Discrete-time simulation and optimization of multi-echelon distribution systems

Master thesis

Simon Riezebos ID: 0655458

Coordinator TU/e: prof.dr.ir. I.J.B.F. Adan
Supervisor TU/e: dr.ir. A.A.J. Lefeber
Advisor OM Partners: dr.ir. C.P.L. Veeger

DC2016.061

June 30, 2016

Dynamics and Control Group
Department of Mechanical Engineering
Eindhoven University of Technology
PO Box 513
5600 MB Eindhoven
The Netherlands

Preface

This thesis is the concluding piece to my Master's degree at Eindhoven University of Technology. I have been a Mechanical Engineering student for almost eight years now. During this time the university has provided me with loads of opportunities and knowledge, and student life has given me some of the most interesting experiences of my life. A central theme in the larger projects of my education is taking something that is happening in real life, full of chaos, unpredictability and exceptions, and translating it into a logically structured model. This can be said about my bachelor final project, my internship in Auckland, New Zealand and now, finally, at my graduation project at OM Partners.

This project has, like the rest of my education, taken slightly longer than originally planned. This is not something I regret, there have always been side-opportunities to studying, which take some time but give experiences and energy in return that I would never want to trade for a slighty quicker completion. Nevertheless, the project has been completed.

This would not have been possible without the support of my supervisors. I would like to thank Erjen Lefeber for his steady weekly guidance throughout the project, helping with hard decisions and keeping me on my toes when necessary. Next, I would like to thank Casper Veeger for the very pleasant collaboration at OM Partners and for all the conversational sparring, on- and off-topic, during the drives to and from the office. I would also like to thank Ivo Adan for the guidance throughout the project, and throughout my whole master, helping with the choice of Manufacturing Networks, the amazing internship in New Zealand and of course the graduation project at OM Partners.

Finally, I want to thank my family, my friends, and my fellow students for their patience and support, and the pleasant distractions from time to time.

Simon Riezebos
Eindhoven, June 30, 2016

Summary

An important topic for all companies that distribute products is inventory. There needs to be enough to be able to supply the product demand, but not too much, since that would take up valuable resources. For distribution networks that are becoming more and more complex, it is difficult to assess how much inventory there should be held throughout the network. OM Partners creates software that provides a solution to this problem. It can calculate the optimal amount of safety stock to aim for in each echelon. A simulation model was created in this project, which OM Partners can use to gain insight in the validity of analytical methods to calculate safety stocks in a single/multi-echelon network.

A local distribution center (LDC) supplies its customers, meeting their (daily) demand, and is replenished by a central distribution center (CDC) or factory, sending larger replenishment shipments every now and then, to make sure LDC stock does not run out. To prevent running out it keeps safety stock, the amount of safety stock is determined by a balance between costs and customer service. A certain amount of safety stock is supposed to guarantee a level of service towards customers. Calculations determine the necessary amount of safety stock.

Discrete-time simulations are used in this project to validate these calculations. A Matlab model that follows the same rules as a distribution center would is simulated and gives similar results as the calculations do. This validates the equations used for demand patterns with a high demand frequency.

When there is a distribution network the problem becomes more complex, stock must be minimized by deciding whether to keep more stock centrally or locally. The discrete-time model was expanded, using curve-fitting and optimization, to also find an optimal safety stock for two echelons. The multi-echelon model is compared to analytical models. Results are not equal, but all methods show that a high service level from CDC to LDC is costly and unnecessary. Furthermore, there are plausible hypotheses to explain the differences between the analytical approximation and the discrete-time model.

The results of this project give OM Partners a new way to support their calculations towards customers, and helps to reduce superfluous inventory.

Contents

Symbols

P_1 := probability of no stockout just before the arrival of a replenishment order

P_2 := fraction of demand satisfied directly from the shelf

R := review period (time)

s := re-order point (number of products)

S := order-up-to level (number of products)

Q := order quantity (number of products)

$X(t)$:= inventory level at time t (number of products)

$Y(t)$:= inventory position at time t (number of products)

$O(t)$:= inventory in transit at time t (number of products)

L := lead time (time)

B := backorders (number of products)

D := demand (number of products)

SS := safety stock (number of products)

k := safety factor (inverse of probability distribution)

$E[x]$:= expected value of x

σ_x^2 := variance of x

COV := coefficient of variation, $\dfrac{\sigma_x}{E[x]}$

$Gam_{CDF}(x) := \dfrac{1}{\beta^\alpha \Gamma(\alpha)} \int_0^x t^{\alpha-1} e^{\frac{-t}{\beta}} dt$

$\Gamma(x) := \int_0^\infty e^{-t} t^{x-1} dt$

$\Phi(x) := \dfrac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2}\, dt$

Chapter 1

Introduction

This thesis is the result of a collaboration between Eindhoven University of Technology and OM Partners. OM Partners develops supply chain planning software that is used by many companies all over the world. Part of this software focuses on the flow of products through a distribution network. An example of a distribution network is a factory which produces products to be used worldwide, these products are sent to central distribution centers (CDCs), which are located in strategic locations around the world, in large batches (e.g. freighters). From here, the products are sent to local distribution centers (LDCs) in smaller batches (e.g. trucks), from which they are sent to customers/resellers in even smaller batches (e.g. pallets/boxes).

If there were no ordering policies or inventory optimization, when a customer orders a box of products, they would have to wait until the factory produces it, sends a freighter to a CDC, which sends a truckload to an LDC, which can then send the box to the customer. This option is far from ideal.

Zooming in on one DC, the stock level can be defined as the number of products (a number of units of 1 products) currently available in inventory. Products depart from the DC, towards a lower level DC or a customer, which reduces stock. To create a more effective product flow compared to the previous example, a replenishment policy can also be defined to make sure that stock generally does not go below zero (negative stock is defined as backorders). This way, in reaction to the reduction in stock caused by customer demand, a replenishment order is placed at a higher level DC or the factory, which arrives with a delay defined as lead time, and increases the inventory level. An example of this process can be seen in Figure 1.1. It also shows that inventory level is the number of available products, which becomes negative if there is customer demand when there is no stock, and inventory position which is a combination of inventory level and inventory that has been ordered but has not arrived.

When customer demand and all lead times can be predicted perfectly, timely orders can be placed so that stock never runs out but is also never more than necessary to fulfill the orders until the next replenishment. Unfortunately, this is generally impossible due to stochastic lead times and demand. This is why, in addition to the stock that would be necessary to fulfill the average demand between replenishments, it can be helpful to hold extra safety stock.

Holding extra inventory causes higher inventory holding costs, but running out of stock causes backorder costs and decreased customer satisfaction. A relation can be defined between safety stock and (customer) service level. This relation is the main subject of this project. The goal is to validate inventory optimization equations for safety stock calculation and determine optimal safety stock levels through discrete-time simulation.

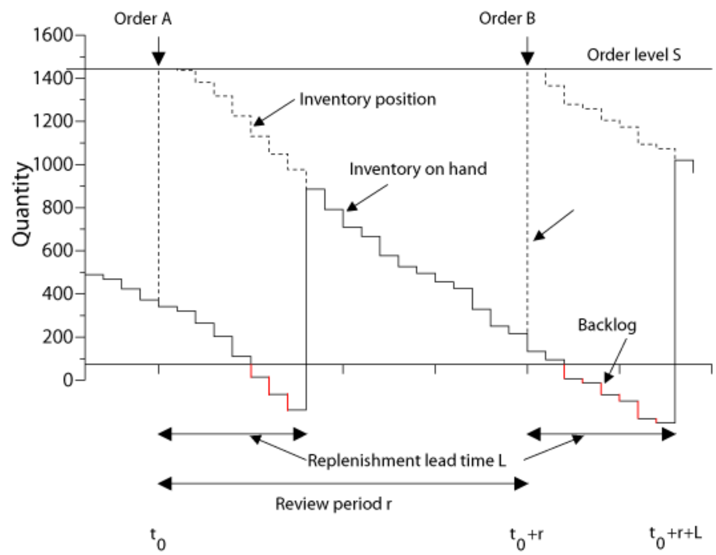


Figure 1.1: Example of stock level trajectory

The first step is simulating the situation that was explained previously, by zooming in on one DC, using time intervals. Two common replenishment policies and four demand patterns are implemented.

The next step is creating a discrete-time model that simulates a distribution network instead of a single DC. By expanding to multiple echelons, which means that DCs are connected in series, the relation between safety stock and service level becomes more complex. The only place where backorders create costs and customer dissatisfaction is at the lower echelons, where products are shipped to customers. Instead of balancing one safety stock level against a service level, in this case the safety stocks of all higher echelons determine the service level towards the customers. This multi-echelon situation is also simulated using a discrete-time model, and the results are used to validate analytical equations. The discrete-time model is developed to be able to simulate 1 CDC with multiple LDCs, the LDCs can not exchange products between each other. Input parameters for the developed discrete-time models can be defined in an Excel file.

The remaining part of this thesis starts with an elaboration of the theory behind the analytical model and the way the discrete-time model is created using literature in Chapter 2. The next step is the design and details of the single-echelon discrete time model, followed by its validation, in Chapter 3. In Chapter 4, this model is then expanded to multi-echelon, where the expanded discrete-time model is also used in a curve-fitting and optimization sequence. Multi-echelon results are validated, and lastly, in Chapter 5, conclusions are drawn and recommendations are made.

Chapter 2

Literature review

This chapter gives an overview of the theory from literature that is used in this project. The first section describes the formulas used in single-echelon inventory optimization to calculate safety stocks for desired service levels, the second section expands into multi-echelon theory applied to the single-echelon equations. The third section explains discrete-time simulation as used in this project and the last section is about the optimization techniques used in the multi-echelon model.

2.1 Single-echelon inventory optimization equations

In a distribution center where demand D and lead time L are stochastic, when placing an order to replenish inventory, an estimate must be made of what will happen until the order arrives and until the next order can be placed to keep service towards customers up. This estimation is done using inventory optimization equations. To understand the equations, the terminology used in the previous chapter is explained further. The equations can be used to calculate service level based on several variables. In [2], the service levels are defined and calculated using statistical deductions on inventory systems. The service levels P_1 and P_2 are defined in the second chapter.

- P_1 := probability of no stockout just before the arrival of a replenishment order.
- P_2 := fraction of demand satisfied directly from the shelf.

Which means that looking back at past results for a DC, service can be calculated using:

$$P_1 = 1 - \frac{n_{so}}{n_{rpl}},$$

$$P_2 = 1 - \frac{B_{tot}}{D_{tot}}.$$

n_{so} is the number of times that there was no more stock right before it was replenished, n_{rpl} is the total number of replenishments, B_{tot} is the total amount of backorders that occurred and D_{tot} is the total amount of demand that was requested. This assumes that backordered demand is sent out as soon as it becomes available.

When looking at the future, these variables are unknown, but it is possible to calculate expected values for them using data about demand ($E[D]$ and σ_D), lead time ($E[L]$ and σ_L) and policy. For (R, s, Q) and (R, S) policies, which are used in this project, service is based on what happens during the uncertainty period:

- In case of an (R, s, Q) policy, after amount of time R has passed, an order of Q products is placed if the inventory position ($Y(t)$) is below re-order point s. Service levels can be calculated by deducing the probability distribution of demand that is ordered by customers between the moment a replenishment order is placed and the moment it arrives. The expected uncertainty period therefore is $E[L]$, with standard deviation σ_L.

- For the (R, S) policy, where the difference between the current inventory position $Y(t)$ and the order-up-to level S is placed each time that review period R has passed, service levels can be calculated the same way. Instead of looking between placing and arrival of an order, it is necessary to look between placing the first order and arrival of the next order, since inventory will keep decreasing after the arrival of an order, and should be prevented from going empty in between orders. The expected uncertainty period therefore is $R + E[L]$, with standard deviation σ_L.

Due to the stochastic behavior of demand and lead time, the number of products that is necessary during this uncertainty period is variable, as can be seen in Figure 2 of [2]. The amount of fluctuation depends on the variances. If a lot of demand occurs after placement but before arrival of a replenishment a stockout can occur, but if demand is average and lead time is long a stockout can still occur. The expected value and variance of demand during uncertainty period ($DDUP$) can be calculated using:

$$E[DDUP, (R, s, Q)] = E[L]E[D],$$
$$E[DDUP, (R, S)] = (E[L] + R)E[D],$$
$$\sigma^2_{DDUP,(R,s,Q)} = \sigma^2_D E[L] + \sigma^2_L E^2(D),$$
$$\sigma^2_{DDUP,(R,S)} = \sigma^2_D(E[L] + R) + \sigma^2_L E^2(D).$$

Using $E[DDUP]$ as re-order point or order-up-to level is defined as having 0 safety stock, which means that:

$$s_{(R,s,Q)} = E[DDUP, (R, s, Q)] + SS,$$
$$S_{(R,S)} = E[DDUP, (R, S)] + SS,$$

where s is the re-order point, S is the order-up-to level and SS is the amount of safety stock. If all DC parameters are known, and a re-order point or order-up-to level are chosen, the probability that $DDUP$ is below this value can be determined using the cumulative distribution function (CDF) of the demand pattern. This project uses the Gamma distribution, so this calculation is done using:

$$Gam_{CDF}(S, \alpha, \beta) := \frac{1}{\beta^\alpha \Gamma(\alpha)} \int_0^S t^{\alpha-1} e^{\frac{-t}{\beta}}\, dt,$$

where

$$\Gamma(\alpha) := \int_0^\infty e^{-t} t^{\alpha-1} dt,$$
$$\alpha = \left(\frac{E[DDUP]}{\sigma_{DDUP}}\right)^2,$$
$$\beta = \frac{\sigma^2_{DDUP}}{E[DDUP]}.$$

Here, S is used for both the re-order point and the order-up-to-level. Using Gam_{CDF} this way, the P_1 service level can be calculated for (R, s, Q) and (R, S) distribution centers. Similarly, the required safety stock to obtain a service level can be calculated using the inverse of this CDF. P_2 is determined by calculating the expected value of the number of backorders per replenishment. Using equation (73) from [2], it can be determined for (R, S) policies that:

$$P_2 = 1 - \frac{Gam_1 - Gam_2}{E[D]R},$$

with:

$$Gam_1 = E[DDUP](1 - Gam_{CDF}(S, \alpha + 1, \beta)) - S(1 - Gam_{CDF}(S, \alpha, \beta)),$$

$$Gam_2 = E[DDUP](1 - Gam_{CDF}(S + E[D]R, \alpha + 1, \beta)) - (S + E[D]R)(1 - Gam_{CDF}(S + E[D]R, \alpha, \beta)),$$

where $E[D]R$ and S can be replaced with Q and s respectively, to use this equation for (R, s, Q) policies instead of (R, S).

2.1.1 Undershoot

The equations used for service level calculations previously had the assumption that the inventory level is exactly at s at the moment a replenishment is ordered in case of an (R, s, Q) policy. There are two reasons why this is almost never the case:

- Demand is not continuous but discrete, so before and after 1 customer order the inventory level can go from above s to below s.
- The inventory level is checked each review period, if it is slightly above s at the moment of a check, it is probably going to be a lot lower than s a full review period later.

For this reason, undershoot is introduced in Chapter 5.2 of [2]. Undershoot is defined as the number of products that the inventory level is below s at the moment a replenishment order is placed. $E[DDUP]$ and σ_{DDUP} can be replaced the following way:

$$E[U] = \frac{\sigma_D^2 R + E[D]^2 R^2}{2E[D]R},$$

$$\sigma_U^2 = (1 + \frac{(\frac{\sigma_D}{E[D]})^2}{R})(1 + 2\frac{(\frac{\sigma_D}{E[D]})^2}{R}(\frac{(E[D]R)^2}{3} - E^2[U])),$$

$$E[DDUP_U] = E[U] + E[DDUP],$$

$$\sigma_{DDUP_U}^2 = \sigma_U^2 + \sigma_{DDUP}^2.$$

Here, U represents undershoot. Using the equations described in this section the P_1 and P_2 service levels for single-echelon inventory optimization can be calculated for any safety stock for the (R, s, Q) and (R, S) policy.

2.1.2 Demand pattern classification

In [6], demand patterns are classified according to four quadrants. The parameters used to classify demand are the coefficient of variation and the average inter-demand interval. For both parameters there is a threshold that determines which quadrant a demand pattern belongs to. Demand can be classified as:

- Smooth demand: demand with a low coefficient of variation ($COV < 0.5$) and a demand greater than 0 in $> 75\%$ of time periods.
- Erratic demand: demand with a high coefficient of variation ($COV \geq 0.5$) and a demand greater than 0 in $> 75\%$ of time periods.
- Intermittent demand: demand with a low coefficient of variation ($COV < 0.5$) and a demand of 0 in $\geq 25\%$ of time periods.
- Lumpy demand: demand with a high coefficient of variation ($COV \geq 0.5$) and a demand of 0 in $\geq 25\%$ of time periods.

2.2 Multi-echelon inventory optimization equations

In the previous section, a single DC was considered. Replenishments were represented by a lead time with an expected value and a standard deviation. By that way of modeling, the assumption is made that in the echelon above the modeled DC, each order can be deployed immediately and completely. Since the higher ranking echelon is often also a DC (with occasional stockouts) this assumption is incorrect. This section describes the extensions that are added to the single-echelon equations so that more echelons can be modeled.

2.2.1 Multi-echelon approach by Desmet

When looking at a two-echelon distribution system with one CDC and multiple LDCs, the only difference with the single-echelon system occurs when CDC stock is insufficient to fulfill LDC demand directly. This causes a delay in the delivery of products ordered by LDCs. The exact impact of this delay is difficult to determine. It is approximated in [4].

The delay is approximated using an adjustment to the lead time parameters of LDCs, making $E[L_{LDC}]$ and $\sigma_{L_{LDC}}$ dependent of CDC service level and lead time. Using the equations in Section 2.1, $P_{1_{CDC}}$ can be determined for a certain amount of CDC safety stock, using the combined demand of all LDCs as demand for the CDC. This service level is used to calculate adjusted lead time parameters for LDCs:

$$E[L^*_{LDC}] = E[L_{LDC}] + (1 - P_{1,CDC})E[L_{CDC}],$$
$$\sigma^2_{L^*_{LDC}} = \sigma^2_{L_{LDC}} + (1 - P_{1,CDC})^2 \sigma^2_{L_{CDC}}.$$

Here, L^* represents the adjusted lead time. In this representation, a stockout at the CDC means that the LDC will have to wait an extra L_{CDC}, which occurs $(1 - P_1)$ times on average. By replacing the values used for lead time in the equations in Section 2.1 with the result of the equations above, adjusted LDC service levels can be calculated.

When the objective is to minimize costs while maximizing customer service, the sum of all safety stocks should be minimized (possibly with a correction for differing inventory costs in CDCs/LDCs) while maintaining a desired LDC service level. Since SS_{CDC} directly correlates to $P_{1,CDC}$, which is used in the equations to determine LDC service levels, the optimal configuration can be found by searching through configurations with high SS_{CDC} and low SS_{LDC} to the opposite, to find the minimal total safety stock that meets LDC service level criteria.

2.2.2 Adjusted $P_{1,CDC}$ by Dendauw

An adjustment to the approximation is proposed in [3]. The reasoning made here is that when the CDC has a large lotsize, most of the time when an order from the LDC comes in, there is a higher chance than P_1 that stock is still available, since chances are very small that stock runs out right after replenishment of CDC inventory. It uses an adjusted formula for $P_{1,CDC}$:

$$P^*_{1,CDC} = \Phi(\frac{Q_{CDC}/2 + SS_{CDC}}{\sigma_{DDUP,CDC}}),$$

where Q_{CDC} is either Q from CDC parameters in case of an (R, s, Q) policy or $E[D]R$ in case of an (R, S) policy. Φ represents the standardized Normal cumulative distribution function:

$$\Phi(x) \ = \ \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\, dt.$$

By replacing $P_{1,CDC}$ in the equations in Section 2.1 with the result from the calculation of $P^*_{1,CDC}$, and using this value in the equations in Section 2.2.1, a new optimal safety stock configuration can be determined.

2.2.3 Adjusted waiting time by Desmet

In [5], an adjusted representation of the waiting time is proposed. In Section 2.2.1, the proposed waiting time is L_{CDC}. This is replaced by a waiting time approximation that uses an Exponential distribution with an average that is dependent on CDC parameters. A formula for calculation of this adjusted waiting time is not shown, the adjusted waiting time is approximated in Section 4.4.5.

2.3 Discrete-time simulation for distribution systems

Discrete-time simulation can be used to make inventory models of distribution networks, in this case, this modeling type is used to simulate inventory systems. When analytical models are used, stochasticity and expansive models can quickly increase complexity in accompanying calculations. In discrete-time models this mostly increases the workload for the computer. Processing power in computers increases every year, which makes simulation an accessible way of modeling supply chains and distribution networks.

As can be seen in Chapter 3.2 of [1], discrete-time models function by iterating through time-steps and updating variables each step. When applied to inventory models, the basic equations needed are:

$$X(k+1) = X(k) - D(k) + Q(k-L),$$
$$Y(k+1) = Y(k) - D(k) + Q(k),$$

where k represents each time interval, X and Y are inventory level and position respectively, D is demand which can be stochastic, Q is stock ordered and L is lead time which can also be stochastic.

A roadmap for using discrete-event simulation to simulate inventory models can be found in [7]. This can also be applied to discrete-time simulation. The steps of the plan presented in this article are:

1. Formulate problem
2. Specify independent and dependent variables
3. Develop and validate conceptual model
4. Collect data
5. Develop and verify computer-based model
6. Validate the model
7. Perform simulations
8. Analyze and document results

These steps are quite representative for the project, and broadly lay out the steps that are taken. This process can be applied for single- and multi-echelon simulation and can partially be applied in optimization as well.

The modeling environment that is used in this project is Matlab[8], specifically the statistics, curve-fitting and optimization toolbox. Scripts are made using the Matlab programming language to create the models and do simulations.

2.4 Constrained nonlinear multivariable optimization

This section describes the optimization method and curve fitting method that are used in this project. Optimization problems are defined based on the methods introduced in Section 1.2 of [9]:

$$\begin{aligned} \text{minimize} \;\; & f(\mathbf{x}), \\ \text{subject to} \;\; & \mathbf{h}(\mathbf{x}) = \mathbf{0}, \\ & \mathbf{g}(\mathbf{x}) \leq \mathbf{0}. \end{aligned}$$

The goal is to minimize the outcome of function $f(\mathbf{x})$ which is dependent of variable(s) $\mathbf{x}$, while constraints can apply to the variable(s) $\mathbf{x}$. The constraints can be equality constraints ($\mathbf{h}(\mathbf{x})$), e.g. $x_1 = 2x_2$, or inequality constraints ($\mathbf{g}(\mathbf{x})$), e.g. $x_1 \geq 4$. Applying this to, for example, the problem in Section 2.2, the optimization problem could be to minimize total safety stock, while making sure the service level does not go below a chosen requirement.

One of the challenges in optimization is described in Chapter 3 of [9]. When a function has multiple local minima, like the function in Figure 3.3 (p. 97), the minimum that is found might not be the true minimum. A minimum is found using methods like searching along a line (Section 4.6 on p. 154), which generally report the first minimum that is observed. It helps if a function is monotonic, which means that it is either increasing or decreasing for each value (e.g. $f(x_2) > f(x_1)$ for each $x_2 > x_1$). For example, since each extra unit of safety stock in a DC decreases the probability that a stockout occurs, the relation between safety stock and service level is monotonically increasing. To solve an optimization problem it is necessary to check these properties and to make sure that the optimum that is found is not just a local optimum.

These methods and properties for optimization problems are applied in this project using Matlab. The function `fmincon` is used to find the minimum of a constrained nonlinear multivariable function. Which is an optimization problem as described in this section.

2.4.1 Least squares curve fitting

Curve fitting can be seen as an optimization problem, as described in Section 2.2 of [9] (p. 49). When a set of x- and y-values is available, and the relation between x and y should be found, this can be done by curve fitting. When there is a linear correlation (e.g. $y = ax$), the problem becomes an optimization problem where an optimal value for a is found by minimizing the sum of squared residuals, which is the difference between values calculated using the formula and data points, by changing the fitting coefficients (e.g. a). This can be done for nonlinear functions with multiple coefficients as well.

Curve fitting in this project is done using Matlab, which has the function `fit`, where curve fitting can be done for a wide range of problems using methods like least squares curve fitting as described in this section.

The single- and multi-echelon theory has been described in this chapter, as well as the simulation and optimization methods and software used in the project. In the next chapter discrete-time simulation will be used to simulate a single echelon distribution system.

Chapter 3

Single-echelon Matlab model

This chapter applies the theory on discrete-time simulation explained in Section 2.3, and compares the results to the results of the single-echelon equations in Section 2.1. A discrete-time Matlab model is created to simulate a distribution center for various policies and demand patterns.

3.1 Conceptual model

This first section gives a description of the desired input and output of the discrete-time model, as well as an overview of the steps that are taken in the model. This is based on the theory in Section 2.3.

3.1.1 Specification of model variables

The discrete-time model should be able to give the same output as the equations in Section 2.1, based on the same input variables. The output of the model should therefore be the service levels P_1 and P_2. A discrete-time model uses timesteps, therefore the input variables are specified as:

- $E[D]$ and σ_D are the expected value and standard deviation of the customer demand per time interval. These are determined using a Gamma distribution, a Normal distribution is common but that distribution can have values below 0. The model is going to randomly generate demand values using $E[D]$ and σ_D. To prevent "negative" demand, the Gamma distribution as described in Section 2.1 is chosen.

