robust Optimization algorithm. After an initialization phase, the next step is an iterative process in which the new sampling locations are selected using an

adaptive sampling method. Given the nature of this problem as a worst-case robust configuration, we define a control variable space and an uncertain variable space. In this analysis, we refer to the order parameters and the discounting variables as the control variables. The parameters representing the Beta distribution of the uncertain parameter that models the utility, as well as the coefficient of variation on the distribution of consumers entering the store in some experiments, are considered to be uncertain variables. Two versions of the Expected Improvement Criterion are selected for sampling in the designated spaces. The significance of this work lies in the scarcity of literature dealing with the handling of perishable products in a worst-case, robust environment. This makes it a valuable contribution to the field. The results demonstrate the effectiveness of a metamodeling approach to the topic, as well as its adaptability in different settings.

# Table of Contents

**List of Tables** VII

**List of Figures** VIII

**Acronyms** XI

**1 Introduction** 1

**2 Literature review** 3
2.1 Inventory management and pricing policies . . . 3
2.2 Discrete choice models and utility functions . . . 4

**3 Context and problem definition** 7
3.1 Problem overview . . . 7
3.2 Decision policies . . . 11
3.3 Problem Statement . . . 12

**4 Kriging Surrogate Modeling** 14
4.1 Modeling Assumptions . . . 14
4.1.1 Example: Kriging fit and prediction with confidence bounds 15
4.2 Efficient Global Optimization . . . 16
4.3 Efficient Global Robust Optimization . . . 18
4.3.1 Initial Sampling . . . 18
4.3.2 Selection in $\mathcal{X}$ . . . 18
4.3.3 Selection in $\mathcal{U}$ . . . 19

**5 Implementation details** 22
5.1 Welch's method for steady-state mean estimation . . . 23
5.1.1 Steady-state mean estimation . . . 24
5.2 Main optimization algorithm . . . 25

**6 Computational experiments and results** 32
6.1 Base Stock Policy No Discount . . . 33
6.1.1 Discrete domain for $(\alpha, \beta)$ and discrete domain for $cv$ . . . 33
6.1.2 Discrete domain for $(\alpha, \beta)$ and continuous domain for $cv$ . . 37
6.1.3 Continuous domain for $(\alpha, \beta)$ and fixed value for $cv$ . . . 40
6.2 Base Stock Simple . . . 43
6.2.1 Continuous domain for $(\alpha, \beta)$ and fixed $cv$ . . . 43
6.3 Constant Order No Discount . . . 46
6.3.1 Continuous domain for $(\alpha, \beta)$ and fixed $cv$ . . . 46
6.4 Comparison between policies . . . 48

**7 Conclusions** 52

**A Simulation environment** 54
A.1 Agents . . . 54
A.1.1 `BaseStockNoDiscount` . . . 54
A.1.2 `BaseStockSimple` . . . 54
A.1.3 `ConstantOrderNoDiscount` . . . 55
A.1.4 `ConstantOrderSimple` . . . 55
A.2 `DailySimulation` Environment . . . 56
A.3 Managers . . . 57
A.3.1 `CustomerManager` . . . 57
A.3.2 `InventoryManager` . . . 58
A.3.3 `StatManager` . . . 58
A.3.4 `SupplyManager` . . . 59

**Bibliography** 60

# List of Tables


3.1 Products A and B with SL = 3. . . . . 7
3.2 Products A and B with SL = 5. . . . . 8
3.3 Products A and B with SL = 7. . . . . 8

6.1 Final solutions summary for each SL. . . . . 34
6.2 Domains of the base stock level ($bs$) for each product. . . . . 37
6.3 Final solutions summary for each SL. . . . . 38
6.4 Domains of the base stock level ($bs$) for each product. . . . . 40
6.5 Final solutions summary for each SL. . . . . 41
6.6 Domains of the discount variables for each product. . . . . 43
6.7 Final solutions obtained for each value of SL. . . . . 43
6.8 Final solutions summary for each SL. . . . . 46
6.9 Comparison of key daily metrics across the four ordering policies for SL = 3. . . . . 50
6.10 Comparison of key daily metrics across the four ordering policies for SL = 5. . . . . 50
6.11 Comparison of key daily metrics across the four ordering policies for SL = 7. . . . . 50

# List of Figures


3.1 Utilities and Beta(2,3) for products of $\mathsf{SL} = 3$ and varying $\tau$. . . . . 10
3.2 Utilities and Beta(2,3) for products with $\mathsf{SL} = 3$ and $\tau = 3$, when a 40% discount is applied on product B with $\tau = 1$. . . . . 11

4.1 Non linear function and Kriging approximation. . . . . 16

5.1 Moving averages in Welch's method. . . . . 25
5.2 95% confidence intervals and means computed at each iteration. . . 25

6.1 Uncertainty set for $(\alpha, \beta)$ in the continuous case. . . . . 33
6.2 Initial sampled ($bs_A$, $bs_B$) colored by $cv$ ($\mathsf{SL} = 3, 5, 7$). . . . . 34
6.3 Frequency of $(\alpha, \beta)$ pairs in sampled points ($\mathsf{SL} = 3, 5, 7$). . . . . 34
6.4 Robust solution value per iteration. . . . . 35
6.5 Maximum $EI_c$ values per iteration. . . . . 36
6.6 Confidence intervals and means for evaluated points with same uncertainty parameters as final solution. . . . . 36
6.7 Initial sampled ($bs_A$, $bs_B$) colored by $cv$ for $\mathsf{SL} = 3$. . . . . 37
6.8 Frequency of $(\alpha, \beta)$ pairs in sampled points for $\mathsf{SL} = 3$. . . . . 38
6.9 Robust solution value per iteration. . . . . 39
6.10 Maximum $EI_c$ values per iteration. . . . . 39
6.11 Confidence intervals and means for evaluated points with same uncertainty parameters as final solution. . . . . 40
6.12 Initial sampled ($bs_A$, $bs_B$) colored by $cv$ for $\mathsf{SL} = 3$. . . . . 41
6.13 Robust solution value per iteration. . . . . 41
6.14 Maximum $EI_c$ values per iteration. . . . . 42
6.15 Confidence intervals and means for evaluated points with same uncertainty parameters as final solution. . . . . 42
6.16 Robust solution value per iteration. . . . . 44
6.18 Confidence intervals and means for evaluated points with same uncertainty parameters as final solution. . . . . 44
6.17 Maximum $EI_c$ values per iteration. . . . . 45

6.19 Robust solution value per iteration. . . . . 46
6.20 Maximum $EI_c$ values per iteration. . . . . 47
6.21 Confidence intervals and means for evaluated points with same uncertainty parameters as final solution. . . . . 47
6.22 Comparison of confidence intervals of the solutions' average daily profits. . . . . 48

# Acronyms

**LIFO**
Last-In-First-Out

**FIFO**
First-In-First-Out

**B2C**
Business to Consumer

**LEFO**
Last-Expired-First-Out

**FEFO**
First-Expired-First-Out

**MNL**
Multinomial Logit

**cv**
coefficient of variation

**BSPnd**
Base Stock Policy No Discount

**BSPs**
Base Stock Policy Simple

**COPnd**
Constant Order Policy No Discount

**COPs**

Constant Order Policy Simple

**EGRO**

Efficient Global Robust Optimization

**SMT**

Surrogate Modeling Toolbox

**EGO**

Efficient Global Optimization

**EI**

Expected Improvement

**DCM**

Discrete Choice Model

# Chapter 1

# Introduction

The management and pricing of perishables can be a fundamental task for companies selling them. Firstly, it is important to know the classification of products into the groups of *Obsolescence* and *Deterioration*. When referring to obsolescence, a product faces a reduction in price due to market or technological reasons, whereas an item can go through deterioration only because of some internal characteristics as in the case of ageing. Therefore perishability belongs to the second group (Gioia et al. 2022). Products with a maximal shelf life (use-by date) of a few weeks are considered to be perishable products (Buisman et al. 2019). A relevant operational dilemma arises: should retailers sell older products together with fresher ones, possibly offering a discount? How should replenishment be managed in such situations? There is no easy answer to these questions. Demand uncertainty and consumer behavior, such as preferences for new or older products, complicate inventory control. Consumers may follow different issuing policies (LIFO or FIFO), but retail settings often lack tight control over which items are chosen. Studies on the management of perishable goods emphasize that pricing is a key lever for balancing profitability and waste reduction. Developing effective pricing methods is essential not only for newer products, but also for older inventory. What should not take for granted is the impact food waste has on the environment. Managing the balance between the two can significantly improve profitability and reduce waste. To encourage sales of older items and reduce waste, discounts on aging products can be effective. A key aspect in designing such pricing strategies is understanding the dynamics of product perishability and customer demand. The first step is to determine the product's shelf life. When shelf life is considered deterministic, the remaining days before expiration are known with certainty. Demand can be either deterministic or stochastic. In real-world settings, however, demand is rarely predictable. Assuming deterministic demand often results in oversimplified models that fail to capture actual variability. Therefore, this study considers stochastic, independent demand that evolves over time while accounting for delivery lead times.

Furthermore, the analysis is extended to a multi-item context with substitution between similar products. Specifically, the analysis focuses on two substitutable products with deterministic shelf life.
Discrete choice models can be employed to model demand, as they can be capable of capturing vertical or horizontal product differentiation. Vertical differentiation reflects quality-based differences, where consumers share a common preference ranking. In contrast, horizontal differentiation accounts for idiosyncratic preferences, such as taste or brand loyalty, where no product is universally preferred. This study focuses on vertical differentiation, which is particularly relevant in grocery retail, where product age or residual shelf life strongly influence consumer choices.
indent We assume that customers are myopic and base their purchasing decisions solely on current utility without anticipating future price changes. Consumer utility is modeled using a linear discrete choice model in which original product prices are treated as exogenously determined (Gioia et al. 2022; Fadda et al. 2024).

# Chapter 2

# Literature review

In this study, we address inventory management and pricing decisions together by considering a scenario in which the initial product price is set, but discounts can be applied as items age. This integrated approach captures the interplay between pricing and inventory dynamics to increase profitability and reduce waste. Traditionally, these two areas have been studied separately. Inventory management models often assume constant, exogenous prices, while dynamic pricing literature solely considers how price adjustments affect demand without accounting for stock levels or aging. Below, we review key contributions from both streams of literature. Besides pricing and inventory management, modeling demand through consumer choice behavior plays a crucial role. The literature focuses on various choice models that describe how consumers make purchasing decisions. Thus, we also present some relevant work on discrete choice models and utility functions.

## 2.1 Inventory management and pricing policies

Chua et al. (2017) analyze inventory and discounting policies for perishable goods with short shelf life and uncertain demand. The base model assumes a two-period shelf life and shows that a threshold discount policy reduces waste: discounts are applied only when old stock is below a threshold. Extensions include endogenous discount decisions, new customer segments (bargain hunters), and longer shelf lives. Results show that flexible discounting improves profits, attracts more customers, and helps manage waste, with two-period models offering useful approximations even for longer shelf lives. Moving on, Sainathan (2013) analyzes a retailer's simultaneous pricing and ordering decisions for a perishable product with a two-period shelf life over an infinite horizon. The retailer sets prices for both product ages and orders new stock each period. Key findings include: demand uncertainty can make selling old products profitable, and allowing only the order quantity

to vary dynamically captures most benefits of full flexibility. The optimal price of the new product does not always decrease with old inventory, contradicting common clearance strategies. Market segmentation by selling old products with constant pricing yields much higher benefits than fully flexible policies, which can be costly to implement. The model highlights when and how retailers should vary decision variables, offering practical insights such as avoiding price changes due to potential negative customer reactions. Differently, Solari et al. (2024) explore how discount strategies can help reduce waste and improve profitability in B2C retail settings where perishable products are managed. In such environments, consumers typically prefer items with longer remaining shelf lives, increasing the likelihood that near-expiry products go unsold. The authors develop a simulation-based model incorporating a mixed LEFO-FEFO issuing policy and analyze how discounts can shift demand toward expiring products. Their findings suggest that well-designed discount policies can effectively mitigate waste, improve shelf utilization, and enhance overall profitability. Turning to Chew et al. (2014), the authors develop a model to jointly determine optimal order quantities and pricing for perishable products with multiple-period lifetimes, explicitly accounting for demand substitution among products of different ages. For products with a two-period lifetime, they formulate a stochastic dynamic programming model and demonstrate that considering demand transfers between old and new products significantly enhances total profit. For products with longer lifetimes, the authors propose a heuristic based on the optimal single-period solution to address the complexity. Next, Haijema and Minner (2015) show that traditional stock-level policies, like base-stock and constant order, are popular for perishable inventory due to their simplicity, relying only on total stock without age differentiation. This study compares hybrid policies via simulation optimization, finding that some enhanced versions outperform traditional ones by smoothing orders and cutting costs under specific lead time and demand scenarios. Finally, Buisman et al. (2019) develop a simulation-based optimization model to improve replenishment and discounting policies for perishable foods. They show that using a Dynamic Shelf Life based on actual product quality significantly reduces waste and improves profit compared to a Fixed Shelf Life. Combining a Dynamic Shelf Life with dynamic discounting yields the best performance in terms of waste reduction, profit and food safety.

## 2.2 Discrete choice models and utility functions

Understanding how consumers make purchasing decisions is essential in problems involving substitutable products, especially when product attributes like price and quality (or freshness) vary. To model this behavior, we rely on discrete choice

models, which allow us to represent the utility each consumer associates with the available alternatives. The literature proposes a variety of such models, differing in their assumptions about consumer heterogeneity, preference structure and product characteristics. We present several approaches from the literature, each defining a different utility function to model consumer preferences.

A first group of models adopts a simple linear structure for utility, where consumer preferences depend on product quality and price, without incorporating horizontal differentiation. In Sainathan (2013), each consumer $n$ chooses between two versions of the same perishable product (i.e., the older unit or the fresher one) or no purchase based on the following utility function for each version $i$

$$U_{ni} = \theta_n q_i - p_i,$$

where $q_i$ represents the quality associated with freshness, $p_i$ the price, and $\theta_n$ is drawn from a uniform distribution on [0,1] to capture heterogeneity across consumers. A similar linear structure is found in Transchel et al. (2022), although in a non-perishable setting. What distinguishes this model is the substitution mechanism: if the preferred product (i.e., the one with the highest utility) is unavailable, the consumer moves to the next-best option, and so on. While perishability is not explicitly modeled, this behavior realistically reflects how consumers may respond to stockouts.

A different formulation is proposed in Akçay et al. (2010), where the utility function includes a stochastic component to account for idiosyncratic preferences toward specific products. The utility for consumer $n$ and product $i$ is given by

$$U_{ni} = \theta_n q_i - p_i + \mu \xi_{ni}, \tag{2.1}$$

with $\xi_{ni}$ capturing product-specific preference shocks and $\mu$ controlling their magnitude. This model introduces horizontal differentiation, reflecting differences in individual tastes beyond price and quality. This formulation is useful for modeling unobservable heterogeneity in large assortments. When $\mu = 0$ and $\theta_n \sim \mathcal{U}[0,1]$ in 2.1, the model reduces to the previous case of vertical differentiation only.

Finally, a widely used class of models is based on the Multinomial Logit (MNL) framework (Sifringer et al. 2020; Ben-Akiva et al. 2003; Herring et al. 2020). These models express the utility of consumer $n$ for product $j$ as

$$U_{ni} = V_{ni} + \epsilon_{ni},$$

where $V_{ni} = \beta^\top x_{ni}$, with $x_{ni}$ denoting the observed attributes of product $i$ (e.g., price, freshness) and $\beta$ the associated preference parameters.

Assuming that the error terms $\epsilon_{ni}$ are i.i.d. Gumbel-distributed, the probability that consumer $n$ selects product $i$ is

$$P_n(i) = \frac{e^{V_{ni}}}{\sum_{k \in C_n} e^{V_{nk}}},$$

where $C_n$ denotes the set of available products for consumer $n$. The parameters $\beta$ are estimated via maximum likelihood, maximizing the following function

$$L = \sum_{n=1}^{N} \sum_{i \in C_n} y_{ni} \log \left[ P_n(i) \right],$$

where $y_{nj} = 1$ if consumer $n$ chooses product $j$, and 0 otherwise. MNL models allow for flexible substitution patterns and are particularly effective when modeling choices among alternatives with multiple attributes.

Together, these models offer a range of tools for capturing consumer behavior in the presence of price differences, freshness, and availability. The choice of model depends on the specific application and trade-offs between analytical tractability and behavioral realism.

The policies adopted in this work are inspired by inventory models such as those discussed above, which demonstrate that simple and structured approaches can effectively balance profitability and operational complexity under uncertainty. For the modeling of demand, we follow the framework proposed by Gioia et al. (2022), which is closely aligned with the approaches developed by Sainathan (2013) and Transchel et al. (2022). The specific modeling choices are described in detail in Chapter 3.

# Chapter 3

# Context and problem definition

We begin by detailing the problem under consideration and the adopted approach. Starting from the ideas in Fadda et al. (2024) and Gioia et al. (2022), we develop a robust optimization framework that handles multiple items.

## 3.1 Problem overview

We examine a combined pricing and replenishment problem within a discrete periodic review setting for two perishable products, featuring deterministic shelf lives, replenishment lead times and stochastic demand. The two products, namely A and B, are substitutable, meaning that they serve a similar purpose from the customer is perspective and compete in the same category. Each item of each product $i \in \{\mathrm{A}, \mathrm{B}\}$ is characterized by its purchasing cost $c^i$, selling price $p^i$, a fixed discrete delivery lead time $\mathsf{LT}^i$, and a deterministic discrete shelf life $\tau$ (in days), which is equal to $\mathsf{SL}^i$ upon delivery. Here, $\tau$ denotes the residual shelf life of a unit of product $i$; the superscript $i$ is omitted for clarity and applies throughout. Every item with $\tau > 0$ is perceived with a quality $q^i_\tau < q^i_{\tau+1}$, (quality increasing with $\tau$); when $\tau = 0$, the item is scrapped. We define three possibilities for the shelf life: 3, 5, 7 days. The characteristics of the two products for the possible values of shelf life are shown in Tables 3.1, 3.2 and 3.3.

