LUNDS UNIVERSITET
Lunds Tekniska Högskola

Mathematical Modeling of Inventory Control Systems with Lateral Transshipments

Lina Johansson

Master Thesis

Department of Industrial Management and Logistics
Division of Production Management
Lund university, Faculty of Engineering LTH

Supervisor: Fredrik Olsson
Examiner: Peter Berling

Keywords: Inventory Control, Lateral Transshipments, Mathematical Modeling.

Lina Johansson (π 06)
lina.johansson@iml.lth.se

Lund 2011

Abstract

This master thesis is about the mathematical modeling of an inventory control system. The system is a single-echelon two-location system with one type of item. The locations replenish from a supplier that is assumed to have an ample stock. The demand is modeled as customers arriving according to a Poisson process with a known intensity and each customer only demands one item. In addition to replenishments from the normal supplier, the two locations also have the opportunity of using bidirectional lateral transshipments. This means that a location facing stock out may get a delivery from the other location, given that the sending location has stock on hand. The lateral transshipments are faster than normal replenishments, but the transshipment cost is larger than the normal order cost. The locations both apply continuous review and either an $(S-1,S)$ ordering policy or an (R,Q) policy. A mathematical model has been developed to simulate the inventory control system. Three different versions of the model have been made; one without lateral transshipments, one where the lateral transshipments have zero leadtime and one where the transshipment leadtime is positive. These three alternatives have then been evaluated and compared for the two ordering policies.

Preface

This report is the result of my master thesis which has been made at the Division of Production Management at LTH. I would like to thank my supervisor Fredrik Olsson for his help throughout this project. Getting to learn about inventory control has been a very interesting experience and I look forward to learning more.

I also would like to thank Christian for his support and especially my parents for always backing me up.

Contents

Chapter 1

Introduction

In the first chapter of this report the problem in focus of this project is formulated. In Chapter 2 an introduction to the theory of inventory control is given. A short literature review is presented in the following chapter and in Chapter 4 the method used is discussed. After that, the results are presented in Chapter 5 and the conclusions in Chapter 6. A list of the notations used can be found in Appendix A.

1.1 Problem formulation

The aim of this master thesis is to model a specific inventory system with lateral transshipments, evaluate the system under a given transshipment policy and optimize the decision variables.

The inventory system in question is described in the next section.

1.1.1 The inventory system in focus

The inventory system is a divergent and single-echelon system with two locations and a common central supplier. The inventory system is presented in Figure 1.1.

For this project, there are some basic assumptions and delimitations regarding the model.

- The stochastic customer demand occur only at the two locations and the demand process at the locations follows two independent Poisson processes. Customers arrive one at a time and they demand only one item.

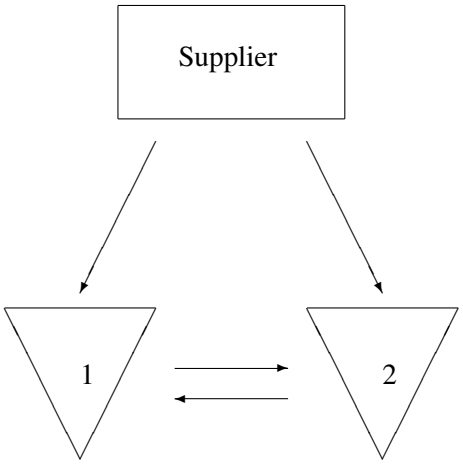

Figure 1.1: The inventory system at hand.

- The two locations normally replenish stock from the supplier according to an $(S-1,S)$ or (R,Q) policy.
- The supplier is assumed to have ample or infinitely large stock, which means that there never is stock-out at the supplier and all ordered items will arrive after the lead time.
- Lateral transshipments are allowed between the two locations. The transshipments are bidirectional and the two locations share all their inventory.
- The replenishment lead time for the two locations and the transshipment lead time are fixed.
- If the demand cannot be satisfied, the items are backordered.

Lateral transshipments are, in this project, made when a customer arrives to a location that faces stock out while the other location has stock on hand. In cases where the transshipment lead time is positive, a lateral transshipment is only made if the residual time until an item already ordered from the supplier arrives at the

location exceeds the transshipment lead time. That is, if it takes less time waiting for an incoming item than to make a lateral transshipment. Other transshipment rules may for instance be to make a lateral transshipment if the inventory level drops to a certain low level, to prevent stock out.

Chapter 2

Theory

"The purpose of an inventory control system is to determine when and how much to order. This decision should be based on the stock situation, the anticipated demand, and different cost factors." (Axsäter, 2006, p. 46)

2.1 Inventory control

When dealing with inventory control, it is of importance to know how many items there are in inventory and also the timing of items. It is also important to know how many items that are ordered and on their way in as well as knowing how many items that already has been demanded but not delivered. The items can be categorized into stock on hand, outstanding orders and backorders.