- $P(D)$ is used by a Geometric distribution to determine the interval until the next period with customer demand. This is used in cases where not each time interval contains demand. After each demand, the distribution $P(n_0 = k) = (1-p)^k\,p$ is sampled to check the number of time intervals n_0 that have no demand, until the next period that contains demand. When $P(D) = 1$, the results is always 0, so that there are 0 periods without demand.

- R is the review period, which is represented as the number of time intervals between reviews.

- Q is the number of products that is ordered when inventory order position is lower than the re-order point in case of an (R, s, Q) policy.

- $E[L]$ and σ_L are the expected value and standard deviation of customer demand in time periods, also generated from a Gamma distribution to prevent negative values.

- SS is the number of products that is added to $E[DDUP]$ (Section 2.1) to either determine the re-order point or the order-up-to level depending on the replenishment policy.

In the declaration above, all variables except SS are properties of the distribution center that is simulated. SS is the independent variable that is changed to influence output service levels.

3.1.2 Conceptual model overview

A model must be created that uses the input variables described in the previous section to determine the service level as output. This is accomplished by doing a discrete-time simulation, where the distribution system is simulated for a large number of time intervals. If the DC is modeled accurately, the number of stockouts and backorders that occurred during simulation can be used to determine the service levels.

For each discrete time interval the model variables and the values used to calculate the service levels are updated, as presented in Figure 3.1.

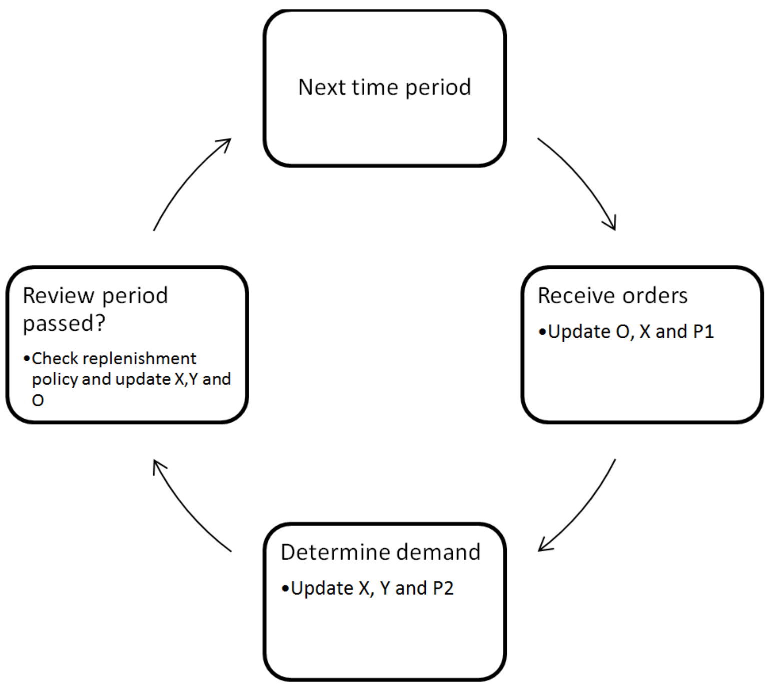


Figure 3.1: Flowchart of communication in single-echelon model

- For each time period the first step is adding any received orders to the inventory level X and removing those orders from the orders in transit O. The model should keep track of the number of received replenishments and the number of stockouts so that P_1 can be calculated after simulation.
- Then the demand for that time period is handled by subtracting it from inventory level X and inventory position Y. The model should keep track of the total amount of demand and the number of backorders that occur so that P_2 can be calculated after simulation.
- When a review period has passed, the current inventory position Y is checked with the order policy to determine whether an order should be placed or how large the order should be, depending on the policy. The inventory position Y and orders in transit O are updated when necessary.

A more detailed overview is shown in Listing 3.1, where the structure of the model can be seen in pseudocode. The Matlab model that is described in the next section follows this structure.

```
initialize

for each safety stock value
    for the defined number of experiments
        for the total number of review periods that fit in an experiment
            for the number of time periods in a review period
                %% Receive replenishment orders
                if an order on order list is receivable
                    if net stock is below zero
                        update number of replenishments and stockouts
                    end
                    add order to net stock
                end
                %% Process demand
                subtract current time period demand from net stock and ...
                    inventory position
                if net stock is below zero
                    update backorders
                end
            end
            %% Check replenishment policy
            if replenishment policy indicates new order
                add order to order list and inventory position
            end
        end
        calculate service levels (current experiment)
    end
    calculate mean and confidence interval (all experiments, current ...
        safety stock value)
end

plot mean and confidence interval of all safety stock values
```

Listing 3.1: Pseudocode model overview

3.2 Matlab model

This section gives an overview of the Matlab model created to do discrete-time simulation for single-echelon distribution systems with 1 DC. Details of the model are explained for relevant parts of the model structure as shown in Listing 3.1. The full code is shown in Appendix A.

3.2.1 Initialization

The code starts with clearing memory and setting simulation parameters as shown in Listing 3.2. Simulation accuracy is determined by the simulation scale, the number of time intervals that are simulated and the total number of simulations that are done to determine result confidence interval. `Nstart` is the number of time intervals that do not count for service level results to prevent startup effects.

```
clearvars; close all;

%Simulation properties
simscale = 1; %Time periods are split into smaller steps using scale
Nstart = 500*simscale; %Number of time periods that do not count for results
N = (10000+Nstart)*simscale; %Total number of time periods
NX = 15; %Number of experiments
simgraph = 0; %If 1 the simulation runs once and makes a graph of ...
    inventory levels
```

Listing 3.2: Clear memory and set simulation parameters

Then, the code in Listing 3.3 calculates DC parameters using input values from excel and the simulation properties determined before. Some basic formulas are used to determine the values of re-order point/order-up-to level S and order size Q. S is used as the expected demand during the uncertainty period $E[DDUP]$ to which safety stock is added during simulation. Q is a factor of the expected demand during a review period.

```
input = xlsread('InputSE.xlsx','Sheet1');

%Calculate system properties from input
type = input(1); %0 = RsQ policy, 1 = RS policy
ED = input(2)/simscale; %Expected value of demand per time period
sigD = input(3)/simscale; %standard deviation of demand per time period
PD = input(4); %lambda of Poisson distribution that determines demand interval
EL = input(5)*simscale; %Expected  value of lead time in time periods
sigL = input(6)*simscale; %Standard deviation of lead time in time periods
R = input(7)*simscale; %Review period in time periods
Q = input(8)*ED*R*PD; %Order quantity level using standard formula (only RsQ)
sslow = input(9); %Lower bound of safety stock
ssint = input(10); %Interval of safety stock values
sshigh = input(11); %Upper bound of safety stock

%Initialization
```

Listing 3.3: Calculate system properties

Instead of generating demand values during simulation, a demand matrix is generated before simulation, this is more efficient computationally, and allows each simulation with a different SS value to use the same demand matrix. The generation of the demand vector is shown in Listing 3.4. `gamrnd` generates demand values using the input values of $E[D]$ and σ_D. If $P(D) < 1$, the intervals between time periods that have demand are generated using `geornd`. The periods without demand change the parameters of the demand used in simulation. Therefore, the average and standard deviation of the demand are calculated from the demand matrix after it is generated.

```
    %used for all values of ss
%When a lambda is given, a geometric distribution calculates the time periods
    %until the next demand
if PD == 1
    seed = gamrnd((ED/sigD)^2,sigD^2/ED,N,NX);
else
    seed = zeros(N,NX);
    for x = 1:NX
        next = geornd(PD); %Determine time periods until next order
        for y = 1:N %Generate demand  or fill in pause for all demands
            if next == 0
                seed(y,x) = gamrnd((ED/sigD)^2,sigD^2/ED);
                next = geornd(PD);
            else
                seed(y,x) = 0;
                Nzero = Nzero + 1;
                next = next - 1;
            end
        end
    end
end
avgdemand = mean(mean(seed)); %Resulting average of demands
prcntdemand = 1-Nzero/(N*NX); %Resulting % of filled time periods
sigdemand = mean(std(seed)); %Resulting standard deviation of demands
```

Listing 3.4: Demand generation

3.2.2 Receiving replenishment

Next is the main part of the model. For each value in the safety stock range defined by input, `NX` simulations are conducted. For 1 simulation of 1 SS value, the model starts by resetting variables like inventory level and position, after which a loop over all time periods starts.

The loop over each time period starts by checking whether a replenishment order has arrived between the previous and the current time period, as shown in Listing 3.5. Replenishment orders that are in transit are stored in cell `O` which contains a 2-by-n_o matrix where n_o is the number of replenishments in transit. This matrix could contain, for example:

$$\begin{bmatrix} 100 & 3.7 \\ 100 & 11.5 \end{bmatrix}.$$

This means that there are currently 2 replenishments in transit that both contain 100 products, the first arriving after 4 time periods and the next after 12 time periods.

If `O` is not empty, 1 is subtracted from the second column. If the number in the second column of the first replenishment becomes negative, it has arrived between the previous and the current time period. The number of products in that replenishment order is added to the inventory level, and the numbers of replenishments and stockouts are updated for calculation of P_1. The replenishment order is then removed from `O`.

```
            end
            %Check if there are orders that have arrived
            %Update values for P1
            if size(O{1},1) > 0 %Are there orders?
                O{1}(:,2) = O{1}(:,2) - 1;
                if O{1}(1,2) < 0
                    NQtot = NQtot + 1*start;
                    if X < 0
                        Nstockout = Nstockout + 1*start;
                    end
                    X = X + O{1}(1,1);
                    O{1}(1,:) = [];
                end
```

Listing 3.5: Receive orders

3.2.3 Demand

Demand is fulfilled in Listing 3.6. The demand value comes from the matrix generated in Listing 3.4, total demand is updated, and if the current inventory level is insufficient, backorders are calculated, for calculation of P_2. Then, demand is subtracted from `Y` and `X`. The replenishment and demand part of the model are repeated until the end of a review period.

```
            end
            %Determine demand
            D = seed(R*(k-1)+i,xper);
            %Update values for P2
            Dtot = Dtot + D*start;
            if X ≤ 0
                Btot = Btot + D*start;
            else
                if X < D
                    Btot = Btot + start*(D - X);
                end
            end
            %Update inventory levels and record data for visualization
            Y = Y - D;
```

Listing 3.6: Check demand

3.2.4 Replenishment policy

After each review period, the replenishment policy is checked, this is shown in Listing 3.7. In case of an (R, s, Q) model (`type 0`), a multiple of `Q` products is ordered if the inventory position is below `s`, if the inventory position is more than `Q` products below `s`, `Q` is ordered twice (or more if necessary). For (R, S) models (`type` 1), the difference between `Y` and `s` is ordered. Orders are placed by adding a line to the matrix in cell `O`, containing the order quantity and the lead time.

```
            end
            %After each review period update orders
            if Y < s && type == 0
                L = gamrnd((EL/sigL)^2,sigL^2/EL);
                O{1} = [O{1};ceil((s-Y)/Q)*Q L];
                Y = Y + ceil((s-Y)/Q)*Q;
            elseif type == 1
                L = gamrnd((EL/sigL)^2,sigL^2/EL);
                Q = s-Y;
                O{1} = [O{1};Q L];
                Y = Y + Q;
```

Listing 3.7: Replenishment policy

3.2.5 Results calculation

After simulating all review periods, results for P_1 and P_2 are recorded in matrix `P`, this is done for all `NX` simulations that are done per value of safety stock, as shown in Listing 3.8. If the model is in "simulation graph" mode (`simgraph` 1), it exits the loop after the first run, since enough data to create a diagram of a single simulation is available.

```
        end
        %Record service levels
        P(xper,:) = [(1-Nstockout/NQtot) (1-Btot/Dtot)];
        %Print result of simulation for visualization and terminate loops
        if simgraph == 1
            break;
        end
    end
    if simgraph == 1
        break;
```

Listing 3.8: Record results

The last step in simulation is shown in Listing 3.9, for both P_1 and P_2 the average and confidence interval of all experiments for a safety stock value is calculated and collected in the results matrix.

```
    end
    %Determine mean and std
    Results(r,:) = [s mean(P) tinv(0.975,NX-1)/sqrt(NX)*std(P)];
```

Listing 3.9: Collect results

3.2.6 Results processing

After all simulation loops have finished a plot of the results is created. If the model is in "simulation graph" mode, data of `Y` and `X` is collected for a plot of a single simulation. Measurements are before each time period loop, after receiving orders, after serving demand and after updating the replenishment policy. An example of the visualized data for a single run is shown in Figure 3.2. Here, another function can also be seen, to increase accuracy the time resolution of the simulation can be decreased. The first diagram shows an (R, s, Q) policy with normal resolution, the second an (R, S) policy with a 24x higher resolution.

This Matlab model can be simulated for different DC parameters, results for a representative set of different situations are compared to the analytical equation results in Section 3.4

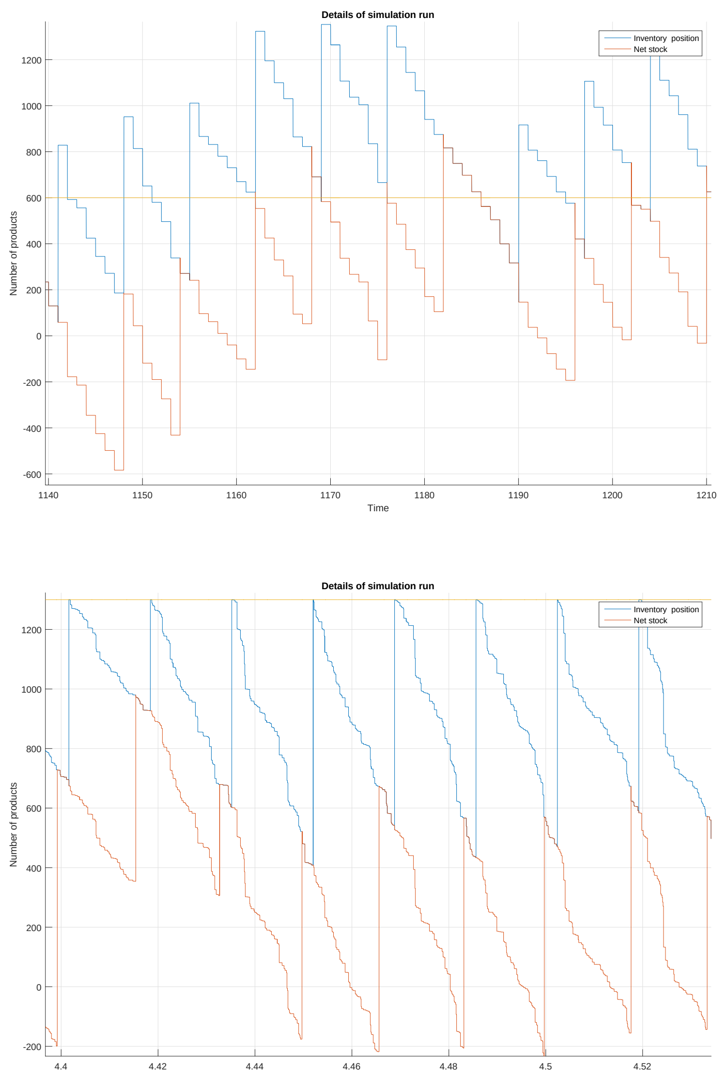


Figure 3.2: Single run diagram of normal resolution (R, s, Q) model and 24x resolution (R, S) model

3.3 Single-echelon optimization

The single-echelon Matlab discrete-time model is used to simulate and plot safety stock vs. service level curves. The next step is using the discrete-time model to find the optimal safety stock value for a set of input variables and a service level requirement. This section handles an adjusted version of the single-echelon Matlab model that can be used to find the optimal safety stock value. This Matlab script determines the optimal safety stock using the following steps:

- Determine approximate safety stock range where service level goes from 0% to 100%.
- Simulate in intervals across relevant range.
- Determine approximate service level equation by curve-fitting on simulation results.
- Calculate optimum from equation.

Some adjustments are made to the original single-echelon script to accommodate the steps above. To quickly be able to run the original single-echelon script for different input parameters that are not predefined, it is turned into a nested function. This way the optimization part and the simulation part of the script can use the same (excel) input. The full script can be found in Appendix A.2. The last part of that script contains function `SEIO_func`, which is very similar to the script explained in the previous section. This function is used throughout the script and is not explained in detail. To accommodate a nested function, the optimization script itself is also a function, which is invoked by another script using the Excel input as input for the function `SEIO_optim_fit`.

3.3.1 Find relevant safety stock range

The Matlab model is supposed to find an optimum without prior information about service level outcomes. Initially, it is therefore assumed that there is no information about the safety stock at which a service level can be achieved. Before any curve-fitting can be done, the relevant range of safety stocks must be determined. The first part of the optimization looks for the approximate point where the service level is 50%, which is the steepest point, in order to determine the safety stock range where the service level goes from 0% to 100%. It starts at $E[DDUP]$, as defined in Section 2.1, and searches in increments of $E[D]$ until the 50% point is crossed. The re-order point/order-up-to level at which this occurs is used as a basis for the simulation range.

```
%Find point where P = 50%
Stest = EDDUP; %Stest is updated towards the optimum
P = SEIO_func(Stest); %Find initial P guess
if P > 50 %Iterate towards 50% by steps of ED
    while P > 50
        Stest = Stest - ED;
        P = SEIO_func(Stest);
    end
    Stest = Stest + ED;
else
    while P < 50
        Stest = Stest + ED;
        P = SEIO_func(Stest);
```

```
    end
end
```

Listing 3.10: Roughly estimate 50% point

3.3.2 Simulate relevant safety stock range

The next part creates vectors for the simulations within a range around the 50% point and performs simulations in that range. A wide range from -1 to 2 times $E[DDUP]$ is chosen to ensure that simulations are performed around the service level requirement, which is usually between 80% and 98%.

```
%Create grid for curve fitting area
Xvalues = ...
    linspace(Stest-ED*(EL+type*R)*PD,Stest+3*ED*(EL+type*R)*PD,Nintervals)'; ...
    %Nintervals around 50% point
Yvalues = zeros(size(Xvalues,1),1);

%Find results within grid
for j = 1:Nintervals
    Yvalues(j) = SEIO_func(Xvalues(j));
end
```

Listing 3.11: Simulate around 50% point

3.3.3 Curve-fitting on simulation results

In the next part, Listing 3.12, the results are processed. The results are service level values for a range of safety stock values. These have to be transformed into a model that can be used to determine the optimal value. This is accomplished using non-linear least squares curve-fitting as described in Section 2.4. A function called the logistic function is fitted on the results of the simulations. The function that is fitted to the data is:

$$P_x = \frac{100\%}{1 + e^{\alpha(S - S_{50\%})}},$$

where P_x can be either P_1 or P_2, α is a fitting coefficient that determines the slope, and S is the re-order point/order-up-to level (x-axis) with $S_{50\%}$ a fitting coefficient with the value where the result is 50%. Among several functions that can be used to describe an S-curve, this function follows the simulation results most accurately. The main alternative was the cumulative distribution function of the Normal distribution, which follows simulation results slightly less accurately. With a small number of simulations, an accurate fit is achieved. A sample of the fitting results can be seen in Figure 3.3, where the simulation result points are shown together with the fitted line.

Since the logistic function is monotonically increasing and there are no constraints except for the service level target, the target with accompanying safety stock can be found using an `fzero` command, which searches along a line until the result is found. The confidence interval of the service level result around the resulting safety stock value is calculated. At this point the optimal safety stock and the service level confidence interval are returned as results of the optimization script.

```
%Fit results to logistic function and
linfun = fit(Xvalues,Yvalues,'100/(1+exp(b*(x-a)))','StartPoint',[Stest ...
    0.001]);
objective = @(x) linfun(x) - Ptarget;
Stest = fzero(objective,EDDUP); %Find optimal S value
ci = predint(linfun,Stest,0.95,'functional'); %Find confidence interval ...
    for optimum
```

Listing 3.12: Fit logistic function and determine optimum

The script that calls the optimization script performs simulations using `SEIO_func` to determine whether the results from fitting are corrects. If results are too low, the optimal safety stock is increased by increments of $E[D]$ until the true optimum is found. If results are too high, the same happens in decreasing direction.

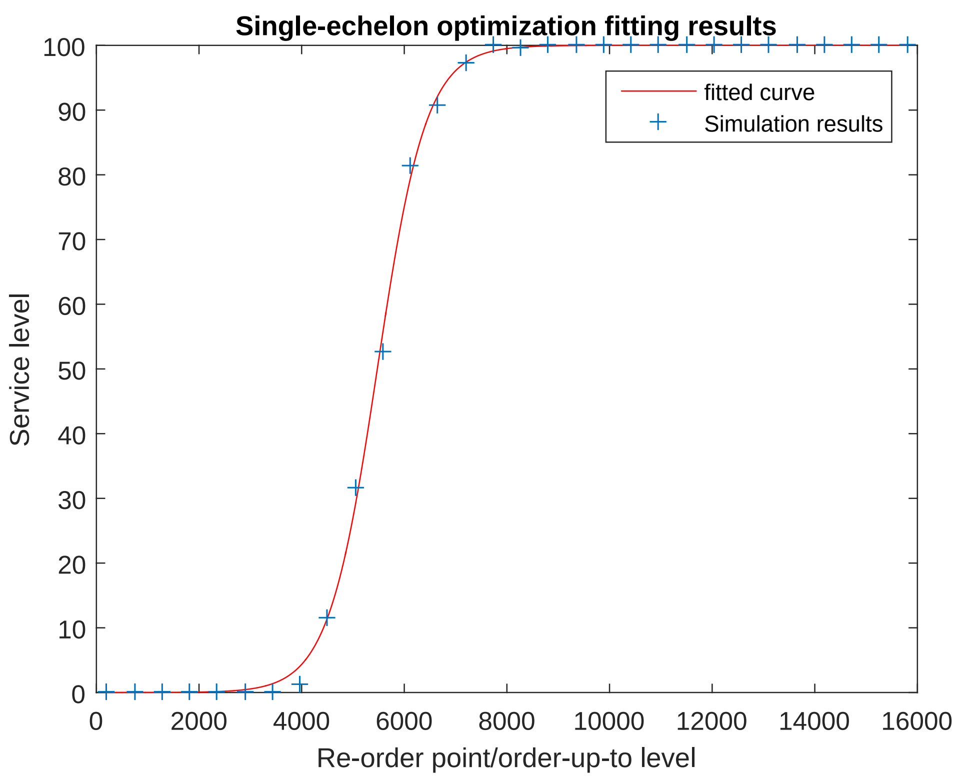


Figure 3.3: Fit result and original data

3.4 Validation of single-echelon results

To validate the single-echelon discrete-time model, the simulation results and the analytical equations can be compared. Before doing so, some possible causes of errors need to be taken into account:

- Due to the discrete timesteps in simulation, demand and lead time are discretized as well. Occasionally, a stockout that might have occurred if demand and lead time were continuous is prevented. If a lot of demand would have occurred near the start of a time interval while the a replenishment would have arrived too late, a stockout should occur, but since the total demand for a time period and the total replenishments for a time period are checked after the time interval, the stockout is not counted. This raises service level results.
- If σ_L is larger than approximately 1/3 of R, delays in replenishments occur due to the prevention of replenishment overtaking. Allowing overtaking would lead to incorrect results, because the effective lead time of some shipments would decrease. This is prevented by only allowing the oldest replenishment to be received. If a recently ordered replenishment has a lower lead time than an older one, it is prevented from being received until the older one is received.
- The equations used in analytical calculation assume that demand is distributed according to a Gamma distribution. If $P(D) < 1$, the demand pattern in the simulation becomes a Gamma distribution with Geometrically distributed zeros in between. To calculate analytical results in these cases, the expected value and standard deviation of demand are recalculated including the time periods containing 0 demand. The new $E(D)$ and σ_D are used in the equations that assume Gamma distributed demand, while demand does not follow an exact Gamma distribution.

To assess the size of these errors and check the accuracy of the simulation, a comparison with the analytical equations is made, where different sets of input parameters are checked. The analytical equations are calculated exactly following the description in Section 2.1.

3.4.1 Method of comparison

To provide a broad comparison of the analytical and discrete-time method, different sets of input parameters are used. A starting set of parameters is defined from which different variables are varied in order to see where differences are larger and where they are smaller. The initial input parameters are shown in Figure 3.4.

Using these parameters as a basis, comparative diagrams and tables are made of the results for different changes in parameters:

- All changes are compared for both the (R, s, Q) and (R, S) policy, if applicable.
- A visual comparison is made for sets of input parameters that represent the demand pattern quadrants specified in Section 2.1.2. The analytical results are added to the simulation results using the script in Appendix A.1.
- A comparison is made for (R, s, Q) without taking undershoot into account.
- The effect of simulating with smaller time intervals is tested.