**Table 3.1:** Products A and B with $\mathsf{SL} = 3$.

| Product | LT | Price | Cost | Qualities (per period) |
|---|---|---|---|---|
| A | 1 | 6 | 4 | 24.5, 23, 18 |
| B | 1 | 5.5 | 3.55 | 23.5, 22, 17 |

**Table 3.2:** Products A and B with $\mathsf{SL} = 5$.

| Product | LT | Price | Cost | Qualities (per period) |
|---|---|---|---|---|
| A | 1 | 6 | 4 | 30, 29, 28, 26, 24 |
| B | 1 | 5.5 | 3.55 | 29, 28, 27, 25, 23 |

**Table 3.3:** Products A and B with $\mathsf{SL} = 7$.

| Product | LT | Price | Cost | Qualities (per period) |
|---|---|---|---|---|
| A | 1 | 6 | 4 | 30, 29.5, 29, 28, 26, 24, 22 |
| B | 1 | 5.5 | 3.55 | 29, 28.5, 28, 27, 25, 23, 21 |

The simulation is divided discrete time steps, where each step corresponds to one day. The retailer makes decisions every day concerning new orders and the possible application of discounts on the items. At the start of day $t$, prices for existing items are updated based on applied discounts; we denote by $p_\tau$ the price of items that have a residual shelf life of $\tau$ days. The quantity of items scheduled for delivery in $l \in \{0, \dots, \mathsf{LT}\}$ days is represented by $O_t^l$, while the on-hand inventory of items with residual shelf life $\tau \in \{1, \dots, \mathsf{SL}\}$ days at time $t$ before sales is indicated by $I_t^\tau$.
Considering two products, we track these quantities separately for each product to accurately represent inventory dynamics. Thus, for product $i \in \{A, B\}$, we denote the delivered items by $O_t^{l,i}$ and the on-hand inventory by $I_t^{\tau,i}$. Prices are also adjusted independently per product and residual shelf life, with $p_\tau^i$ representing the price for product $i$ with shelf life $\tau$. In each experiment, we fix $\mathsf{SL}$ to a given value, equal for both products. This modeling choice makes the two products more comparable, as it removes differences arising from perishability duration.
This discrete-time approach enables detailed modeling of inventory aging, delivery scheduling and dynamic pricing strategies for multiple products simultaneously.
The number of customers entering the shop is modeled as a Negative Binomial random variable. As done in Fadda et al. (2024), we set the mean daily number of customers $\mu = 30$. The standard deviation $\sigma$ is derived based on the coefficient of variation ($cv$), which reflects the variability in demand beyond the mean. In particular, following the calculations in Fadda et al. (2024), we consider two levels of demand uncertainty: a baseline setting with $cv = 0.3$ (i.e., $\sigma = 9$) and, in some experiments, a more challenging setting with $cv = 0.7$ (i.e., $\sigma = 21$). This approach allows modeling daily arrivals as independent Negative Binomial random variables with fixed mean but adjustable variance to capture different demand uncertainty scenarios. Each day the shop opens, customers arrive and decide whether to buy one item from any of the available products or to buy nothing. Following Gioia et al. (2022), we adopt a discrete choice framework to model consumer behavior. Each

customer $n$ evaluates the utility of any available unit of product $i$ with residual shelf life $\tau$ through a linear function

$$U_{n\tau}^{i} = \theta_n q_{\tau}^{i} - p_{\tau}^{i},$$

where $\theta_n \sim \text{Beta}(\alpha, \beta)$ represents the consumer's price-quality sensitivity. $\tau$ uniquely identifies each alternative within a product category. Thus, items of the same product with different remaining shelf lives are treated as distinct alternatives.
A purchase occurs if at least one utility value is positive; otherwise, the customer opts for the no-purchase option, modeled with utility $U_{n0} = 0$. The product variant chosen is the one yielding the highest utility

$$\text{Choice}_n = \arg\max \left\{ U_{n0}, U_{n\tau}^{i} \right\}, \quad \text{for all } i \in \{A, B\},\ \tau \in \{1, \ldots, \mathsf{SL}^{i}\}.$$

Therefore, to account for perishability, we model quality degradation due to ageing as an additional vertical differentiation within each product category. Specifically, we disaggregate each product into multiple variants distinguished by their residual shelf life $\tau$, treating them as distinct alternatives in the consumer choice process. This effectively enriches the quality space and allows consumers to differentiate between fresher and older items of the same product, while the retailer continues to manage inventory at the product level.
As we can see from Tables 3.1, 3.2 and 3.3, the product A is defined by a higher price, cost and higher qualities (at a given $\tau$) than product B. To better understand the differences in product choice, we analyze the previously defined utility function The consumer prefers product $A$ over $B$ when

$$U_{n\tau}^{A} > U_{n\tau}^{B},$$

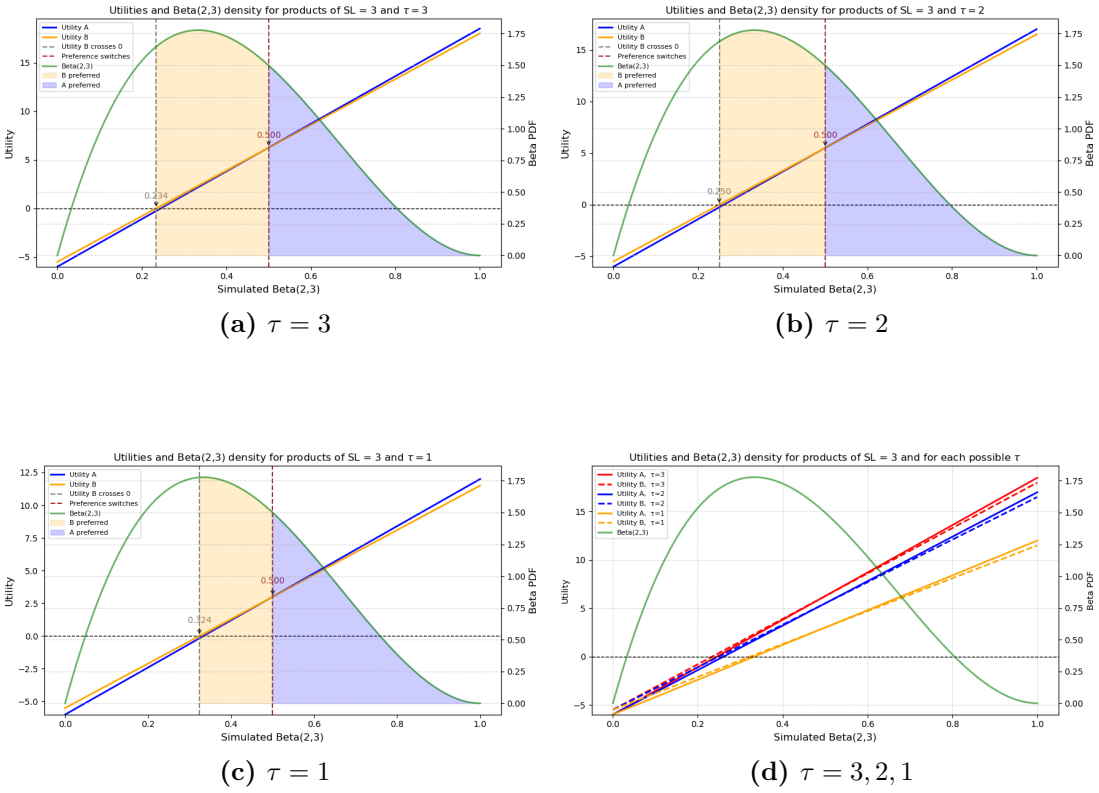
which leads to the switching point in $\theta_n$:

$$\theta_n^{*}(\tau) = \frac{p_{\tau}^{A} - p_{\tau}^{B}}{q_{\tau}^{A} - q_{\tau}^{B}}.$$

For the given prices and qualities at each $\tau$, this switching value $\theta_n^{*}(\tau)$ is always 0.5. Moreover, in the absence of discounts, an item with lower residual life never yields a higher utility than an item of the same product with higher residual life. That is

$$\tau' < \tau \quad \Rightarrow \quad U_{n,\tau'}^{i} \leq U_{n,\tau}^{i} \quad \text{for all } \theta_n \in [0,1].$$

We can see from the plots in Figure 3.1 what happens in the case of the products, with $\mathsf{SL} = 3$, when no discounts are applied and when varying the simulated $\theta$, in the case of $\theta \sim \text{Beta}(2,3)$. It is quite interesting to observe how things change when applying a discount. For instance, in Figure 3.2, we apply a 40% discount




**(a)** $\tau = 3$ **(b)** $\tau = 2$

**(c)** $\tau = 1$ **(d)** $\tau = 3, 2, 1$

**Figure 3.1:** Utilities and Beta(2,3) for products of $\mathsf{SL} = 3$ and varying $\tau$.

on product B with $\tau = 1$. As a result, product B becomes the preferable choice over every other item for lower values of $\theta$, and consistently remains higher than product A with the same $\tau$, unlike the scenario without any discount. This example illustrates the significant impact that discounts can have on customer choices and is be explored on some policies in Section 3.2.

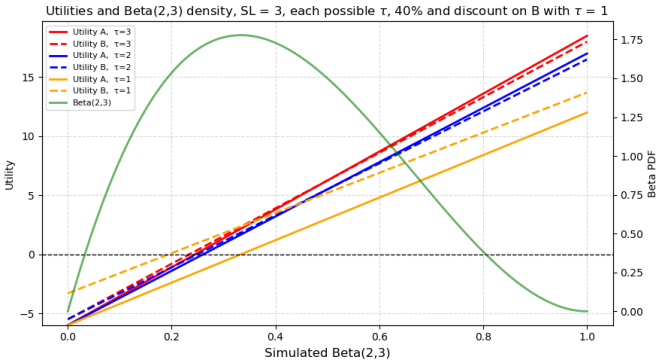



**Figure 3.2:** Utilities and Beta(2,3) for products with $\mathsf{SL} = 3$ and $\tau = 3$, when a 40% discount is applied on product B with $\tau = 1$.

## 3.2 Decision policies

Before diving into the actual objective, we show the policies that are used in the problem. As partially done in Fadda et al. (2024), we define the following parametric policies where decisions are made simultaneously for each product and therefore may differ between them:

- **Base Stock Policy No Discount** (BSPnd): orders at every time step $t$ quantity for every product $\forall i$ based on the following formula:

$$x_t^i = \max\left[z^i - \sum_{l=0}^{\mathsf{LT}} O_t^{l,i} - \sum_{\tau=1}^{\mathsf{SL}-1} I_t^{\tau,i}, 0\right].$$

  $z^i$ represents the base stock level for product $i$. The policy does not apply discounts; it requires 2 parameters to be set.

- **Base Stock Policy Simple** (BSPs): follows the same ordering policy as BSPnd, while applying a discount $\delta^i$ to items older than a threshold $\tau_0$ $\forall i$; it requires 6 parameters to be set.

- **Constant Order Policy No Discount** (COPnd): orders at every time step $t$ a fixed constant quantity for every product $\forall i$ and does not apply discounts; it requires 2 parameters to be set.

- **Constant Order Policy Simple** (COPs): orders at every time step $t$ a fixed constant quantity $c_t^i$ for every product and applies a discount $\delta^i$ to items older than a threshold $\tau_0$ $\forall i$; it requires 6 parameters to be set.

In all cases the orders are placed at the end of the day after the store closes.

## 3.3 Problem Statement

Decision-making under uncertainty can be approached in various ways. In our case, we formulate a robust optimization problem and use a simulation-based approach to evaluate policy performance against worst-case parameter realizations. The objective is to maximize the expected daily profit under this uncertainty for each given policy. The decision variables $\mathbf{x}$ are the parameters defining the policy (e.g., base stock levels, discounts-related parameters), while the uncertain parameters $\mathbf{u}$ include the shape parameters $(\alpha, \beta)$ of the Beta distribution that models the utility function. In some experiments, we also include the coefficient of variation $cv$ of arrivals as an additional uncertain parameter.
In addition to the symbols introduced in the previous sections, we define the following variables:

- $S_t^{\tau,i}$: units of product $i$ with residual shelf life $\tau$ sold at day $t$.
- $M_t^i$: units of product $i$ scrapped at day $t$.
- $p^i_{\text{markdown}}$: markdown price of product $i$.

The daily profit obtained on day $t$ under decision $\mathbf{x}$ and uncertainty realization $\mathbf{u}$ is then defined as

$$\Pi_t(\mathbf{x}, \mathbf{u}) = \sum_{i \in \mathcal{I}} \left( \underbrace{\sum_{\tau=1}^{\mathsf{SL}^i} p_\tau^i \, S_t^{\tau,i}}_{\text{revenue from units sold}} + \underbrace{p^i_{\text{markdown}} \, M_t^i}_{\text{revenue from scrapped units}} - \underbrace{c^i \, O_t^{\mathsf{LT}^i,i}}_{\text{ordering cost}} \right).$$

In our experiments we set $p^i_{\text{markdown}} = 0$, so the term associated with scrapped units vanishes.
Using this definition, the worst-case robust optimization problem can be formulated as

$$\max_{\mathbf{x} \in \mathcal{X}} \min_{\mathbf{u} \in \mathcal{U}} \mathbb{E}\Big[\Pi_t(\mathbf{x}, \mathbf{u})\Big].$$

Here, the expectation is taken with respect to the stochastic elements of the system, which include the random arrival of customers and the sampling of utility parameters from a Beta distribution. The inner minimization captures the worst-case

performance over the set of uncertain parameters $\mathcal{U}$, while the outer maximization optimizes the policy parameters to be robust against such uncertainty. To tackle the computational complexity, we use a metamodel (surrogate model) that approximates the objective function, enabling efficient simulation-based optimization. The algorithm that we employed is called Efficient Global Robust Optimization (EGRO) with the Kriging metamodel, and is described in detail in the next sections.

# Chapter 4

# Kriging Surrogate Modeling

## 4.1 Modeling Assumptions

Kriging is a surrogate modeling technique widely used in simulation-based optimization to approximate computationally expensive objective functions. The core idea is to construct a predictive model $\hat{f}(\mathbf{x})$ that interpolates known data points and provides uncertainty estimates in unobserved regions of the domain.
We assume that the unknown expensive function $f(\mathbf{x})$ is a realization of a Gaussian process $Y(\mathbf{x})$, modeled as

$$Y(\mathbf{x}) \sim \mathcal{GP}(\mu, \sigma^2 R(\theta)),$$

where $\mu$ is the constant mean of the process, $\sigma^2$ is its variance, and $R(\theta)$ is a correlation matrix defined by a correlation function parameterized by $\theta$. The correlation between two inputs $\mathbf{x}_i$ and $\mathbf{x}_j$ is given by

$$\mathrm{Corr}(Y(\mathbf{x}_i), Y(\mathbf{x}_j)) = \exp\left(-\sum_{q=1}^{k} \theta_q |x_{iq} - x_{jq}|^p\right),$$

where $k$ is the dimensionality of the input space, $\theta_q$ controls the correlation decay rate along dimension $q$, and $p$ is typically set to 2. Larger values of $\theta_q$ imply that the function varies more rapidly along the corresponding dimension.
Given observed responses $\mathbf{y} = (y_1, \ldots, y_n)^\top$ at training points $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, the joint distribution of $\mathbf{y}$ is multivariate normal with mean $\mu\mathbf{1}$ and covariance matrix $\sigma^2\mathbf{R}$, where $\mathbf{R}$ is the correlation matrix with entries $R_{ij} = \mathrm{Corr}(Y(\mathbf{x}_i), Y(\mathbf{x}_j))$.
The log-likelihood function is

$$\log L(\mu, \sigma^2, \boldsymbol{\theta}) = -\frac{n}{2}\log(2\pi) - \frac{n}{2}\log\sigma^2 - \frac{1}{2}\log|\mathbf{R}| - \frac{1}{2\sigma^2}(\mathbf{y} - \mu\mathbf{1})^\top \mathbf{R}^{-1}(\mathbf{y} - \mu\mathbf{1}).$$

For fixed correlation parameters $\boldsymbol{\theta}$, the maximum likelihood estimators (MLEs) for $\mu$ and $\sigma^2$ are obtained by setting the derivatives of the log-likelihood to zero

$$\hat{\mu} = \frac{\mathbf{1}^\top \mathbf{R}^{-1} \mathbf{y}}{\mathbf{1}^\top \mathbf{R}^{-1} \mathbf{1}},$$

$$\hat{\sigma}^2 = \frac{(\mathbf{y} - \mathbf{1}\hat{\mu})^\top \mathbf{R}^{-1} (\mathbf{y} - \mathbf{1}\hat{\mu})}{n}.$$

Substituting these back, the profile log-likelihood to maximize over $\boldsymbol{\theta}$ becomes

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta} \in \Lambda} \left( -\frac{n}{2} \log \hat{\sigma}^2 - \frac{1}{2} \log |\mathbf{R}| \right),$$

where $\Lambda$ is the feasible parameter space for $\boldsymbol{\theta}$.
Once the parameters are estimated, the Kriging predictor for a new point $\mathbf{x}$ is given by

$$\hat{y}(\mathbf{x}) = \hat{\mu} + \mathbf{r}^\top \mathbf{R}^{-1} (\mathbf{y} - \mathbf{1}\hat{\mu}),$$

where $\mathbf{r}$ is the vector of correlations between $\mathbf{x}$ and the training points.
A key feature of Kriging is its ability to quantify prediction uncertainty. The mean squared error of the prediction at $\mathbf{x}$ is

$$s^2(\mathbf{x}) = \hat{\sigma}^2 \left[ 1 - \mathbf{r}^\top \mathbf{R}^{-1} \mathbf{r} + \frac{(1 - \mathbf{1}^\top \mathbf{R}^{-1} \mathbf{r})^2}{\mathbf{1}^\top \mathbf{R}^{-1} \mathbf{1}} \right].$$

This variance represents the Kriging model's uncertainty in predicting the response at $\mathbf{x}$.