For example when determining costs connected with holding and shortage, one take the inventory level in consideration,

$$\text{inventory level} = \text{stock on hand} - \text{backorders}.$$

When considering when to order new items, one cannot only look at the stock on hand, but one must also consider the incoming items as well as backorders. Therefore, the inventory position is taken into consideration when making such decisions,

$$\text{inventory position} = \text{stock on hand} + \text{outstanding orders} - \text{backorders}.$$

To keep track of the inventory position, it can be observed continuously or at certain predetermined points in time. The first case is called continuous review. If

the inventory position is observed at given points in time with a constant interval, this is called periodic review.

2.2 Demand

The customers arrive randomly, one at a time, to each of the two locations and the demand is modeled to follow a Poisson process. The customers arrives with a constant arrival rate $\lambda_i, i = 1,2$. The Poisson process is extensively used for modeling customers arriving to a queuing system. (Law & Kelton, 2000)

The Poisson distribution is used to describe events that occur randomly and independently. If X denotes the number of events that occurs in a given interval of length t, we have that $X \in \text{Po}(\lambda t)$. The time intervals, T, between two consecutively arriving customers are then independent and exponentially distributed, $T \in \text{Exp}(\lambda)$. The reverse relation also holds; if $T \in \text{Exp}(\lambda)$ then $X \in \text{Po}(\lambda t)$. (Blom et al., 2005)

The demand may be a process in which events occur randomly over time and the intermediate times between arriving customers are iid. The stochastic process $N(t), t \geq 0$ is the number of events that occur until time t. When the times between events are independent exponentially distributed random variables, the process is a Poisson process. (Nahmias, 2009)

The Poisson process (describing arrivals) is defined to be a stochastic process where customers arrive one at a time, $N(t+s) - N(t)$ is independent of $N(u), 0 \geq u \geq t$ and the distribution of $N(t+s) - N(t)$ is independent of $t, \forall\, t, s \geq 0$. If customers arrive in batches, a so called compound Poisson process is used, since the first property is not fulfilled. The second property means that the number of customers that has arrived in the time interval $(t, t+s]$ does not depend on how many customers that have arrived earlier, in the interval $[0, t]$, or on the times of these arrivals. This is a simplification of the demand of the real system, since customers may choose to turn away if there already are a lot of customers in the system (so called balking). The third property means that the number of arrivals in a time interval of length s is independent of time t. This might not be the case in a real-life situation since demand for example can differ over the day or the week, but if the arrival rate is reasonably constant, the Poisson process is a good model. For example, the process can model a certain shorter time period that is particularly interesting and where arrival rate can be assumed to be constant. If we let the arrival rate depend on time, $\lambda(t)$, a nonstationary Poisson process can

be used. (Law & Kelton, 2000)

For all times t, $N(t)$ is Poisson distributed with mean λt. This implies that the mean arrival rate is λ, simply because λt customers arrives during a time period of t time units end therefore the average interarrival time is $1/\lambda$. This corresponds to the fact that the time between two successive arrivals is exponentially distributed with parameter λ ($T \in \text{Exp}(\lambda)$), i.e. mean $1/\lambda$. (Laguna & Marklund, n/a)

2.3 Ordering policies

When the locations order items, they usually apply a specific ordering policy. This policy determines when items should be ordered and how many items that should be ordered. Here follows the two most common ordering policies along with a closely related third policy, which has been used in this project.

2.3.1 (R, Q) policy

When using this policy, an order is placed when the inventory position reaches or goes below the reorder point, R. At each order, a batch of quantity Q items is ordered. Several batches can be ordered at the same time, if one batch would not be sufficient to make the inventory position reach over R. Therefore, the policy is sometimes referred to as (R, nQ) policy.

In case of continuous review, the reorder point R will always be reached exactly if the demand is one unit at a time. In the case with discrete demand process, such as a Poisson process, the inventory position lies in the interval $[R+1, R+Q]$. On the other hand, if the demand regards more than one item or if the review is periodic, the inventory position can fall below the reorder point before an order is placed. In these cases the ordering of more than one batch can be in question. Consequently, it is not certain that the inventory position reaches $R+Q$ after a new order has been made. (Axsäter, 2006; Axsäter, 1991)

2.3.2 (s, S) policy

With this policy, new items are ordered when the inventory position has dropped to the reorder point s or below. Then, a sufficiently number of items are ordered, so that the inventory position increases to S. The maximum level is therefor always achieved after ordering new items.