- Several values of σ_L are tested.
- The service level requirement is changed.
- Order quantity is changed for a P_2 requirement with an (R, s, Q) policy.

type (0=RsQ, 1=RS)	0
E(D)	100
sigma(D)	10
P(D)	1
E(L)	6
sigma(L)	2
R	7
Q factor (*ED*R)	1
P1 or P2?	1
Service level lower bound	95

Figure 3.4: Initial input parameters

3.4.2 Results comparison

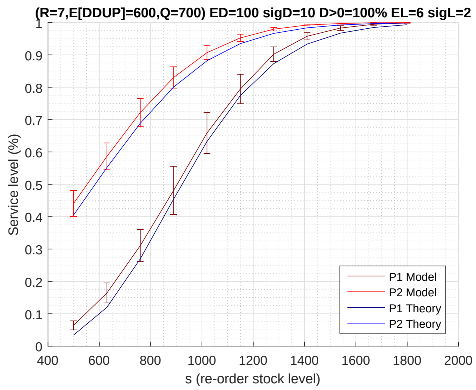
3.4.3 4 Quadrant comparison

The first comparison is done by simulating in the quadrants of demand as described in Section 2.1.2. The initial set of parameters in Figure 3.4 is used as smooth demand. For erratic demand and lumpy demand $\sigma_D = 100$ is used, so that $COV = 1$, and for intermittent demand an lumpy demand $P(D) = 0.1$ is used. To provide a general overview of model vs. equation performance, results are shown graphically in Figure 3.5 for the (R, s, Q) policy and in Figure 3.6 for the (R, S) policy. Furthermore, the optimal safety stock results for $P_1 = 95\%$ can be found in Table 3.1. For each figure or table, similarities and differences are discussed.

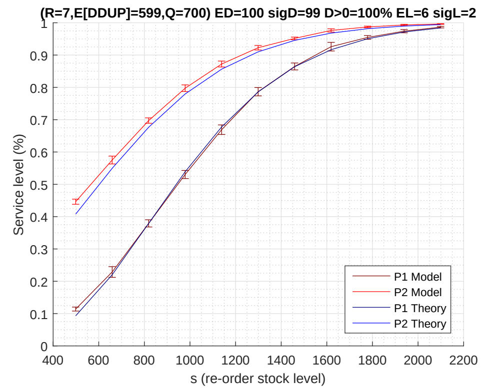
Figure 3.5a shows the results for an (R, s, Q) replenishment policy with a smooth demand pattern, the 95% confidence interval of the simulated safety stocks is also shown below and above the line. The positive bias in service level that was explained at the start of this section is clearly present in this case, although in the 30% to 90% range for P_1 the analytical result is still within the simulation confidence interval. In the higher service level range, which is more relevant than the lower range, differences in safety stock become larger as the curve becomes more horizontal.

In case of erratic demand as shown in Figure 3.5b, P_1 results are almost equal while the positive bias is still present in P_2. Intermittent and lumpy demand in Figures 3.5c and 3.5d seem accurate, even though the equations assume differently distributed demand, as explained at the start of this section. Intermittent demand results fluctuate in accuracy. In high service level ranges the same trend as in the other quadrants can be observed.

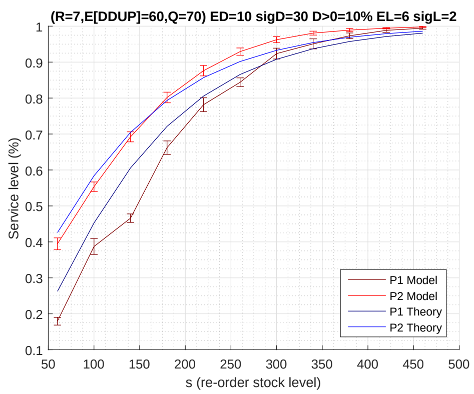
Compared to the (R, s, Q) results, the smooth quadrant (R, S) results in Figure 3.6a are almost the opposite. Between 30% and 90% P_1 service, there are large differences, while the



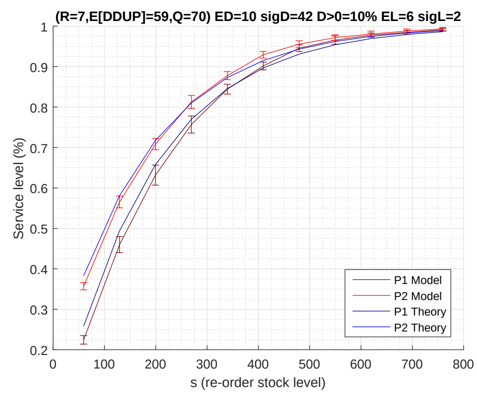
(a) Smooth demand



(b) Erratic demand



(c) Intermittent demand



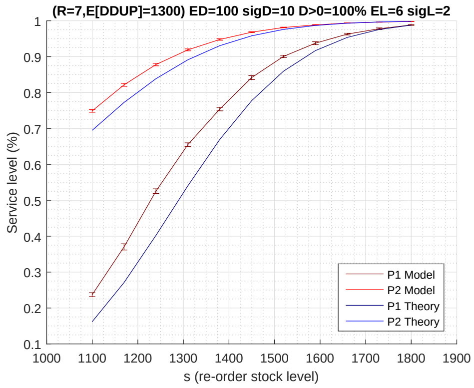
(d) Lumpy demand

Figure 3.5: (R, s, Q) 4 quadrant results

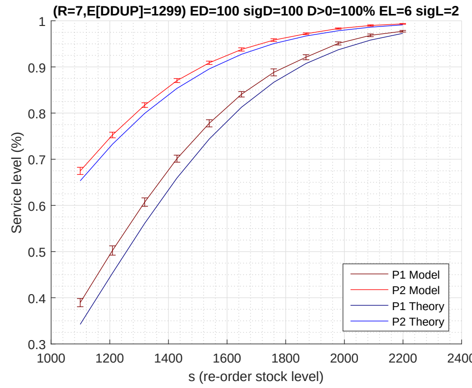
highest service levels seem the most accurate. This might be due to the fixed order quantity in (R, s, Q) models, which is the main difference between (R, s, Q) and (R, S). The same effect can be seen in Figure 3.6b for erratic demand.

For intermittent and lumpy demand in Figures 3.6c and 3.6d, it can be seen that the simulation has a lower P_2 service level than P_1 for most safety stocks. This is generally not the case, but might be caused by the large fluctuations that can occur in demand during the uncertainty period. The Geometric distribution that is used to generate demand intervals create stockouts early in the uncertainty period, which can cause a lot of demand in that period to be backordered. Only 1 stockout can occur per replenishment, while the demand of multiple time intervals can be backordered in the same replenishment period. This effect is not taken into account in the analytical equations, since they assume Gamma distributed demand for every time period, which causes large differences in P_2 while differences in P_1 are smaller.

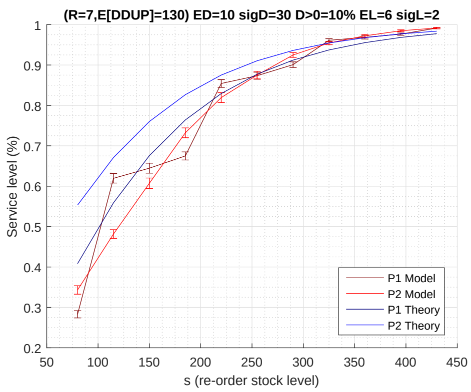
Table 3.1 shows the results of optimization for the input parameters described for each quadrant where the optimum is found for $P_1 = 95\%$. For (R, s, Q), differences have a small positive bias for the simulation, except for erratic demand, where the results are very accurate. This



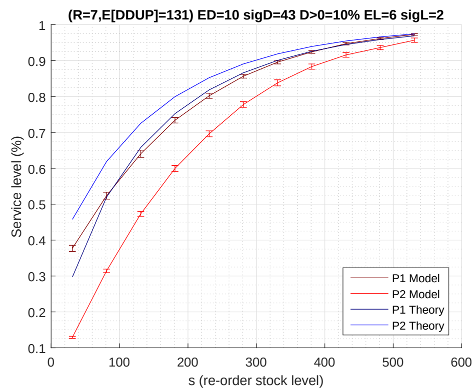
(a) Smooth demand



(b) Erratic demand



(c) Intermittent demand



(d) Lumpy demand

Figure 3.6: (R, S) 4 quadrant results

corresponds to the results discussed for Figure 3.5. For (R, S), differences for smooth and erratic demand are relatively larger.

	$SS_{(R,s,Q)}$		$SS_{(R,S)}$	
$P_1 = 95\%$	Calc	Sim	Calc	Sim
Smooth	864	761	351	318
Erratic	1185	1192	734	696
Intermittent	304	275	221	184
Lumpy	469	454	321	310

Table 3.1: Comparison of optimal safety stock per qudrant

Undershoot

Undershoot is sometimes excluded in calculations, because in practice the strict guidelines of an (R, s, Q) policy are often stretched, orders are already placed if the inventory position is still slightly above the re-order point at the moment it is reviewed, and if it is below the re-order point, the order quantity can be increased. The discrete-time model follows the policy exactly. To compare differences, the additions from Section 2.1.1 are left out of the calculations for the initial set of input parameters. Results can be seen in Table 3.2. There is a large difference in results, this should be taken into account when choosing to calculate safety stock without undershoot.

	$SS_{(R,s,Q)}$	
	Calc	Sim
With undershoot	864	761
Without undershoot	365	761

Table 3.2: Comparison of optimal safety stock without undershoot

Simulation scale

As explained in Section 3.2, the positive bias caused by discretization of demand can be decreased by making the discrete time intervals smaller. The simulation scale can be adjusted to accomplish this. In Table 3.3 it can be seen that for higher scales, the difference becomes smaller. However, increasing the scale also increases computational load.

	$SS_{(R,s,Q)}$		$SS_{(R,S)}$	
$P_1 = 95\%$	Calc	Sim	Calc	Sim
Scale 1	864	761	351	318
Scale 2	864	778	351	320
Scale 5	864	780	351	326
Scale 20	864	811	351	330

Table 3.3: Comparison of optimal safety stock for different simulation scale

Lead time variance

As described at the start of this section, a bias occurs when σ_L increases towards R, this is validated in Table 3.4. The simulation safety stocks increase relative to the analytical results, becoming similar for $\sigma_L = 4$ and significantly higher for $\sigma_L = 8$ compared to analytical equations. This way the bias caused by discrete demand and the bias caused by replenishment overtaking prevention coincidentally can negate each other.

	$SS_{(R,s,Q)}$		$SS_{(R,S)}$	
$P_1 = 95\%$	Calc	Sim	Calc	Sim
$\sigma_L = 2$	864	761	351	318
$\sigma_L = 4$	1189	1205	722	804
$\sigma_L = 8$	1980	2578	1532	2249

Table 3.4: Comparison of optimal safety stock for different lead time variance ($R = 7$)

Service level requirement

The results are compared for different service level requirements in Table 3.5. For (R, S) is seems like higher service levels produce more accurate results, while the opposite is true for (R, s, Q). This corresponds to the results from Figures 3.5 and 3.6.

	$SS_{(R,s,Q)}$		$SS_{(R,S)}$	
	Calc	Sim	Calc	Sim
$P_1 = 80\%$	579	519	167	114
$P_1 = 95\%$	864	761	351	318
$P_1 = 98\%$	1025	895	450	444
$P_2 = 80\%$	289	231	-102	-156
$P_2 = 95\%$	603	523	126	88
$P_2 = 98\%$	775	657	242	212

Table 3.5: Comparison of optimal safety stock for different service level requirements

This chapter describes the conceptual and Matlab model of the single-echelon discrete-time simulation. Furthermore, the version of this model that is used to find optimal safety stock values for various inputs is described. Finally, the single-echelon results are compared to the analytical equations from Chapter 2. The next step is to make the transition from single- to multi-echelon.

Chapter 4

Multi-echelon Matlab model

In the previous chapter a discrete-time model for a single DC was presented. That model had unlimited supply (although with a lead time) and 1 input (safety stock) with 1 output (service level). Using this model it was possible to determine the optimal safety stock for the single-echelon case. However, the supplier of the DC usually does not have a service level of 100%. In most cases an LDC is supplied by a CDC, which both have safety stock and service levels. Since the relevant performance indicator in distribution networks is service level towards the customer, instead of calculating the safety stock for echelons individually, it is possible to take multiple echelons of safety stock into account when determining the minimal safety stock to fulfill service level requirements.

This chapter presents a discrete-time model for multi-echelon distribution systems. The single-echelon model is expanded, and able to simulate networks of 1 CDC with N LDCs. To find the minimal cumulative safety stock of all DCs, a model is fitted to simulation results, which is used in constrained nonlinear optimization. The minimal total safety stock that fulfills LDC confidence interval requirements is determined. In the last section, resulting optima are compared to analytical approximations.

4.1 Conceptual model

The single-echelon model needs to be expanded to multi-echelon, this causes conceptual changes that need to be addressed. The supply to the CDC and the demand fulfillment of the LDC are similar to the single-echelon model, the big change from single-echelon is the connection between CDC and LDC. Demand for the CDC is generated by the replenishment policy of LDCs, this causes two problems:

- When the CDC has insufficient stock to meet the demand of LDCs, it needs to be divided between LDCs.
- In the above case, backorders are created at the CDC, these need to be sent to the LDCs that did not get their demand fulfilled as soon as a new replenishment arrives.

The solution to these problems is addressed in this section.

4.1.1 Specification of model variables

The input and output variables are mostly equal to the ones mentioned in Section 3.1.1. There are a couple of differences:

- R, Q, $E[D]$, σ_D, $P(D)$, $E[L]$, σ_L, SS, P_1 and P_2 have a separate value for each DC. The DC they belong to is added as a suffix, e.g. R_{CDC} or $E[L_{LDC_2}]$.
- $E[D_{CDC}]$, $\sigma_{D,CDC}$ and $P(D_{CDC})$ are based on the replenishment orders of the LDCs.

- Since LDCs are supplied by the CDC, SS_{CDC} indirectly influences LDC service levels. If CDC service levels are low, LDCs will experience delays in replenishment, which can subsequently decrease LDC service levels.
- The single-echelon model used backorders only to calculate P_2, in the multi-echelon model backorders from CDC to LDC still need to be sent to LDCs as soon as new stock is available. B_{LDC_n} is defined as the number of backorders from the CDC to each LDC.

Due to the differences between single- and multi-echelon an extra step needs to be implemented for handling the (back)orders from LDC to CDC. This is elaborated hereafter.

4.1.2 Conceptual model overview

Expanding from to multi-echelon brings changes to the original concept, which can be seen in Figure 4.1. Two new steps are added, and existing steps need to be done for each DC. Changes are shown in **bold** font.

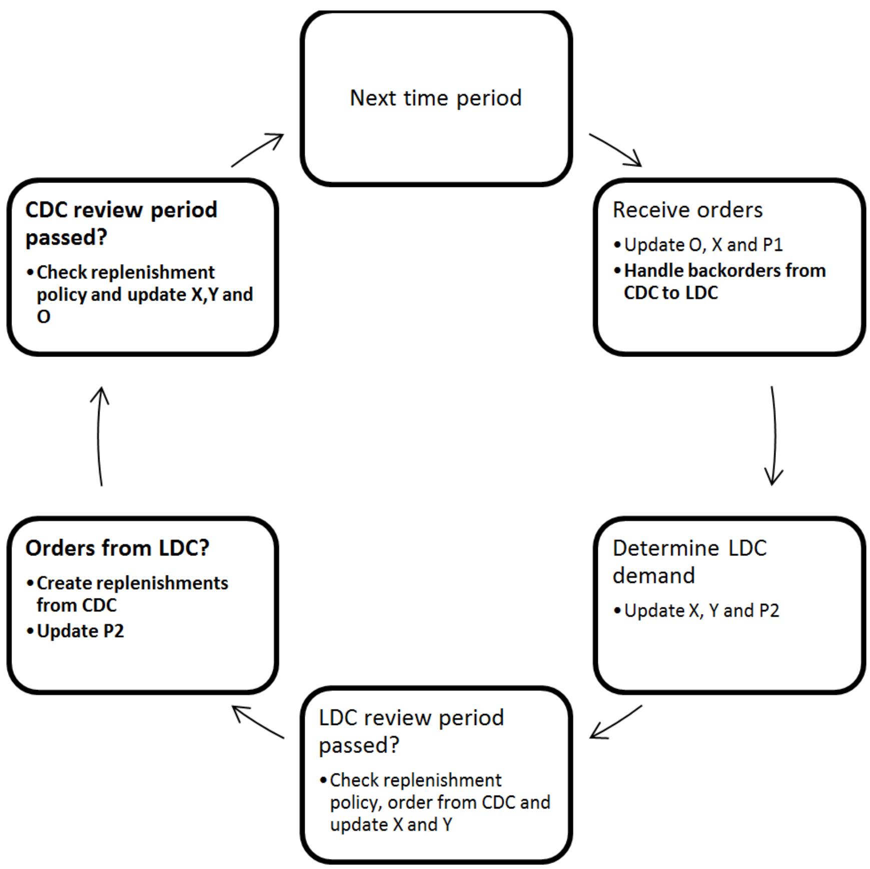


Figure 4.1: Flowchart of communication in multi-echelon model

The first three steps of each time period are similar to the original single-echelon steps. Orders are received if necessary, demand is subtracted from inventory level and replenishment orders are made when necessary. All of this happens for all LDCs at the same time. However, instead of just adding a replenishment order to the order list, these orders need to be supplied by the CDC. Therefore, after calculation LDC replenishment orders the CDC checks whether it has enough stock to satisfy all orders. If it does, the orders are sent to the LDCs, if it has insufficient stock, two things happen:

- The shortage of stock becomes a backorder for the ordering LDC, to be sent when a new replenishment arrives at the CDC.
- The remaining stock needs to be divided between the ordering LDCs, depending on the sharing policy. This can either be done by "fair sharing", where every LDC gets an equal percentage of its order, or by priority, which means each LDC has a rank, and higher ranked LDCs receive their complete order, at the cost of lower ranked LDC orders.

Backorders need to be kept track of, so that when a replenishment order is received by the CDC in the first step of a time period, the received products are sent to the LDCs with outstanding backorders according to the sharing policy.

The CDC replenishment policy is checked separately, because the demand from LDCs, which is determined by checking their replenishment policy, has to be subtracted from the CDC inventory level before the CDC order quantity can be determined.

The changes proposed in this section are implemented in a pseudocode model based on the single-echelon pseudocode model. It is shown in Listing 4.1. Compared to the single-echelon model, the largest additions are sending backorders from CDC to LDC and processing LDC demand in the CDC. The next section contains a detailed description of the effects these changes on the Matlab model.

```
initialize
for each experiment
    for each time period
        %% Receive replenishment orders
        for each DC
            if orders in order list
                if order is ready to be received
                    update DC P1 parameters and add order to net stock
                end
            end
        end
        %% Handle backorders
        if CDC just received an order and has backorders
            send backorders to LDCs
        end
        %% Process LDC demand and replenishment
        subtract LDC demand and update LDC P2 parameters
        for each LDC
            if review period passed
                if replenishment policy satisfied
                    place order at CDC
                end
                add to inventory position
            end
```

```
        end
        %% Process CDC demand and replenishment
        if orders from any LDC to CDC
            if CDC stockout
                update CDC P2 parameters and save backorders
            elseif sufficient CDC stock
                for each LDC
                    if LDC ordered from CDC
                        add order to LDC order list
                    end
                end
            else
                if fair share policy
                    place equal percentage of order on each LDC order list
                    subtract LDC demand from CDC net stock and update P2
                elseif priority policy
                    for each LDC
                        place orders on LDC order list based on priority
                    end
                end
            end
            subtract LDC demand from CDC inventory position and net stock
        end
        if CDC review period passed
            if replenishment policy satisfied
                add order to CDC order list
            end
        end
    end
    for each DC
        calculate final service levels
    end
end
determine mean and confidence interval of service levels and make plots
```

Listing 4.1: Pseudocode model overview

4.2 Matlab model

The multi-echelon discrete-time Matlab model is an extension of the single-echelon model. The conceptual changes discussed in the previous section are implemented in the single-echelon Matlab model, the full script is shown in Appendix B. The details of new sections in the Matlab script are explained in this section.

4.2.1 Parameters for multiple DCs

The structure of the script and variables is changed to accommodate multiple DCs. An example of the input parameters that the model needs to process can be seen in Figure 4.2, for each parameter that has a single value in the single-echelon model, there now is a row of values for each DC. The first column contains CDC data, which is calculated partially from LDC data, $E[D_{CDC}]$, $\sigma_{D,CDC}$ and $P(D_{CDC})$ are calculated from LDC demand, but these are mostly used as a reference, since the CDC demand in the model is actually generated by replenishment orders of LDCs. Every column of data that is added to the Excel sheet represents as an extra LDC. The order of the columns of LDC data also determines the order of LDC priority, in case of a priority sharing policy. The LDC ranking goes from left to right, meaning the data in column C contains the highest priority LDC and each subsequent column represents a lower rank.

	A	B	C	D	E	F
1	type (0=RsQ,1=RS)	1	1	1	1	1
2	E(D)	400	100	100	100	100
3	sigma(D)	100	50	50	50	50
4	P(D)	1	1	1	1	1
5	E(L)	40	5	5	5	5
6	sigma(L)	8	1	1	1	1
7	R	28	7	7	7	7
8	Q	1,1	1,1	1,1	1,1	1,1
9	SS	2000	500	500	500	500

Figure 4.2: Example of excel input

The first changes that are made to the Matlab model to handle a CDC and N LDCs, each having input parameters and time-sensitive variables, are:

- All input parameters are changed to vectors, e.g. $E[D] = (400, 100, 100, 100, 100)$. The first value in vectors with a value for all DCs always is the CDC value, the second value is the value for the first LDC, the third for the second LDC, going on until the final LDC.
- Variables that were already stored in vectors are changed into matrices, e.g. the vector that contains the LDC demand values generated for each time period becomes a matrix with each column representing the demand values for one LDC.
- The order list matrix becomes a set of matrices that each represent the order list of a DC. Since order matrix length changes by number of orders, as explained in Section 3.2.2, these matrices are stored in a cell. A cell can handle multiple matrices of varying sizes.
- B_{LDC_n} and D_{CDC} are defined as vectors containing the backorders to LDCs that still need to be sent and the demand of LDCs to the CDC.

Using these multi-DC variables, the steps taken in the single-echelon Matlab model can be changed to multi-echelon.