### 4.1.1 Example: Kriging fit and prediction with confidence bounds

To illustrate how Kriging approximates a function and predicts new values, we consider a nonlinear function defined as

$$f(x) = \sin(3\pi x) e^{-5x} + 0.2x^2.$$

We generate 10 training points using Latin Hypercube Sampling over the domain $[0, 1]$, and fit a Kriging model using the Surrogate Modeling Toolbox (SMT; Bouhlel et al. 2019).
The resulting surrogate not only interpolates the training points but also provides predictions with associated uncertainty estimates. In Figure 4.1, we observe how the Kriging model closely approximates the true function and provides a 95%

confidence interval, which becomes narrower in regions with training data and wider in unexplored areas.

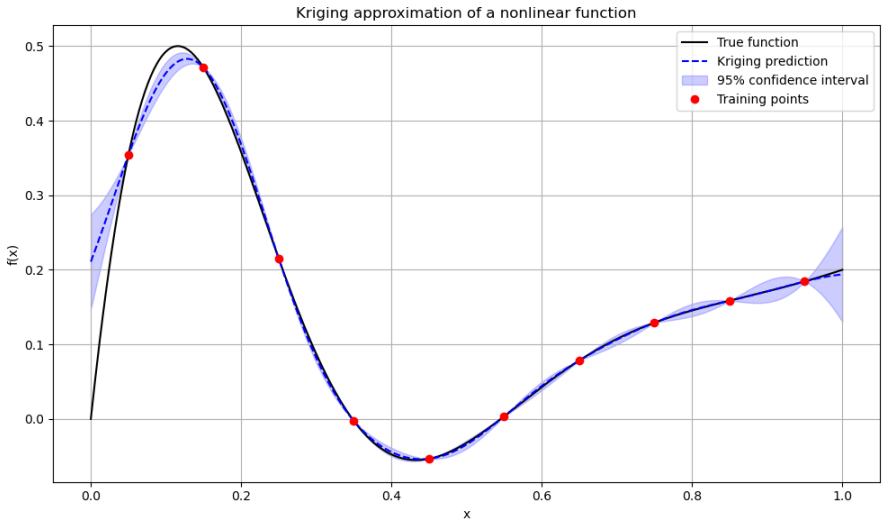



**Figure 4.1:** Non linear function and Kriging approximation.

In our work, Kriging is employed to build a surrogate model of the simulation-based objective function. We now introduce the EGO and EGRO algorithms, in which Kriging can be used to tackle optimization problems, with EGRO being selected for our case due to its suitability in a robust setting.

## 4.2 Efficient Global Optimization

Consider the global optimization problem

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

where $f : \mathcal{X} \subseteq \mathbb{R}^k \to \mathbb{R}$ is an expensive-to-evaluate black-box function defined over a bounded domain $\mathcal{D}$. The objective is to find the global maximizer $\mathbf{x}^*$ such that

$$f(\mathbf{x}^*) = \max_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}).$$

The Efficient Global Optimization (EGO) algorithm, introduced by Jones et al. (1998), addresses this problem by iteratively constructing a surrogate model, typically a Kriging model, to approximate $f(\mathbf{x})$ based on previously evaluated points. This surrogate provides both a prediction $\hat{f}(\mathbf{x})$ and an estimate of uncertainty, enabling efficient exploration of the design space.

To guide the search for the global maximum, EGO employs the Expected Improvement ($EI$) acquisition function defined as

$$\mathrm{EI}(\mathbf{x}) = \mathbb{E}\left[\max\left(0, \hat{f}(\mathbf{x}) - f_{\max}\right)\right],$$

where $f_{\max}$ is the best (maximum) observed objective value so far, and the expectation is taken over the predictive distribution of $\hat{f}(\mathbf{x})$.
The $EI$ function quantifies the expected increase in objective value achievable by sampling at $\mathbf{x}$, thus balancing exploitation of known promising regions and exploration of uncertain areas.
At each iteration, the next evaluation point is chosen by maximizing the $EI$ function

$$\mathbf{x}_{\text{next}} = \arg\max_{\mathbf{x}\in\mathcal{X}} \mathrm{EI}(\mathbf{x}).$$

The true function $f(\mathbf{x}_{\text{next}})$ is then evaluated, the surrogate is updated with the new data, and the process repeats until a stopping criterion is met. A pseudo-code is presented in Algorithm 1.

**Algorithm 1** Efficient Global Optimization (EGO) (Jones et al. 1998)

**Input:** domain $\mathcal{X}$, expensive black-box $f$, initial sample size $n_0$, max iterations $N_{\max}$, stopping tolerance $\varepsilon$
Generate initial design $\mathcal{D}_{n_0} = \{\mathbf{x}_1, \ldots, \mathbf{x}_{n_0}\}$ (e.g., LHS)
Evaluate $y_i \leftarrow f(\mathbf{x}_i)$ for all $\mathbf{x}_i \in \mathcal{D}_{n_0}$
Fit Kriging surrogate $\hat{f}(\mathbf{x})$ using $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n_0}$
**for** $n \leftarrow n_0 + 1$ **to** $N_{\max}$ **do**
    Solve surrogate max problem to find $f_{\max} = \max \hat{f}$
    Define acquisition: $\mathrm{EI}(\mathbf{x})$ using current $\hat{f}$ and observed $f_{\max}$
    $\mathbf{x}_{\text{next}} \leftarrow \arg\max_{\mathbf{x}\in\mathcal{X}} \mathrm{EI}(\mathbf{x})$
    Evaluate $y_{\text{next}} \leftarrow f(\mathbf{x}_{\text{next}})$
    $\mathcal{D}_n \leftarrow \mathcal{D}_{n-1} \cup \{(\mathbf{x}_{\text{next}}, y_{\text{next}})\}$
    **if** stopping criterion met (e.g., $\mathrm{EI}(\mathbf{x}_{\text{next}}) < \varepsilon$) **then**
        **break**
    **else**
        Re-fit Kriging surrogate $\hat{f}(\mathbf{x})$ with $\mathcal{D}_n$
    **end if**
**end for**
**return** $\mathbf{x}^* = \arg\max \hat{f}$

## 4.3 Efficient Global Robust Optimization

The idea of EGRO has been proposed in Rehman and Langelaar (2015) as an extension of the EGO algorithm to robust design problems involving parametric uncertainties. The method aims to efficiently identify the global robust optimum of an expensive-to-evaluate black-box function through a nested optimization procedure using surrogate modeling and adaptive sampling. While the original formulation in Rehman and Langelaar (2015) adopts a `min-max` structure, in this work we present the algorithm in its equivalent `max-min` form, which aligns with our problem setting.
Given an objective function $f(\mathbf{x}, \mathbf{u})$ where $\mathbf{x} \in \mathcal{X}$ are control variables and $\mathbf{u} \in \mathcal{U}$ are uncertain variables, the robust optimization problem is formulated as

$$\max_{\mathbf{x}\in\mathcal{X}} \min_{\mathbf{u}\in\mathcal{U}} f(\mathbf{x}, \mathbf{u}).$$

To reduce the number of expensive evaluations, a Kriging surrogate model $\hat{f} = K_f(\mathbf{x}, \mathbf{u})$ is built. The robust optimization is thus approximated as

$$\max_{\mathbf{x}\in\mathcal{X}} \min_{\mathbf{u}\in\mathcal{U}} K_f(\mathbf{x}, \mathbf{u}).$$

The following sections outline the algorithm in detail.

### 4.3.1 Initial Sampling

Before starting the iterative optimization process, an initial set of sample points is selected in the joint space of control and uncertain variables, $\mathcal{X} \times \mathcal{U}$. These initial samples are typically chosen using space-filling designs such as Latin Hypercube Sampling (LHS) or low-discrepancy sequences to ensure a good coverage of the input domain. The objective function $f(\mathbf{x}, \mathbf{u})$ is evaluated at these points to generate the initial dataset. This dataset is then used to build the initial Kriging surrogate model $K_f(\mathbf{x}, \mathbf{u})$, which serves as the foundation for subsequent adaptive sampling steps within the EGRO algorithm.

### 4.3.2 Selection in $\mathcal{X}$

The EGRO method modifies the classical $EI$ criterion to fit the `max-min` robust setting. For any $\mathbf{x}$, we define the worst-case Kriging prediction as

$$\hat{y}_{\min}(\mathbf{x}) = \min_{\mathbf{u}\in\mathcal{U}} K_f(\mathbf{x}, \mathbf{u}).$$

Then, the current robust optimum on the surrogate is

$$r_K = \max_{\mathbf{x}\in\mathcal{X}} \hat{y}_{\min}(\mathbf{x}).$$

which approximates the solution of the inner `max-min` problem over the surrogate. Following the principles of EGO, EGRO assumes that the uncertain value of $\hat{y}_{\min}(\mathbf{x})$ can be modeled as a normally distributed random variable

$$Y_{\min} \sim \mathcal{N}\left(\hat{y}_{\min}(\mathbf{x}), s^2(\mathbf{x}, \mathbf{u}_{\min})\right),$$

where $s^2(\mathbf{x}, \mathbf{u}_{\min})$ is the Kriging mean squared error at $(\mathbf{x}, \mathbf{u}_{\min})$.
An improvement occurs when the predicted worst-case outcome $Y_{\min}$ at a candidate decision $\mathbf{x}$ exceeds the current robust estimate $r_K$, i.e.,

$$Y_{\min} > r_K,$$

since the robust objective is defined as a maximization over the worst-case scenario. Thus we can compute an expected Improvement as

$$\mathbb{E}[I_c(\mathbf{x})] = \mathbb{E}[\max(Y_{\min} - r_K, 0)].$$

For simplicity, we refer to it as $EI_c$. The formula of the $EI_c$ simplifies to

$$EI_c(\mathbf{x}) = (\hat{y}_{\min} - r_K(\mathbf{x}))\,\Phi\left(\frac{\hat{y}_{\min} - r_K(\mathbf{x})}{s(\mathbf{x})}\right) + s(\mathbf{x})\,\phi\left(\frac{\hat{y}_{\min} - r_K(\mathbf{x})}{s(\mathbf{x})}\right),$$

where $\Phi(\cdot)$ and $\phi(z)$ are respectively the cumulative distribution function and probability density function of the standard normal distribution.
The point $\mathbf{x}^{\text{new}}$ that maximizes $EI_c(\mathbf{x})$ is selected as the next control candidate

$$\mathbf{x}^{\text{new}} = \arg\max_{\mathbf{x}\in\mathcal{X}} EI_c(\mathbf{x}).$$

### 4.3.3 Selection in $\mathcal{U}$

Once the new candidate control point $\mathbf{x}^{\text{new}}$ has been selected, the EGRO algorithm proceeds by identifying the most informative uncertain variable $\mathbf{u}^{\text{new}}$ at which to evaluate the expensive objective function. This step focuses on selecting $\mathbf{u} \in \mathcal{U}$ such that the evaluation at $(\mathbf{x}^{\text{new}}, \mathbf{u})$ is most likely to improve the current robust estimate.
At the fixed control location $\mathbf{x}^{\text{new}}$, we define the worst-case Kriging prediction as the minimum value of the surrogate function over the uncertainty domain

$$g_K = \min_{\mathbf{u}\in\mathcal{U}} K_f(\mathbf{x}^{\text{new}}, \mathbf{u}),$$

which serves as the reference worst-case cost for the current $\mathbf{x}^{\text{new}}$.
The goal is now to find a point $\mathbf{u}^{\text{new}}$ such that the Kriging prediction $K_f(\mathbf{x}^{\text{new}}, \mathbf{u})$ exceeds the current worst-case value $g_K$, indicating a potential for improvement in the inner minimum.

As before, we model the Kriging prediction at each $(\mathbf{x}^{\text{new}}, \mathbf{u})$ as a Gaussian random variable

$$Y \sim \mathcal{N}(\hat{y}, s^2),$$

where $\hat{y} = K_f(\mathbf{x}^{\text{new}}, \mathbf{u})$, and $s^2 = s^2(\mathbf{x}^{\text{new}}, \mathbf{u})$ is the Kriging mean squared error. Since we are searching for a global minimum in the uncertain variable space $\mathcal{U}$, an improvement occurs when $Y < g_K$. Here, the $EI$ is becomes

$$EI_u(\mathbf{x}^{\text{new}}, \mathbf{u}) = \mathbb{E}[\max(g_K - Y, 0)].$$

This expectation simplifies to

$$EI_u(\mathbf{x}^{\text{new}}, \mathbf{u}) = (g_K - \hat{y})\,\Phi\left(\frac{g_K - \hat{y}}{s}\right) + s\,\phi\left(\frac{g_K - \hat{y}}{s}\right).$$

The new uncertain variable sample is selected by maximizing this criterion

$$\mathbf{u}^{\text{new}} = \arg\max_{\mathbf{u}\in\mathcal{X}_e} EI_u(\mathbf{x}^{\text{new}}, \mathbf{u}).$$

Once the expensive function is evaluated at the new candidate point, the algorithm checks whether the sampling budget has been exhausted (or a maximum number of iterations is reached) or if the maximum $EI_c^{max}$ is below a predefined threshold. If not, the surrogate model is updated with the new sample, the new Kriging metamodel is constructed and the search continues. Otherwise, the algorithm terminates, and the current robust optimum $r_K$ is returned as the final solution $\mathbf{x}_{\text{best}}$. A pseudo-code of the EGRO algorithm is detailed in Algorithm 2.

**Algorithm 2** Efficient Global Robust Optimization (EGRO) (Rehman and Langelaar 2015)

**Input:** Decision domain $\mathcal{X}$, uncertainty domain $\mathcal{U}$, expensive black-box $f$, initial sample size $n_0$, max iterations $N_{\max}$, stopping tolerance $\varepsilon$
Generate initial design $\mathcal{D}_{n_0} = \{(\mathbf{x}_i, \mathbf{u}_i)\}_{i=1}^{n_0}$ using space-filling design (e.g., LHS)
Evaluate $y_i \leftarrow f(\mathbf{x}_i, \mathbf{u}_i)$ for all $(\mathbf{x}_i, \mathbf{u}_i) \in \mathcal{D}_{n_0}$
Fit Kriging model $K_f$ using $\mathcal{D}_{n_0}$
**for** $n \leftarrow n_0 + 1$ **to** $N_{\max}$ **do**
    Solve surrogate max-min problem on $\hat{f}$ to compute $r_K(\mathbf{x}) = \max_{\mathbf{x}\in\mathcal{X}} \min_{\mathbf{u}\in\mathcal{U}} \hat{f}(\mathbf{x}, \mathbf{u})$
    Compute expected improvement $EI_c(\mathbf{x})$ and select $\mathbf{x}^{\text{new}} = \arg\max_{\mathbf{x}\in\mathcal{X}} EI_c(\mathbf{x})$
    For fixed $\mathbf{x}^{\text{new}}$, compute $EI_u(\mathbf{u})$ and select $\mathbf{u}^{\text{new}} = \arg\max_{\mathbf{u}\in\mathcal{U}} EI_u(\mathbf{u})$
    Evaluate true function $y_{\text{new}} \leftarrow f(\mathbf{x}^{\text{new}}, \mathbf{u}^{\text{new}})$
    Augment dataset: $\mathcal{D}_n \leftarrow \mathcal{D}_{n-1} \cup \{(\mathbf{x}^{\text{new}}, \mathbf{u}^{\text{new}}, y_{\text{new}})\}$
    **if** stopping criterion met (e.g., $EI_c(\mathbf{x}^{\text{new}}) < \varepsilon$) **then**
        **break**
    **else**
        Re-fit Kriging surrogate $\hat{f}$ with $\mathcal{D}_n$
    **end if**
**end for**
**return** robust solution $\mathbf{x}^* = \arg r_K$

# Chapter 5

# Implementation details

This chapter provides an insight into the code structure. Before presenting the optimization algorithm, we first describe the simulation framework, outlining the main classes that underpin the simulation process:

- `Agents`: each agent represents a parametric policy governing inventory replenishment and pricing decisions. The agents implement the decision rules evaluated during optimization.

- `Environment`: it simulates daily system operations such as demand arrivals, inventory updates, order processing, and sales fulfillment. We use the `DailySimulation` environment in this work.

- `Managers`: these classes manage specific subsystems within the environment, including customer demand (`CustomerManager`), inventory tracking (`InventoryManager`), order queues and lead times (`SupplyManager`) and statistics collection (`StatManager`).

Together, these classes constitute the simulation algorithm used in this work and are described in detail in Appendix A.
We now explain how the expected value of the daily profit

$$\mathbb{E}\Big[\Pi_t(\mathbf{x}, \mathbf{u})\Big].$$

is estimated experimentally using an algorithm based on Welch's method. Lastly, this chapter concludes with a detailed explanation of the main EGRO algorithm implemented with Kriging.
The code is implemented in Python language.