The (s,S) policy resembles the (R,Q) policy, with the difference that items are not ordered in a multiple of a batch of a certain size. Given that the inventory position falls exactly to the reorder point, which it does in case of continuous demand and continuous review, the two policies are equal when $s = R$ and $S = R+Q$. (Axsäter, 2006; Axsäter, 1991)

2.3.3 $(S-1,S)$ policy

The $(S-1,S)$ policy is a variation of the (s,S) policy but in contrast to the (s,S) policy, this policy will trigger an order regardless of the inventory position. When there is continuous review, an order will be made each time a demand has occurred. The amount of items ordered will be the same number as the number of demanded items, This means that the inventory position always is S. In case of periodic review, items will be ordered at all times except when the demand has been zero since the last inspection. This policy is also called S policy, order-up-to-S policy or base stock policy. (Axsäter, 2006; Axsäter, 1991)

If $s = S-1$, the S policy is equivalent to the (s,S) policy and if $R = S-1$ and $Q = 1$, it is the same as the (R,Q) policy. (Axsäter, 2006; Axsäter, 1991)

2.4 The economic order quantity model

The economic order quantity model is used for lot sizing and there is a well-known formula that gives the optimal order quantity for an (R,Q) policy. The formula is known as the EOQ formula or the Wilson formula.

The formula for the optimal order quantity from the economic order quantity model will be used in our simulation study to approximate Q^* for the models of the inventory system using an (R,Q) policy.

In the model, there are five assumptions: Demand is constant and continuous, order cost and holding costs are constant, the order quantity does not have to be an integer, the whole order quantity is delivered to the location at the same time and no shortages are allowed. The economic order quantity formula will balance the ordering cost and the holding cost to derive the optimal order quantity Q^* and it gives us:

$$Q^* = \sqrt{\frac{2Ad}{h}},$$

where the corresponding optimal cost is $C^* = \sqrt{2Adh}$. The EOQ model is relatively insensitive for errors in the cost parameters. (Axsäter, 2006; Axsäter 1991)

A = order cost
d = demand per time unit
h = holding cost per unit and time unit.

2.5 Lateral transshipments

Lateral transshipments are transshipments between locations at the same echelon (in contrast to normal replenishments from supplier to location). When lateral transshipments are allowed, the locations in some sense share inventory. By establishing transshipment links, the inventory system becomes more flexible and better customer service can sometimes be obtained without the need of increasing inventory levels.

A transshipment link between two locations can either be unidirectional or bidirectional. The difference is whether items can be sent only in one direction or in both. If the transshipment is unidirectional one of the locations serves as an emergency supplier and the other one as a receiver of emergency transshipments. In case of a bidirectional link, they both act as supplier and receiver. When there exists bidirectional transshipment links between all locations, they all share their inventory and this is sometimes referred to as complete pooling. (Olsson, 2010)

As an alternative to lateral transshipments, express deliveries can be used. An order for an express delivery is then made to the normal supplier and the policy might be to request en express delivery when a location faces stock out. The express delivery takes shorter time than a normal replenishment, but they are both delivered from the normal supplier and the locations on the same level do not interact with each other.

When lateral transshipments are allowed, the inventory position will be

$$\begin{aligned}\text{inventory position} = \text{stock on hand} + \text{outstanding orders} \\ + \text{incoming lateral transshipments} - \text{backorders}.\end{aligned}$$

2.6 Cost structure

In this section some common costs in connection with inventory systems are described.

2.6.1 Holding cost

When keeping goods in inventory, this is connected with a cost. Besides the cost of such things as handling, storage and equipment, this cost also depend on the capital tied up in inventory and what else could have been done with this money. The holding cost can be compared to the return on alternative investments, but the holding cost must then also include cost for the risk etc.

2.6.2 Order cost and cost for lateral transshipments

For each order made, there is a cost in form of the order cost, A. The order cost is a cost per batch. Making a lateral transshipment also brings a cost with it. The lateral transshipment cost is denoted τ.

2.6.3 Backorder cost or cost for lost sales

If a customer demands an item for which there is a stock out and the demand cannot be satisfied by a lateral transshipment or any other delivery, the demand can either be lost or backordered. This is connected with a cost since, for example, customers are kept waiting, there is a loss of goodwill and loss of sales. The backorder cost in this report is a cost per unit and time unit.

2.6.4 Cost function

The total cost for the system in this simulation study is computed as a cost per time unit to get a measure that is not dependent of the simulation time. The total cost is the sum of the order cost, the holding cost, the cost for backorders and the cost for transshipments for both locations.