4.2.2 Adjustment of single-echelon steps

The single-echelon steps of receiving replenishment orders, handling customer demand and processing replenishment policy are done in a similar manner for each LDC. The same steps are taken in a `for`-loop that loops over each (L)DC. An example of this can be seen in Listing 4.4, where customer demand is processed. Demand for each LDC is determined from the demand matrix `seed`, total backorders and total demand are updated to calculate P_2 after simulation, and the inventory position `Y` and inventory level `X` are updated.

```
        %Process demand for LDC's
        for z = 1:Ndc-1
            D(z) = seed(i,xper,z); %Read from demand matrix
            Dtot(z+1) = Dtot(z+1) + D(z)*start; %For P2
            %Update values for P2
            if X(z+1) < 0
                Btot(z+1) = Btot(z+1) + D(z)*start;
            else
                if X(z+1) < D(z)
                    Btot(z+1) = Btot(z+1) + (D(z) - X(z+1))*start;
                end
            end
        end
        %Update inventory levels and record data for visualization
        Y(2:end) = Y(2:end) - D;
        X(2:end) = X(2:end) - D;
```

Listing 4.2: LDC demand processing

4.2.3 LDC replenishment order processing

The largest addition to the multi-echelon script is where the LDCs places replenishment orders at the CDC and the CDC divides its stock among those orders. The replenishment orders are determined first in Listing 4.3. The passing of the review period is checked for each LDC separately, since review periods might differ. In the single-echelon model, replenishment orders were added to the order list immediately. Now, they are saved in the vector `Dcdc` which represents demand from LDCs to the CDC. Orders are then added to the inventory position of the LDCs and to the total demand of the CDC.

```
        %After each review period update orders, LDC's first
        r = r + 1;
        for z = 2:Ndc
            if r(z) ≥ R(z)
                if type(z) == 1 %Order up to S for (R,S)
                    Dcdc(z-1) = S(z)-Y(z);
                elseif Y(z) < S(z) %Order Q for (R,s,Q)
                    Dcdc(z-1) = ceil((S(z)-Y(z))/Q(z))*Q(z);
                else
                    Dcdc(z-1) = 0;
                end
                Y(z) = Y(z) + Dcdc(z-1);
```

```
                Dtot(1) = Dtot(1) + Dcdc(z-1)*start;
                r(z) = 0;
            end
        end
```

Listing 4.3: LDC replenishment

Next, the LDC replenishment orders are processed by the CDC in Listing 4.4, which represents lines 27-47 of the pseudocode in Listing 4.1. If there are orders by LDCs, the model performs the following steps:

- Check whether CDC stock is empty, if so, add all LDC demand to the backorder vector `Bcdc`, which contains the number of backorders to be sent to each LDC. Furthermore, the total number of backorders is updated for calculation of $P_{2,CDC}$ and the inventory level is updated.
- If the CDC is not empty, and stock is sufficient to fulfill all LDC demand, add replenishment orders to the order list matrices of the LDCs that ordered replenishments and update CDC inventory level.
- If the CDC has insufficient stock, but is not empty, the remaining stock has to be divided between all ordering LDCs based on rationing policy:
  - If the fair share policy is chosen, each ordering LDC receives an equal percentage of its order. These orders are added to the order list matrices of ordering LDCs, the shortage is added to the backorder vector.
  - If the priority policy is used, each LDC order is fulfilled by LDC rank, as long as stock remains. When stock is insufficient to fulfill the demand of the LDC that is currently being processed, the remaining stock goes to that LDC, all lower ranking LDCs receive nothing.
  - For this step CDC inventory level is updated per LDC that is being processed, so that remaining CDC stock can be checked for each LDC individually.
- Finally, CDC inventory position is updated and the vector of demand from LDCs is emptied.

```
        if max(Dcdc) > 0 %Check orders from LDC to CDC
            if X(1) ≤ 0 %In case of stockout everything backorders
                if Tempty == 0
                    Tempty = i;
                end
                Btot(1) = Btot(1) + sum(Dcdc)*start;
                for z = 1:Ndc-1
                    Bcdc(z) = Bcdc(z) + Dcdc(z);
                end
                X(1) = X(1) - sum(Dcdc);
            elseif X(1) ≥ sum(Dcdc) %If stock is sufficient everyting is sent
                for z = 2:Ndc
                    if Dcdc(z-1) > 0
                        L = gamrnd((EL(z)/sigL(z))^2,sigL(z)^2/EL(z));
                        O{z} = [O{z};Dcdc(z-1) L];
```

```
                    end
                end
                X(1) = X(1) - sum(Dcdc);
            else%When no stockout but also insufficient stock
                if Tempty == 0
                    Tempty = i;
                end
                if fairshare == 1
                    Opart = X(1)/sum(Dcdc);
                    for z = 2:Ndc %Send partial order if fair share
                        if Dcdc(z-1) > 0
                            L = gamrnd((EL(z)/sigL(z))^2,sigL(z)^2/EL(z));
                            O{z} = [O{z};Opart*Dcdc(z-1) L];
                            Bcdc(z-1) = Bcdc(z-1) + (1-Opart)*Dcdc(z-1);
                            Btot(1) = Btot(1) + (1-Opart)*Dcdc(z-1)*start;
                        end
                        X(1) = X(1) - Dcdc(z-1);
                    end
                else
                    for z = 2:Ndc %Check DC's in order
                        if Dcdc(z-1) > 0 && X(1) ≥ Dcdc(z-1) %Send if ...
                            sufficient
                            L = gamrnd((EL(z)/sigL(z))^2,sigL(z)^2/EL(z));
                            O{z} = [O{z};Dcdc(z-1) L];
                        elseif X(1) ≤ 0 %Backorder when finished
                            Bcdc(z-1) = Bcdc(z-1) + Dcdc(z-1);
                            Btot(1) = Btot(1) + Dcdc(z-1)*start;
                        elseif Dcdc(z-1) > 0 %Else send remaining stock
                            L = gamrnd((EL(z)/sigL(z))^2,sigL(z)^2/EL(z));
                            O{z} = [O{z};X(1) L];
                            Bcdc(z-1) = Bcdc(z-1) + Dcdc(z-1) - X(1);
                            Btot(1) = Btot(1) + (Dcdc(z-1) - X(1))*start;
                        end
                        X(1) = X(1) - Dcdc(z-1);
                    end
                end
            end
            Y(1) = Y(1) - sum(Dcdc);
            Dcdc = zeros(Ndc-1,1);
        end
```

Listing 4.4: CDC to LDC shipment

4.2.4 Fulfilling LDC backorders

Since backorders from the CDC still need to be sent to the LDCs when available, Listing 4.5 runs after the CDC receives a replenishment. When a replenishment is received by the CDC, `Qcdc` is set to the number of products in that replenishment in the section where orders are received. The steps taken are similar to the previous part, in the case that there was insufficient but also not empty stock. For the fair share policy, everything is divided evenly, if the replenishment that was received is insufficient to fulfill all backorders. For the priority policy backorders are sent as long as the replenishment suffices.

```
        %Send backorders to LDC's
        if Qcdc > 0
            if fairshare == 1 && sum(Bcdc) > Qcdc
                Opart = Qcdc/sum(Bcdc);
                for z = 2:Ndc
                    L = gamrnd((EL(z)/sigL(z))^2,sigL(z)^2/EL(z));
                    O{z} = [O{z};Opart*Bcdc(z-1) L];
                    Bcdc(z-1) = (1-Opart)*Bcdc(z-1);
                end
            else
                for z = 2:Ndc
                    if Bcdc(z-1) ≤ Qcdc && Bcdc(z-1) > 0 %If there is ...
                        enough send B
                        L = gamrnd((EL(z)/sigL(z))^2,sigL(z)^2/EL(z));
                        O{z} = [O{z};Bcdc(z-1) L];
                        Qcdc = Qcdc - Bcdc(z-1);
                        Bcdc(z-1) = 0;
                    elseif Qcdc > 0 && Bcdc(z-1) > Qcdc %Send remaining Q ...
                        when finished
                        L = gamrnd((EL(z)/sigL(z))^2,sigL(z)^2/EL(z));
                        O{z} = [O{z};Qcdc L];
                        Bcdc(z-1) = Bcdc(z-1) - Qcdc;
                        Qcdc = 0;
                    end
                end
            end
            Qcdc = 0;
        end
```

Listing 4.5: CDC backorder processing

With adjustments discussed in this section, the Matlab model is capable of simulating multi-echelon distribution systems of 1 CDC with a variable number of LDCs, while keeping the same options of demand patterns and replenishment policies used in the single-echelon model.

Furthermore, functionality was added to the multi-echelon model to keep count of the number of time periods between a stockout and a replenishment of the CDC, this represents the delay that LDCs experience due to CDC stock shortages. The delay is discussed further in Section 4.4.

It is still possible to collect data of a single run of the experiment, similar to the single-echelon model. This can provide a detailed view of the changes in inventory during simulation. A diagram is shown in Figure 4.3, which contains details for a simulation with 4 LDCs and an (R, S) policy. This figure shows that inventory fluctuations occur in the same way that they occurred in the single-echelon model, as seen in Figure 3.2. Furthermore, the effect of a CDC stockout on LDC inventory levels can be seen at the inventory level minimum that each DC experiences between time periods 8300 and 8400.

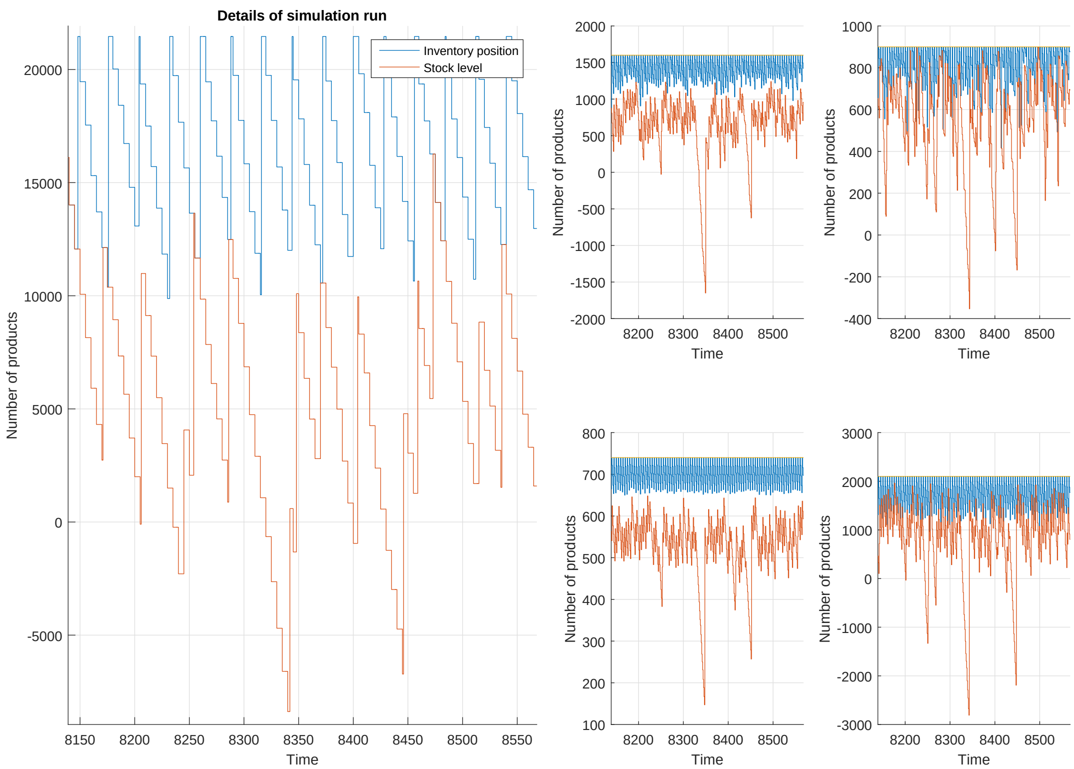


Figure 4.3: Details of multi-echelon simulation results

4.3 Multi-echelon optimization

The multi-echelon discrete-time model can determine service levels for a set of CDC and LDC safety stocks. Although this can be useful, the minimal combined safety stock of all DCs that meets a service level requirement is still unknown. The next step is determining this optimal safety stock configuration by solving the following optimization problem:

$$\begin{aligned} \text{minimize} \;\; & f(SS_{CDC}, SS_{LDC_n}) = SS_{CDC} + \sum_{n=1}^{N_{LDC}} SS_{LDC_n}, \\ \text{subject to} \;\; & g_n = P_{x,LDC_n}(SS_{CDC}, SS_{LDC_n}) \geq SL_{x,LDC_n} \quad \text{for } n = 1, ..., N_{LDC}, \end{aligned}$$

where SS is safety stock, N_{LDC} is the number of LDCs, g_n are inequality constraints, P is the service level corresponding to a combination of CDC and LDC safety stock and SL_{x,LDC_n} are the service level requirements for LDCs based on input parameters.

To solve this optimization problem, a surrogate model is fitted to results of multiple simulations for varying safety stock combinations. Using a surrogate model mediates stochastic errors in simulation results and provides approximations for values between simulated points. This section describes the approach to the surrogate model as well as a Matlab script in which this model is implemented, and the optimization problem is solved.

4.3.1 Surrogate model approach

The surrogate model should represent a relation between CDC and LDC safety stocks, and LDC service levels, based on results from simulations. A change in LDC safety stock only affects the service level of that LDC. It does not affect CDC service level or service levels of other LDCs, these would be affected if the order pattern from an LDC would be dependent on safety stock, which it is not. The amount ordered by an LDC depends on $E[DDUP]$, which is independent of safety stock.

Since LDC service levels are independent of each other, the surrogate model can be split into a separate model for each LDC. The model then should determine LDC service level based on CDC safety stock and LDC safety stock. To find this relation, data from simulations is visualized as a 3D model with CDC safety stock on the x-axis, LDC safety stock on the y-axis and LDC service level on the z-axis. A sample of this data can be seen in Figure 4.4.

A first approach for fitting a surrogate model on this data is a surface model. While testing this the closest fit was found using a triple logistic function, the function that was also used for the single-echelon model. The surface model is based on the same function as the single-echelon model, with shifting slope and 50% point parameters. The logistic function is used again to represent the shift in slope and 50% point:

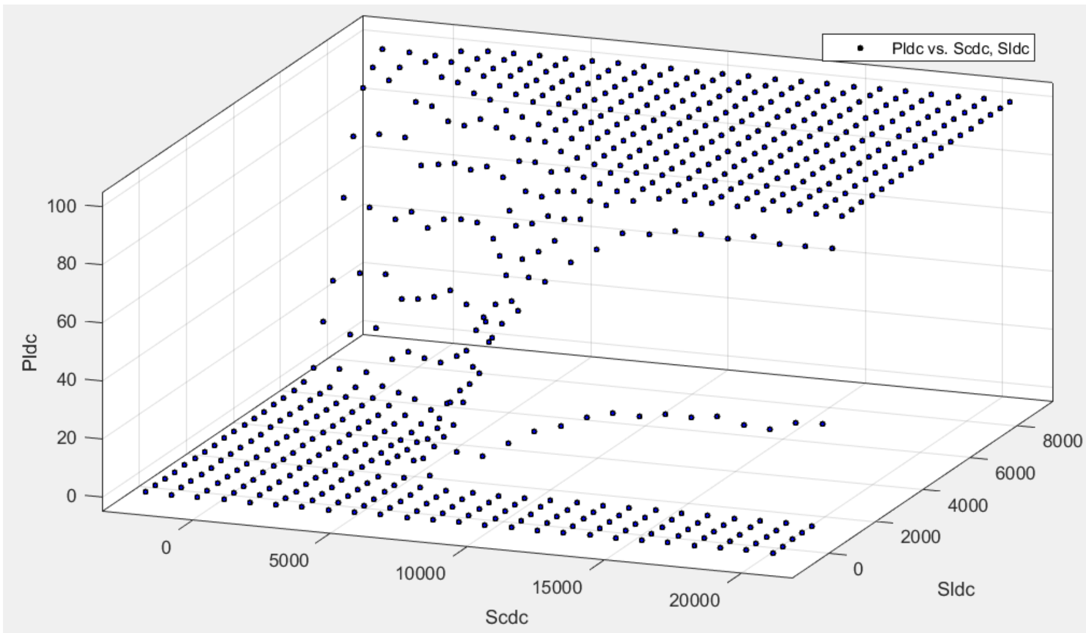


Figure 4.4: Multi-echelon simulation results

$$P_x(S_{LDC_n}, S_{CDC}) = \frac{100\%}{1 + e^{-\alpha(S_{CDC})(S_{LDC_n} - S_{LDC_{50\%}}(S_{CDC}))}},$$

$$\text{with: } \alpha(S_{CDC}) = c_1 + \frac{c_2}{1 + e^{-c_3(S_{CDC} - c_4)}},$$

$$S_{LDC_{50\%}}(S_{CDC}) = c_5 + \frac{c_6}{1 + e^{-c_7(S_{CDC} - c_8)}},$$

where c_i are fitting coefficients, which can be roughly estimated using input parameters. S represents either the re-order point or the order-up-to level based on replenishment policy. An example of the resulting surface model fit and errors can be seen in Figure 4.5. Unfortunately, the highest error in this surface model occurs in the $70 - 98\%$ range, which is a common range for LDC service levels. To improve accuracy in that area, another approach is formulated.

Sequential curve-fitting approach

Modeling the shifting slope and 50% point parameters in the surface model using more logistic functions seems to be causing inaccuracies. These inaccuracies can be circumvented by determining the LDC safety stock corresponding to the service level requirement directly for each CDC safety stock value. By looking at LDC safety stock vs. service level separately for each CDC safety stock, fitting can be done using the same approach as described in Section 3.3. The resulting points each represent the LDC safety stock corresponding to the LDC service level requirement for a CDC safety stock. These results are shown in Figure 4.6. This figure also shows a curve fitted on the resulting points, contrary to the previously fitted curves, the Normal cumulative distribution function has a closer fit here, although the difference is very small.

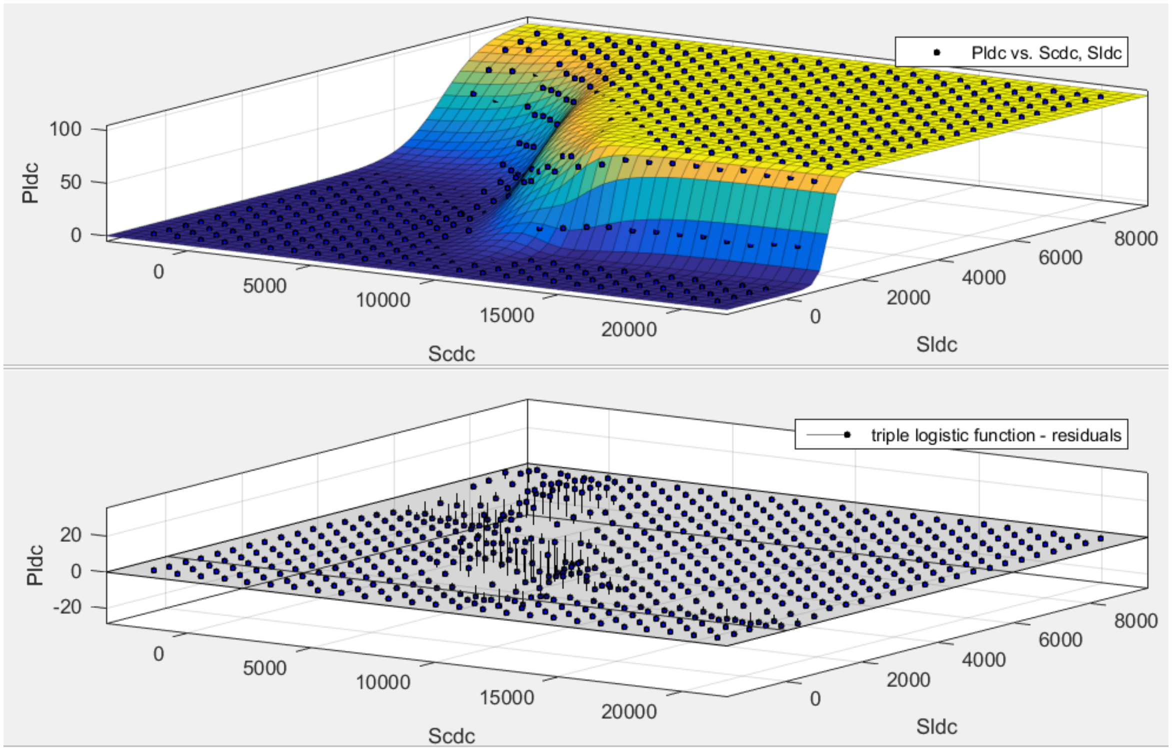


Figure 4.5: Multi-echelon surface fitting

Figure 4.6: Multi-echelon sequential curve-fitting

The function that is fitted to the data is:

$$S_{LDC_{SL\%}}(S_{CDC}) = c_1 + c_2\Phi\left(\frac{S_{CDC} - c_3}{c_4}\right),$$

$$\text{with: } \Phi(x) \;=\; \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-t^2/2}\,dt.$$

Here, $S_{LDC_{SL\%}}$ is the re-order point/order-up-to level corresponding to the service level requirement of an LDC, c_i are all fitting coefficients. This surrogate model is used for optimization in a script that finds the optimal safety stock configuration.

4.3.2 Optimization script

A Matlab script is made, which uses the multi-echelon discrete-time model and the surrogate model described in this section to determine the optimal safety stock configuration for a set of input parameters. The steps taken in this script are:

1. Use input parameters in single-echelon optimization to determine CDC and LDC safety stock boundaries to simulate within.
2. Perform simulations over a grid of CDC and LDC safety stock intervals within boundaries.
3. Perform sequential curve fitting on simulation results.
4. Find optimal safety stock configuration using fitted curves.
5. Perform simulations to find service level confidence interval of resulting optimum.
6. Compare confidence interval of optimum to requirement using simulation.
7. Find linear fit and perform optimization to determine new optimum.
8. Repeat steps 5, 6 and 7 until service level requirement is within optimum confidence interval.

The script performing these steps is shown in Appendix C. Details on the steps performed in the script are explained in the rest of this section.

4.3.3 Simulation grid and boundaries

The results of the simulations that are performed in this script should contain the CDC and LDC safety stock ranges in which LDC service level goes from 0% to 100%. To find these ranges, the single-echelon optimization function is used for the CDC and twice for each LDC, as can be seen in Listing 4.6.

Based on the input parameters it is not yet clear in which (CDC and LDC) safety stock ranges the multi-echelon simulation should be performed. Since the service level types are known from input, it can be stated that the range should go from 0% to 100% in the required service level type (P_1 or P_2) for CDC and LDCs. It is also known that when the S value of the CDC approaches 0 the lead time parameters of the CDC can be added to those of the LDCs, since at that point a replenishment order from the LDC is ordered by the CDC the moment it comes in, without any stock to buffer. The CDC range can be determined by taking the approximate 0% and 100% points, using the combined LDC demand as input. The LDC ranges by taking the approximate 0% point, and the 100% point for a simulation with added CDC lead time.

The function `SE_fit` is a variation of the single-echelon optimization script in Section 3.3. It returns the safety stocks corresponding to 0.01% and 99.9% service. The safety stock ranges that are simulated are determined by taking the safety stocks from `SE_fit` and determining a range of points between them of `Nintervals` intervals, `Nintervals` is a simulation parameter.

```
%% Determine simulation ranges
%SE_fit returns approximate 0% and 100% safety stocks
CDCbounds = SE_fit(input(:,1));
CDCrange = linspace(round(CDCbounds(1)),round(CDCbounds(2)),Nintervals);
%CDC lead time (variance) is added to LDC to determine upper LDC bounds
CDCLvalues = zeros(size(input,1),1);
CDCLvalues(5) = input(5,1);
CDCLvalues(6) = input(6,1);
LDCranges = cell(Nldc,1);
for i = 2:Ndc %Determine LDC ranges
    output = SE_fit(input(:,i));
    low = round(output(1));
    output = SE_fit(input(:,i)+CDCLvalues);
    high = round(output(2));
    LDCranges{i-1} = linspace(low,high,Nintervals);
end
```

Listing 4.6: Boundary calculation

As explained at the start of this section, LDC service levels are independent. This means that LDC safety stock ranges can be simulated in parallel, so that the grid in which simulations are performed becomes a square of `Nintervals`-by-`Nintervals`.

For each CDC safety stock and for each set of LDC safety stocks per CDC safety stock, a simulation is performed using a script similar to the Matlab model described in the previous section, which can be seen in in Listing C.2. The results are saved in the `Results` matrix, which is built as:

$$\begin{bmatrix} S_{CDC} & S_{LDC_1} & S_{LDC_2} & S_{LDC_n} & \cdots & P_{1,CDC} & P_{2,CDC} & P_{1,LDC_1} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}.$$

where each row contains results from a simulation.

4.3.4 Sequential curve fitting

Using the simulation results in the matrix `Results`, the surrogate model can be fitted. As described at the start of this section, this is done by performing the fitting function from the single-echelon model for each CDC safety stock value. For each LDC, the main script prepares input data for a fitting function in Listing 4.7. It runs through all results for a set of S_{CDC}, S_{LDC} and $P_{x,LDC}$ and gives a higher fitting weight to simulations with a service level between 70% and 99%, to increase fitting accuracy in that range. The function `FindTargetLine`, which is shown in Listing C.3, performs curve fitting as described in Section 4.3.1. The fitting coefficients are stored in matrix `fitvalues`, which is save to a `.mat` file which is used for optimization.

```
%% Fit service level target line for each LDC
fitvalues = zeros(4,Nldc);
Scdc = Results(:,1);
for i = 1:Nldc
    Weigths = ones(size(Results,1),1);
    Sldc = Results(:,i+1);
    Pldc = Results(:,Ndc+2*i+input(9,i+1)); %Choose P1 or P2
    for j = 1:size(Scdc,1)
        if Pldc(j) > 70 && Pldc(j) < 99
            Weigths(j) = Pldc(j)/70*2;%Increase weight in relevant range
        end
    end
    fitvalues(:,i) = ...
        FindTargetLine(Scdc,Sldc,Pldc,Weigths,CDCrange,LDCranges{i},Ptarget(i));
end
```

Listing 4.7: Service level requirement line fitting

4.3.5 Constrained nonlinear optimization

Using the optimal safety stock line determined by sequential curve-fitting for each LDC, a total optimum can now be found using optimization. The optimization problem is defined as:

$$\text{minimize} \;\; f(S_{CDC}, S_{LDC_n}) = S_{CDC} + \sum_{n=1}^{N_{LDC}} S_{LDC_n},$$

$$\text{subject to} \;\; h_n = 0 = S_{LDC} - \left(c_{1,n} + c_{2,n} \Phi \left(\frac{S_{CDC} - c_{3,n}}{c_{4,n}} \right) \right) \quad \text{for } n = 1, ..., N_{LDC}.$$

Here, S is the re-order point (for (R, s, Q)) or the order-up-to level (for (R, S)) of a DC, N_{LDC} is the total number of LDCs, h_n are equality constraints based on fitted data and $c_{i,n}$ are fitting coefficients. This optimization is performed in the main script using `fmincon` as shown in Listing 4.8.

```
%% Initiate optimization
options = optimset('fmincon');
options = optimset(options,'MaxFunEvals',10000,'TolFun',1E-3);
x0 = zeros(1,Ndc);
x0(1) = CDCrange(end); %Start at P=100%
for i = 1:Nldc %Find appropriate LDC values from fits
    x0(i+1) = ...
        fitvalues(1,i)-fitvalues(2,i)*normcdf(x0(1),fitvalues(3,i),fitvalues(4,i));
end
```

```
lb = zeros(1,Ndc);
ub = value; %Last simulation point

%Perform constrained optimization
[CurrOptimum,fval,exitflag,output,lambda,grad]=...
    fmincon(@objfunMEIO,x0,[],[],[],[],lb,ub,@confunMEIOPline,options);

if exitflag < 1 %Stop if no optimum available
    print = 'Optimum not found, pleasy retry or change parameters'
    return
end
```

Listing 4.8: Multi-echelon optimization call

The objective function and constraint function used for this optimization are shown in Listing 4.9 and 4.10.

```
function f = objfunMEIO(x)
% Objective function
f = sum(abs(x)); %Don't go below zero stock
```

Listing 4.9: Multi-echelon optimization objective function

The objective function calculates the sum of the values of S for each DC that are given as input. The constraint function uses the previously saved service level targets and fitting coefficients to construct the equality constraints.

```
function [g,h] = confunMEIOPline(x)
%Load target service levels and fit values
load('Ptarget.mat');
load('fitvalues.mat');
Nldc = size(fitvalues,2);
h=zeros(Nldc,1);
% Constraints
g = [];
%Equality constraints for target service level lines
for i = 1:Nldc
    h(i) = x(i+1) - ...
        (fitvalues(1,i)-fitvalues(2,i)*normcdf(x(1),fitvalues(3,i),fitvalues(4,i)));
end
```

Listing 4.10: Multi-echelon optimization constraint function

The initial guess is at a CDC service level of 100%, so that the optimization looks for the minimum with the highest CDC service level. A lower minimum might be found when looking close to $S_{CDC} = 0$, but this minimum is not preferable as a result. When $S_{CDC} \to 0$, CDC holding costs are no longer being reduced, since holding costs can not decrease below 0, which occurs when the inventory level is at 0.