## 5.1 Welch's method for steady-state mean estimation

In simulation-based optimization, particularly when dealing with stochastic systems that evolve over time, it is important to distinguish between transient and steady-state behavior. When estimating performance metrics such as the expected daily profit for a given policy and fixed uncertainty parameters, using data from the initial transient phase may introduce bias and lead to inaccurate conclusions. To address this, we adopt Welch's method (Law 2014), a standard procedure for detecting and eliminating the initial transient period in simulation outputs.
Welch's method relies on the ideas of replication averaging and moving average smoothing. First, $n$ independent replications of the simulation are performed, each generating a time series of length $m$. Let $Y_{ji}$ denote the $i$-th observation of the $j$-th replication. For each time step $i = 1, \ldots, m$, we compute the average across replications

$$\bar{Y}_i = \frac{1}{n} \sum_{j=1}^{n} Y_{ji},$$

resulting in an averaged time series $\bar{Y}_1, \bar{Y}_2, \ldots, \bar{Y}_m$. This step reduces the variance of the process, making trends and structural patterns easier to detect.
To further suppress high-frequency noise, a moving average filter is applied to the averaged series. For a window size $w$, the smoothed value at time $i$ is given by

$$\bar{Y}_i(w) = \begin{cases} \dfrac{1}{2w+1} \displaystyle\sum_{s=-w}^{w} \bar{Y}_{i+s} & \text{if } i = w+1, \ldots, m-w, \\ \dfrac{1}{2i-1} \displaystyle\sum_{s=-(i-1)}^{i-1} \bar{Y}_{i+s} & \text{if } i = 1, \ldots, w. \end{cases}$$

This smoothing step helps to reveal the long-term trend of the system and facilitates the identification of the steady-state region.
Once the smoothed series is obtained, a cutoff point $t_0$ is selected to mark the end of the transient period. The transient length $t_0$ is chosen as the smallest index beyond which the smoothed series $\bar{Y}_i(w)$ appears to have stabilized, indicating that the system has reached steady state. Several values of $w$ can be tested to find a window size that balances smoothness and detail in the curve. Typically, plots of $\bar{Y}_i(w)$ against $i$ are inspected visually to determine $t_0$. Increasing the number of replications $n$ also helps to reduce noise and improve the clarity of this identification.
In our case, we use Welch's method to estimate the *steady-state mean* $v = \mathbb{E}(Y)$ of the process $Y_1, Y_2$.... Assuming that the simulation replications are sufficiently long ( $m \gg t_0$) and the number of replications $n$ is adequate, the steady-state mean can

be estimated by considering only the observations beyond the warm-up period in each replication. Specifically, for each replication $j = 1, \ldots, n$, we compute

$$X_j = \frac{1}{m - t_0} \sum_{i=t_0+1}^{m} Y_{ji}.$$

The values $X_j$ are treated as independent and identically distributed (i.i.d.) random variables. An unbiased point estimator of $v$ is given by

$$\bar{X}_{(n)} = \frac{1}{n} \sum_{j=1}^{n} X_j,$$

which satisfies $\mathbb{E}[\bar{X}_{(n)}] \approx v$ . Similarly, an unbiased estimator of the variance $\sigma^2$ is given by the sample variance

$$S^2_{(n)} = \frac{1}{n-1} \sum_{j=1}^{n} \left( X_j - \bar{X}_{(n)} \right)^2.$$

An approximate $100(1 - \alpha)\%$ confidence interval for $v$ is then

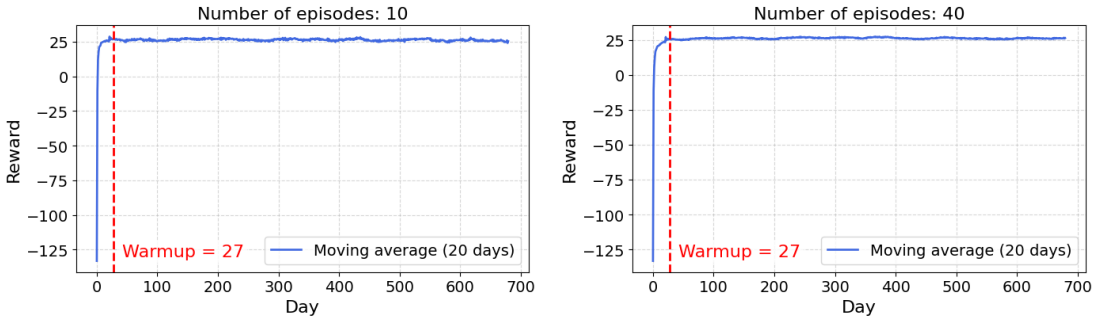
$$\bar{X}_{(n)} \pm t_{n-1,1-\alpha/2} \frac{S_{(n)}}{\sqrt{n}},$$

where $t_{n-1,1-\alpha/2}$ is the upper $(1 - \alpha/2)$ quantile of the Student is $t$-distribution with $n - 1$ degrees of freedom.
Welch's method provides a simple yet effective approach to steady-state estimation without requiring prior knowledge of the system dynamics.

### 5.1.1 Steady-state mean estimation

Following the proposed methodology based on Welch's method, we developed an automated algorithm to detect the warm-up period and estimate the steady-state mean of the expected daily profit (reward), $\mathbb{E}\Big[\Pi_t(\mathbf{x}, \mathbf{u})\Big]$. The overall procedure is summarized in Algorithm 3. We set a maximum number of 200 simulation episodes. The window size used to compute the moving average for transient detection was fixed at 20, which was empirically found to yield a sufficiently smooth curve during preliminary tests. Additionally, we required the relative width of the 95% confidence interval for the estimated mean to fall below a threshold of 0.02 before stopping the simulation. For example, consider the case with $\mathsf{SL} = 3$, $\theta_n \sim \text{Beta}(2, 3)$, $cv = 0.7$, and base stock levels $(10, 25)$, respectively for products A and B. Starting from 10 episodes and increasing by 10 at each step, the required confidence interval is achieved after 40 episodes, with a relative width of 1.61%. The plots in Figure 5.1

show the moving average curves for 10 and 40 episodes, while Figure 5.2 illustrates all the computed confidence intervals.




**(a)** 10 episodes **(b)** 40 episodes

**Figure 5.1:** Moving averages in Welch's method.

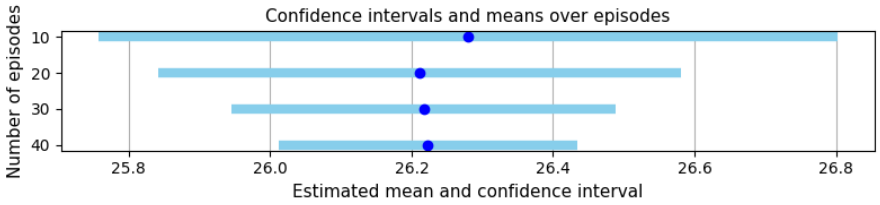



**Figure 5.2:** 95% confidence intervals and means computed at each iteration.

## 5.2 Main optimization algorithm

The core idea behind the optimization algorithm is the implementation of the EGRO algorithm suited for the problem at hand. The algorithm is contained in a class and is adapted to different settings depending on the domains of the uncertain variables.
The general idea is presented in the pseudo-code in Algorithm 4.

### Implementation notes

The algorithm is implemented in the `Kriging` class. In particular, the Kriging surrogate model is explicitly built using the `MixedIntegerKrigingModel` or `KRG` methods from the SMT library, which is important for handling mixed discrete-continuous variables in our implementation. In fact, experiments were developed for three settings:

**Algorithm 3** Automatic estimation of steady-state mean daily profit

**Input:** Environment $\mathcal{E}$, policy $\pi$, initial episodes $n_0$, step size $\Delta n$, max episodes $n_{\max}$, CI threshold $\epsilon$, warm-up window $w$, stability threshold $\tau$, required stable points $s$, minimum length of post-transient scenario $L_{\min}$
Initialize $n \leftarrow n_0$, $n_{\text{old}} \leftarrow 0$
Initialize rewards matrix $R \in \mathbb{R}^{n_{\max} \times T}$
**while** $n \leq n_{\max}$ **do**
    Simulate $\Delta n = n - n_{\text{old}}$ new episodes of length $T$
    Append results to $R$
    Compute average reward process over episodes: $\bar{Y}_t \leftarrow \frac{1}{n} \sum_{i=1}^{n} R_{i,t}$
    Compute moving average $M_t$ of $\bar{Y}_t$ using window $w$
    Compute differences $D_t = |M_{t+1} - M_t|$
    **if** there exists $i$ such that $D_i, D_{i+1}, \ldots, D_{i+s-1} < \tau$ **then**
        Set warm-up $L \leftarrow i + w$
    **else**
        Set fallback warm-up
    **end if**
    **if** $T - L < L_{\min}$ **then**
        **continue** ▷ not enough post-transient data
    **end if**
    Compute steady-state samples: $Z_i \leftarrow \frac{1}{T-L} \sum_{t=L+1}^{T} R_{i,t}$
    Compute sample mean $v_n$, sample variance $s^2$, and CI width $w_{\text{CI}}$
    Compute relative CI width: $r = \frac{w_{\text{CI}}}{|v_n|}$
    **if** $r \leq \epsilon$ **then**
        **return** $v_n$, CI, $r$, $L$, $n$
    **else**
        $n_{\text{old}} \leftarrow n$
        $n \leftarrow n + \Delta n$
    **end if**
**end while**
**return** last computed statistics

- `Discrete domain for` $(\alpha, \beta)$ `and discrete domain for` $cv$ `("dd");`
- `Discrete domain for` $(\alpha, \beta)$ `and continuous domain for` $cv$ `("dc");`
- `Continuous domain for` $(\alpha, \beta)$ `and fixed value for` $cv$ `("cf").`

Because of this, some methods in the `Kriging` class may differ across settings (e.g., treatment of uncertainty variables); the current code listing implements the `dd` case with $cv \in \{0.3, 0.7\}$ encoded categorically as `{'03','07'}` and is presented in

**Algorithm 4** Proposed EGRO Algorithm with Kriging surrogate

Initialize design space, data structures and parameters
Generate initial training samples
Evaluate the expensive function on initial samples
**for** iteration = 1 to $N_t$ **do**
    Encode samples and train the Kriging surrogate model
    Compute robust performance values and identify the current best solution
    Evaluate the control variable space $EI_c$ for all candidate controls
    Select a new candidate control point $\mathbf{x}^{\text{new}}$ from $EI_c$ maximization
    Evaluate the uncertain variable space $EI_u$ with $\mathbf{x}^{\text{new}}$ as control variable
    Select a new candidate uncertain point $\mathbf{u}^{\text{new}}$ from $EI_u$ maximization
    Define the new candidate point: $(\mathbf{x}^{\text{new}}, \mathbf{u}^{\text{new}})$
    Evaluate the expensive function at the new sample and update the dataset
    Check convergence criteria (stagnation / $EI$ threshold)
    **if** converged **then**
        **break**
    **end if**
**end for**
Return the robust solution and all evaluated samples

the following sections.

### Class overview

`SL`: products' shelf life (`SL3`, `SL5`, `SL7`).

`num_prods`: number of products (2 in the experiments).

`policy`: replenishment policy.

`X_beta`: set of discrete $(\alpha, \beta)$ pairs.

`X_cv`: possible coefficients of variation (`{'03','07'}` in `dd`).

`N_t` : maximum algorithm iterations; `n_tot_samples`: total initial samples to use.

`step_bs_levels`: step for base-stock integer grid.

`store_setting_03/07`: store infos depending on cv.

`prod_settinge`: products infos. `eps_ei` $EI_c$ tolerance for stopping.

### Setup and sampling

- `__init__`(...): initializes hyperparameters, experiment settings, and logger. Sets *cv* levels to `{'03','07'}` (categorical) and time horizon to 700 days.

- `build_design_space()`: constructs the mixed design space for SMT.
  - For `bsnds`: variables are $(bs_A, bs_B, \mu, \sigma, cv)$ with `IntegerVariable` for base stocks, `FloatVariable` for $(\mu, \sigma)$ computed from $(\alpha, \beta)$, and `CategoricalVariable` for $cv \in \{0{,}1\}$ (mapping $0 \leftrightarrow 0.3$, $1 \leftrightarrow 0.7$).
  - For `bssimple`: variables are $(bs_A, day_A, start_{day_A}, bs_B, start_{day_B}, disc_B, \mu, \sigma, cv)$, where the discounts and start days are encoded as `CategoricalVariables`, since they take values from a discrete, non-consecutive set (e.g., days 1, 3,...) rather than a continuous or consecutive range.A possible mapping could assign consecutive integers to these days if they were to be treated as integers.
  - For `conds`: variables are analogous to `bsnds`, except that the base stock variables $(bs_A, bs_B)$ are replaced by fixed order quantities $(co_A, co_B)$ for each product.
  - For `cosimple`: variables are analogous to `bssimple`, except that the base stock variables $(bs_A, bs_B)$ are replaced by fixed order quantities $(co_A, co_B)$ for each product. Tests for this policy are not reported in this thesis.

  Also precomputes a dictionary `dict_ab_enc` mapping $(\alpha, \beta) \mapsto (\mu, \sigma)$ and stores $\mu, \sigma$ bounds from all $X_\beta$.
- `discrete_lhs(domains, seed)`: samples over discrete domains (control $X_c$, $X_\beta$, $X_{cv}$). Draws `n_samples_to_add` indices per domain (with/without replacement as needed), shuffles, zips across dimensions and removes duplicates.
- `encode_samples_continuous(samples)`: encodes a list of triples $(x_c, (\alpha, \beta), cv)$ into SMT input rows. It replaces $(\alpha, \beta)$ with $(\mu, \sigma)$ from `dict_ab_enc` and maps $cv \in \{\text{'03'}, \text{'07'}\}$ to $\{0{,}1\}$. The control-part layout depends on `policy`.
- `_possible_orders(env)`: returns feasible order quantities $\{0{,}6{,}12, \dots\}$ per product up to a newsvendor bound.

### Evaluation

- `setup_environment(x)`, given $x = (x_c, (\alpha, \beta), cv)$:
  1. Selects store settings by $cv$ (`store_setting_03/07`), sets `DCM.alpha`, `DCM.beta`.
  2. Instantiates managers and the environment `DailySimulation` for a fixed time horizon.
- `setup_agent(x, env)`: builds an agent matching the chosen policy. The options are:

  - `bsnds`: `BaseStockNoDiscount` with `order_param`.
  - `bssimple`: `BaseStockSimple` with `order_param`, `discount_from`, `discount_amount`.
  - `conds`: `ConstantOrderNoDiscount` with `order_param`.
  - `cosimple`: `ConstantOrderSimple` with `order_param`, `discount_from`, `discount_amount`.

- `compute_real_y(x)`: evaluates the reward through simulation. It proceeds this way:
  1. Builds `env` and `agent`; calls `auto_evaluate_policy` (Welch's automated method).
  2. Returns $(\bar{r}, n_{\text{episodes}}), \widehat{\sigma}^2_{\text{noise}}, \{r_e\}_e, \text{CI}$, where $\bar{r}$ is the estimate of the mean steady-state daily reward , $\widehat{\sigma}^2_{\text{noise}}$ is the sample variance and CI is the confidence interval computed by the method for the mean reward.

**Surrogate predictions and robust objective**

- `comp_dict_preds_forxc(y_preds)`: reshapes batched predictions over $(x_c, (\alpha, \beta), cv)$ into a nested dictionary

$$\texttt{dict\_results\_xc}[x_c][(\alpha, \beta)] = [\widehat{y}(cv{=}0.3),\ \widehat{y}(cv{=}0.7)].$$

- `comp_rk_ymin_forxc(dict_results_xc)`: for each $x_c$, finds the minimum value across $(\alpha, \beta)$ and $cv$, returning
  - `dict_ymin[`$x_c$`]` $= \Big(\min_{(\alpha,\beta),cv} \widehat{y},\ (\alpha, \beta)^\star,\ cv^\star\Big)$,
  - the robust value $r_k = \max_{x_c} \min_{(\alpha,\beta),cv} \widehat{y}$,
  - the current robust point $x^\star = (x_c^\star, (\alpha, \beta)^\star, cv^\star)$.

**Expected Improvement**

- `expected_improvement_batch(y_min, rk, sigma)` computes the vectorized $EI$ formula. Numerically guards $\sigma \geq 10^{-8}$.
- `comp_all_EIs_xc(dict_ymin, rk)` computes $EI$ for each $x_c$ ($EI_c$) by:
  1. taking $y_{\min}(x_c)$ from `dict_ymin`,
  2. encoding $(x_c, (\mu, \sigma), cv)$,
  3. querying predictive variance to build $\sigma$,
  4. returning a map $x_c \mapsto EI_c$.

- `comp_gk_ymin_forxe(xc_new)`, with $x_c$ fixed, predicts $\widehat{y}(x_c^{new}, \cdot)$ for all $(\alpha, \beta), cv$ and returns
  - `dict_results_xcnew[`$(\alpha, \beta)$`]` $= [\widehat{y}(cv{=}0.3), \widehat{y}(cv{=}0.7)]$,
  - $g_k(x_c^{\text{new}}) = \min_{(\alpha,\beta),cv} \widehat{y}$.
- `comp_all_EIs_xe(xc_new, gk, dict_results_xcnew)`, for each $(\alpha, \beta)$ and $cv$ at fixed $x_c$ (the candidate $xc^{new}$), computes $EI$ using $y_{\min} = \widehat{y}(xc^{new}, (\alpha, \beta), cv)$ and target $g_k$, returning a dictionary $(\alpha, \beta) \mapsto [\text{EI}(cv{=}0.3), \text{EI}(cv{=}0.7)]$.

### Outer loop

`main_computation()`: contains the overall EGRO loop.

1. Builds the design space; creates the candidate grid $X_c$ and full prediction batch over $X_c \times X_\beta \times X_{cv}$.
2. Generates initial samples via `discrete_lhs`, evaluate `compute_real_y` for each and collects per–sample noise estimates.
3. Trains `MixedIntegerKrigingModel` from SMT (`KRG` surrogate) on the training encoded samples and averaged rewards and use it to predict mean rewards on all encoded samples.
4. Computes robust incumbent $(r_k, x^\star)$ on the surrogate via `comp_rk_ymin_forxc`.
5. Computes $EI_c$ for all $x_c$ via `comp_all_EIs_xc`; picks $x_c^{\text{new}}$ maximizing $EI_c$.
6. Fixs $x_c^{\text{new}}$, computes $g_k$ and $EI$ over $((\alpha, \beta), cv)$ via `comp_gk_ymin_forxe` and `comp_all_EIs_xe`; selects $(\alpha, \beta)^{\text{new}}, cv^{\text{new}}$ with maximum $EI_u$; evaluates the new triplet if unseen.
7. Checks stopping: either (i) $EI_c^{\max} < \varepsilon_{\text{EI}}$, or (ii) stagnation in $x^\star$ (i.e., the best solution has not improved for several iterations; here we chose an arbitrary value of 5 iterations) with stagnation_counter $\geq 5$, $EI_c^{\max} < 10\,\varepsilon_{\text{EI}}$, and $i > 50$.

Accumulates the sequence of robust solutions.

### Notes on the three uncertainty settings

- `dd` (current code): $(\alpha, \beta)$ and $cv$ are discrete. Encodes $(\alpha, \beta) \mapsto (\mu, \sigma)$ and $cv \in \{0.3, 0.7\}$ as a categorical variable $\{0,1\}$. The inner `min` is realized by scanning $(\alpha, \beta)$ and $cv$ on the surrogate.