The total cost for orders and lateral transshipments are computed as the the number of orders or transshipments that have been made during the simulation times the order cost respectively the transshipment cost divided by the simulation time. The cost for holding items in inventory are computed as the accumulated time in inventory for all items times the holding cost divided by the simulation time. The cost for backorders are similarly the accumulated time customers were kept waiting times the backorder cost divided by the simulation time.

Chapter 3

Literature review

Here follows a brief literature review.

Axsäter (2003) studies a number of locations in a single-echelon inventory system where lateral transshipments between the warehouses are allowed. All the warehouses have continuous review (R, Q) policies and transshipment lead times are zero. Axsäter derives and optimizes a decision rule that tells when demand should be covered with a lateral transshipment in order to minimize the expected costs facing the warehouse. This type of inventory control is especially of interest for systems where the distribution is spread over a larger geographical area with several local warehouses which are supplied externally.

Lateral transshipments in a two-echelon inventory system with repairable items are modeled and analyzed by Axsäter (1990). Here, the warehouses applies continuous review and one-for-one replenishments and faces Poisson demand. Axsäter introduces a new approach to model the problem and compares the results from this technique to the results of Lee, who previously studied the same system.

A single-echelon system with two identical warehouses is under study in Olsson (2009). Both locations faces Poisson demand and the lateral transshipment rule is given. The ordering policies for normal replenishments are optimized and Olsson shows that even though the locations are symmetric, the optimal policies are not necessarily so. First, the ordering policies is assumed to be (R,Q) policies and second, the optimal policies are obtained without this assumption.

Alfredsson and Verrijdt (1999) study a two-echelon inventory system for service parts. Emergency supply can be delivered to the local warehouses both in

form of lateral transshipments as well as direct deliveries from the central warehouse. An analytical model is constructed and simulations are carried out. When costs are introduced in the model, it is shown that the authors strategy for satisfying demand leads to large savings compared to the situation where only normal supply is available. The results from the model is also compared to the results from other models and this comparison suggests that the combination of lateral transshipments and direct deliveries is a good idea.

Yang and Dekker (2010) adapt a different approach in their study of an inventory system with lateral transshipments. They consider lateral transshipments times that are not negligible compared to lead times for normal supply. The method they use is approximate. Their inventory system is a two-echelon system with lateral transshipments under central control. The supplier is assumed to have infinite production capacity and supplies a central depot with a one-for-one policy. The depot then supplies several local service centers on a one-for-one, first come-first served basis. All lead times are constant in the model and continuous review is applied everywhere. The local service centers face a Poisson demand. Yang and Dekker also perform a case study regarding a large company in the dredging industry.

Wong et al. (2006) studies an inventory system with several items. The single-echelon system handling repairable spare parts consists of two locations with one-for-one replenishment and continuous review. Both lateral transshipments from the other location and emergency shipments from an infinite source can be made. The total system cost is minimized in the optimization of ordering policies for all items. A multi-item service measure is used, which means that all items must be considered when deciding on the best policy. There is a target level for the average waiting time for a demanded spare part.

A model for slow-moving, expensive items in a single-echelon inventory system with several locations is developed by Kukreja et al. (2001). The locations apply one-for-one ordering policies and continuous review. Their study is based on a utility company that has plants that can use lateral transshipments and it shows that with a different policy the total system cost can be reduced by seventy percent. To achieve that, the lateral transshipments are taken into account when determining stock levels.

Howard (2010) builds a model using information about the incoming items in the pipeline. The inventory system is a multi-echelon system with local warehouses, a support warehouse and a supplier that provides both the local warehouses as well as the support warehouse. Emergency transshipments can be made

to the local warehouses from the support warehouse and, as a second alternative, from the supplier.

For a more general review of the literature on the subject of lateral transshipments, see Paterson et al. (2010).

Chapter 4

Method

4.1 Choice of method

A system is defined as being a collection of entities. The entities can for example be customers, products or machines and these entities interact with each other. The behavior of the system can be studied in several ways to learn more about the system, to see how the performance might be improved or to see how changes will affect the system. There are two ways of doing experiments to study a system. One way is to do experiments with the actual system and the other way is to do experiments with a model of the system. (See Figure 4.1.) For different systems, one or the other alternative may be preferred. For example it can be very costly, time-consuming or even impossible to make changes to the actual system. (Law & Kelton, 2000)

The model that is built of the system can be a physical or mathematical model. A mathematical model consists of logical and quantitative relations that describe the system. These relations can be altered to see what effect this will have to the model and consequently to the system, given that the model is valid. (Law & Kelton, 2000)

For some mathematical models, if the relations are not to complicated, an exact analytical solution can be found. Other models can be too complex, so that an analytical solution cannot be obtained. The solution may be too hard to compute or it may be non-existent. Then the model can be examined by simulation. By doing simulations, it is numerically evaluated how the output changes for given input. (Law & Kelton, 2000)