If the optimization succeeds, the resulting optimum is stored so that it can be checked using simulations.

4.3.6 Local linear optimization

The optimum found using constrained optimization is the optimum based on the fitted functions found by curve-fitting. This might not be the true optimum, since fitting contains errors. Therefore, the found optimum is compared to the service level requirement using the confidence interval of multiple simulations. If the service level requirement is within the confidence interval of those simulations, the optimum is accepted, otherwise a more suitable optimum is searched for using multiple simulations and local linear optimization. The code for this is shown in Listing 4.11. It uses script `MEIO_withCI` (Appendix C) which runs the multi-echelon simulation `Nexp` times for a given safety stock configuration and determines the service level confidence interval.

If the optimum is not yet found, script `ReOptimize` is executed, which can be seen as a small version of the main script. The script can be found in Appendix C. It does the following:

- Determine new simulation points based on the difference between the current optimum and the service level requirements. If the resulting service level is higher than the requirement the percentage difference is subtracted from S_{LDC_n}, if it is lower the percentage is added.
- Simulate using `MEIO_withCI` to prevent stochastic inaccuracy. The results form a 2-by-2 matrix which should be around the service level requirement.
- Fit a linear function on new points. Fitting is done this way since results on small scale are indistinguishable from nonlinear functions like the logistic function.
- Perform optimization to find a new estimate for the optimum. `fmincon` is used in the same way as the main script, with linear equality constraints based on new curve-fitting results.

The loop then starts again by checking the new optimum, and reiterates until it has found a simulation-proven value or until it has taken too many iterations, to prevent an infinite loop. The final optimum is returned when the script finishes.

```
%% Initiate optimum iteration
Pdiff = zeros(1,Nldc); %Difference between requirement and results
finished = 0; %Becomes 1 when final optimum is found
Iterations = 0; %Increases up to a defined maximum
while finished == 0
    %Initiate new results matrix
    RES2 = zeros(4,2*(Ndc)-1);
    x = CurrOptimum; %x is used in simulations in MEIO_withCI
    MEIO_withCI; %Determine Pavg and Pci, the average and CI of x
    RES2(1,:) = [x Pavg];

    %Determine difference between current optimum and service level target
    for i = 1:Nldc
        if Pavg(i)+Pci(i) > Ptarget(i) && Pavg(i)-Pci(i) < Ptarget(i)
            Pdiff(i) = 0; %Count as 0 if within confidence interval
        else
            Pdiff(i) = Ptarget(i)-Pavg(i);
        end
```

```
    end

    %If result not within confidence interval, reiterate
    if not(isequal(Pdiff,zeros(1,Nldc)))
        ReOptimize;
        Iterations = Iterations + 1;
    else
        finished = 1; %exit loop when optimum found
    end
    if Iterations ≥ 10 %Exit if optimum is not found within 10 times
        print = 'Optimum not found, pleasy retry or change parameters'
        return
    end
end
FinalOptimum = CurrOptimum - EDDUP' %Print final safety stock results
```

Listing 4.11: Multi-echelon final optimum iteration

4.4 Validation of multi-echelon results

A multi-echelon discrete-time model and optimization script have been developed. For a distribution system consisting of 1 CDC and N LDCs, an optimal configuration of safety stock can be found. This optimum can be compared to the result of an analytical approximation, which also determines the optimal safety stock configuration. The comparison can be used to validate the discrete-time model and the analytical approximation with each other. This section shows results from both methods, and analyzes possible sources of differences between simulation results and analytical calculation results.

4.4.1 General observations on simulation results

Before comparing analytical and simulation results, a few general remarks about accuracy of the multi-echelon discrete-time model need to be taken into account:

- The discrepancies between single-echelon simulation and analytical calculation that were noted in Section 3.4 also apply to the multi-echelon model.
- When CDC safety stock is increased so that the service level becomes 100%, results for LDCs should be equal to single-echelon results, since it never needs to wait for a replenishment longer than the standard lead time. Simulations have shown that this is indeed what happens in the multi-echelon discrete-time model.
- The discrete-time Matlab model was tested with many different sets of input parameters to check for bugs and inconsistencies. Since simulations are stochastic, multiple simulations with equal input parameters often do not give identical results. Differences in total safety stock are negligible, but the balance between CDC and LDC safety stock sometimes by noticeable amounts, an example of this can be seen later in Table 4.6. Since service levels and the sum of safety stocks do not change significantly, this is not deemed a problem.
- To facilitate the "fair share" policy, review periods of LDCs must end in the same time period. This causes the demand towards the CDC to coincide at intervals of R_{LDC} time periods, while analytical equations assume continuous demand. Replenishments can arrive between demand moments, occasionally preventing a stockout that would have happened if demand would have been subtracted each time period. This causes CDC service to be higher than would be expected for a certain average demand, an example of this is shown in Figure 4.7. This effect is similar to the effect that was noted in the single-echelon model, where discrete demand caused deviations compared to continuous demand.

This influences the accuracy of the results, but should not cause major errors.

4.4.2 Analytical approximation

This section describes how the analytical results, that are used to compare to, are acquired. An Excel tool is being developed at OM Partners that can be used to determine optimal safety stock, and corresponding $P_{1,CDC}$ for a set of input parameters similar to the input parameters of the multi-echelon discrete-time model (Figure 4.2).

The Excel tool calculates the total average inventory for a $P_{1,CDC}$ range from 0% to 100%,

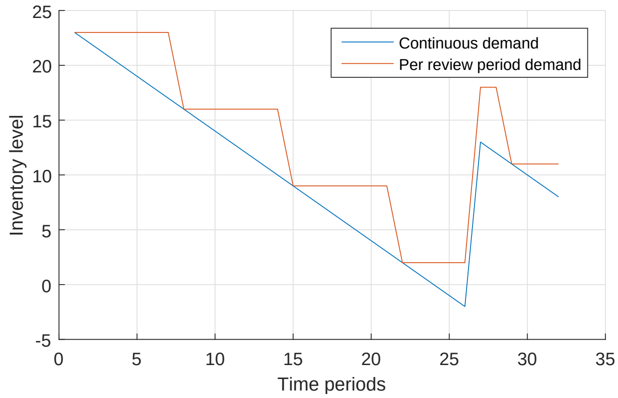


Figure 4.7: Continuous demand vs. Demand per review period

while LDC service levels are kept at the requirement. Average inventory is calculated by adding the safety stock to half of the average order quantity and subtracting the average number of backorders. For each value of $P_{1,CDC}$, the safety stock necessary in the CDC and all LDCs is calculated based on the theories described in Section 2.2.1. From all total average inventory levels the lowest value is picked as the optimum. The resulting value of $P_{1,CDC}$ and safety stock at each DC can be compared to results of discrete-time simulation.

Figure 4.8 shows an example of results of the Excel tool. The x-axis contains the safety factor z, which is the inverse of a standard probability distribution, corresponding to the P_1 of the CDC. Taking the Normal distribution as an example, the z-value for a percentage is determined by calculating the inverse of the standard Normal cumulative distribution function, which is shown in Section 2.2.2.

Increasing in the x-direction corresponds to decreasing $P_{1,CDC}$. The y-axis represents average inventory. As can be seen, a decrease in $P_{1,CDC}$ corresponds to less safety stock in the CDC (blue line), as would be expected from single-echelon results (Section 3.4). The other colors, cumulatively representing average inventory in the LDCs, increase when CDC inventory decreases, to make up for waiting times caused by decreasing CDC service. The optimal safety stock configuration can be seen at $z \approx 0.0$. The $P_{1,CDC}$ and safety stock values corresponding to this z-value are used to compare analytical approximation to discrete-time simulation.

4.4.3 Method of comparison

Optima found using the discrete-time model and optima found using the Excel tool can be compared for various sets of input parameters. To obtain a broad overview of results, ranges are specified for input parameters. Some restrictions apply due to model constraints which are also discussed.

Input parameters were specified in Section 4.1.1. By choosing different values for each parameter and comparing results, it is possible to see the effect of these input parameters on differences between the compared methods. First, a “standard” set of input parameters is

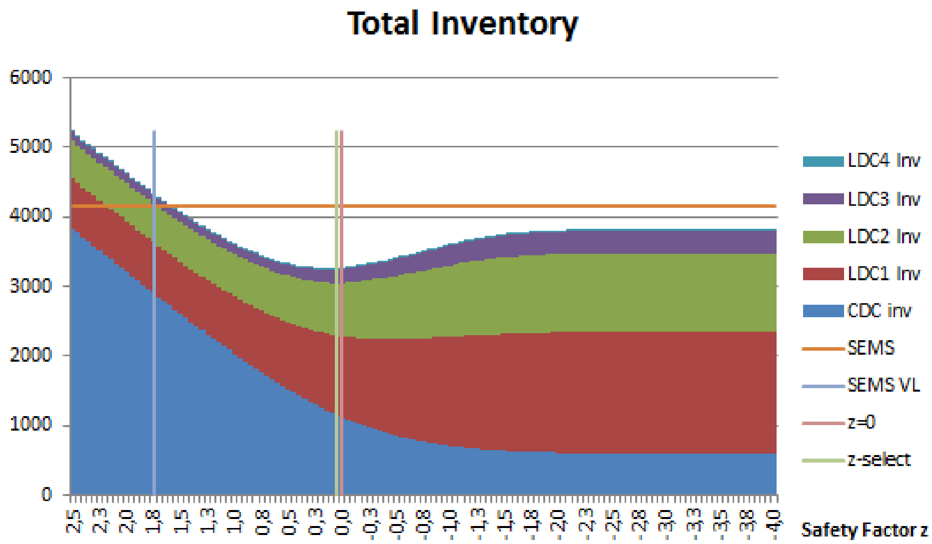


Figure 4.8: Multi-echelon excel results

created as a starting point from which the deviations are made. This set is set up with:

- 1 CDC and 4 LDCs, with differing demand and lead time, but similar replenishment policy, demand variance, and lead time variance, which are changed later.
- Smooth demand for all 4 LDCs, since this is most accurate in single echelon form, with $E(D) = (5, 10, 20, 40)$, $\sigma_D = (0.5, 1, 2, 4)$ and $P(D) = (1, 1, 1, 1)$. These values corresponds to LDCs 1-4.
- Initially, a lead time with low variance is chosen, since high variance also led to errors in the single-echelon model. For the CDC, which has a longer review period, it can be changed later. $E(L) = (20, 12, 10, 8, 6)$, $\sigma_L = (3, 1, 1, 1, 1)$. The first value corresponds to the CDC, with subsequent values corresponding to LDCs 1-4.
- The LDCs start with a review period of a week $R = 7$, the CDC initially has $R = 30$, a month.
- All DCs have an (R, S) review policy. (R, s, Q) can not be compared, because the analytical approximation does not take undershoot into account, while the simulation automatically contains undershoot, as described in Section 3.4.
- The initial service level requirement is $P_1 = 95\%$, which is common in practice. P_2 is not compared as the current version of the analytical approximation only supports P_1.

Using this as a starting point, optima are determined using analytical approximation and discrete-time simulation. This is repeated for varying input parameters, which are described in the next section. The initial input parameters are shown in Figure 4.9.

4.4.4 Results comparison

The optima determined by the analytical and simulation method are shown in the following tables. Differences, trends and possible causes thereof are discussed per set of input parameters.

	CDC	LDCx	LDCx	LDCx	LDCx
type (0=RsQ,1=RS)	1	1	1	1	1
E(D)	75	5	10	20	40
sigma(D)	5	0,5	1	2	4
P(D)	1	1	1	1	1
E(L)	20	12	10	8	6
sigma(L)	3	1	1	1	1
R	30	7	7	7	7
Q	1	1	1	1	1
P1 or P2?	2	1	1	1	1
Service level target	50%	95%	95%	95%	95%

Figure 4.9: Initial multi-echelon input parameters

4 quadrant comparison

For each of the 4 quadrants that are used to classify demand, as described in Section 2.1.2, the optimal safety stock configuration is determined using analytical approximation (Calc) and discrete-time simulation (Sim). The input parameters are specified:

- Smooth demand is equal to the initial settings described in the previous section.
- Erratic demand has a σ_D equal to $E(D)$ so that $COV = 1$, compared to smooth demand the standard deviation is increased tenfold.
- Intermittent and lumpy demand have $P(D) = 0.2$. To keep average overall demand equal $E(D)$ (and σ_D to keep COV equal) is multiplied by 5, to compensate for only having demand once every five days on average.

The results are shown in Table 4.1.

	Smooth		Erratic		Intermittent		Lumpy	
	Calc	Sim	Calc	Sim	Calc	Sim	Calc	Sim
$P_{1,CDC}$	57%	0%	57%	5.6%	57%	11%	57%	10%
SS_{CDC}	0	-930	0	-739	0	-946	0	-1285
SS_{LDC_1}	16	57	50	60	104	114	159	183
SS_{LDC_2}	29	115	94	118	195	226	294	360
SS_{LDC_3}	58	229	183	227	382	444	578	726
SS_{LDC_4}	115	459	354	453	740	882	1101	1450
SS_{tot}	218	-69	681	118	1421	720	2132	1434

Table 4.1: Comparison of resulting optimal safety stock configuration per quadrant

Before evaluating these results an explanation must be added to the input parameters. For the intermittent and lumpy quadrant, it is not clear in advance what exactly the mean and standard deviation of the demand pattern is, since it is generated from a mean, a standard deviation and a probability that there is demand in a time period. The measured $E(D)$ and σ_D are shown in Table 4.2. These are the values used in the analytical approximation to calculate intermittent and lumpy results. As was argued in Section 3.4, the analytical approximation is meant for a demand pattern without intervals between demand, which can cause inaccuracies in intermittent and lumpy results.

	Intermittent		Lumpy	
	$E(D)$	σ_D	$E(D)$	σ_D
CDC	75	95	75	138
LDC 1	5	10	5	15
LDC 2	10	20	10	30
LDC 3	20	40	20	60
LDC 4	40	80	40	120

Table 4.2: Measured multi-echelon demand

The comparison shows significant differences between analytical results and simulation results. Even though these differences are large, some observations can be made:

- For both methods, from left to right the total safety stock increases, which corresponds to the measured $\sigma_D = (5, 50, 95, 183)$. When excluding smooth demand the total safety stock increases by roughly the same amount, which might be coincidence. From smooth to erratic the difference for the calculation is roughly double when compared to the simulation. This difference between simulation and analytical method could be caused by the analytical approximation keeping the CDC service level at 57%.

- Smooth and intermittent demand respectively have similar safety stock at the CDC for both methods, but LDC safety stock increases sevenfold for the analytical method while it increases twice for the simulation method. This difference between simulation and analytical method could be caused by the analytical approximation keeping the CDC service level at 57%.

- $P_{1,CDC}$ stays the same for the calculation while it seems to increase for the simulation. This is probably caused by the calculation of average inventory in the analytical approximation, which includes average backorders in this calculation.

- For the simulation, LDC safety stock stays equal from smooth to erratic, the increased demand uncertainty is compensated by increased CDC safety stock. The calculation keeps CDC safety stock equal while increasing LDC safety stock.

CDC lead time variance

In the initial parameters $\sigma_{L,CDC} = 3$, to simulate different lead time variances it is multiplied and divided by 3. Results are shown in Table 4.3.

	$\sigma_{L,CDC} = 1$		$\sigma_{L,CDC} = 3$		$\sigma_{L,CDC} = 9$	
	Calc	Sim	Calc	Sim	Calc	Sim
$P_{1,CDC}$	41%	0%	57%	0%	60%	9%
SS_{CDC}	-29	-1297	0	-930	68	-833
SS_{LDC_1}	11	80	16	57	36	70
SS_{LDC_2}	22	160	29	115	63	140
SS_{LDC_3}	43	320	58	229	126	284
SS_{LDC_4}	85	639	115	459	252	566
SS_{tot}	132	-98	218	-69	544	228

Table 4.3: Comparison of resulting optimal safety stock configuration per $\sigma_{L,CDC}$

It looks like the simulation mostly compensates increased lead time variance by increasing CDC safety stock. The calculation also increases CDC safety stock for increased $\sigma_{L,CDC}$, but LDC safety stock also increases, $\sigma_{L,CDC}$ has a large effect on $\sigma_{L,LDC}$ in the equations, which might account for this difference in SS_{LDC}.
The differences between $\sigma_{L,CDC} = 3$ and $\sigma_{L,CDC} = 9$ are roughly equal for both methods.

Relative CDC review period

To test the effect of a longer review period in the CDC relative to LDCs, the initial R_{CDC} of a month is changed to two weeks and to three months. Results are shown in Table 4.4.

	$R_{CDC} = 14$		$R_{CDC} = 30$		$R_{CDC} = 90$	
	Calc	Sim	Calc	Sim	Calc	Sim
$P_{1,CDC}$	57%	1.4%	57%	0.0%	57%	0.0%
SS_{CDC}	0	-538	0	-930	0	-1393
SS_{LDC_1}	16	19	16	57	16	67
SS_{LDC_2}	29	37	29	115	29	135
SS_{LDC_3}	58	76	58	229	58	268
SS_{LDC_4}	115	151	115	115	252	534
SS_{tot}	218	-255	218	-69	218	-390

Table 4.4: Comparison of resulting optimal safety stock configuration for different review period ratios

For the analytical approximation, results do not change by changing the review period. The simulation has a decreasing CDC safety stock, but this could be explained by the increased uncertainty period causing a larger order-up-to level. Total safety stock also changes in the simulation, but the middle case is highest. The low total in the $R_{CDC} = 14$ case could be caused by the endings of review periods being synchronized for CDC and LDCs, which means that 1 out of every 2 orders by the LDCs is immediately re-ordered by the CDC, since the simulation first checks LDC review periods and then CDC review period. The lower SS_{CDC} in the simulation for the highest review period could corresponds to an increasing $E[DDUP_{CDC}]$, which includes R_{CDC} in its calculation.

Service level requirement

The effect of different service level requirements can be seen in Table 4.5. The P_1 of all LDCs is set to the value in the table.

	$P_1 = 80\%$		$P_1 = 95\%$		$P_1 = 98\%$	
	Calc	Sim	Calc	Sim	Calc	Sim
$P_{1,CDC}$	41%	0.0%	57%	0.0%	60%	0.1%
SS_{CDC}	-80	-1033	0	-930	23	-676
SS_{LDC_1}	9	30	16	57	19	52
SS_{LDC_2}	18	59	29	115	35	105
SS_{LDC_3}	35	117	58	229	69	208
SS_{LDC_4}	70	235	115	459	138	417
SS_{tot}	52	-593	218	-69	284	106

Table 4.5: Comparison of resulting optimal safety stock configuration per quadrant

Both methods increase CDC safety stock, and total safety stock, for increased P_1 of the LDCs. LDC safety stock increases for the analytical approximation, while not clearly increasing or decreasing for the simulation. This might be caused by the stochastic component of results that was mentioned at the start of this section.

Number of LDCs

Using the input parameters of LDC 3 to find results for 1, 2, 3 and 4 equal LDCs, the optima in Table 4.6 were found.

	1 LDC		2 LDCs		3 LDCs		4 LDCs	
	Calc	Sim	Calc	Sim	Calc	Sim	Calc	Sim
$P_{1,CDC}$	57%	0%	57%	0%	57%	0%	57%	0%
SS_{CDC}	0	-297	0	-595	0	-793	0	-1181
SS_{LDC_1}	63	222	63	220	63	186	63	215
SS_{LDC_2}			58	220	58	187	58	215
SS_{LDC_3}					58	186	58	215
SS_{LDC_4}							58	215
SS_{tot}	63	-74	121	-155	178	-234	236	-320

Table 4.6: Comparison of resulting optimal safety stock configuration per quadrant

The expected result of this experiment would be that many results change linearly with increasing LDCs, which is roughly what happens in the results. The analytical approximation has a slightly differing safety stock at the first LDC compared to the others. The simulation, for the 3 LDC option, shifts the CDC-LDC safety stock balance towards CDC safety stock, but total safety stock keeps the same trend, as explained by the stochastic component of results that was mentioned at the start of this section.

SS_{CDC} for the simulation decreases in direct relation to $E[DDUP_{CDC}]$, since this doubles for each added LDC, so that SS_{CDC} stays equal in relation to $E[DDUP_{CDC}]$ except for the situation with 3 LDCs, which was already discussed.

In most cases, the analytical approximation chooses an optimum with $SS_{CDC} = 0$, which is likely to be caused by the large effect of backorders on average stock level. Also using a measurement of the average inventory in the simulation would be a useful future step. The next section compares adjustments to the analytical approximation to the simulation results.

4.4.5 Analytical calculation extensions

The comparison showed a lot of large differences between the discrete-time model and the analytical approximation. Causes of inaccuracies of the simulation were addressed at the start of Section 4.4.1. Adjustments to the equations in Section 2.2.1 are described in Sections 2.2.2 and 2.2.3. Assumptions made in the analytical approximation are compared to simulation results. Proposals for adjusted equations are described and compared.

The equations from Section 2.2.1 are:

$$E[L^*_{LDC}] = E[L_{LDC}] + (1 - P_{1,CDC})E[L_{CDC}],$$

$$\sigma^2_{L^*_{LDC}} = \sigma^2_{L_{LDC}} + (1 - P_{1,CDC})^2\sigma^2_{L_{CDC}}.$$

The addition of $(1 - P_{1,CDC})L_{CDC}$ to LDC lead time is analyzed. This can be split into the assumption that the delay in case of a stockout is L_{CDC}, and the assumption that the LDC is affected negatively at each stockout:

- When a CDC has a make-to-order (MTO) strategy, a delay of L_{CDC} would be the case: An LDC orders from the CDC, and at that moment the CDC orders from e.g., a factory. The CDC lead time passes, and the replenishment arrives at the CDC only to be sent immediately to the LDC. Then the LDC lead time passes, and the shipment arrives at the LDC with lead time $L = L_{CDC} + L_{LDC}$. However, with a replenishment policy like (R, s, Q) or (R, S) this delay does not apply. A replenishment is ordered after a review period, and by the time a stockout occurs, this order has almost arrived in most cases. Therefore, the "waiting time" the LDC is discussed further in this section.
- The other assumption is that the extra waiting time of the LDCs is a function of $P_{1,CDC}$. This seems to infer that at each CDC stockout, the replenishment to the LDC should have a measurable delay. This is partially true, except that when the stockout occurs, demand is still partially met (depending on the policy). In case of a fair share policy, each LDC still gets an equal percentage of its replenishment order. If this percentage is high enough to cover the delay of the rest of the shipment, there is no measurable effect at all on LDC service levels. Adjusted versions of this equation are tested hereafter.

Analytical approximation in Matlab

To see the effect of adjusted versions of the analytical approximation, a script is made that calculates the analytical results for the same CDC safety stock range as the simulation, and makes a diagram to compare results.

In Matlab, a single-echelon analytical calculation is already available (Appendix A.1). To transform this calculation to multi-echelon, a few small adjustments are made:

- For each CDC safety stock, the CDC P_1 needs to be calculated.
- This $P_{1,CDC}$ is used in the calculation of an adjusted $E(L)$ and σ_L for each LDC, for each CDC safety stock.
- Using the adjusted lead time parameters, the necessary LDC safety stock to maintain service level requirements can be calculated.

After these adjustments the script can be used to calculate data points to make a similar diagram to the one used in the Excel tool, using total safety stock instead of total average inventory. The script expanded from single-echelon can be seen in Appendix D. The analytical results from Matlab are presented in Figure 4.11, the LDC safety stocks are added to the CDC safety stock, the lines in the figure represent the total safety stock. The input parameters from Figure 4.9 were used.

Adjusted $P_{1,CDC}$

An adjustment to the Desmet formula that is currently being developed, is an adjusted value that can be used instead of $P_{1,CDC}$, as described in Section 2.2.2. The proposed equation is:

$$P^*_{1,CDC} = \Phi(\frac{Q_{CDC}/2 + SS_{CDC}}{\sigma_{DDUP,CDC}}),$$

where σ_{DDUP} is the standard deviation of demand during the uncertainty period. This equation is implemented in the Matlab model, results are compared to previous results in Figure 4.11. As can be seen, the resulting values are closer to those of the simulation compared to the initial analytical method.

Adjusted variance equation

It was noticed that the equation for $\sigma^*_{L,LDC}$ is different from the result of the mean of square minus square of mean rule, as shown in Appendix E, the derived adjustment is:

$$\sigma^2_{L^*_{LDC}} = \sigma^2_{L_{LDC}} + (1 - P_{1,CDC})^2\sigma^2_{L_{CDC}} + (1 - P_{1,CDC})P_{1,CDC}E(L^2_{CDC}).$$

The results of this method are also compared to previous results in Figure 4.11. It seems that the resulting values are not closer to those of the simulation compared to the initial analytical approximation.

Waiting time

In Section 2.2.3, an extended version of the Desmet formula is discussed. It uses an exponential distribution with a different μ for each P_1 of the CDC. The calculation of the exponential distribution is not mentioned, but it is possible to compare this to simulation results for the waiting time.

An addition to the multi-echelon simulation keeps track of the time at which a stockout occurs, and subtracts this time from the time of the next replenishment arrival, so that a list of waiting times is generated. Waiting time histograms for several P_1 values can be seen in Figure 4.10.

Waiting times measured from simulation also seem to follow a probability distribution, by testing different distributions it was found that the Gamma distribution most closely represents the waiting times found during simulation. A Gamma distribution is fitted to the waiting time data for each CDC safety stock value, resulting in an $E[W]$ and σ_W for each CDC safety stock.