- `dc`: $(\alpha, \beta)$ discrete but models $cv$ as a `FloatVariable` on [0.3,0.7]. Replace the categorical $cv$ in `build_design_space` and `encode_samples_continuous` with its numeric value; samples the points with a LHS for the $cv$ domain. The inner minimization over $cv$ is performed using the global optimizer `shgo` from SciPy (Virtanen et al. 2020). In general, that making $cv$ continuous introduces slight changes in the methods compared to the original discrete version.

- `cf`: $(\alpha, \beta)$ continuous with a fixed $cv$. Fixes $cv$ and replaces $X_\beta$ with a continuous parameterization (e.g., variables $(\mu, \sigma)$, respectively mean and std of the Beta). Chooses the domains to sample $\alpha$ and $\beta$ from (we sample $\mu$ and $t = \alpha + \beta$ from two arbitrary domains). Implements continuous solvers using `shgo`; an additional encoder/decoder for $(\alpha, \beta)$. Since the uncertain variables are continuous, there are some changes in the methods and structures compared to the original ones, especially when computing $EI_u$.

# Chapter 6

# Computational experiments and results

The experiments for the different policies are reported by setting and by shelf life. In general, three settings were considered, corresponding to different domains of the uncertain parameters:

- `Discrete domain for` $(\alpha, \beta)$ `and discrete domain for` $cv$:

  $$(\alpha, \beta) \in \{(0.5,0.5), (1,1), (1,2), (2,1), (2,2), (2,3), (3,2), (3,3), (3,4), (4,3), (4,4), (4,5), (5,4), (5,5), (2,4), (2,5)\},$$
  $$cv \in \{0.3, 0.7\}.$$

- `Discrete domain for` $(\alpha, \beta)$ `and continuous domain for` $cv$:

  $$(\alpha, \beta) \in \{(0.5,0.5), (1,1), (1,2), (2,1), (2,2), (2,3), (3,2), (3,3), (3,4), (4,3), (4,4), (4,5), (5,4), (5,5), (2,4), (2,5)\},$$
  $$cv \in [0.3,\ 0.7].$$

- `Continuous domain for` $(\alpha, \beta)$ `and fixed value for` $cv$:

  $$(\alpha, \beta) \in \{(\mu t, (1-\mu)t) \ \mid \ \mu \in [0.3,\ 0.7],\ t = \alpha + \beta \in [4,\ 10]\},$$
  $$\Rightarrow \alpha \in [1.2,\ 7.0],\ \beta \in [1.2,\ 7.0],$$
  $$cv = 0.3.$$

  In the continuous case, the domains for $\mu$ and $t$ were chosen to guarantee $\alpha, \beta \geq 1$, avoiding anomalous behavior at the distribution boundaries. The uncertainty set is the polytope, represented in Figure 6.1.

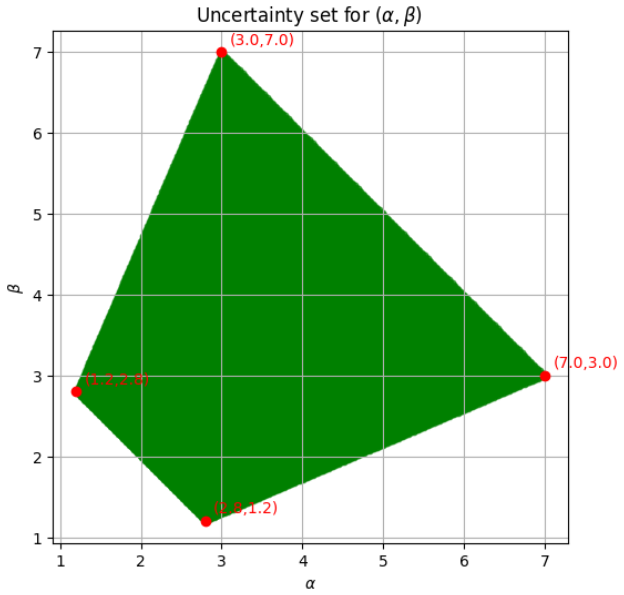



**Figure 6.1:** Uncertainty set for $(\alpha, \beta)$ in the continuous case.

The last domain was the one where every policy was tested. For each experiment, a specific set of decision variables was defined according to the chosen policy and uncertainty setting, reflecting the particular requirements of that setup. The initial number of sampled points is given following the rule in Rehman and Langelaar 2015, meaning $10d$, where $d$ represents the number of variables.

## 6.1 Base Stock Policy No Discount

Recall that we only have two decision variables in this policy: base stock level for product A and base stock level for product B.

### 6.1.1 Discrete domain for $(\alpha, \beta)$ and discrete domain for $cv$

For this first set of experiments, the number of initial sampled points is 50.

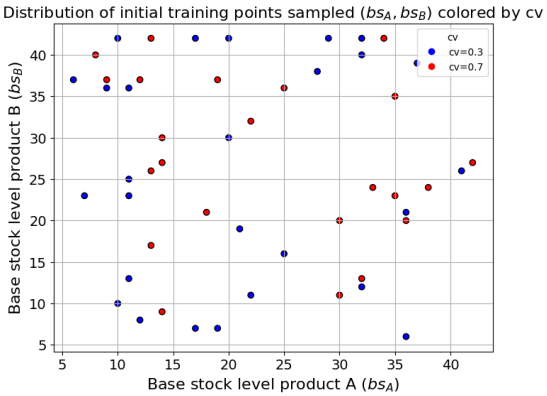



**Figure 6.2:** Initial sampled ($bs_A$, $bs_B$) colored by $cv$ (SL = 3, 5, 7).

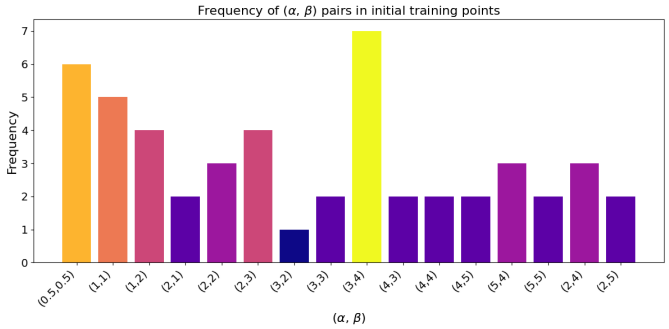


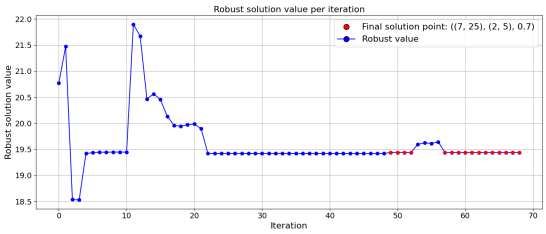

**Figure 6.3:** Frequency of $(\alpha, \beta)$ pairs in sampled points (SL = 3, 5, 7).

The decision variable domains for the base stock levels are defined for each product. Specifically, for products A and B, the base stock levels range from 6 to 42 units. Table 6.1 presents the results, including the number of iterations to reach the final solution, the solution decision variables and uncertain parameters and the true mean daily reward, i.e., the performance observed directly from the system rather than the EGRO robust estimate.

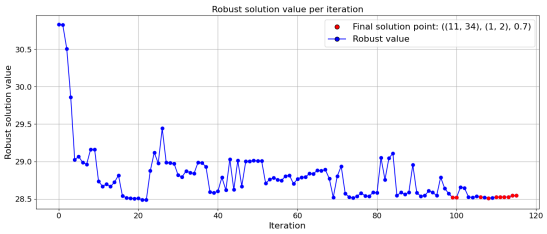
| SL | **#iters** | ($bs_A$, $bs_B$) | (($\alpha, \beta$), $cv$) | **value** |
|---|---|---|---|---|
| 3 | 69 | (7, 25) | ((2,5), 0.7) | 19.44 |
| 5 | 116 | (11, 34) | ((1,2), 0.7) | 28.55 |
| 7 | 34 | (9,42) | ((1,2), 0.7) | 31.12 |

**Table 6.1:** Final solutions summary for each SL.

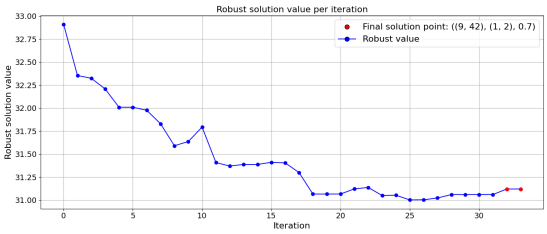
To illustrate the progression of the algorithm, we present the plots in Figures 6.4, 6.5 and 6.6.




**(a)** SL = 3




**(b)** SL = 5




**(c)** SL = 7

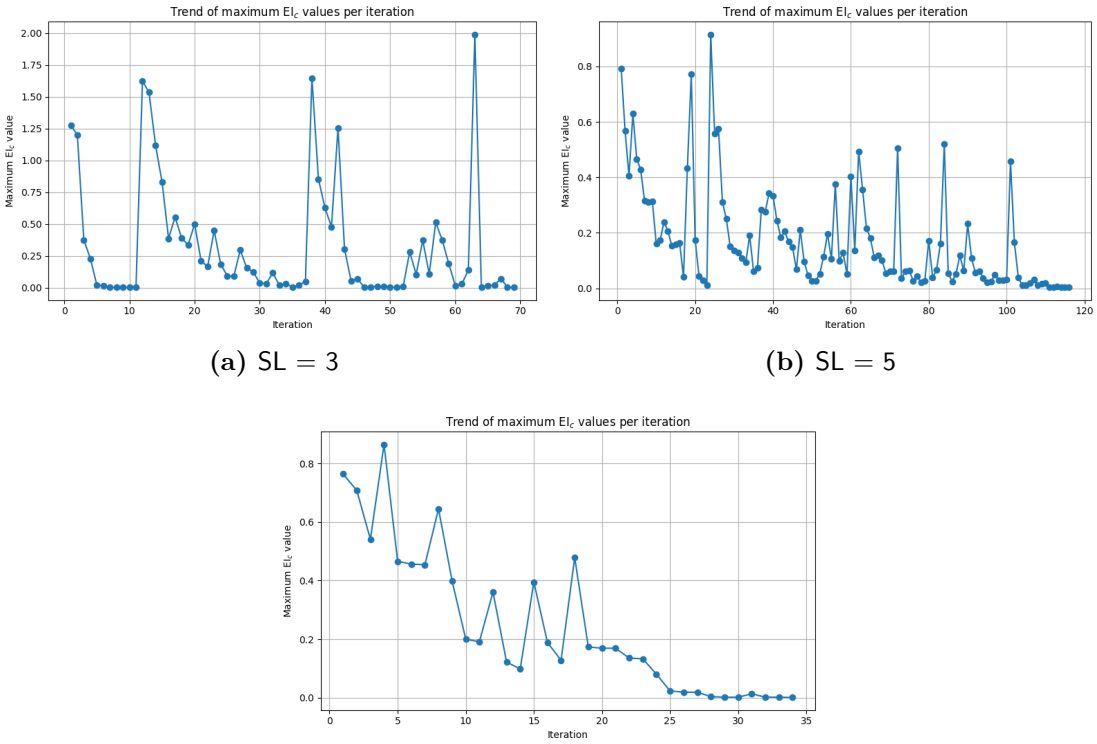
**Figure 6.4:** Robust solution value per iteration.

From Figure 6.4, it is evident that during the first iterations the algorithm makes optimistic assumptions, starting with large values for the initial robust solutions when the exploration phase has just begun. For SL = 3, the algorithm first identifies the final robust solution after 49 iterations, but requires an additional 20 iterations to confirm its value while considering alternative candidates as robust solutions. Overall, we observe a stable trend from early iterations, as the algorithm already evaluates a point very close to the final robust solution, namely (6,25).

In contrast, for SL = 5, the robust values fluctuate considerably, leading to a larger number of required iterations. Only in the very last iterations does the process stabilize, with the final solution value itself changing until the last two iterations, since the point had not been evaluated earlier by the algorithm.

Finally, for SL = 7, we observe a generally decreasing trend and a rather fast convergence to the solution.

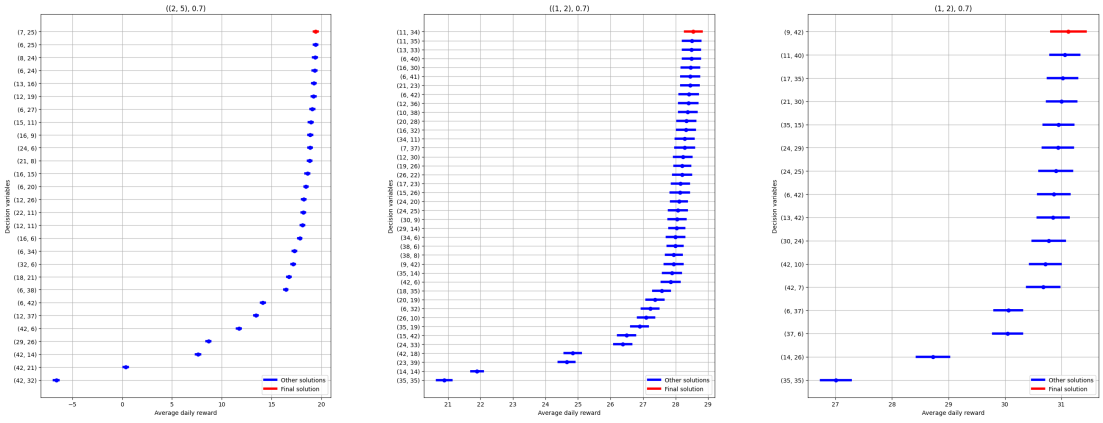
The observed fluctuations in Figure 6.5 in $EI_c$ for SL = 3 and SL = 5 across the control variables are entirely expected. This is because the Expected Improvement reflects not only the predicted mean from the Kriging model, but also its associated uncertainty. Points with higher uncertainty can contribute significantly to the Expected Improvement, even if their predicted mean is not optimal. Consequently, $EI_c$ exhibits an alternating pattern as the algorithm explores different regions of the control variable space. In the case of SL = 7, we get a more stable trend.




**(a)** SL = 3 **(b)** SL = 5

**(c)** SL = 7

**Figure 6.5:** Maximum $EI_c$ values per iteration.




**(a)** SL = 3 **(b)** SL = 5 **(c)** SL = 7

**Figure 6.6:** Confidence intervals and means for evaluated points with same uncertainty parameters as final solution.

In Figure 6.6, we show all the points evaluated throughout the algorithm using the same uncertain parameters as the final solution. The 95% confidence intervals and mean values, computed using the aforementioned Welch algorithm, are also displayed in a increasing reward order. It is quite expected that many points yield very similar results, and it is interesting to observe that the algorithm evaluates a substantial number of them before converging.

### 6.1.2 Discrete domain for $(\alpha, \beta)$ and continuous domain for $cv$

For this set of experiments, the number of initial sampled points is once again 50. The decision variables domains are defined for each shelf life in Table 6.2.

| **Product** | SL | **bs level domains** |
|---|---|---|
| A | 3 | [3,...,11] |
| B | 3 | [21,...,29] |
| A | 5 | [7,...,15] |
| B | 5 | [30,...,38] |
| A | 7 | [5,...,13] |
| B | 7 | [38,...,46] |

**Table 6.2:** Domains of the base stock level ($bs$) for each product.

Unlike the case with only discrete domains, here we chose to differentiate between the products and to narrow the sets based on the results of the prior analysis. We see the distribution of initial sampled point in the case of $\mathsf{SL} = 3$ in Figures 6.7 and 6.8.

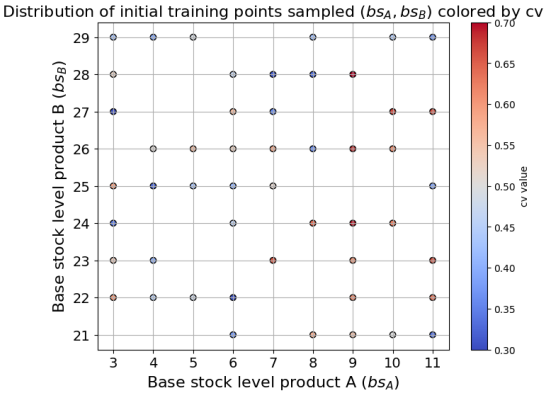



**Figure 6.7:** Initial sampled ($bs_A$, $bs_B$) colored by $cv$ for $\mathsf{SL} = 3$.

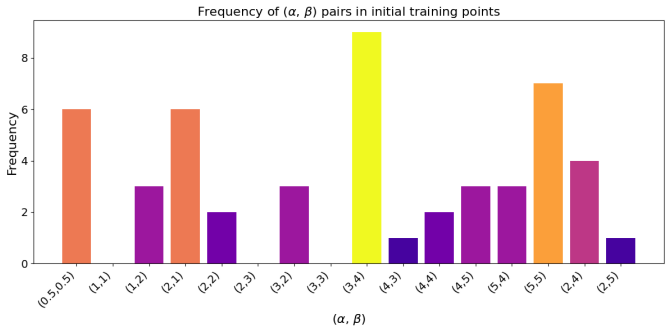



**Figure 6.8:** Frequency of $(\alpha, \beta)$ pairs in sampled points for $\mathsf{SL} = 3$.

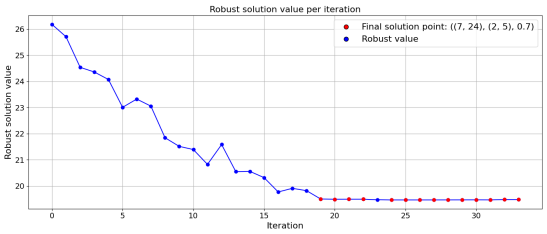
| SL | **#iters** | $(bs_A, bs_B)$ | $((\alpha, \beta), cv)$ | **value** |
|---|---|---|---|---|
| 3 | 34 | (7, 24) | ((2,5), 0.7) | 19.48 |
| 5 | 21 | (10, 35) | ((1,2), 0.7) | 28.54 |
| 7 | 25 | (9,41) | ((1,2), 0.7) | 31.15 |

**Table 6.3:** Final solutions summary for each SL.

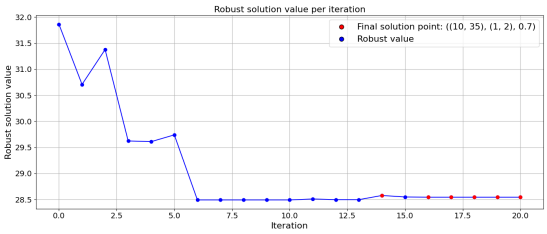
Looking at Table 6.3, we observe results that are consistent with the case where $cv$ is treated as a discrete variable. In all instances, the worst-case scenario corresponds to $cv = 0.7$, as expected, since a higher coefficient of variation introduces greater uncertainty in the arrivals. Overall, the algorithm converges in fewer iterations across all cases, which can be attributed to the narrower domains of the base stock levels. The solution points across every SL are very similar to those found in the previous case.