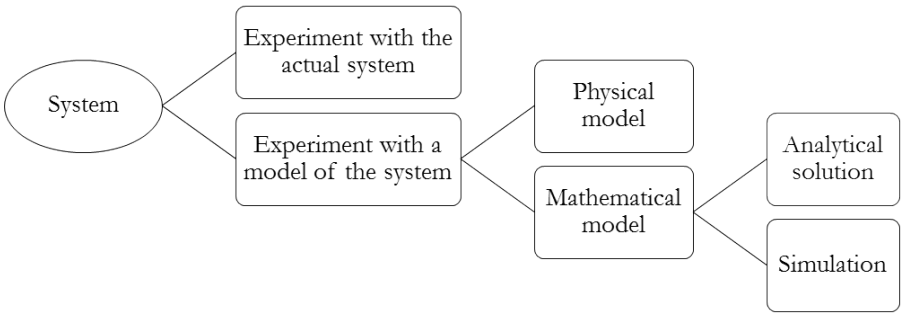

Figure 4.1: Different ways of studying a system. (Law & Kelton, 2000)

The system analyzed in this project is an inventory control system for which a mathematical model has been built. Since the system is somewhat complex, it has been studied numerically. A number of simulations have been made and different alterations of the model have been tested and evaluated. The model of the inventory system has in this particular case been developed in MATLAB, but it could have been done in any other programming language. (See Appendix B for implementation.)

To describe a system, one or several variables are necessary and the collection of these variables defines the state of the system. Depending on what aspects of the model that are of interest in a particular simulation, different state variables can be used for describing the system. (Banks et al., 2005)

An event is defined to be an instantaneous occurrence that may affect the state of the system. (Banks et al., 2005; Law & Kelton, 2000) Events that occur within a system are called endogenous and events occurring outside of the system are called exogenous (Banks et al., 2005). In our model, an arrival of a customer is an exogenous event, while the delivery of an item to a customer is an example of an endogenous event.

4.2 Mathematical modeling

The mathematical model describes how the real system would behave, if it had the same prerequisites. (Law & Kelton, 2000) A model represents the system so that the system can be examined. The model is a simplification of the system and not all characteristics of the system might be represented in the model; sometimes

only the ones that have an effect on the problem at hand are included. (Banks et al., 2005)

Mathematical models that are examined by simulation are called simulation models and they are categorized in the three following ways. (Banks et al., 2005; Law & Kelton, 2000)

4.2.1 Static or dynamic simulation models

A static model represents a system at a certain time or a system where time does not matter, whereas a dynamic model describes a system that changes over time.

4.2.2 Deterministic or stochastic simulation models

The difference between a deterministic and a stochastic model is the presence of randomness. A deterministic model does not contain any random parts. The input is known and when the model has been specified there is a given output. Stochastic models, on the other hand, do consist of some randomness and the result of such a model is therefore also random and must be considered as an estimate of the true characteristics. (Even for stochastic models there is a unique output for a given input, but the difference from deterministic models is that the stochastic model will have different input each time a simulation is run.)

4.2.3 Continuous or discrete simulation models

For a discrete model, the state variables change instantaneously only at a discrete set of points in time. That is, the state can change at a countable number of points in time. If the state variables instead change continuously with time, the model is called continuous. In some cases a model that combines both continuous and discrete modeling can be made to represent a system that does not fall into one of the two categories. It is not certain that a discrete system is modeled with a discrete model or that a model of a continuous system also is continuous. The choice of using a discrete or continuous model depends on the objectives of the study as well as the characteristics of the system.

4.2.4 Our model

Our model is discrete, dynamic and stochastic. Such models are called discrete-event simulation models (Law & Kelton, 2000) and they will be in focus in this

report. The state variables are discrete in this case, since they only change at a set of separate points in time when a customer arrives or when an item is moved. It is obvious that the model changes over time, which makes it dynamic and since the demand is modeled by a random process, the model is also stochastic.

4.2.5 Building a mathematical model

A simulation study is performed in a series of different steps. The process is not always straight forward; sometimes earlier steps must be gone through again.

The first step is the problem formulation. In this step the problem is formulated, the objectives and outcomes of the study are stated and an overall plan is made.

The next step is to collect data about the system. The data requirements depend on the objectives of the study. Performance measures that have been selected to evaluate the system and the availability of data also have an impact. If it is possible, data on the performance of the existing system is collected. This data can be used to validate the model.

These first two steps involve determining the boundaries for the model and which parts of the system that should be included in the model. The data collection and the model building often goes hand in hand. Depending on how the model might change, new data might be needed.