The resulting values follow a pattern similar to Figure 4.6, which is why the same Normal cumulative distribution function is fitted to the Gamma parameters for each CDC safety stock. This fit is implemented in the calculation of $E^*[L_{LDC}]$ and $\sigma_{L^*_{LDC}}$. The results of this method are also compared to previous results in Figure 4.11, the corresponding line seems a bit closer to the simulation results compared to the initial approximation. However, these results are partially based on simulation results, which might be the cause of the reduced difference. The script used to find the waiting time data can be found in Appendix D.1.

In addition to the individual adjustments to the analytical approximation, the combination of $P^*_{1,CDC}$, variance calculation using the mean of square minus square of mean rule and waiting time measurement from simulation is added to the diagram. It seems that this combination is closest to simulation results overall, with minimal total safety stock at roughly the same CDC safety stock as the final simulation optimum.

In this chapter, the single-echelon discrete-time model was expanded to be able to simulate multi-echelon distribution networks. A method to find the optimal safety stock configuration was described, and results were compared to analytical methods. It is clear that the currently available methods used to calculate multi echelon safety stock, and the simulation, do not give equal results. However, both results indicate that a high service level from CDC to LDCs is undesirable. Since using high CDC service levels is currently common, this can be a useful opportunity in inventory (cost) reduction.

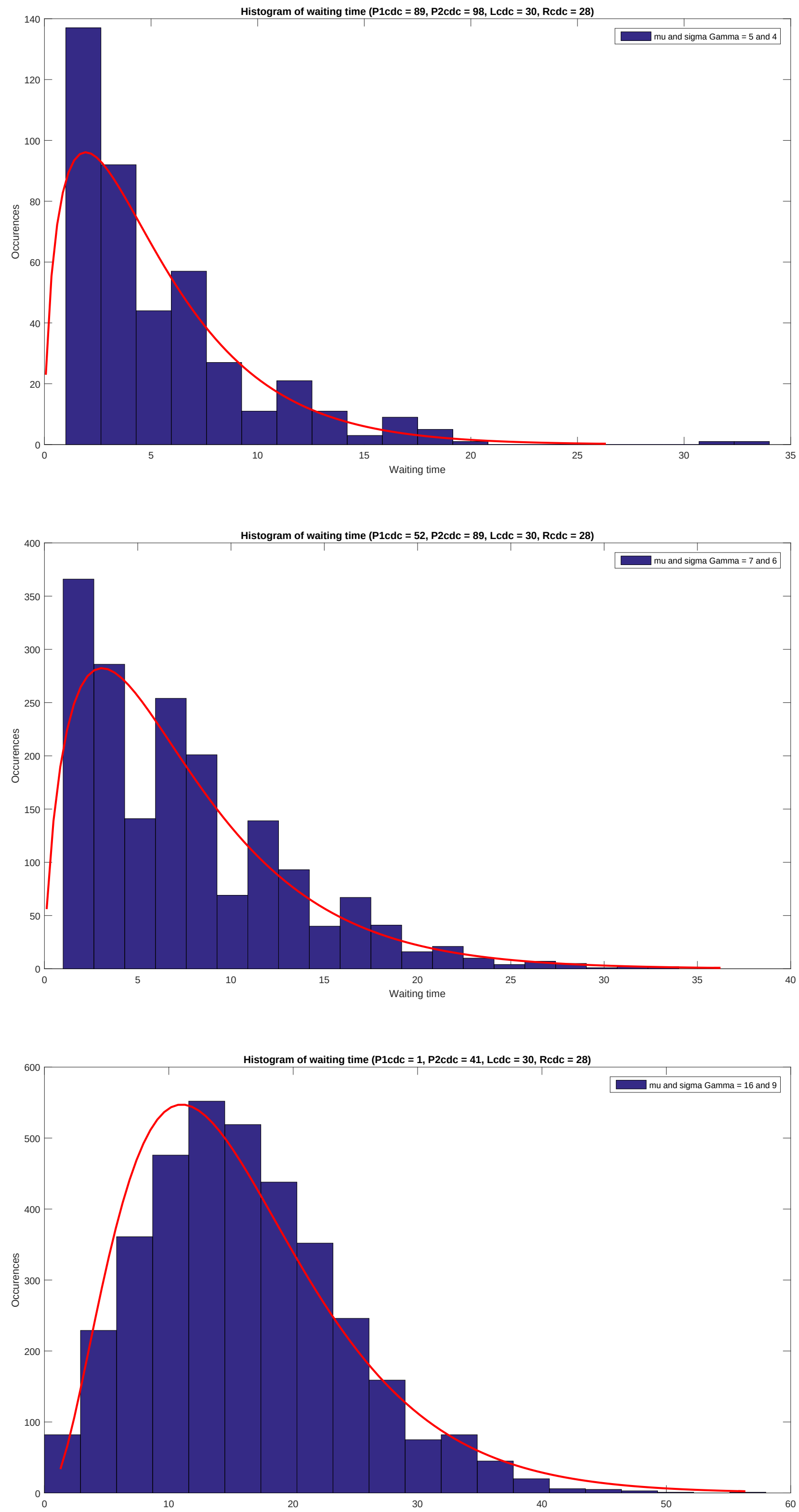


Figure 4.10: Waiting time histograms

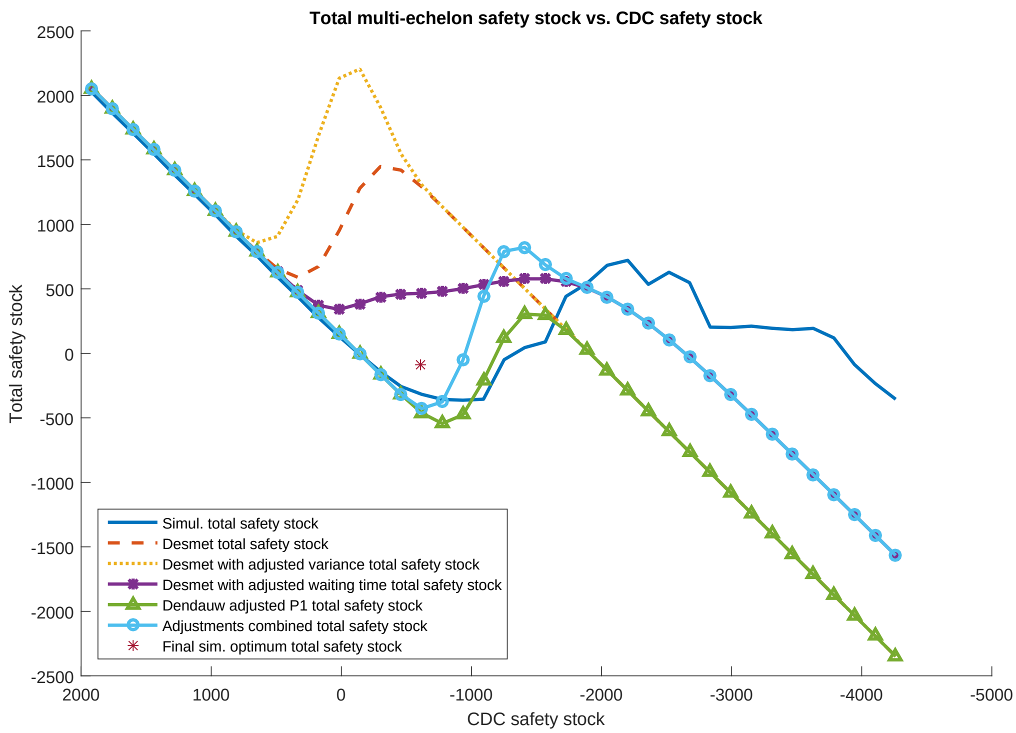


Figure 4.11: Multi-echelon calculations comparison

Chapter 5

Conclusion & Recommendations

While the previous chapters provided an in-depth perspective of the discrete-time models and their detailed functions, this chapter discusses the results of this whole project. Drawing conclusions on results vs. expectations that were created in the problem statement, and comparing simulation results to analytical results. Based on possibilities to improve and expand the current model, some recommendations are formulated.

5.1 Conclusion

Before discussing the details of model results, the objectives of the project are discussed:

- A single- and multi-echelon model were created, a switch was made from discrete-event simulation to discrete-time simulation to have a better match with discrete demand intervals.
- Single-echelon functionality was added. Input parameters can be used to simulate (R, S) and (R, s, Q) policies, with stochastic lead times and stochastic demand in all four demand quadrants. The exception is forecast based replenishment, which was not added. Since forecast based replenishment is done by adjusting demand parameters for different forecast periods this can be done by using differing input parameters in the current model.
- The multi-echelon model keeps the functions of the single-echelon model. It is able to simulate networks of 2 echelons, 1 CDC can be simulated with N LDCs one echelon lower. It would still be possible to optimize larger networks manually, by doing sequential optimizations. By determining optimal CDC service levels for multiple CDCs in a larger network, and afterwards using these CDC parameters as "LDC" parameters in a simulation where the "CDC" is the higher echelon CDC/factory. This might not be as accurate as adding extra echelons to the model.
- Optimization was added for both the single-echelon and multi-echelon model. Minimal safety stock can be determined for a set of input parameters and a service level requirement.

5.1.1 Results validation

Both the single-echelon and multi-echelon results of analytical equations and simulations have been compared in several ways. The single echelon results are similar, some differences were noted:

- A systematic error is created by using discrete time intervals, when as a first step replenishments are checked for an interval, and as a second step demand is checked for that interval. This causes situations where stock might have run out if it were a continuous process. Using a smaller time interval decreases this error but increases computing time.

- Lead time variances larger than approximately 1/3 of the review period lead to differences in results, due to replenishments that are delayed to prevent overtaking.

- For intermittent and lumpy demand, differences in results are larger. This is probably due to the formulas that were used, these are not intended for use with infrequent demand.

Based on overall results it seems that simulation and analytical single-echelon results are equivalent. The multi-echelon results are less consistent. Since the multi-echelon discrete-time model is purely an expansion of the single-echelon model, which is validated with analytical results, this indicates that the main source of inequalities is in the analytical equations. Some options to increase the accuracy of the results were considered and tested, resulting in less difference, but the most similar options are still unequal. Errors in simulation results, on top of errors transfered from the single-echelon model, can be caused by simultaneous orders from all LDCs. This causes clustered demand at the end of the review period of the LDCs. Situations can occur when a stockout would have happened if demand was less clustered, increasing the results in CDC service levels. This could not be prevented, because the use of a fair share policy needed the demands to arrive simultaneously in order to determine each LDCs share.

Nevertheless, it can be concluded that in most cases, CDC service level should be a lot lower than it is in current practice, which is agreed upon by both analytical equations and simulations. Having two resources which give this advice can help to convince customers of this conclusion. Furthermore, in this project, two tools have been created that OM Partners can use to get a second opinion on the results of their calculations.

5.2 Recommendations

Based on the points of the initial problem description that were not implemented, and the sources of errors that were identified, better results might be possible by doing further research in the following ways:

- The problem of overtaking replenishments is caused by a large lead time variance. In practice large lead time variance often means that production is planned in batches, which causes some replenishments orders to be available quickly and some after a long wait. To solve this problem in the simulation, it might be possible to implement this batching process into the simulation. Instead of large lead time variance a large batch size with smaller variance could be used, with pauses between batches.

- The error caused by time intervals might be reduced by switching the order of steps in the simulation for half of the simulation and taking the average result. A more rigorous solution would be a continuous-time model. If demand is modeled as a continuous process this could be done using discrete-event simulation.

- The fair share policy and priority policy at the CDC in the multi-echelon simulation impose restrictions on the model. A more robust solution might be possible that does not require simultaneous LDC orders.

- The discrete-time multi-echelon optimization model minimizes total safety stock, which implies that there are negative holding costs for negative stock. Using the average positive inventory level measured during simulation would provide a more accurate safety stock optimum.

- The initial goal of N echelons, to simulate large distribution networks, should also be possible to model. The same expansion that was implemented for going from one to two echelons could be implemented to add a third echelon. If this expansion could be automated N echelons would be possible.
- The multi-echelon optimization uses a sequential curve-fitting method. Resulting fits are close to the data, but not equal. In regions where service level curves are almost horizontal, a small inaccuracy can lead to a big difference in safety stock. Increasing the accuracy of the fitted curves or surface would make the initially guesses optimum a lot more accurate, or possibly equal to the simulation data.
- By aiming the multi-echelon simulations in the area where service levels fluctuate, more relevant data could be acquired using less computing time.

This concludes the last chapter of this thesis.

Bibliography

[1] Aström, KJ, W. B. (1990). *Computer-Controlled Systems: Theory and Design*. Prentice-Hall International Editions.

[2] de Kok, A. (1998). Analysis of one product/one location inventory control models. *Unpublished course notes at Technische Universiteit Eindhoven.*

[3] Dendauw, P. (2016). Multi-echelon inventory optimization safety stock positioning. *Unpublished notes at OM Partners/Universiteit Gent.*

[4] Desmet, B. (2009). A normal approximation model for safety stock optimization in multi-echelon supply chains. *PhD Thesis.*

[5] Desmet, B., Aghezzaf, E. H., and Vanmaele, H. (2010). A normal approximation model for safety stock optimization in a two-echelon distribution system. *Journal of the Operational Research Society*, 61(1):156–163.

[6] Kostenko, A. and Hyndman, R. (2006). A note on the categorization of demand patterns.

[7] Manuj, I., Mentzer, J. T., and Bowers, M. R. (2009). Improving the rigor of discrete-event simulation in logistics and supply chain research. *International Journal of Physical Distribution and Logistics Management*, 39(3):172–201.

[8] MATLAB (2015). *version 8.6.0.267246 (R2015b).* The MathWorks Inc., Natick, Massachusetts.

[9] Papalambros, P. Y. and Wilde, D. J. (2000). *Principles of optimal design: modeling and computation.* Cambridge university press.

Appendix A

Single-echelon matlab scripts

This appendix contains the Matlab scripts used for single-echelon simulation, validation and optimization. The first script is the simulation described in Section 3.2.

```
clearvars; close all;

%Simulation properties
simscale = 1; %Time periods are split into smaller steps using scale
Nstart = 500*simscale; %Number of time periods that do not count for results
N = (10000+Nstart)*simscale; %Total number of time periods
NX = 15; %Number of experiments
simgraph = 0; %If 1 the simulation runs once and makes a graph of ...
    inventory levels

input = xlsread('InputSE.xlsx','Sheet1');

%Calculate system properties from input
type = input(1); %0 = RsQ policy, 1 = RS policy
ED = input(2)/simscale; %Expected value of demand per time period
sigD = input(3)/simscale; %standard deviation of demand per time period
PD = input(4); %lambda of Poisson distribution that determines demand interval
EL = input(5)*simscale; %Expected  value of lead time in time periods
sigL = input(6)*simscale; %Standard deviation of lead time in time periods
R = input(7)*simscale; %Review period in time periods
Q = input(8)*ED*R*PD; %Order quantity level using standard formula (only RsQ)
sslow = input(9); %Lower bound of safety stock
ssint = input(10); %Interval of safety stock values
sshigh = input(11); %Upper bound of safety stock

%Initialization
Results = zeros((sshigh-sslow)/ssint,5); %Results matrix
r = 1; %Counter for results matrix
P = zeros(NX,2); %Service level matrix
Nzero = 0; %Count # zeros in case of Lambda
if simgraph == 1
    data = zeros((round(N/R)-1)*(1+R*3),3); %Matrix to save inventory data
    c = 1; %counter
end

%One vector of demands is generated for each experiment, this vector is
    %used for all values of ss
%When a lambda is given, a geometric distribution calculates the time periods
    %until the next demand
if PD == 1
    seed = gamrnd((ED/sigD)^2,sigD^2/ED,N,NX);
else
    seed = zeros(N,NX);
    for x = 1:NX
        next = geornd(PD); %Determine time periods until next order
```

```
        for y = 1:N %Generate demand  or fill in pause for all demands
            if next == 0
                seed(y,x) = gamrnd((ED/sigD)^2,sigD^2/ED);
                next = geornd(PD);
            else
                seed(y,x) = 0;
                Nzero = Nzero + 1;
                next = next - 1;
            end
        end
    end
end
avgdemand = mean(mean(seed)); %Resulting average of demands
prcntdemand = 1-Nzero/(N*NX); %Resulting % of filled time periods
sigdemand = mean(std(seed)); %Resulting standard deviation of demands

EDDUP = avgdemand*(EL+type*R); %EDDUP according to standard formula

%Loop over values of s
for s = sslow+EDDUP:ssint:sshigh+EDDUP
    %Loop over number of simulations in 1 experiment
    for xper = 1:NX;
        %Initialize simulation values
        Y = s + (1-type)*Q;%Inventory position initialization
        X = s + (1-type)*Q;%Net stock initization
        Dtot = 0;%Used in calculation of P2
        Btot = 0;%Used in calculation of P2
        NQtot = 0;%Used in calculation of P1
        Nstockout = 0;%Used in calculation of P1
        O = cell(1,1);%Orders in transit are added to this matrix
        start = 0;%Start will become 1 after a startup period
        %Loop over all review periods
        for k = 1:N/R
            if k*R > Nstart %Start measuring P1 and P2 after startup period
                start = 1;
            end
            %Loop over demands in 1 review period
            for i = 1:R;
                %Record data for visualization
                if simgraph == 1
                    data(c,:)=[R*(k-1)+i Y X];
                    c = c + 1;
                end
                %Check if there are orders that have arrived
                %Update values for P1
                if size(O{1},1) > 0 %Are there orders?
                    O{1}(:,2) = O{1}(:,2) - 1;
                    if O{1}(1,2) < 0
                        NQtot = NQtot + 1*start;
                        if X < 0
                            Nstockout = Nstockout + 1*start;
                        end
                        X = X + O{1}(1,1);
                        O{1}(1,:) = [];
                    end
                end
                %Record data of delivered orders for visualization
```

```
                if simgraph == 1
                    data(c,:)=[R*(k-1)+i+0.01 Y X];
                    c = c + 1;
                end
                %Determine demand
                D = seed(R*(k-1)+i,xper);
                %Update values for P2
                Dtot = Dtot + D*start;
                if X ≤ 0
                    Btot = Btot + D*start;
                else
                    if X < D
                        Btot = Btot + start*(D - X);
                    end
                end
                %Update inventory levels and record data for visualization
                Y = Y - D;
                X = X - D;
                if simgraph == 1
                    data(c,:)=[R*(k-1)+i+0.02 Y X];
                    c = c + 1;
                end
            end
            %After each review period update orders
            if Y < s && type == 0
                L = gamrnd((EL/sigL)^2,sigL^2/EL);
                O{1} = [O{1};ceil((s-Y)/Q)*Q L];
                Y = Y + ceil((s-Y)/Q)*Q;
            elseif type == 1
                L = gamrnd((EL/sigL)^2,sigL^2/EL);
                Q = s-Y;
                O{1} = [O{1};Q L];
                Y = Y + Q;
            end
            %Update data for visualization
            if simgraph == 1
                    data(c,:)=[R*(k-1)+i+0.03 Y X];
                    c = c + 1;
            end
        end
        %Record service levels
        P(xper,:) = [(1-Nstockout/NQtot) (1-Btot/Dtot)];
        %Print result of simulation for visualization and terminate loops
        if simgraph == 1
            break;
        end
    end
    if simgraph == 1
        break;
    end
    %Determine mean and std
    Results(r,:) = [s mean(P) tinv(0.975,NX-1)/sqrt(NX)*std(P)];
    r = r + 1;
end
%Plot single graph of visualization or service level results
if simgraph == 1
    figure(1)
```

```
    hold on
    grid on
    for px = 1:2
        plot(data(:,1),data(:,1+px));
    end
    plot(data(:,1),s*ones(size(data(:,1),1),1))
    legend('Inventory  position','Net stock');
    title('Details of simulation run');
    xlabel('Time');
    ylabel('Number of products');
else
    figure
    hold on
    grid on
    grid MINOR
    for px = 1:2
        errorbar(Results(:,1),Results(:,1+px),Results(:,3+px),'Color',[px/2 ...
            0 0],'DisplayName',['P' num2str(px) ' Model']);
    end
    legend(gca,'show')
    if type == 0
        title(['(R=' sprintf('%0.0f',R) ',E[DDUP]=' sprintf('%0.0f',EDDUP) ...
            ',Q=' sprintf('%0.0f',Q) ') ED=' sprintf('%0.0f',avgdemand) ' ...
            sigD=' sprintf('%0.0f',sigdemand) ' D>0=' ...
            sprintf('%0.0f',100*prcntdemand) '% EL=' sprintf('%0.0f',EL) ' ...
            sigL=' sprintf('%0.0f',sigL)]);
    else
        title(['(R=' sprintf('%0.0f',R) ',E[DDUP]=' sprintf('%0.0f',EDDUP) ...
            ') ED=' sprintf('%0.0f',avgdemand) ' sigD=' ...
            sprintf('%0.0f',sigdemand) ' D>0=' ...
            sprintf('%0.0f',100*prcntdemand) '% EL=' sprintf('%0.0f',EL) ' ...
            sigL=' sprintf('%0.0f',sigL)]);
    end
    xlabel('s (re-order stock level)');
    ylabel('Service level (%)');
end
```

Listing A.1: Single-echelon matlab model

A.1 Single-echelon analytical results code

The second script contains the script used to calculate analytical results for validation of the discrete-time model, described in Section 3.4.

```
%% Initialize
SEIO_RsQ_RS_v1;

AnalyticalResults = zeros((sshigh-sslow)/ssint,3);
a = 1;
sigDDUP = sqrt((EL+type*R)*sigdemand^2+avgdemand^2*sigL^2); %Standard formula
Ushoot = 1;

%% Determine Gamma parameters
if type == 0 && Ushoot == 1 %include undershoot in (R,s,Q)
    EU = (sigdemand^2*R+avgdemand^2*R^2)/(2*R*avgdemand);
```

```
    sigU = sqrt((1+((sigdemand/avgdemand)^2)/R) * ...
        (1+2*(sigdemand/avgdemand)^2/R) * (avgdemand*R)^2/3-EU^2);
    EUDDUP = EU + EDDUP;
    sigUDDUP = sqrt(sigU^2+sigDDUP^2);
    alpha = (EUDDUP/sigUDDUP)^2;
    beta = sigUDDUP^2/EUDDUP;
else
    alpha = (EDDUP/sigDDUP)^2;
    beta = sigDDUP^2/EDDUP;
end

%% Determine service level results for safety stock range
for s = sslow+EDDUP:ssint:sshigh+EDDUP
    P1 = gamcdf(s,alpha,beta);
    if type == 0 && Ushoot == 1
        GAMMA1 = EUDDUP * (1-gamcdf(s,alpha+1,beta))- s * ...
            (1-gamcdf(s,alpha,beta));
        GAMMA2 = EUDDUP * (1-gamcdf(s+Q,alpha+1,beta)) - (s+Q) * ...
            (1-gamcdf(s+Q,alpha,beta));
        P2 = 1-(GAMMA1-GAMMA2)/Q;
    elseif type == 0 && Ushoot == 0
        GAMMA1 = EDDUP * (1-gamcdf(s,alpha+1,beta))- s * ...
            (1-gamcdf(s,alpha,beta));
        GAMMA2 = EDDUP * (1-gamcdf(s+Q,alpha+1,beta)) - (s+Q) * ...
            (1-gamcdf(s+Q,alpha,beta));
        P2 = 1-1/(Q)*(GAMMA1-GAMMA2);
    else
        GAMMA1 = EDDUP * (1-gamcdf(s,alpha+1,beta)) - s * ...
            (1-gamcdf(s,alpha,beta));
        GAMMA2 = EDDUP * (1-gamcdf(s+avgdemand*R,alpha+1,beta)) - ...
            (s+avgdemand*R) * (1-gamcdf(s+avgdemand*R,alpha,beta));
        P2 = 1-1/(avgdemand*R)*(GAMMA1-GAMMA2);
    end
    AnalyticalResults(a,:) = [s P1 P2];
    a = a + 1;
end
AnalyticalResults = AnalyticalResults(1:a-1,:)

%% Add results plot to simulation results
for px = 1:2
    plot(AnalyticalResults(:,1),AnalyticalResults(:,px+1),'Color',[0 0 ...
        px/2],'DisplayName',['P' num2str(px) ' Theory']);
end
legend('off');
legend(gca,'show');
legend('Location','southeast');
```

Listing A.2: Single-echelon results comparison

A.2 Single-echelon optimization code

The last script is the script used to determine the optimum amount of safety stock for a given set of input parameters and a service level requirement as used in Section 3.3.

```
function output = SEIO_optim_fit(xlsinput)