Examining the plots in Figure 6.9, we notice an initially optimistic behavior that rapidly decreases toward values close to the final solution. In the case of $\mathsf{SL} = 3$, the final solution point is identified as robust at an early stage and consistently confirmed as the robust solution well before the algorithm terminates, while for $\mathsf{SL} = 5$ and $\mathsf{SL} = 7$ the final solution is first evaluated in later stages.

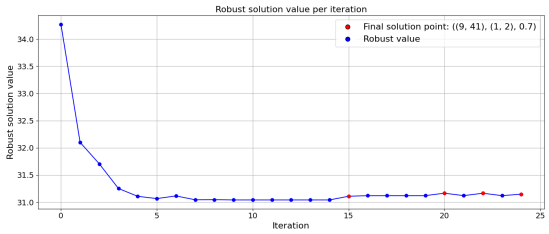
Regarding the trends of the maximum $EI_c$ in Figure 6.10, we observe a relatively smooth behavior across all cases, without pronounced oscillations. This contributes to a consistently fast convergence in every instance.




**(a)** SL = 3

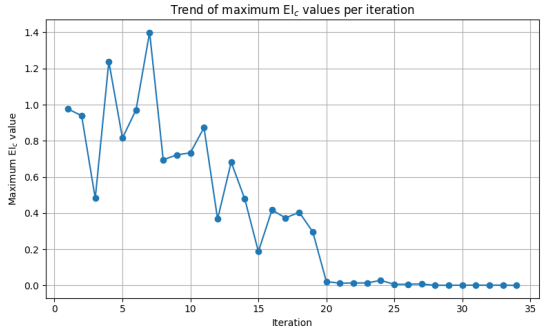



**(b)** SL = 5

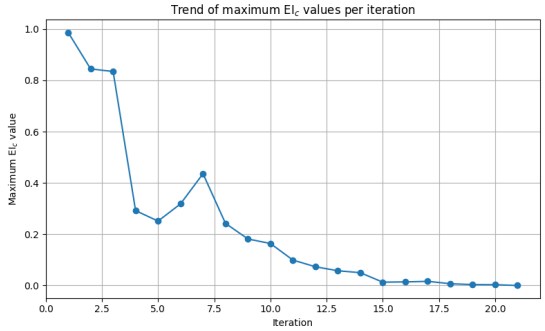



**(c)** SL = 7

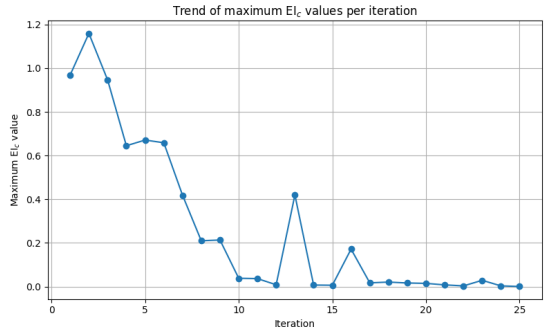
**Figure 6.9:** Robust solution value per iteration.




**(a)** SL = 3

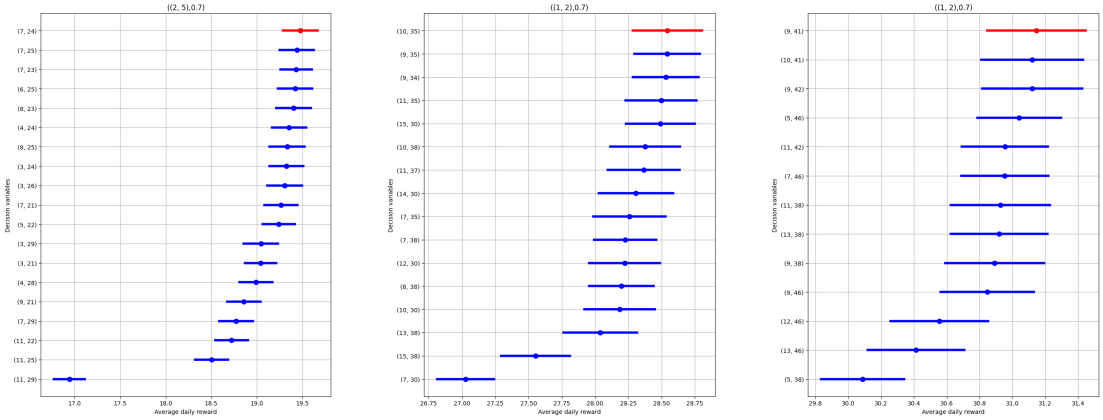



**(b)** SL = 5




**(c)** SL = 7

**Figure 6.10:** Maximum $EI_c$ values per iteration.



**(a)** SL = 3 **(b)** SL = 5 **(c)** SL = 7

**Figure 6.11:** Confidence intervals and means for evaluated points with same uncertainty parameters as final solution.

Based on the 95% confidence intervals of the points evaluated with the same uncertainty parameters as the final solution in Figure 6.11, we again observe very close values in terms of both means and confidence intervals. This is even clearer here, since the possible base stock levels are restricted to a small set.

### 6.1.3 Continuous domain for $(\alpha, \beta)$ and fixed value for $cv$

For this set of experiments, the number of initially sampled points is now set to 40, as $cv$ is fixed at 0.3. The domains of the decision variables have been redefined to better reflect the current setting, given that $cv$ is fixed. The updated domains are summarized in Table 6.4.

| Product | SL | bs level domains |
|---|---|---|
| A | 3 | [3,...,11] |
| B | 3 | [21,...,29] |
| A | 5 | [4,...,12] |
| B | 5 | [27,...,35] |
| A | 7 | [3,...,11] |
| B | 7 | [36,...,44] |

**Table 6.4:** Domains of the base stock level ($bs$) for each product.

We can examine the distribution of initial sampled point in the case of SL = 3 in Figure 6.12, where, given the continuous nature of $(\alpha, \beta)$, we color the points by $t = \alpha + \beta$.

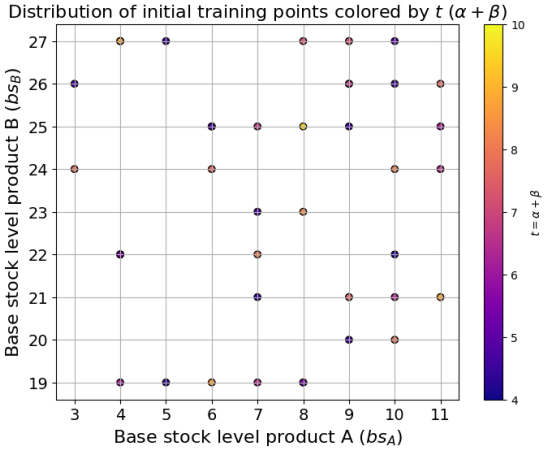
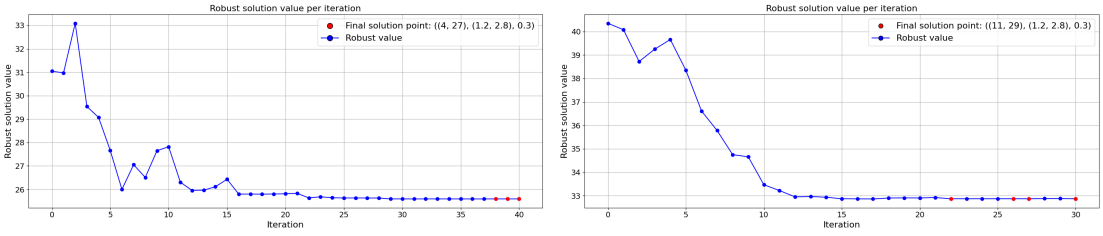



**Figure 6.12:** Initial sampled ($bs_A$, $bs_B$) colored by $cv$ for SL = 3.

| SL | #iters | ($bs_A$, $bs_B$) | ($\alpha, \beta$) | value |
|---|---|---|---|---|
| 3 | 41 | (4, 27) | (1.2, 2.8) | 25.59 |
| 5 | 31 | (11, 29) | (1.2, 2.8) | 32.87 |
| 7 | 13 | (3,40) | (1.2, 2.8) | 34.09 |

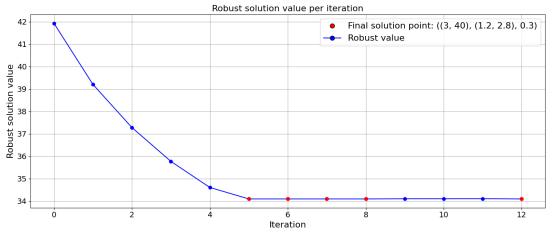
**Table 6.5:** Final solutions summary for each SL.




**(a)** SL = 3

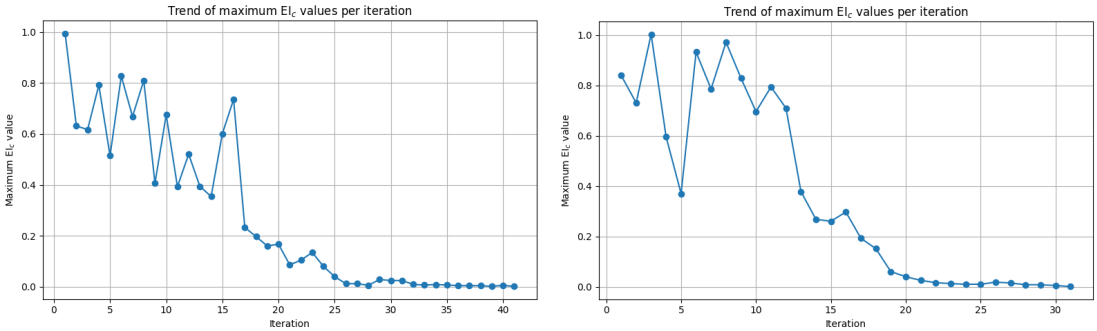



**(b)** SL = 5

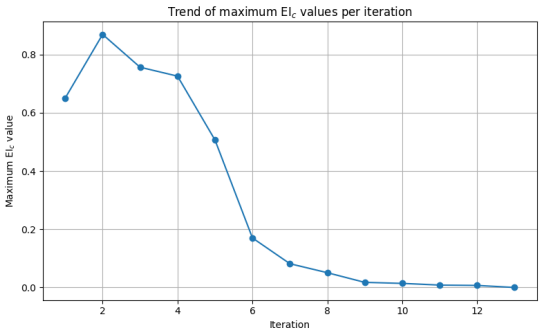



**(c)** SL = 7

**Figure 6.13:** Robust solution value per iteration.

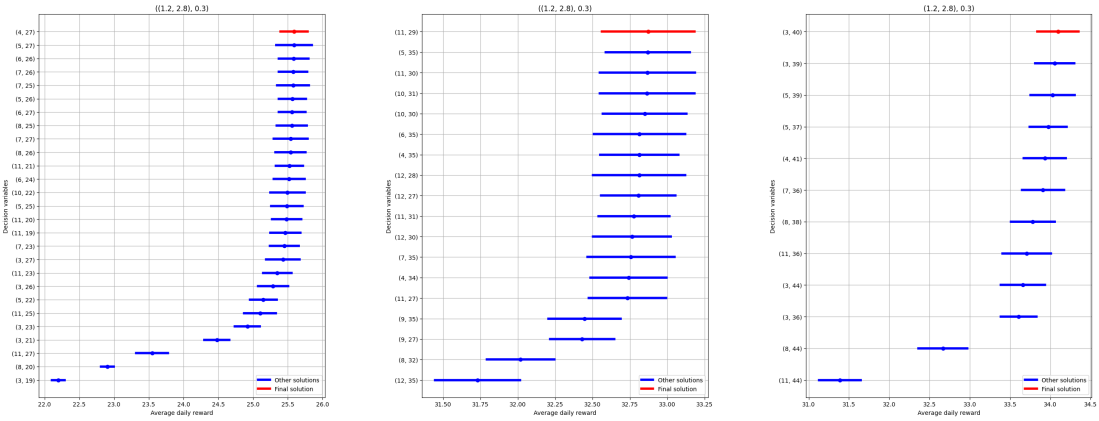



**(a)** SL = 3 **(b)** SL = 5




**(c)** SL = 7

**Figure 6.14:** Maximum $EI_c$ values per iteration.




**(a)** SL = 3 **(b)** SL = 5 **(c)** SL = 7

**Figure 6.15:** Confidence intervals and means for evaluated points with same uncertainty parameters as final solution.

Table 6.5 reports the results. In all cases, the worst-case $(\alpha, \beta)$ corresponds to $(1.2, 2.8)$, which is the vertex of the polytope with the lowest values of $\mu$ (with the higher variance) and $t$. Interestingly, for $\mathsf{SL} = 3$ and $\mathsf{SL} = 7$, the base stock levels in the optimal solutions are close to the minimum values in their respective domains, whereas for $\mathsf{SL} = 5$ this is not observed.
The plots in Figures 6.13, 6.14, 6.15 are consistent with the results and align well with our expectations.

## 6.2 Base Stock Simple

For the Base Stock Simple policy, recall the four additional decision variables: the set of possible discounts and the days on which the discounts start. For this policy, as well as for the following ones, we only performed three experiments in the last case, i.e., with continuous $(\alpha, \beta)$ domains and a fixed $cv$, since we focused only on the continuous case.

### 6.2.1 Continuous domain for $(\alpha, \beta)$ and fixed $cv$

The number of initial sampled points is 80. The domains for the base stock levels are kept unchanged from the case of the Base Stock No Discount Policy, while the discount-related variables are defined in Table 6.6. The set of possible discounts follows the choice in Fadda et al. 2024. Note that the first day of sale is indexed as 0.

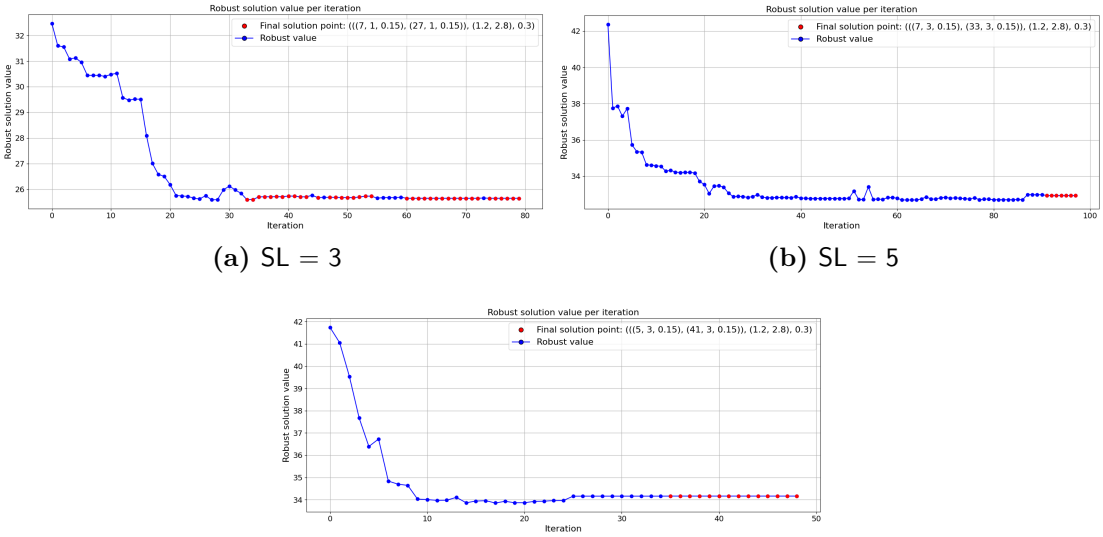
| **Product** | SL | **discounts** | **discount start days** |
|---|---|---|---|
| A; B | 3 | [0.15, 0.25, 0.50] | [1, 2] |
| A; B | 5 | [0.15, 0.25, 0.50] | [1, 3, 4] |
| A; B | 7 | [0.15, 0.25, 0.50] | [1, 3, 5, 6] |

**Table 6.6:** Domains of the discount variables for each product.

The results are summarized in Table 6.7.

| **SL** | **#iters** | **($bs_A$, $bs_B$)** | ($disc_A$, $disc_B$) | ($start_{day_A}$, $start_{day_B}$) | $(\alpha, \beta)$ | **value** |
|---|---|---|---|---|---|---|
| 3 | 80 | (7, 27) | (0.15, 0.15) | (1, 1) | (1.2, 2.8) | 25.65 |
| 5 | 98 | (7, 33) | (0.15, 0.15) | (3, 3) | (1.2, 2.8) | 32.62 |
| 7 | 49 | (5, 41) | (0.15, 0.15) | (3, 3) | (1.2, 2.8) | 34.16 |

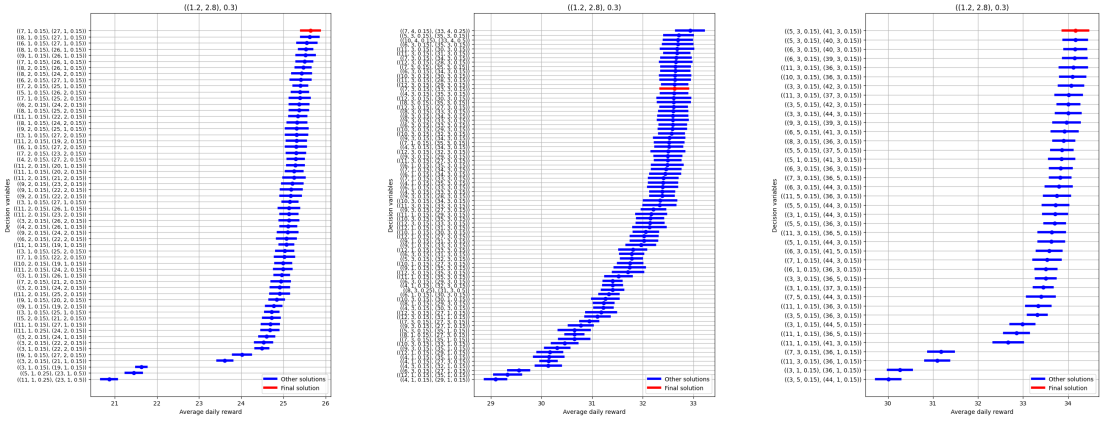
**Table 6.7:** Final solutions obtained for each value of SL.