After having made a conceptual model, it is time to check if the assumptions that have been made are reasonable before the conceptual model is translated into a programming model.

The programming model must be both verified and validated to make sure that the conceptual model is translated properly and that the programming model represents the system correctly (given the objectives of the simulation study).

When there is a simulation model working properly, the experiments that it will be used in are designed. The outcome of these experiments is then analyzed.

4.3 Simulation

Simulation is one of the most commonly used tools in several fields of research. The simulation imitates a real-world process and the output from the simulation should be approximately the same as if the real system were studied. Data is collected during the simulation and is used to make estimates of the real systems measures of performance. (Banks et al., 2005) The results are only estimates

of the true characteristics of a model and this is one of the disadvantages with simulation. But on the other hand, simulation may be the only way to examine a model. Discrete-event simulation is used for studying dynamic, discrete models where the state variables only can change when an event occurs. (Law & Kelton, 2000)

With discrete-event simulation models, track must be kept of the simulated time since the state changes over time. Moving forward the simulated time can principally be done in two ways; next-event time advance or fixed-increment time advance. The first of these are by far the most common and means that time is advanced from one event to the next one closest in time. The other option is to make time advance in fixed steps. Then, all events that has occurred during the time step is assumed to occur at the end of the interval. (Law & Kelton, 2000)

Since the state of our model only changes at the time of an event, the first method is preferred and has been used for the simulations in this project. The value of the simulation clock is increased to the time of the next arrival of a customer or the next arrival of an incoming item, whichever comes first. A set of variables are used to keep track of which event that is the most imminent.

Chapter 5

Results

The inventory system has been evaluated for both S policies and (R, Q) policies. To find the optimal values of the decision variables, the mathematical model has been used in simulations where combinations of S_1 and S_2 respectively R_1 and R_2 has been tested. The combination that leads to the lowest total cost for the system is the optimal policy.

Different sets of values for the parameters in the models has been given as input to the simulations resulting in twenty different problems to optimize for the three versions of the model. Note that the results for the case when τ is zero and the case when τ is positive, all other variables the same, is the same case for the model where lateral transshipments are not allowed.

The results for the inventory system where the locations apply an S policy is given in Table 5.1. Three versions of the model is used; one where lateral transshipments are allowed and the transshipment lead time is positive, one where the transshipment lead time is zero and one where transshipments are not allowed. Different values of the parameters in the model are used in 18 combinations.

When assuming (R, Q) policies and determining the optimal values of the reorder point and order quantity, the process is started by approximating Q^*. This is done by using the EOQ formula. This is a simplification that has been made in the optimization. After that the batch quantity is given, the reorder point is optimized through simulations. In Table 5.2, R^* is given for the twenty different settings both with and without the allowing of lateral transshipments.

The optimal pair of S and R differs when lateral transshipments are allowed or when not allowed as well as when the lead time is zero or positive. Therefore

simulations have been made with the model with lateral transshipments and positive lead time using the order up to levels and reorder point that is optimal for the other two models. This is done to see the effect on the cost when using the wrong optimal ordering policy. The result can be seen in Table 5.3 for the S policy model and in Table 5.4 for the (R,Q) policy model.

How many lateral transshipments that are executed depends on the lateral transshipment lead time. The number of transshipments as a function of l is shown in Figures 5.1, 5.2 and 5.3. The first figure corresponds to problem number 1 for the S policy system where the lead time for normal replenishments are 2 for both locations. The second and third figure is showing problem number 1 and 8 for the (R,Q) policy system, where $L_1 = L_2 = 1$.

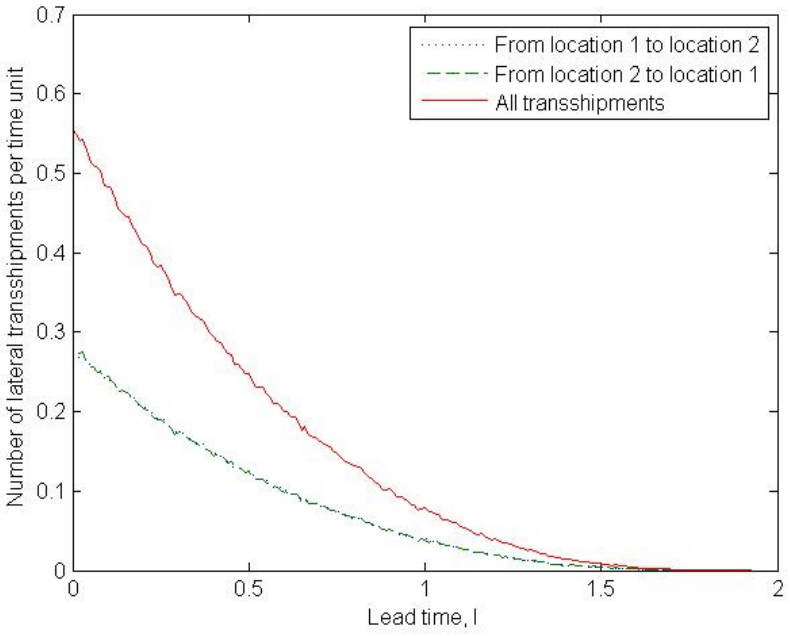

Figure 5.1: The number of lateral transshipments as a function of the transshipment lead time for problem no 1 of the S policy system, $L_1 = L_2 = 2$.