```

```
%Simulation properties
simscale = 1; %Time periods are split into smaller steps using scale
Nstart = 500*simscale; %Number of time periods that do not count for results
N = (10000+Nstart)*simscale; %Total number of time periods
Nintervals = 30; %Number of intervals used in simulating fitting data

%Calculate system properties from input
type = xlsinput(1); %0 = RsQ policy, 1 = RS policy
ED = xlsinput(2)/simscale; %Expected value of demand per time period
sigD = xlsinput(3)/simscale; %standard deviation of demand per time period
PD = xlsinput(4); %lambda of Poisson distribution that determines demand ...
    interval
EL = xlsinput(5)*simscale; %Expected value of lead time in time periods
sigL = xlsinput(6)*simscale; %Standard deviation of lead time in time periods
R = xlsinput(7)*simscale; %Review period in time periods
EDDUP = ED*PD*(EL+type*R); %S according to standard formula
Q = xlsinput(8)*ED*PD*R; %Order quantity level using standard formula ...
    (only RsQ)
Ptype = xlsinput(9); %P1 or P2 requirement
Ptarget = xlsinput(10); %Target service level

%Find point where P = 50%
Stest = EDDUP; %Stest is updated towards the optimum
P = SEIO_func(Stest); %Find initial P guess
if P > 50 %Iterate towards 50% by steps of ED
    while P > 50
        Stest = Stest - ED;
        P = SEIO_func(Stest);
    end
    Stest = Stest + ED;
else
    while P < 50
        Stest = Stest + ED;
        P = SEIO_func(Stest);
    end
end

%Create grid for curve fitting area
Xvalues = ...
    linspace(Stest-ED*(EL+type*R)*PD,Stest+3*ED*(EL+type*R)*PD,Nintervals)'; ...
    %Nintervals around 50% point
Yvalues = zeros(size(Xvalues,1),1);

%Find results within grid
for j = 1:Nintervals
    Yvalues(j) = SEIO_func(Xvalues(j));
end

%Fit results to logistic function and
linfun = fit(Xvalues,Yvalues,'100/(1+exp(b*(x-a)))','StartPoint',[Stest ...
    0.001]);
objective = @(x) linfun(x) - Ptarget;
Stest = fzero(objective,EDDUP); %Find optimal S value
ci = predint(linfun,Stest,0.95,'functional'); %Find confidence interval ...
    for optimum
% plot(linfun)
% hold on
```

```
% plot(Xvalues,Yvalues,'+','DisplayName','Simulation results')
% xlabel('Re-order point/order-up-to level')
% ylabel('Service level')
% title('Single-echelon optimization fitting results')
% legend('off');
% legend(gca,'show');
output = [Stest Ptarget ci]; %Return S value, P value and confidence interval

%%Single run version of single-echelon script
    function P = SEIO_func(s)
        if PD == 1
            seed = gamrnd((ED/sigD)^2,sigD^2/ED,N,1);
        else
            seed = zeros(N,1);
            next = geornd(PD);
            for y = 1:N
                if next == 0
                    seed(y) = gamrnd((ED/sigD)^2,sigD^2/ED);
                    next = geornd(PD);
                else
                    seed(y) = 0;
                    next = next - 1;
                end
            end
        end

        Y = s + (1-type)*Q;%Inventory position initialization
        X = s + (1-type)*Q;%Net stock initizalization
        Dtot = 0;%Used in calculation of P2
        Btot = 0;%Used in calculation of P2
        NQtot = 0;%Used in calculation of P1
        Nstockout = 0;%Used in calculation of P1
        O = cell(1,1);%Orders in transit are added to this matrix
        start = 0;%Start will become 1 after a startup period

        %Loop over all review periods
        for k = 1:N/R
            if k*R > Nstart %Start measuring P1 and P2 after startup period
                start = 1;
            end
            %Loop over demands in 1 review period
            for i = 1:R;
                %Check if there are orders that have arrived
                %Update values for P1
                if size(O{1},1) > 0
                    O{1}(:,2) = O{1}(:,2) - 1;
                    if O{1}(1,2) < 0
        %                           if size(O{1},1) > 1 && O{1}(1,2) > O{1}(2,2)
        %                               test = [test;O{1}(1,2) - O{1}(2,2)];
        %                           end
                        NQtot = NQtot + 1*start;
                        if X < 0
                            Nstockout = Nstockout + 1*start;
                        end
                        X = X + O{1}(1,1);
                        O{1}(1,:) = [];
                    end
```

```
                end
                %Determine demand
                D = seed(R*(k-1)+i);
                %Update values for P2
                Dtot = Dtot + D*start;
                if X ≤ 0
                    Btot = Btot + D*start;
                else
                    if X < D
                        Btot = Btot + start*(D - X);
                    end
                end
                Y = Y - D;
                X = X - D;
            end
            %After each review period update orders
            if Y < s && type == 0
                L = gamrnd((EL/sigL)^2,sigL^2/EL);
                O{1} = [O{1};ceil((s-Y)/Q)*Q L];
                Y = Y + ceil((s-Y)/Q)*Q;
            elseif type == 1
                L = gamrnd((EL/sigL)^2,sigL^2/EL);
                Q = s-Y;
                O{1} = [O{1};Q L];
                Y = Y + Q;
            end
        end
        %Record service levels
        P = (2-Ptype)*100*(1-Nstockout/NQtot)+(Ptype-1)*100*(1-Btot/Dtot);
    end
end
```

Listing A.3: Single-echelon inventory optimization code

Appendix B

Multi-echelon discrete-time Matlab script

The script in this appendix is used to simulate the multi-echelon discrete-time model, as described in Section 4.2.

```
clearvars; close all

%Simulation properties
simscale = 1; %Changes the amount of steps in 1 demand period
Nstart = 500*simscale;
N = (10000+Nstart)*simscale; %Number of demands generated
NX = 10; %Number of experiments
simgraph = 0; %if this is 1 the simulation will run once while recording ...
    values of Y and X
fairshare = 1;

xlsvalues = xlsread('Input.xlsx','Sheet1');%Read input from excel

%System parameters are vectors containing the properties for each DC
Ndc = size(xlsvalues,2); %Number of DC's = Number of columns
ED = zeros(Ndc,1);
sigD = zeros(Ndc,1);
PD = zeros(Ndc,1);
EL = zeros(Ndc,1);
sigL = zeros(Ndc,1);
R = zeros(Ndc,1);
Q = zeros(Ndc,1);
S = zeros(Ndc,1);
type = zeros(Ndc,1);

%Extract correct values from input matrix for each DC
for z = 1:Ndc
    type(z) = xlsvalues(1,z);
    ED(z) = xlsvalues(2,z)/simscale;
    sigD(z) = xlsvalues(3,z)/simscale;
    PD(z) = xlsvalues(4,z)/simscale;
    EL(z) = xlsvalues(5,z)*simscale;
    sigL(z) = xlsvalues(6,z)*simscale;
    R(z) = xlsvalues(7,z)*simscale;
    Q(z) = xlsvalues(8,z)*ED(z)*R(z);
    S(z) = ED(z)*(EL(z)+type(z)*R(z))+xlsvalues(9,z);
end
%Initialization
P = zeros(NX,2*Ndc);%Service level matrix
W = zeros(10^4,3);
cW = 1;
seed = zeros(N,NX,Ndc-1);
```

```
%One table of demands is generated per experiment, different values of s
%use the same table When a lambda is given, a poisson distribution
%calculates the time until the next demand

for z = 1:(Ndc-1)
    if PD(z+1) == 1 %If each period contains demand, all values are ...
        generated at once
        seed(:,:,z) = gamrnd((ED(z+1)/sigD(z+1))^2,sigD(z+1)^2/ED(z+1),N,NX);
    else
        for x = 1:NX %If demand is intermittent or lumpy exponential ...
            intervals are put in
            next = geornd(PD(z+1));
            for y = 1:N
                if next == 0
                    seed(y,x,z) = ...
                        gamrnd((ED(z+1)/sigD(z+1))^2,sigD(z+1)^2/ED(z+1),N,NX);
                    next = geornd(PD(z+1));
                else
                    seed(y,x,z) = 0;
                    next = next - 1;
                end
            end
        end
    end
end

%This option is used to visualize 1 simulation, data is recorded in a matrix
if simgraph == 1
    data = zeros(N*2,3,Ndc);
end
%% Loop over NX experiments
for xper = 1:NX;
    %Initialize simulation values
    Y = zeros(Ndc,1);
    for z = 1: Ndc
        Y(z) = S(z) + (1-type(z))*Q(z);
    end
    X = Y;
    D = zeros(Ndc-1,1);
    Dcdc = zeros(Ndc-1,1);
    Dtot = zeros(Ndc,1);
    Btot = zeros(Ndc,1);
    Bcdc = zeros(Ndc-1,1);
    NQtot = zeros(Ndc,1);
    Nstockout = zeros(Ndc,1);
    O = cell(Ndc,1);
    r = zeros(Ndc,1);
    Qcdc = 0;
    start = 0;
    Tempty = 0;
    %Loop over all demands
    for i = 1:N;
        if i > Nstart
            start = 1;
        end
        %Record data for visualization
```

```
        if simgraph == 1
            for z = 1:Ndc
                data((i-1)*2+1,:,z)=[i Y(z) X(z)];
            end
        end
        %Check if there are orders that have arrived
        %Update values for P1
        for z = 1:Ndc
            if size(O{z}) > 0 %If O contains anything there are orders waiting
                O{z}(:,2) = O{z}(:,2) - 1; %Subtract 1 for next time period
                if O{z}(1,2) < 0 %Check for order due
                    NQtot(z) = NQtot(z) + 1*start; %For P1
                    if X(z) < 0
                        Nstockout(z) = Nstockout(z) + 1*start; %For P1
                    end
                    X(z) = X(z) + O{z}(1,1); %Add to net stock
                    if z == 1
                        Qcdc = O{z}(1,1); %Used for CDC backorders
                        if Tempty > 0
                            W(cW,:) = [i-Tempty sum(Bcdc) Qcdc];
                            cW = cW + 1;
                            Tempty = 0;
                        end
                    end
                    O{z}(1,:) = []; %Delete order from O
                end
            end
        end
        %Send backorders to LDC's
        if Qcdc > 0
            if fairshare == 1 && sum(Bcdc) > Qcdc
                Opart = Qcdc/sum(Bcdc);
                for z = 2:Ndc
                    L = gamrnd((EL(z)/sigL(z))^2,sigL(z)^2/EL(z));
                    O{z} = [O{z};Opart*Bcdc(z-1) L];
                    Bcdc(z-1) = (1-Opart)*Bcdc(z-1);
                end
            else
                for z = 2:Ndc
                    if Bcdc(z-1) ≤ Qcdc && Bcdc(z-1) > 0 %If there is ...
                            enough send B
                        L = gamrnd((EL(z)/sigL(z))^2,sigL(z)^2/EL(z));
                        O{z} = [O{z};Bcdc(z-1) L];
                        Qcdc = Qcdc - Bcdc(z-1);
                        Bcdc(z-1) = 0;
                    elseif Qcdc > 0 && Bcdc(z-1) > Qcdc %Send remaining Q ...
                            when finished
                        L = gamrnd((EL(z)/sigL(z))^2,sigL(z)^2/EL(z));
                        O{z} = [O{z};Qcdc L];
                        Bcdc(z-1) = Bcdc(z-1) - Qcdc;
                        Qcdc = 0;
                    end
                end
            end
            Qcdc = 0;
        end
        %Process demand for LDC's
```

```
        for z = 1:Ndc-1
            D(z) = seed(i,xper,z); %Read from demand matrix
            Dtot(z+1) = Dtot(z+1) + D(z)*start; %For P2
            %Update values for P2
            if X(z+1) < 0
                Btot(z+1) = Btot(z+1) + D(z)*start;
            else
                if X(z+1) < D(z)
                    Btot(z+1) = Btot(z+1) + (D(z) - X(z+1))*start;
                end
            end
        end
        %Update inventory levels and record data for visualization
        Y(2:end) = Y(2:end) - D;
        X(2:end) = X(2:end) - D;
        %After each review period update orders, LDC's first
        r = r + 1;
        for z = 2:Ndc
            if r(z) ≥ R(z)
                if type(z) == 1 %Order up to S for (R,S)
                    Dcdc(z-1) = S(z)-Y(z);
                elseif Y(z) < S(z) %Order Q for (R,s,Q)
                    Dcdc(z-1) = ceil((S(z)-Y(z))/Q(z))*Q(z);
                else
                    Dcdc(z-1) = 0;
                end
                Y(z) = Y(z) + Dcdc(z-1);
                Dtot(1) = Dtot(1) + Dcdc(z-1)*start;
                r(z) = 0;
            end
        end
        if max(Dcdc) > 0 %Check orders from LDC to CDC
            if X(1) ≤ 0 %In case of stockout everything backorders
                if Tempty == 0
                    Tempty = i;
                end
                Btot(1) = Btot(1) + sum(Dcdc)*start;
                for z = 1:Ndc-1
                    Bcdc(z) = Bcdc(z) + Dcdc(z);
                end
                X(1) = X(1) - sum(Dcdc);
            elseif X(1) ≥ sum(Dcdc) %If stock is sufficient everyting is sent
                for z = 2:Ndc
                    if Dcdc(z-1) > 0
                        L = gamrnd((EL(z)/sigL(z))^2,sigL(z)^2/EL(z));
                        O{z} = [O{z};Dcdc(z-1) L];
                    end
                end
                X(1) = X(1) - sum(Dcdc);
            else%When no stockout but also insufficient stock
                if Tempty == 0
                    Tempty = i;
                end
                if fairshare == 1
                    Opart = X(1)/sum(Dcdc);
                    for z = 2:Ndc %Send partial order if fair share
                        if Dcdc(z-1) > 0
```

```
                            L = gamrnd((EL(z)/sigL(z))^2,sigL(z)^2/EL(z));
                            O{z} = [O{z};Opart*Dcdc(z-1) L];
                            Bcdc(z-1) = Bcdc(z-1) + (1-Opart)*Dcdc(z-1);
                            Btot(1) = Btot(1) + (1-Opart)*Dcdc(z-1)*start;
                        end
                        X(1) = X(1) - Dcdc(z-1);
                    end
                else
                    for z = 2:Ndc %Check DC's in order
                        if Dcdc(z-1) > 0 && X(1) ≥ Dcdc(z-1) %Send if ...
                            sufficient
                            L = gamrnd((EL(z)/sigL(z))^2,sigL(z)^2/EL(z));
                            O{z} = [O{z};Dcdc(z-1) L];
                        elseif X(1) ≤ 0 %Backorder when finished
                            Bcdc(z-1) = Bcdc(z-1) + Dcdc(z-1);
                            Btot(1) = Btot(1) + Dcdc(z-1)*start;
                        elseif Dcdc(z-1) > 0 %Else send remaining stock
                            L = gamrnd((EL(z)/sigL(z))^2,sigL(z)^2/EL(z));
                            O{z} = [O{z};X(1) L];
                            Bcdc(z-1) = Bcdc(z-1) + Dcdc(z-1) - X(1);
                            Btot(1) = Btot(1) + (Dcdc(z-1) - X(1))*start;
                        end
                        X(1) = X(1) - Dcdc(z-1);
                    end
                end
            end
            Y(1) = Y(1) - sum(Dcdc);
            Dcdc = zeros(Ndc-1,1);
        end
        if r(1) ≥ R(1) %Check orders for CDC when review period passed
            if type(1) == 1 %Order up to S for (R,S)
                Qp = S(1)-Y(1);
                Y(1) = Y(1) + Qp;
                L = gamrnd((EL(1)/sigL(1))^2,sigL(1)^2/EL(1));
                O{1} = [O{1};Qp L];
            elseif Y(1) < S(1) %Or order N*Q for (R,s,Q)
                L = gamrnd((EL(1)/sigL(1))^2,sigL(1)^2/EL(1));
                O{1} = [O{1};ceil((S(1)-Y(1))/Q(1))*Q(1) L];
                Y(1) = Y(1) + ceil((S(1)-Y(1))/Q(1))*Q(1);
            end
            r(1) = 0;
        end
        %Update data for visualization
        if simgraph == 1
            for z = 1:Ndc
                data(2*i,:,z)=[i Y(z) X(z)];
            end
        end
    end
    %Record service levels
    for z = 1:Ndc
        P(xper,2*z-1:2*z) = [(1-Nstockout(z)/NQtot(z))*100 ...
            (1-Btot(z)/Dtot(z))*100];
    end
    if simgraph == 1
        P(1,:)
        break;
```

```
    end
    %Print result of simulation for visualization and terminate loops
end
%Determine confidence interval of service levels
Pavg = mean(P);
Pci = tinv(0.975,NX-1)/sqrt(NX)*std(P);
W = W(1:cW-1,:);
save('WaitingTime','W');
%Plot single graph of visualization or service level results
if simgraph == 1
    figure
    ha(1) = subplot(2,4,[1,2,5,6]);
    hold on
    grid on
    for px = 1:2
        plot(data(:,1,1),data(:,1+px,1));
    end
    legend('Inventory position','Stock level');
    title('Details of simulation run');
    xlabel('Time');
    ylabel('Number of products');
    for z=2:Ndc
        if z > 3
            pl = 3;
        else
            pl = 1;
        end
        ha(z) = subplot(2,4,z+pl);
        hold on
        grid on
        for px = 1:2
            plot(data(:,1,z),data(:,1+px,z));
        end
        plot(data(:,1),S(z)*ones(size(data(:,1),1),1))
        xlabel('Time');
        ylabel('Number of products');
    end
    linkaxes(ha,'x');
else
    figure
    hold on
    grid on
    grid MINOR
    for px = 1:2
        errorbar(1:Ndc,Pavg(px:2:end),Pci(px:2:end),'.','Color',[px/2 0 ...
            0],'DisplayName',['P' num2str(px) 'Matlab']);
    end
%    plot(RES(:,1),95*ones(size(RES,1),1),'k','DisplayName','95% line');
    legend(gca,'show')
    title('model results');
    xlabel('Number of DC (1=CDC)');
    ylabel('Service level (%)');
end
```

Listing B.1: Multi-echelon model

Appendix C

Multi-echelon optimization scripts

This appendix contains all scripts used for multi-echelon optimization, the main script and all subscripts are described in Section 4.3.

```
clearvars; close all;

%% Initialize
input = xlsread('Input.xlsx','Sheet1'); %Read input parameters
Ptarget = 100*input(10,2:end); %Turn service level targets into percentage
save('Ptarget.mat','Ptarget'); %Save for use in constraint functions
Ndc = size(input,2); %Number of DCs
Nldc = Ndc-1; %Number of LDCs
Nexp = 10; %Number of experiments for confidence interval calculation
Nintervals = 10; %Number of intervals in safety stock simulation range

%Determine standard formula order-up-to levels or re-order points
EDDUP = zeros(Ndc,1);
for i = 1:Ndc
    EDDUP(i) = input(2,i)*(input(7,i)*input(1,i)+input(5,i))*input(4,i);
end

%% Determine simulation ranges
%SE_fit returns approximate 0% and 100% safety stocks
CDCbounds = SE_fit(input(:,1));
CDCrange = linspace(round(CDCbounds(1)),round(CDCbounds(2)),Nintervals);
%CDC lead time (variance) is added to LDC to determine upper LDC bounds
CDCLvalues = zeros(size(input,1),1);
CDCLvalues(5) = input(5,1);
CDCLvalues(6) = input(6,1);
LDCranges = cell(Nldc,1);
for i = 2:Ndc %Determine LDC ranges
    output = SE_fit(input(:,i));
    low = round(output(1));
    output = SE_fit(input(:,i)+CDCLvalues);
    high = round(output(2));
    LDCranges{i-1} = linspace(low,high,Nintervals);
end

Results = zeros(10^3,3*(Ndc)); %Create results matrix
k=1;
%% Perform simulations for whole range
value = zeros(1,Ndc);
for i = 1:Nintervals
    for j = 1:Nintervals
        value(1) = CDCrange(i);
        for l = 1:Nldc
            value(l+1) = LDCranges{l}(j);
        end
        Results(k,:)=MEIO_optim_v3(value,input);
```

```
        k = k+1;
    end
end
Results = Results(1:k-1,:); % Trim away zeros

%% Fit service level target line for each LDC
fitvalues = zeros(4,Nldc);
Scdc = Results(:,1);
for i = 1:Nldc
    Weigths = ones(size(Results,1),1);
    Sldc = Results(:,i+1);
    Pldc = Results(:,Ndc+2*i+input(9,i+1)); %Choose P1 or P2
    for j = 1:size(Scdc,1)
        if Pldc(j) > 70 && Pldc(j) < 99
            Weigths(j) = Pldc(j)/70*2;%Increase weight in relevant range
        end
    end
    fitvalues(:,i) = ...
        FindTargetLine(Scdc,Sldc,Pldc,Weigths,CDCrange,LDCranges{i},Ptarget(i));
end
save('fitvalues.mat','fitvalues'); %Save for use in constraint functions

%% Initiate optimization
options = optimset('fmincon');
options = optimset(options,'MaxFunEvals',10000,'TolFun',1E-3);
x0 = zeros(1,Ndc);
x0(1) = CDCrange(end); %Start at P=100%
for i = 1:Nldc %Find appropriate LDC values from fits
    x0(i+1) = ...
        fitvalues(1,i)-fitvalues(2,i)*normcdf(x0(1),fitvalues(3,i),fitvalues(4,i));
end
lb = zeros(1,Ndc);
ub = value; %Last simulation point

%Perform constrained optimization
[CurrOptimum,fval,exitflag,output,lambda,grad]=...
    fmincon(@objfunMEIO,x0,[],[],[],[],lb,ub,@confunMEIOPline,options);

if exitflag < 1 %Stop if no optimum available
    print = 'Optimum not found, pleasy retry or change parameters'
    return
end

%% Initiate optimum iteration
Pdiff = zeros(1,Nldc); %Difference between requirement and results
finished = 0; %Becomes 1 when final optimum is found
Iterations = 0; %Increases up to a defined maximum
while finished == 0
    %Initiate new results matrix
    RES2 = zeros(4,2*(Ndc)-1);
    x = CurrOptimum; %x is used in simulations in MEIO_withCI
    MEIO_withCI; %Determine Pavg and Pci, the average and CI of x
    RES2(1,:) = [x Pavg];

    %Determine difference between current optimum and service level target
    for i = 1:Nldc
        if Pavg(i)+Pci(i) > Ptarget(i) && Pavg(i)-Pci(i) < Ptarget(i)
```

```
            Pdiff(i) = 0; %Count as 0 if within confidence interval
        else
            Pdiff(i) = Ptarget(i)-Pavg(i);
        end
    end

    %If result not within confidence interval, reiterate
    if not(isequal(Pdiff,zeros(1,Nldc)))
        ReOptimize;
        Iterations = Iterations + 1;
    else
        finished = 1; %exit loop when optimum found
    end
    if Iterations ≥ 10 %Exit if optimum is not found within 10 times
        print = 'Optimum not found, pleasy retry or change parameters'
        return
    end
end
FinalOptimum = CurrOptimum - EDDUP' %Print final safety stock results
SStot = sum(FinalOptimum)
P1final
```