**(a)** SL = 3 **(b)** SL = 5

**(c)** SL = 7

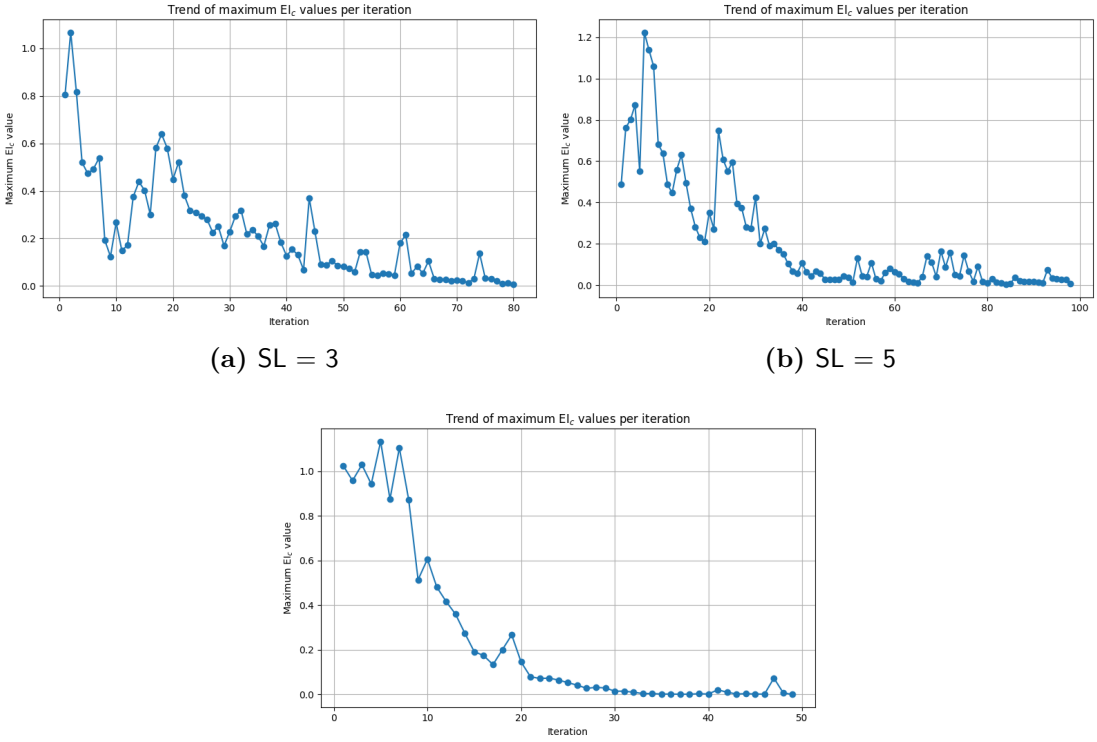
**Figure 6.16:** Robust solution value per iteration.




**(a)** SL = 3 **(b)** SL = 5 **(c)** SL = 7

**Figure 6.18:** Confidence intervals and means for evaluated points with same uncertainty parameters as final solution.

What can be observed is that the number of iterations required increases before the algorithm reaches the final solutions. In the plots of Figure 6.16, the case with SL = 3 illustrates that the algorithm identified the final solution relatively early, yet a substantial number of additional iterations were necessary to confirm it as optimal. In contrast, for SL = 7, the maximum $EI_c$ decreases rapidly with




**(a)** SL = 3

**(b)** SL = 5

**(c)** SL = 7

**Figure 6.17:** Maximum $EI_c$ values per iteration.

limited oscillations, resulting in smoother convergence and a considerably smaller number of iterations. The case with SL = 5, however, exhibits a more challenging behavior: the algorithm required a significantly longer run and ultimately provided a less satisfactory solution, as evidenced by the confidence intervals reported in Figure 6.18. Under the same uncertainty conditions, alternative solutions led to higher mean profits. This phenomenon can be attributed to the fact that the actual final solution was never directly evaluated by the system. Consequently, the robust estimate of the average daily profit does not accurately reflect the simulated value, which leads to the observed performance degradation. In general, the Base Stock Simple setting introduces a larger number of decision variables, which increases the complexity of the optimization problem and, in turn, results in a higher number of iterations being required by the algorithm.

## 6.3 Constant Order No Discount

Similar to the Base Stock No Discount Policy, this policy has two decision variables, which in this case correspond to the fixed order quantities for both products.

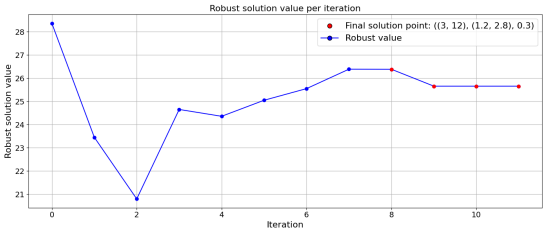
### 6.3.1 Continuous domain for $(\alpha, \beta)$ and fixed $cv$

The number of initial samples is 40. The set of possible order quantities ($co$) is the same across all cases. Specifically, for products A and B, the possible order quantities range from 3 to 27 units in steps of 3 (i.e., 3, 6, 9, ..., 27).

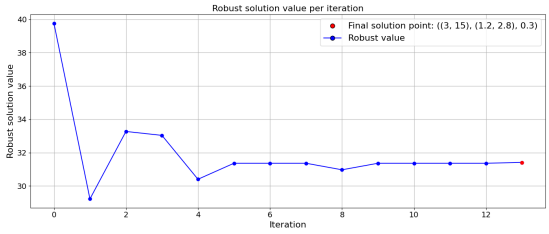
| SL | #iters | $(co_A, co_B)$ | $(\alpha, \beta)$ | value |
|---|---|---|---|---|
| 3 | 12 | (3, 12) | (1.2, 2.8) | 25.66 |
| 5 | 14 | (3, 15) | (1.2, 2.8) | 31.41 |
| 7 | 15 | (3, 15) | (1.2, 2.8) | 32.31 |

**Table 6.8:** Final solutions summary for each SL.

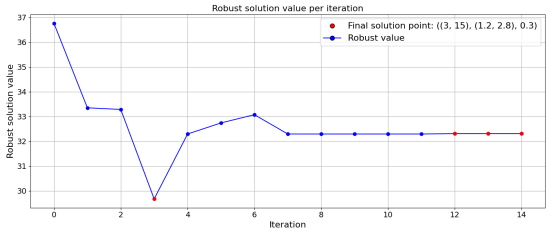
The results are shown in Table 6.8. As observed previously in the Base Stock cases, the worst-case uncertain variable remains consistently $(1.2, 2.8)$. Interestingly, the solutions for both $\mathsf{SL} = 5$ and $\mathsf{SL} = 7$ exhibit the same pattern. As in the Base Stock Policies, there is still a strong preference for product B, which receives substantially higher order quantities in all solutions.




**(a)** SL = 3

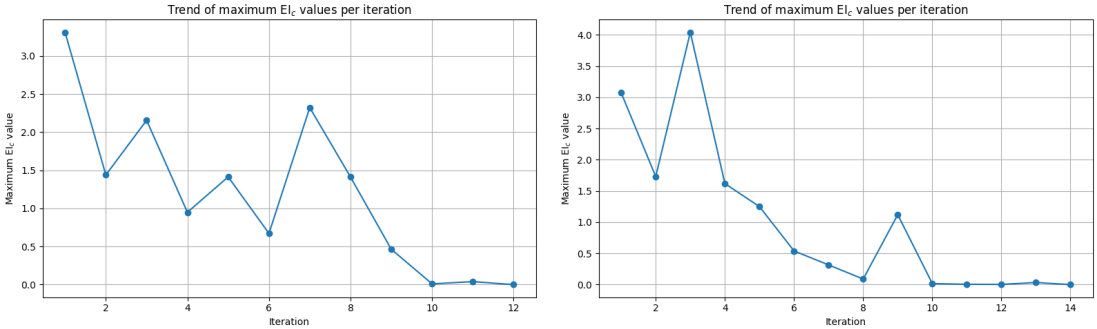



**(b)** SL = 5




**(c)** SL = 7

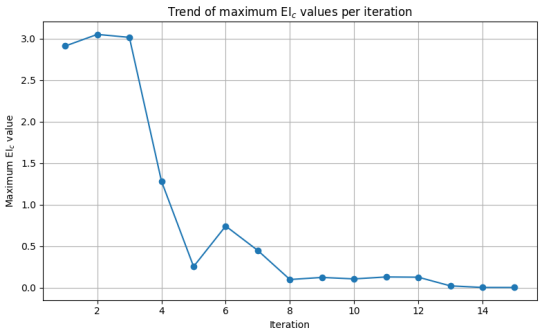
**Figure 6.19:** Robust solution value per iteration.




**(a)** SL = 3

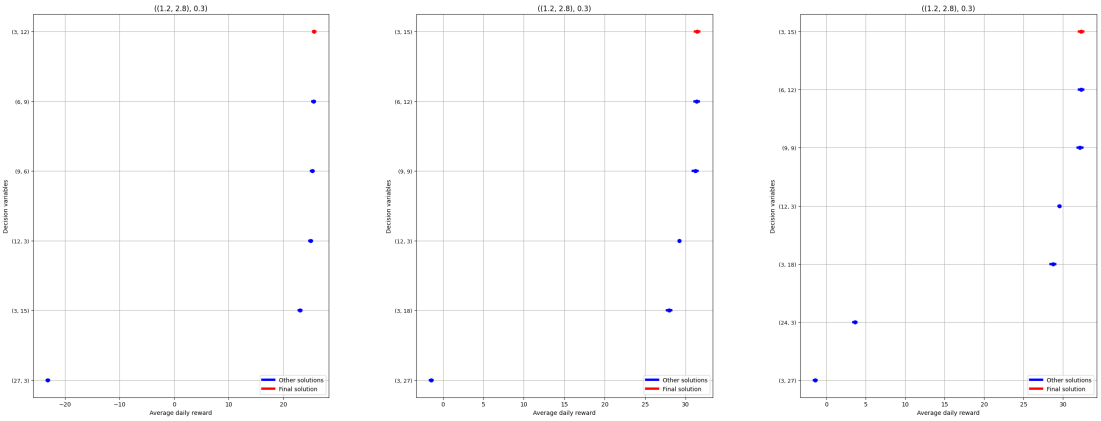



**(b)** SL = 5




**(c)** SL = 7

**Figure 6.20:** Maximum $EI_c$ values per iteration.




**(a)** SL = 3




**(b)** SL = 5



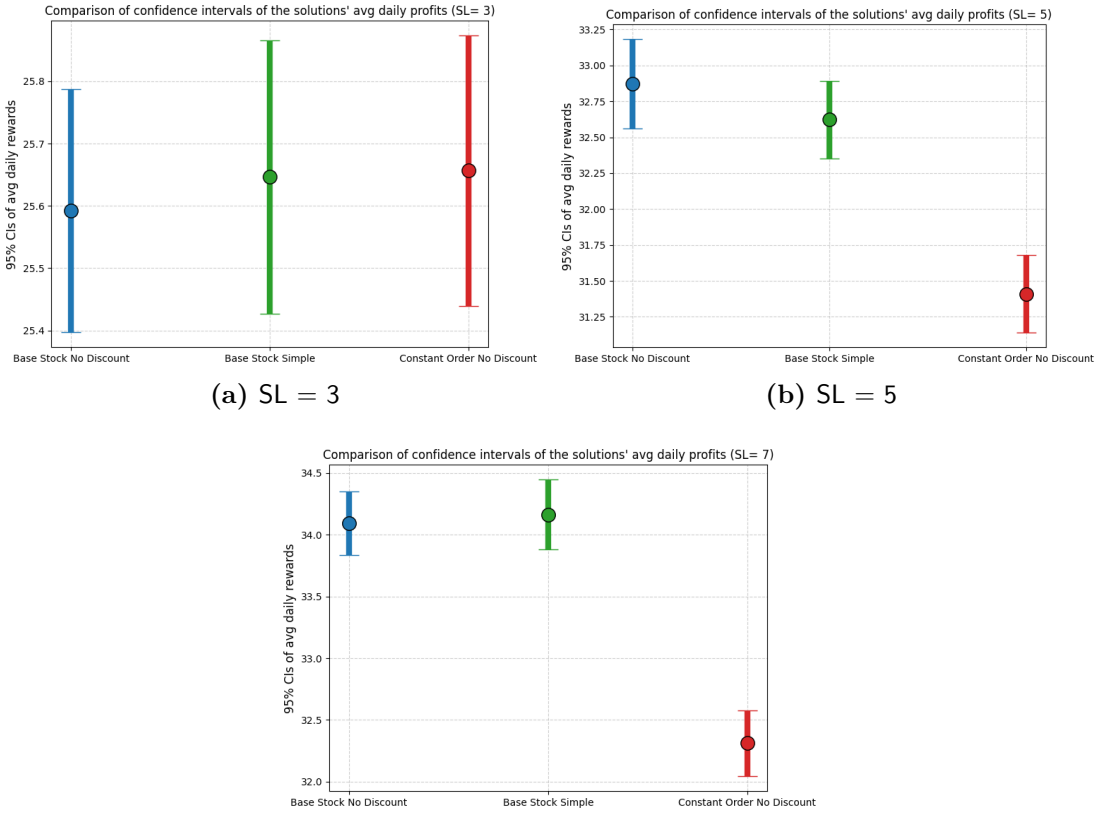

**(c)** SL = 7

**Figure 6.21:** Confidence intervals and means for evaluated points with same uncertainty parameters as final solution.

From the results and Figures 6.19, 6.20, we see that the algorithm reaches a solution quickly in all cases, with the $EI_c$ decreasing rapidly. The final solutions are therefore evaluated over only a few iterations before being confirmed as actual solutions. In plots in 6.21, another observed phenomenon is that some evaluated points yield negative daily mean rewards, resulting in large differences compared to other solutions.

## 6.4 Comparison between policies

In this section, we compare the results obtained from all policies in the setting of continuous $(\alpha, \beta)$ and fixed $cv = 0.3$. First, for each SL, we illustrate the differences between the confidence intervals of the solutions in terms of daily profit in Figure 6.22.




**(a)** SL = 3

**(b)** SL = 5

**(c)** SL = 7

**Figure 6.22:** Comparison of confidence intervals of the solutions' average daily profits.

In all cases, the Base Stock No Discount and Base Stock Simple policies exhibit overlapping confidence intervals, suggesting that, despite differences in their mean values, the observed differences may not be statistically significant.
For $\mathsf{SL} = 3$, as expected, the Base Stock Simple policy has a higher mean profit value than the corresponding policy without discounts, with its confidence interval positioned slightly above. Interestingly, the Constant Order No Discount Policy also appears marginally higher; however, once again, due to the overlap of the confidence intervals, we cannot conclude that these differences are statistically significant.
For $\mathsf{SL} = 5$, the mean value achieved by the solution of the Base Stock Simple policy is lower, which, as discussed in previous sections, likely reflects a suboptimal solution identified by the algorithm. In contrast, the confidence interval for the Constant Order No Discount Policy is lower than those of the other policies and does not overlap with them, indicating that its performance is statistically inferior. Similar patterns are observed for $\mathsf{SL} = 7$, with the exception that the policy incorporating discounts achieves a higher average profit.

In Tables 6.9, 6.10, 6.11. we report results in terms of the following metrics:

- Avg. Sales A: average daily number of units sold for product A.
- Avg. Sales B: average daily number of units sold for product B.
- Avg. Total Sales: total average daily sales across all products.
- Avg. Scrapped A: average daily number of scrapped units of product A.
- Avg. Scrapped B: average daily number of scrapped units of product B.
- Avg. Total Scrapped: total average daily scrapped units across all products.
- Avg. Lost Clients: average daily number of clients who are offered products but do not purchase any, as all utilities are $\leq 0$.
- Avg. Unmet Clients: average daily number of clients who could not be offered any product (e.g., no availability at all).