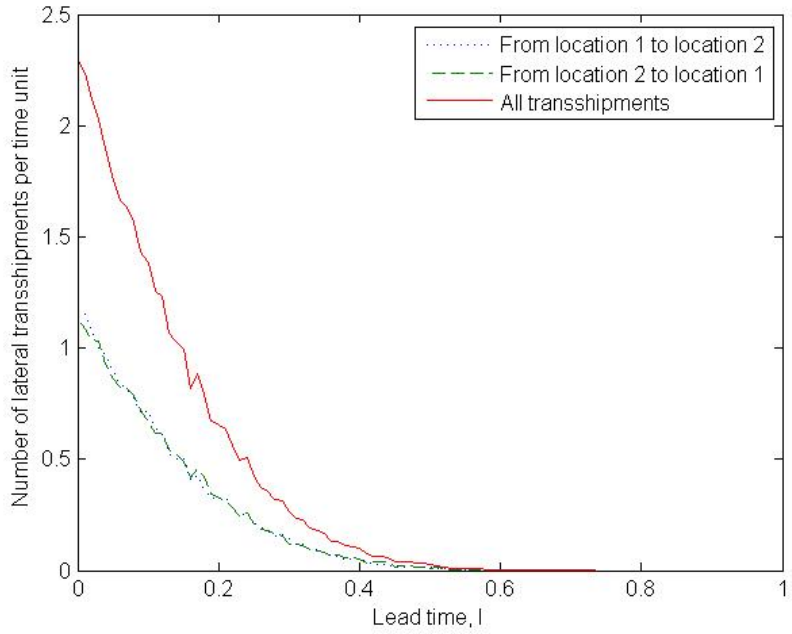

Figure 5.2: The number of lateral transshipments as a function of the transshipment lead time for problem no 1 of the (R,Q) policy system, $L_1 = L_2 = 1$.

Table 5.1: Results for the S policy inventory system, $h_1 = h_2 = 1, L_1 = L_2 = 2$.

Problem no	Variables					$l = 0.5$		$l = 0$		No transshipments	
	λ_1	λ_2	b_1	b_2	τ	S_1^*, S_2^*	Cost (std)	S_1^*, S_2^*	Cost (std)	S_1^*, S_2^*	Cost (std)
1	1	1	20	20	0	4, 4	6.32 (0.02)	4, 4	4.94 (0.02)	5, 5	6.95 (0.01)
2	1	1	20	20	2	4, 4	6.49 (0.02)	4 ,4	5.31 (0.02)	5, 5	6.95 (0.01)
3	1	1	20	40	0	4, 5	6.78 (0.02)	4, 4	5.35 0.04)	5, 5	7.39 (0.03)
4	1	1	20	40	2	4, 5	6.88 (0.02)	4, 4	5.76 (0.02)	5, 5	7.39 (0.03)
5	1	1	40	40	0	5, 5	7.32 (0.03)	4, 5 or 5, 4	5.67 (0.01)	5, 5	7.82 (0.02)
6	1	1	40	40	2	5, 5	7.37 (0.02)	4, 5 or 5, 4	5.94 (0.01)	5, 5	7.82 (0.02)
7	1	2	20	20	0	4, 7	7.58 (0.04)	5, 6	5.90 (0.01)	5, 8	8.19 (0.01)
8	1	2	20	20	2	4, 7	7.67 (0.06)	4, 7	6.39 (0.02)	5, 8	8.19 (0.01)
9	1	2	20	40	0	4, 8	8.15 (0.03)	4, 7	6.37 (0.02)	5, 8	8.88 (0.04)
10	1	2	20	40	2	4, 8	8.36 (0.03)	4, 8	6.92 (0.04)	5, 8	8.88 (0.04)
11	1	2	40	20	0	5, 7	8.06 (0.05)	5, 6	6.26 (0.03)	5, 8	8.69 (0.03)
12	1	2	40	20	2	5, 7	8.16 (0.02)	4, 7	6.79 (0.04)	5, 8	8.69 (0.03)
13	1	2	40	40	0	5, 8	8.74 (0.04)	4, 8	6.78 (0.02)	5, 8	9.27 (0.05)
14	1	2	40	40	2	5, 8	8.83 (0.05)	5, 7	7.12 (0.03)	5, 8	9.27 (0.05)
15	2	2	20	20	0	7, 7	8.76 (0.01)	6, 7 or 7, 6	6.67 (0.02)	8, 8	9.38 (0.01)
16	2	2	20	20	2	7, 7	8.91 (0.05)	6, 7 or 7, 6	7.39 (0.02)	8, 8	9.38 (0.01)
17	2	2	20	40	0	7, 8	9.46 (0.02)	5, 9	7.21 (0.03)	8, 8	10.08 (0.06)
18	2	2	20	40	2	7, 8	9.66 (0.02)	7, 7	7.82 (0.04)	8, 8	10.08 (0.06)
19	2	2	40	40	0	8, 8	10.24 (0.06)	7, 7	7.68 (0.05)	8, 8	10.73 (0.05)
20	2	2	40	40	2	8, 8	10.18 (0.03)	7, 8 or 8, 7	8.14 (0.02)	8, 8	10.73 (0.05)