Listing C.1: Multi-echelon optimization main script

```
function output = MEIO_optim_v3(Svalues,input)

%Simulation properties
simscale = 1; %Changes the amount of steps in 1 demand period
Nstart = 100*simscale;
N = (10000+Nstart)*simscale; %Number of demands generated
simgraph = 0; %if this is 1 the simulation will run once while recording ...
    values of Y and X
fairshare = 1;

values = input;

%System parameters are vectors containing the properties for each DC
Ndc = size(values,2); %Number of DC's = Number of rows
ED = zeros(Ndc,1);
sigD = zeros(Ndc,1);
PD = zeros(Ndc,1);
EL = zeros(Ndc,1);
sigL = zeros(Ndc,1);
R = zeros(Ndc,1);
Q = zeros(Ndc,1);
S = zeros(Ndc,1);
type = zeros(Ndc,1);

%Extract correct values from matrix for each DC
for z = 1:Ndc
    type(z) = values(1,z);
    ED(z) = values(2,z)/simscale;
    sigD(z) = values(3,z)/simscale;
    PD(z) = values(4,z)/simscale;
    EL(z) = values(5,z)*simscale;
    sigL(z) = values(6,z)*simscale;
```

```
    R(z) = values(7,z)*simscale;
    Q(z) = values(8,z)*ED(z)*R(z);
    S(z) = Svalues(z);
end

%Initialization
P = zeros(1,2*Ndc);%Service level matrix
seed = zeros(N,Ndc-1);
%actualdemand = zeros(Ndc,2);

%One table of demands is generated per experiment, different values of s
%use the same table When a lambda is given, a poisson distribution
%calculates the time until the next demand

for z = 1:(Ndc-1)
    if PD(z+1) == 1 %If each period contains demand, all values are ...
        generated at once
        seed(:,z) = gamrnd((ED(z+1)/sigD(z+1))^2,sigD(z+1)^2/ED(z+1),N,1);
    else
        next = geornd(PD(z+1));
        for y = 1:N
            if next == 0
                seed(y,z) = gamrnd((ED(z+1)/sigD(z+1))^2,sigD(z+1)^2/ED(z+1));
                next = geornd(PD(z+1));
            else
                seed(y,z) = 0;
                next = next - 1;
            end
        end
    end
end

%This option is used to visualize 1 simulation, data is recorded in a matrix
if simgraph == 1
    data = zeros(N*2,3,Ndc);
end
%% Run experiment
%Initialize simulation values
Y = zeros(Ndc,1);
for z = 1: Ndc
    Y(z) = S(z) + (1-type(z))*Q(z);
end
X = Y;
D = zeros(Ndc-1,1);
Dcdc = zeros(Ndc-1,1);
Dtot = zeros(Ndc,1);
Btot = zeros(Ndc,1);
Bcdc = zeros(Ndc-1,1);
NQtot = zeros(Ndc,1);
Nstockout = zeros(Ndc,1);
O = cell(Ndc,1);
r = zeros(Ndc,1);
Qcdc = 0;
start = 0;
%Loop over all demands
for i = 1:N;
    if i > Nstart
```

```
            start = 1;
        end
        %Record data for visualization
        if simgraph == 1
            for z = 1:Ndc
                data((i-1)*2+1,:,z)=[i Y(z) X(z)];
            end
        end
        %Check if there are orders that have arrived
        %Update values for P1
        for z = 1:Ndc
            if size(O{z}) > 0 %If O contains anything there are orders waiting
                O{z}(:,2) = O{z}(:,2) - 1; %Subtract 1 for next time period
                if O{z}(1,2) < 0 %Check for order due
                    NQtot(z) = NQtot(z) + 1*start; %For P1
                    if X(z) < 0
                        Nstockout(z) = Nstockout(z) + 1*start; %For P1
                    end
                    X(z) = X(z) + O{z}(1,1); %Add to net stock
                    if z == 1
                        Qcdc = O{z}(1,1); %Used for CDC backorders
                    end
                    O{z}(1,:) = []; %Delete order from O
                end
            end
        end
        %Send backorders to LDC's
        if Qcdc > 0
            if fairshare == 1 && sum(Bcdc) > Qcdc
                Opart = Qcdc/sum(Bcdc);
                for z = 2:Ndc
                    L = gamrnd((EL(z)/sigL(z))^2,sigL(z)^2/EL(z));
                    O{z} = [O{z};Opart*Bcdc(z-1) L];
                    Bcdc(z-1) = (1-Opart)*Bcdc(z-1);
                end
            else
                for z = 2:Ndc
                    if Bcdc(z-1) ≤ Qcdc && Bcdc(z-1) > 0 %If there is enough ...
                        send B
                        L = gamrnd((EL(z)/sigL(z))^2,sigL(z)^2/EL(z));
                        O{z} = [O{z};Bcdc(z-1) L];
                        Qcdc = Qcdc - Bcdc(z-1);
                        Bcdc(z-1) = 0;
                    elseif Qcdc > 0 && Bcdc(z-1) > Qcdc %Send remaining Q when ...
                        finished
                        L = gamrnd((EL(z)/sigL(z))^2,sigL(z)^2/EL(z));
                        O{z} = [O{z};Qcdc L];
                        Bcdc(z-1) = Bcdc(z-1) - Qcdc;
                        Qcdc = 0;
                    end
                end
            end
            Qcdc = 0;
        end
        %Determine demand for LDC's
        for z = 1:Ndc-1
            D(z) = seed(i,z); %Read from demand matrix
```

```
        Dtot(z+1) = Dtot(z+1) + D(z)*start; %For P2
        %Update values for P2
        if X(z+1) < 0
            Btot(z+1) = Btot(z+1) + D(z)*start;
        else
            if X(z+1) < D(z)
                Btot(z+1) = Btot(z+1) + (D(z) - X(z+1))*start;
            end
        end
    end
    %Update inventory levels and record data for visualization
    Y(2:end) = Y(2:end) - D;
    X(2:end) = X(2:end) - D;
    %After each review period update orders, LDC's first
    r = r + 1;
    for z = 2:Ndc
        if r(z) ≥ R(z)
            if type(z) == 1 %Order up to S for (R,S)
                Dcdc(z-1) = S(z)-Y(z);
            elseif Y(z) < S(z) %Order Q for (R,s,Q)
                Dcdc(z-1) = ceil((S(z)-Y(z))/Q(z))*Q(z);
            else
                Dcdc(z-1) = 0;
            end
            Y(z) = Y(z) + Dcdc(z-1);
            Dtot(1) = Dtot(1) + Dcdc(z-1)*start;
            r(z) = 0;
        end
    end
    if sum(Dcdc) > 0 %Check orders to CDC
        if X(1) ≤ 0 %In case of stockout everything backorders
            Btot(1) = Btot(1) + sum(Dcdc)*start;
            for z = 1:Ndc-1
                Bcdc(z) = Bcdc(z) + Dcdc(z);
            end
            X(1) = X(1) - sum(Dcdc);
        elseif X(1) ≥ sum(Dcdc) %If stock is sufficient everyting is sent
            for z = 2:Ndc
                if Dcdc(z-1) > 0
                    L = gamrnd((EL(z)/sigL(z))^2,sigL(z)^2/EL(z));
                    O{z} = [O{z};Dcdc(z-1) L];
                end
            end
            X(1) = X(1) - sum(Dcdc);
        else%When no stockout but also insufficient stock
            if fairshare == 1
                Opart = X(1)/sum(Dcdc);
                for z = 2:Ndc %Check DC's in order
                    if Dcdc(z-1) > 0
                        L = gamrnd((EL(z)/sigL(z))^2,sigL(z)^2/EL(z));
                        O{z} = [O{z};Opart*Dcdc(z-1) L];
                        Bcdc(z-1) = Bcdc(z-1) + (1-Opart)*Dcdc(z-1);
                        Btot(1) = Btot(1) + (1-Opart)*Dcdc(z-1)*start;
                    end
                    X(1) = X(1) - Dcdc(z-1);
                end
            else
```

```
                    for z = 2:Ndc %Check DC's in order
                        if Dcdc(z-1) > 0 && X(1) ≥ Dcdc(z-1) %Send if sufficient
                            L = gamrnd((EL(z)/sigL(z))^2,sigL(z)^2/EL(z));
                            O{z} = [O{z};Dcdc(z-1) L];
                        elseif X(1) ≤ 0 %Backorder when finished
                            Bcdc(z-1) = Bcdc(z-1) + Dcdc(z-1);
                            Btot(1) = Btot(1) + Dcdc(z-1)*start;
                        elseif Dcdc(z-1) > 0 %Else send remaining stock
                            L = gamrnd((EL(z)/sigL(z))^2,sigL(z)^2/EL(z));
                            O{z} = [O{z};X(1) L];
                            Bcdc(z-1) = Bcdc(z-1) + Dcdc(z-1) - X(1);
                            Btot(1) = Btot(1) + (Dcdc(z-1) - X(1))*start;
                        end
                        X(1) = X(1) - Dcdc(z-1);
                    end
                end
            end
            Y(1) = Y(1) - sum(Dcdc);
            Dcdc = zeros(Ndc-1,1);
        end
        if r(1) ≥ R(1) %Check orders for CDC when review period passed
            if type(1) == 1 %Order up to S for (R,S)
                Qp = S(1)-Y(1);
                Y(1) = Y(1) + Qp;
                L = gamrnd((EL(1)/sigL(1))^2,sigL(1)^2/EL(1));
                O{1} = [O{1};Qp L];
            elseif Y(1) < S(1) %Or order N*Q for (R,s,Q)
                L = gamrnd((EL(1)/sigL(1))^2,sigL(1)^2/EL(1));
                O{1} = [O{1};ceil((S(1)-Y(1))/Q(1))*Q(1) L];
                Y(1) = Y(1) + ceil((S(1)-Y(1))/Q(1))*Q(1);
            end
            r(1) = 0;
        end
        %Update data for visualization
        if simgraph == 1
            for z = 1:Ndc
                data(2*i,:,z)=[i Y(z) X(z)];
            end
        end
    end
    %Calculate service levels
    for z = 1:Ndc
        P(2*z-1:2*z) = [(1-Nstockout(z)/NQtot(z))*100 (1-Btot(z)/Dtot(z))*100];
    end

    output = [S' P];
```

Listing C.2: Multi-echelon optimization simulation script

```
function [vals] = FindTargetLine(Scdc, Sldc, Pldc, W, CDCrange, LDCrange, ...
    Ptarget)
%% Initialize
check = Scdc(1); %Used to check for next Scdc value
temp = []; %Will contain results for 1 Scdc value
k = 1; %counter
midpointguess = LDCrange(end);%Should be close to last value in range
```

```
TargetPoints = zeros(10^3,1);

%% Loop fit over all CDC safety stock values and find P targets
for l = 1:size(Scdc,1)
    if Scdc(l) ≠ check %perform fit when at new Scdc value
        [linfun, gof] = ...
            fit(temp(:,1),temp(:,2),'100/(1+exp(-k*(x-x0)))','StartPoint',[0.05 ...
            midpointguess],'Lower',[0 1],'Upper',[1 ...
            20000],'Weights',temp(:,3));
        objective = @(x) linfun(x) - Ptarget;
        TargetPoints(k) = fzero(objective,linfun.x0); %Target Sldc value
        temp = []; %Reinitialize
        check = Scdc(l);
        k = k + 1;
    end
    temp = [temp;Sldc(l) Pldc(l) W(l)]; %Add current Scdc values
    if Pldc(l) > 20 && Pldc(l) < 80
        midpointguess = Sldc(l); %Roughly estimate midpoint guess for fit
    end
end
%Perform fit on final list of values
[linfun, gof] = ...
    fit(temp(:,1),temp(:,2),'100/(1+exp(-k*(x-x0)))','StartPoint',[0.05 ...
    midpointguess],'Lower',[0 1],'Upper',[1 20000],'Weights',temp(:,3));
objective = @(x) linfun(x) - Ptarget;
TargetPoints(k) = fzero(objective,linfun.x0);
TargetPoints = TargetPoints(1:k);%Trim matrix
%Save results for visual comparison
load('TargetPointsLDC.mat')
TargetPointsLDC = [TargetPointsLDC TargetPoints];
save('TargetPointsLDC.mat','TargetPointsLDC')

%% Perform final curvefit
%Roughly estimate coefficients based on target points
aguess = TargetPoints(1);
bguess = TargetPoints(1)-TargetPoints(end);
cguess = CDCrange(round((k)/2));
dguess = cguess - CDCrange(round((k)/4));

%Fit final line and return line coefficients
Pline = ...
    fit(CDCrange',TargetPoints,'a-b*normcdf(x,c,d)','StartPoint',[aguess ...
    bguess cguess dguess],'Lower',0.1*[aguess bguess cguess ...
    dguess],'Upper',10*[aguess bguess cguess dguess]);
vals = coeffvalues(Pline);
```

Listing C.3: Multi-echelon fitting function

```
%Script to run the simulation Nexp times and determine confidence interval
    %for the appropriate service level
P1final = 0; %To use in comparison
P = zeros(Nexp,Nldc);
%% Run experiments
for i = 1:Nexp
    y = MEIO_optim_v3(x,input);
    P1final = P1final + y(Ndc+1);
```

```
    for j = 1:Nldc
        P(i,j) = y(Nldc+3+input(9,j+1)+(j-1)*2);
    end
end

%% Determine average and confidence interval
P1final = P1final/Nexp;
Pavg = mean(P);
Pci = tinv(0.95,Nexp-1)/sqrt(Nexp)*std(P);
```

Listing C.4: Multi-echelon simulation script with confidence interval calculation

```
%First determine new LDC simulation values based on Pdiff
    % interval of twice the difference between result and target
for i = 1:Nldc
    if Pdiff(i) > 0
        lb(i+1) = x(i+1);
        x(i+1) = x(i+1) + 2*0.01*Pdiff(i)*(LDCranges{i}(end)-LDCranges{i}(1));
        ub(i+1) = x(i+1);
    elseif Pdiff(i) < 0
        ub(i+1) = x(i+1);
        x(i+1) = x(i+1) + 2*0.01*Pdiff(i)*(LDCranges{i}(end)-LDCranges{i}(1));
        lb(i+1) = x(i+1);
    else
        if mean(Pdiff) > 0
            lb(i+1) = x(i+1);
            x(i+1) = x(i+1) + ...
                2*0.01*mean(Pdiff)*(LDCranges{i}(end)-LDCranges{i}(1));
            ub(i+1) = x(i+1);
        elseif mean(Pdiff) < 0
            ub(i+1) = x(i+1);
            x(i+1) = x(i+1) + ...
                2*0.01*mean(Pdiff)*(LDCranges{i}(end)-LDCranges{i}(1));
            lb(i+1) = x(i+1);
        end
    end
end

%Simulate LDC new and CDC old values
MEIO_withCI;
RES2(2,:) = [x Pavg];

%Determine new CDC value based on average of Pdiff
if abs(Pdiff) == Pdiff
    lb(1) = x(1);
    x(1) = x(1) + 0.01*max(Pdiff)*(CDCrange(end)-CDCrange(1));
    ub(1) = x(1);
elseif -abs(Pdiff) == Pdiff
    ub(1) = x(1);
    x(1) = x(1) - 0.01*max(abs(Pdiff))*(CDCrange(end)-CDCrange(1));
    lb(1) = x(1);
else
    if mean(Pdiff) > 0
        lb(1) = x(1);
        x(1) = x(1) + 2*0.01*mean(Pdiff)*(CDCrange(end)-CDCrange(1));
        ub(1) = x(1);
```

```
    elseif mean(Pdiff) < 0
        ub(1) = x(1);
        x(1) = x(1) + 2*0.01*mean(Pdiff)*(CDCrange(end)-CDCrange(1));
        lb(1) = x(1);
    end
end

%Simulate new CDC value with new LDC value
MEIO_withCI;
RES2(3,:) = [x Pavg];

%Return LDC values to original
x(2:end) = CurrOptimum(2:end);

%Simulate old LDC with new CDC values
MEIO_withCI;
RES2(4,:) = [x Pavg];



%Linear fitting based on new simulations
fitvalueslin = zeros(3,Nldc);
Scdc = RES2(:,1);
for i = 1:Nldc
    Sldc = RES2(:,i+1);
    Pldc = RES2(:,Nldc+1+i);
    fitvalueslin(:,i) = createFitLin(Scdc,Sldc,Pldc);
end
save('fitvalueslin.mat','fitvalueslin');

options = optimset('fmincon');
options = optimset(options,'MaxFunEvals',1000,'TolFun',1E-3);

% Initial guess in middle of range
x0 = 0.5.*lb + 0.5.*ub;

%Perform optimization
[CurrOptimum,fval,exitflag,output,lambda,grad]=...
fmincon(@objfunMEIO,x0,[],[],[],[],lb,ub,@confunMEIOLin,options);
```

Listing C.5: Multi-echelon optimization iteration

```
function fitvalues = createFitLin(X, Y, Z)

% Fit model to data and return coefficients
[fitresult, gof] = fit( [X, Y], Z, fittype('poly11'));

fitvalues = coeffvalues(fitresult);
```

Listing C.6: Multi-echelon local linear fit function

```
function [ineq,eq] = confunMEIOLin(x)
% Import values
load('Ptarget.mat');
load('fitvalueslin.mat');
```

```
Nldc = size(fitvalueslin,2);
% Recreate fit formulas from fitvalues
s = zeros(Nldc,1);
for i = 1:Nldc
    s(i) = fitvalueslin(1,i) + fitvalueslin(2,i)*x(1) + ...
        fitvalueslin(3,i)*x(i+1);
end
% Constraints
ineq = Ptarget' - s; %Inequality constraints
eq = []; % no equality constraints
```

Listing C.7: Multi-echelon linear fit constraint function

Appendix D

Multi-echelon validation scripts

The script in this appendix is used to compare multi-echelon simulation results to analytical approximations, as described in Section 4.4.

```
%To be run after Main_v2_servicetime.m
load('TargetPointsLDC.mat')
SS = CDCrange - Snorm(1);
SSsimtot = SS;
for i = 1:Nintervals
    SSsimtot(i) = SSsimtot(i) + sum(TargetPointsLDC(i,:)) - sum(Snorm(2:end));
end

%Plot simulation results
figure
hold on
plot(SS,SSsimtot,'LineWidth',2,'DisplayName','Simul. total safety stock')
set(gca,'XDir','reverse')
xlabel('CDC safety stock')
ylabel('Total safety stock')

%% Analytical results
%Get input parameters
ED = zeros(Ndc,1);
sigD = zeros(Ndc,1);
PD = zeros(Ndc,1);
for z = 1:Ndc
    ED(z) = input(2,z);
    sigD(z) = input(3,z);
    PD(z) = input(4,z);
end
seed = zeros(10^4,Nldc);
actualdemand = zeros(Ndc,2);

%Get actual demand in case of intermittent or lumpy
for z = 1:(Nldc)
    if PD(z+1) == 1 %If each period contains demand, all values are ...
        generated at once
        seed(:,z) = gamrnd((ED(z+1)/sigD(z+1))^2,sigD(z+1)^2/ED(z+1),10^4,1);
    else
        next = geornd(PD(z+1));
        for y = 1:10^4
            if next == 0
                seed(y,z) = gamrnd((ED(z+1)/sigD(z+1))^2,sigD(z+1)^2/ED(z+1));
                next = geornd(PD(z+1));
            else
                seed(y,z) = 0;
                next = next - 1;
            end
        end
```

```
    end
    actualdemand(z+1,:) = [mean(seed(:,z)) std(seed(:,z))];
end

actualdemand(1,:) = [sum(actualdemand(2:end,1)) ...
    sqrt(sumsqr(actualdemand(2:end,2)))];
%Strings used in plots
methods{1} = 'Desmet';
methods{2} = 'Desmet with adjusted variance';
methods{3} = 'Desmet with adjusted waiting time';
methods{4} = 'Dendauw adjusted P1';
methods{5} = 'Adjustments combined';

methods{6} = '--';
methods{7} = ':';
methods{8} = '-*';
methods{9} = '-^';
methods{10} = '-o';

for m = 1:5
    SStot = SS;
    %Calculate SS values for same points as simulation results
    for i = 1:Nintervals
        for z = 1:Ndc
            %Turn undershoot calculation on or off for RsQ
            Ushoot = 1;

            %Get final input parameters
            type = input(1,z);
            ED = actualdemand(z,1);
            sigD = actualdemand(z,2);
            R = input(7,z);
            Q = input(8,z)*ED*R;
            P1LDC = input(10,z);

            %Determine adjusted lead time parameters
            if z == 1
                EL = input(5,1);
                sigL = input(6,1);
            else
                if m == 1 || m == 4 %Formula Desmet
                    EL = input(5,z) + (1-P1CDC)*input(5,1);
                    sigL = sqrt(input(6,z)^2 + (1-P1CDC)^2*input(6,1)^2);
                elseif m == 2  %Adjustment Adan & Lefeber
                    EL = input(5,z) + (1-P1CDC)*input(5,1);
                    sigL = sqrt(input(6,z)^2 + (1-P1CDC)^2*input(6,1)^2 + ...
                        (1-P1CDC)*P1CDC*input(5,1)^2);
                elseif m == 3 %Adjusted waiting time
                    EL = input(5,z) + (1-P1CDC)*feval(muline,CDCrange(i));
                    sigL = sqrt(input(6,z)^2 + ...
                        (1-P1CDC)^2*feval(sigline,CDCrange(i))^2);
                else %Adjusted everything
                    EL = input(5,z) + (1-P1CDC)*feval(muline,CDCrange(i));
                    sigL = sqrt(input(6,z)^2 + ...
                        ((1-P1CDC)^2*feval(sigline,CDCrange(i))^2 + ...
                        (1-P1CDC)*P1CDC*feval(mulineWS,CDCrange(i))));
                end
```

```
            end

            %Determine uncertainty period parameters
            sigX = sqrt((EL+type*R)*sigD^2+ED^2*sigL^2);
            DDUP = (EL+type*R)*ED;

            %Calculate Gamma distribution parameters
            if type == 0 && Ushoot == 1
                EU = (sigD^2*R+ED^2*R^2)/(2*R*ED);
                sigU = sqrt((1+((sigD/ED)^2)/R) * (1+2*(sigD/ED)^2/R) * ...
                    (ED*R)^2/3-EU^2);
                EUDDUP = EU + DDUP;
                sigUDDUP = sqrt(sigU^2+sigX^2);
                alpha = (EUDDUP/sigUDDUP)^2;
                beta = sigUDDUP^2/EUDDUP;
            else
                alpha = (DDUP/sigX)^2;
                beta = sigX^2/DDUP;
            end

            %Calculate safety stocks
            if z == 1
                if m > 3
                    sigmaDD = ...
                        sqrt((input(5,1)+input(1,1)*input(7,1))*actualdemand(1,2)^2+actualdemand
                    formuleDD = (0.5*(input(1,1) + ...
                        (1-input(1,1)*input(8,1)))*actualdemand(1,1)*input(7,1) ...
                        + SS(i))/sigmaDD;
                    P1CDC = normcdf(formuleDD,0,1); %Formula Dendauw
                else
                    P1CDC = gamcdf(SS(i)+Snorm(1),alpha,beta); %Formula Desmet
                end
            else
                SStot(i) = SStot(i) + gaminv(P1LDC,alpha,beta) - Snorm(z);
            end
        end
    end
    %Plot in same plot as simulation results
    plot(SS,SStot,methods{m+5},'LineWidth',2,'DisplayName',[methods{m} ' ...
        total safety stock'])
end
% Plot final simulation optimum
plot(FinalOptimum(1),sum(FinalOptimum),'*','DisplayName','Final sim. ...
    optimum total safety stock')
legend('off');
legend(gca,'show');
legend('Location','southwest');
title('Total multi-echelon safety stock vs. CDC safety stock')
```

Listing D.1: Multi-echelon comparison script

D.1 Waiting time processing script

The script in this appendix is used to process lists of waiting times into Gamma distributions, as described in Section 4.4.

```
%Run after main_v2_servicetime;
Coeffs = zeros(Nintervals,2);
CoeffsWS = zeros(Nintervals,2); %Squared waiting time for adjusted variance
WaitingTimeSquared = cell(Nintervals,1);
CDCrangeWT = CDCrange;
ToDelete = [];
% Fit distribution parameters or delete value if not waiting data
for w = 1:Nintervals
    if size(WaitingTime{w}(:,1),1) > 1
        Dist = fitdist(WaitingTime{w}(:,1),'Gamma');
        Coeffs(w,:) = [Dist.a*Dist.b sqrt(Dist.a*Dist.b^2)];
        WaitingTimeSquared{w} = WaitingTime{w}(:,1).^2;
        DistWS = fitdist(WaitingTimeSquared{w},'Gamma');
        CoeffsWS(w,:) = [DistWS.a*DistWS.b sqrt(DistWS.a*DistWS.b^2)];
    else
        ToDelete = [ToDelete;w];
    end
end
td = size(ToDelete,1);
Coeffs = Coeffs(1:w-td,:);
CoeffsWS = CoeffsWS(1:w-td,:);
if td > 0
    for i = 1:td
        CDCrangeWT(ToDelete(td+1-i)) = [];
    end
end


%% Fit mu and sigma curves
%Roughly estimate coefficients
aguess = input(7,1);
bguess = aguess - input(6,1);
cguess = CDCrange(round(Nintervals/2));
dguess = CDCrange(round(Nintervals/4));

muline = ...
    fit(CDCrangeWT',Coeffs(:,1),'a-b*normcdf(x,c,d)','StartPoint',[aguess ...
    bguess cguess dguess],'Lower',0.1*[aguess bguess cguess ...
    dguess],'Upper',10*[aguess bguess cguess dguess]);
mulineWS = ...
    fit(CDCrangeWT',CoeffsWS(:,1),'a-b*normcdf(x,c,d)','StartPoint',[1.5*aguess^2 ...
    1.5*aguess^2 cguess dguess],'Lower',0.1*[1.5*aguess^2 1.5*aguess^2 ...
    cguess dguess],'Upper',100*[1.5*aguess^2 1.5*aguess^2 cguess dguess]);

%Roughly estimate coefficients
aguess = 0.5*input(7,1);
bguess = aguess - 0.5*input(6,1);
cguess = CDCrange(round(Nintervals/2));
dguess = CDCrange(round(Nintervals/4));

sigline = ...
    fit(CDCrangeWT',Coeffs(:,2),'a-b*normcdf(x,c,d)','StartPoint',[aguess ...
    bguess cguess dguess],'Lower',0.1*[aguess bguess cguess ...
    dguess],'Upper',10*[aguess bguess cguess dguess]);
```

Listing D.2: Multi-echelon comparison script

Appendix E

Multi-echelon validation scripts

The calculation of the "mean of square minus square of mean" for the variance equation of the Desmet formula, as mentioned in Section 4.4.5 is shown in this appendix. The "mean of square minus square of mean" rule is defined as:

$$\sigma_x^2 = E(x^2) - E^2(x).$$

The new LDC lead time is:

$$L_{LDC} \text{ with probability } P_{1,CDC},$$

$$L_{LDC} + L_{CDC} \text{ with probability } 1 - P_{1,CDC}.$$

These can be used in the "mean of square minus square of mean" rule:

$$E({L_{LDC}^*}^2) = P_{1,CDC} * E^2(L_{LDC}) + (1 - P_{1,CDC}) * (E(L_{LDC}^2) + E(L_{CDC}^2) + 2E(L_{LDC})E(L_{CDC})),$$

$$E^2(L_{LDC}^*) = E^2(L_{LDC}) + (1 - P_{1,CDC})^2 E^2(L_{CDC}) + 2(1 - P_{1,CDC})E(L_{LDC})E(L_{CDC}),$$

combining into:

$$\sigma_{L_{LDC}^*}^2 = (P_{1,CDC} - 1) * E^2(L_{LDC}) + (1 - P_{1,CDC}) * (E(L_{LDC}^2) + E(L_{CDC}^2) - (1 - P_{1,CDC})^2 E^2(L_{CDC}).$$

Using the rule backwards this can be simplified into:

$$\sigma_{L_{LDC}^*}^2 = \sigma_{L_{LDC}}^2 + (1 - P_{1,CDC})\sigma_{L_{CDC}}^2 + (1 - P_{1,CDC})P_{1,CDC}E^2(L_{CDC}).$$

This can be transformed into a function similar to the Desmet equation:

$$\sigma_{L_{LDC}^*}^2 = \sigma_{L_{LDC}}^2 + (1 - P_{1,CDC})^2\sigma_{L_{CDC}}^2 + (1 - P_{1,CDC})P_{1,CDC}E(L_{CDC}^2).$$