In a manner similar to the estimation of the average daily profit, we compute, for each case, a quantity analogous to the steady-state mean estimation, using the same warm-up period and scenarios as those employed for the profit calculation. The results are reported separately for $\mathsf{SL}$.
What becomes immediately apparent is the impact of discounts on the scrapped items, with a notable difference observed across all shelf lives, particularly when compared to the Constant Order No Discount Policy. This is partly explained by

| **Metric** | BSPnd | BSPs | COPnd |
|---|---|---|---|
| Avg. Sales A | 2.36 | 2.92 | 2.94 |
| Avg. Sales B | 11.97 | 12.16 | 11.39 |
| Avg. Total Sales | 14.33 | 15.07 | 14.33 |
| Avg. Scrapped A | 0.09 | 0.01 | 0.06 |
| Avg. Scrapped B | 0.60 | 0.15 | 0.61 |
| Avg. Total Scrapped | 0.69 | 0.16 | 0.67 |
| Avg. Lost Clients | 12.24 | 11.67 | 11.81 |
| Avg. Unmet Clients | 3.46 | 3.24 | 3.86 |

**Table 6.9:** Comparison of key daily metrics across the four ordering policies for SL = 3.

| **Metric** | BSPnd | BSPs | COPnd |
|---|---|---|---|
| Avg. Sales A | 4.63 | 2.83 | 2.95 |
| Avg. Sales B | 12.81 | 14.59 | 14.36 |
| Avg. Total Sales | 17.45 | 17.42 | 17.31 |
| Avg. Scrapped A | 0.15 | 0.06 | 0.05 |
| Avg. Scrapped B | 0.22 | 0.21 | 0.64 |
| Avg. Total Scrapped | 0.37 | 0.27 | 0.69 |
| Avg. Lost Clients | 10.25 | 10.05 | 10.05 |
| Avg. Unmet Clients | 2.30 | 2.52 | 2.68 |

**Table 6.10:** Comparison of key daily metrics across the four ordering policies for SL = 5.

| **Metric** | BSPnd | BSPs | COPnd |
|---|---|---|---|
| Avg. Sales A | 1.81 | 2.33 | 2.96 |
| Avg. Sales B | 16.22 | 16.24 | 14.51 |
| Avg. Total Sales | 18.04 | 18.58 | 17.47 |
| Avg. Scrapped A | 0.03 | 0.02 | 0.04 |
| Avg. Scrapped B | 0.30 | 0.06 | 0.49 |
| Avg. Total Scrapped | 0.33 | 0.08 | 0.53 |
| Avg. Lost Clients | 10.54 | 10.42 | 10.14 |
| Avg. Unmet Clients | 1.40 | 0.99 | 2.43 |

**Table 6.11:** Comparison of key daily metrics across the four ordering policies for SL = 7.

the fact that the Constant Order Policy orders the same fixed quantities regardless of the current on-order stock and inventory levels. A clear pattern emerges: total sales increase with longer shelf lives, while the number of unmet customers decreases correspondingly. This is due to the longer shelf life allowing products to remain

available for sale over an extended period, increasing the chances of meeting customer demand.
In general, product B exhibits higher sales and scrapped items, owing to its consistently larger order sizes relative to product A.
Finally, regarding lost clients, the Base Stock Simple Policy consistently outperforms the Base Stock No Discount Policy, demonstrating the role of discounts in retaining a larger share of customers.

# Chapter 7

# Conclusions

This thesis addresses the optimization of replenishment policies within a worst-case robust framework. The problem is posed as a max-min optimization, aiming to maximize the expected daily profit under the worst-case realizations of uncertain parameters, in a setting with two perishable and substitutable products: the objective is to maximize the expected profit with respect to the control variables defined by the policy (e.g., base stock levels, discount days, discount percentages), while simultaneously minimizing over the uncertain parameters, including the $(\alpha, \beta)$ parameters of the Beta distribution governing customer utility, and, in some experiments, the coefficient of variation ($cv$) of customer arrivals. Customers arrive daily following a Negative Binomial distribution, and their purchasing decisions depend on the available inventory, the applied discounts, and the utility parameters. The products are analyzed separately for three cases of shelf life: 3,5,7.
The main contribution of this work is the application of the Efficient Global Robust Optimization algorithm, combined with a Kriging metamodel, which enables an efficient surrogate optimization procedure by approximating the solution space while maintaining accuracy. Another important aspect is the estimation of daily average profits, performed using Welch's method to account for stochastic variability. The study focuses on three policies: Base Stock No Discount, Constant Order No Discount, and Base Stock Simple, where discounts are applied. These policies were optimized across a range of scenarios by varying the types of uncertain parameters. The environment is inherently stochastic, reflecting both the random arrival of customers and the uncertainty in the utility parameters associated with each arriving customer.
Particular attention was given to the Base Stock No Discount Policy in the discrete setting, where the parameters $(\alpha, \beta)$ were chosen arbitrarily and the coefficient of variation of arrivals was limited to two possible values or to a continuous range. However, the scenario considered most relevant is the continuous setting in which the pair $(\alpha, \beta)$ of the Beta distribution is constrained within a polytope, while the

*cv* remains fixed. This scenario was the one tested for all the policies.
The results highlight a clear pattern in the uncertain variables: in the continuous setting, the solutions consistently converge to the same Beta distribution parameters, corresponding to one of the vertices with the lowest mean and highest standard deviation. Moreover, the findings illustrate the benefit of incorporating discounts. When comparing the policies, we observe that the simple Base Stock Simple Policy consistently outperforms the others in terms of scrapped items and in most cases in terms of unmet clients, highlighting the role of discounts in mitigating waste. Regarding profit, the confidence intervals usually overlap, but the mean is generally higher, indicating a slight but favorable trend.
In future work, we plan to investigate cases where the minimization step is more challenging and to incorporate observation noise into the construction of the Kriging surrogate. In addition, we aim to extend the Base Stock Simple Policy by introducing more options for discount levels and starting days, thereby enabling optimization over a larger decision space. Beyond this, we also plan to explore new policies that explicitly incorporate discounts.

# Appendix A

# Simulation environment

This section describes the main classes that constitute the simulation environment in our framework.

## A.1 Agents

### A.1.1 `BaseStockNoDiscount`

The `BaseStockNoDiscount` agent is implemented as a Python class. It represents a simple inventory policy that maintains a fixed base stock level for each product by ordering only the quantity needed to replenish inventory up to that level. No discounts are applied in this policy. Its key methods that we use are:

- **`set_parameters`**: receives a dictionary of order parameters and sets the target base stock levels internally for each product.
- **`get_action`**: given the current state observations, (which include on-order quantities, inventory levels for each product, and the current day of the week), this method computes order quantities to replenish inventory up to the base stock targets. The orders are rounded to allowed discrete sizes, and zero discount values are returned alongside the orders.

### A.1.2 `BaseStockSimple`

The `BaseStockSimple` agent extends the basic base stock policy by including the ability to apply time-based discounts on products with limited shelf life. This Pyhton class manages discrete order quantities along with discount levels over a finite horizon. Its main methods are:

- **`set_parameters`**: receives a dictionary containing decision parameters for each product, namely base stock levels, discount amounts, and the start day for discounts, and stores them internally.

- **`get_action`**: given the current state observation, computes order quantities required to replenish inventory up to the base stock targets. It also returns the discount vectors, where all units with age greater or equal to `discount_from` receive the constant discount `discount_amount`, while younger units receive no discount.

Orders for the base stock policies are rounded to the nearest multiple of 6 by rounding up if the remainder when divided by 6 is 3 or more; otherwise, they are rounded down.

### A.1.3 `ConstantOrderNoDiscount`

The `ConstantOrderNoDiscount` agent is a class that implements a simple inventory policy that orders a fixed quantity for each product regardless of the observed state, and applies no discounts.

Its main methods include:

- **`set_parameters`**: receives a dictionary containing constant order quantities for each product and stores them internally.

- **`get_action`**: returns the fixed order quantities for all products along with zero discounts for all shelf life periods.

### A.1.4 `ConstantOrderSimple`

The `ConstantOrderSimple` agent class orders a fixed quantity for each product and applies a constant discount to all products older than a specified age threshold.

The key characteristics are:

- **`set_parameters`**: sets fixed order quantities, discount amounts, and discount start ages for each product from input parameters.

- **`get_action`**: returns the fixed order quantities and a discount schedule where all units with age greater or equal to `discount_from` receive the constant discount `discount_amount`, while younger units receive no discount.

## A.2 `DailySimulation` Environment

The `DailySimulation` environment models inventory and demand dynamics over a finite horizon for multiple products, considering inventory aging, replenishment lead times, pricing, and stochastic demand. It integrates supply and inventory management components with demand simulation to evaluate profit outcomes in daily time steps.

Its main methods are:

- **`__init__`**: initializes the environment with product parameters, consumer model (costumer manager), inventory, supply and stat managers. It sets up internal state variables such as inventory aging vectors, lead times, shelf lives, prices, demand scenarios, and computes a newsvendor bound used to compute the upperbound of possible orders.
- **`clean`**: resets the environment is state before each simulation run, clearing inventories, sales records, prices, and other internal trackers. It is called in the constructor method.
- **`reset`**: initializes the environment for a new simulation episode. It first sets the random seed for the consumer demand generator to ensure reproducibility. Then, it creates a new demand scenario over the defined time horizon by calling the method **`makeScenario`** of the `consumer`. If **`rnd_initial_condition`** is `True`, it modifies the initial inventory and supply conditions randomly by calling **`change_initial_condition`**. Alternatively, if a specific **`initial_state`** is provided, the method sets the environment to that state using **`set_initial_condition`**. Finally, it calls **`restart`** to reset the internal step counter, clear histories, and generate the initial observation, which it returns.
- **`restart`**: resets the simulation environment to its initial state, preparing it for a new run. It sets the internal day counter `current_step` to zero and clears the states of all inventory and supply managers by reinitializing their inventories and orders to their starting values. It also resets the history logs for each product and constructs the initial observation dictionary `obs`. For each product, the observation consists of the current orders normalized by residual lead time, concatenated with the current inventory . Additionally, the observation includes the current day of the week (initialized to zero, representing Monday). Finally, it clears any accumulated statistics calling the **`clearStatistics`** method of the `StatManager` to ensure a fresh start for performance tracking, and returns the initial observation dictionary.
- **`_reset_prices`**: compiles and updates the current prices of all products into a single vector, which is used during demand simulation and profit calculations.

- **changePrice**: updates the price of a specific product dynamically during simulation by applying discounts.

- **step**: simulates a single day of operations by processing incoming inventory, generating stochastic demand based on current prices, fulfilling orders from available inventory, calculating daily profit, updating sales and inventory records, and advancing the simulation clock by one day. It returns the new observation state `obs`, the profit of the day (`reward`), a boolean value called `done` that is True whether this was the last day of the simulation and a dictionary named `info` containing information on the simulation.

## A.3 Managers

### A.3.1 CustomerManager

The `CustomerManager` class models customer arrivals and purchase behavior. It supports different discrete choice models (DCMs) to simulate customer preferences and demand scenarios with seasonal or custom arrival distributions. The class manages utility parameter generation, customer choice given product availability, and sales simulation over a finite time horizon.

Its main methods are:

- **__init__**: initializes the customer manager with store arrival settings and discrete choice model parameters, supporting types such as `LinearBeta`, `Logit`, and `Easy`.

- **_manage_arrival_settings**: processes the seasonal arrival distribution parameters and prepares statistical models (e.g., Negative Binomial) for daily arrivals.

- **makeChoice**: determines the product choice of a customer given availability, prices, qualities and utility parameters. Implements logic for different DCM types to simulate customer purchase decisions (in our case `LinearBeta.`

- **makeScenario**: generates a single demand scenario and corresponding customer utility parameters over a specified time horizon, sampling from the configured seasonal distribution.

- **sell_products**: simulates daily sales by iterating over customers in the demand scenario, offering available products, recording sales, and tracking lost or unmet demand. It is called in the **step_method of the** `DailySimulation` environment.

- **get_availability**: retrieves current product availability from inventory managers.
- **setEnv**, **reset**, **setSeed**: utility methods to link environment, reset random seed, or set a specific seed for reproducibility.

### A.3.2 InventoryManager

The `InventoryManager` class handles inventory tracking for a single product with a fixed shelf life. Each product requires its own `InventoryManager` instance to maintain its stock levels correctly.

Its main methods are:

- **__init__**: initializes the inventory with a specified `ShelfLife`, setting up zero stock for each age category.
- **clearState**: resets the inventory to a given initial state, allowing a clean start for simulations.
- **updateInventory**: advances the inventory ages by shifting stock to older age buckets, scrapping expired units at the end of the shelf life. It is called in the **step** method of `DailySimulation.`
- **receiveSupply**: adds newly ordered units to the freshest age bucket (age 0). It is called in the **step** method of `DailySimulation.`
- **meetDemand**: simulates selling one unit of a specific age if available; raises an error if the requested age is out of stock. It is used in the **sell_products method of the** `CustomerManager.`
- **isAvailable**: checks if any units of the product are currently in stock regardless of age.
- **isAvailableAge**: verifies availability of units at a specified age, raising an error if the age is invalid.
- **getProductAvailabilty**: returns a boolean array indicating the presence of stock at each age bucket.

### A.3.3 StatManager

The `StatManager` class tracks inventory and sales statistics over the simulation horizon.

Some of its main methods, which are employed in `DailySimualtion`, are:

- **`__init__`**: initializes counters for total ordered, sold, scrapped, unmet and lost demand and profit, based on the product settings.
- **`clearStatistics`**: resets all statistics and the internal clock.
- **`setTimeHorizon`**: sets the total time horizon and adjusts active periods considering head and tail transients.
- **`setHeadTail`**: sets the transient periods to exclude from statistics.
- **`updateClock`**: advances the internal time step by one unit.
- **`compute_reward`**: calculates daily profit from ordered, sold and scrapped quantities using product-specific prices and costs.
- **`updateStats`**: updates cumulative statistics if the current time is within the active period; returns the daily profit.
- **`updateUnmet`** and **`updateLost`**: update counts of unmet demand and lost demand respectively.
- **`setEnv`**: assigns an environment reference to access external data such as price arrays.

### A.3.4 `SupplyManager`

The `SupplyManager` class manages outstanding supply orders with respect to lead times for a single product. It maintains a queue of orders in transit and tracks deliveries as time progresses.

Its methods, used in `DailySimulation`, are:

- **`__init__`**: initializes the supply manager with a given lead time and creates an order queue of appropriate length.
- **`clearState`**: resets the order queue to a specified initial state.
- **`deliverSupply`**: advances the order queue by one time step, returning the quantity delivered at the end of the lead time.
- **`GetOrder`**: inserts a new order at the start of the order queue to be delivered after the lead time.

# Bibliography

Akçay, Y., H. P. Natarajan, and S. H. Xu (2010). «Joint Dynamic Pricing of Multiple Perishable Products Under Consumer Choice». In: *Management Science*. DOI: 10.1287/mnsc.1100.1178 (cit. on p. 5).

Ben-Akiva, M., M. Bierlaire, and R. W. Hall (2003). «Discrete Choice Models with Applications to Departure Time and Route Choice». In: *Handbook of Transportation Science*. DOI: 10.1007/0-306-48058-1_2 (cit. on p. 5).

Bouhlel, M. A., J. T. Hwang, N. Bartoli, R. Lafage, J. Morlier, and J. R. R. A. Martins (2019). «A Python surrogate modeling framework with derivatives». In: *Advances in Engineering Software*, p. 102662. ISSN: 0965-9978. DOI: https://doi.org/10.1016/j.advengsoft.2019.03.005 (cit. on p. 15).

Buisman, M. E., R. Haijema, and J. M. Bloemhof-Ruwaard (2019). «Discounting and dynamic shelf life to reduce fresh food waste at retailers». In: *International Journal of Production Economics* 209, pp. 274–284. DOI: 10.1016/j.ijpe.2017.07.016 (cit. on pp. 1, 4).

Chew, E. P., C. Lee, R. Liu, K. Hong, and A. Zhang (2014). «Optimal dynamic pricing and ordering decisions for perishable products». In: *International Journal of Production Economics* 157, pp. 39–48. DOI: 10.1016/j.ijpe.2013.12.022 (cit. on p. 4).

Chua, G. A., R. Mokhlesi, and A. Sainathan (2017). «Optimal Discounting and Replenishment Policies for Perishable Products». In: *International Journal of Production Economics* 186, pp. 8–20. DOI: 10.1016/j.ijpe.2017.01.016 (cit. on p. 3).

Fadda, E., D. G. Gioia, P. Brandimarte, and F. Maggioni (2024). «Joint Discount and Replenishment Parametric Policies for Perishable Products». In: *IFAC-PapersOnLine*. DOI: 10.1016/j.ifacol.2024.09.249 (cit. on pp. 2, 7, 8, 11, 43).

Gioia, D. G., L. K. Felizardo, and P. Brandimarte (2022). «Inventory management of vertically differentiated perishable products with stock-out based substitution». In: *IFAC-PapersOnLine*. DOI: 10.1016/j.ifacol.2022.10.115 (cit. on pp. 1, 2, 6–8).

Haijema, R. and S. Minner (2015). «Stock-level dependent ordering of perishables: A comparison of hybrid base-stock and constant order policies». In: *International Journal of Production Economics* 143, pp. 514–523. DOI: 10.1016/j.ijpe.2015.10.013 (cit. on p. 4).

Herring, J., V. Lurkin, L. A. Garrow, J.-P. Clark, and M. Bierlaire (2020). «Airline customers' connection time preferences in domestic U.S. markets». In: *Journal of Air Transport Management*. DOI: 10.1016/j.jairtraman.2019.101688 (cit. on p. 5).

Jones, D., M. Schonlau, and W. W.J. (1998). «Efficient Global Optimization of Expensive Black-Box Functions». In: *Journal of Global Optimization* 13, pp. 455–492. DOI: 10.1023/A:1008306431147 (cit. on pp. 16, 17).

Law, A. M. (2014). *Simulation Modeling and Analysis*. 5th ed. New York, NY: McGraw-Hill Education (cit. on p. 23).

Rehman, S. u. and M. Langelaar (2015). «Efficient global robust optimization of unconstrained problems affected by parametric uncertainties». In: *Structural and Multidisciplinary Optimization*. DOI: 10.1007/s00158-015-1236-x (cit. on pp. 18, 21, 33).

Sainathan, A. (2013). «Pricing and Replenishment of Competing Perishable Product Variants under Dynamic Demand Substitution». In: *Production and Operations Management*. DOI: 10.1111/poms.12004 (cit. on pp. 3, 5, 6).

Sifringer, B., V. Lurkin, and A. Alahi (2020). «Enhancing discrete choice models with representation learning». In: *Transportation Research Part B: Methodological*. DOI: 10.1016/j.trb.2020.08.006 (cit. on p. 5).

Solari, F., N. Lysova, M. Bocelli, A. Volpi, and R. Montanari (2024). «Perishable Product Inventory Management In The Case Of Discount Policies And Price-Sensitive Demand: Discrete Time Simulation And Sensitivity Analysis». In: *Procedia Computer Science* 232, pp. 1233–1241. DOI: 10.1016/j.procs.2024.01.121 (cit. on p. 4).

Transchel, S., M. E. Buisman, and R. Haijema (2022). «Joint assortment and inventory optimization for vertically differentiated products under consumer-driven substitution». In: *European Journal of Operational Research*. DOI: 10.1016/j.ejor.2021.10.024 (cit. on pp. 5, 6).

Virtanen, P., R. Gommers, T. E. Oliphant, and et al. (2020). «SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python». In: *Nature Methods* 17.3, pp. 261–272. DOI: 10.1038/s41592-019-0686-2 (cit. on p. 31).