Table 5.2: Results for the (R,Q) policy inventory system, $h_1 = h_2 = 1, A = 100, L_1 = L_2 = 1$.

Problem no	Variables						$l = 0.25$		$l = 0$		No transshipments	
	λ_1	λ_2	b_1	b_2	τ	Q_1, Q_2	R_1^*, R_2^*	Cost (std)	R_1^*, R_2^*	Cost (std)	R_1^*, R_2^*	Cost (std)
1	10	10	20	20	30	45, 45	11, 11	100.69 (0.26)	14, 14	106.20 (0.41)	8, 8	96.29 (0.29)
2	10	10	20	20	0	45, 45	5, 5	92.73 (0.21)	4, 4	91.23 (0.30)	8, 8	96.29 (0.29)
3	10	10	20	40	30	45, 45	11, 11	100.83 (0.33)	14, 14	106.39 (0.29)	8, 10	97.77 (0.26)
4	10	10	20	40	0	45, 45	4, 9	94.59 (0.27)	6, 6	92.58 (0.32)	8, 10	97.77 (0.26)
5	10	10	40	40	30	45, 45	11, 11	101.30 (0.28)	13, 14 or 14, 13	106.54 (0.49)	10, 10	99.19 (0.30)
6	10	10	40	40	0	45, 45	7, 8 or 8, 7	96.07 (0.26)	6, 6	93.37 (0.44)	10, 10	99.19 (0.30)
7	10	20	20	20	30	45, 63	11, 21	120.71 (0.63)	13, 26	128.90 (0.70)	8, 17	115.94 (0.48)
8	10	20	20	20	0	45, 63	0, 19	111.61 (0.23)	4, 14	111.32 (0.42)	8, 17	115.94 (0.48)
9	10	20	20	40	30	45, 63	11, 23	121.63 (0.38)	14, 26	128.85 (0.70)	8, 20	118.33 (0.42)
10	10	20	20	40	0	45, 63	3, 19	114.44 (0.53)	6, 6	112.26 (0.31)	8, 20	118.33 (0.42)
11	10	20	40	20	30	45, 63	11, 20	120.89 (0.65)	14, 26	128.96 (0.56)	10, 18	117.75 (0.36)
12	10	20	40	20	0	45, 63	1, 21	114.33 (0.56)	6, 7	112.13 (0.49)	10, 18	117.75 (0.36)
13	10	20	40	40	30	45, 63	11, 21	121.82 (0.31)	14, 26	128.99 (0.60)	10, 20	119,52 (0.39)
14	10	20	40	40	0	45, 63	1, 22	115.69 (0.43)	6, 16	113.70 (0.32)	10, 20	119,52 (0.39)
15	20	20	20	20	30	63, 63	20, 20	140.96 (0.58)	26, 27 or 27, 26	151.73 (0.83)	17, 18 or 18, 17	136.27 (0.30)
16	20	20	20	20	0	63, 63	12, 15 or 15, 12	132.40 (0.52)	9, 9	125.74 (0.79)	17, 18 or 18, 17	136.27 (0.30)
17	20	20	20	40	30	63, 63	21, 21	141.30 (0.56)	26, 26	151.31 (0.74)	17, 20	138.27 (0.46)
18	20	20	20	40	0	63, 63	10, 21	135.09 (0.56)	10, 10	127.83 (1.54)	17, 20	138.27 (0.46)
19	20	20	40	40	30	63, 63	21, 22 or 22, 21	142.40 (0.64)	27, 27	151.73 (0.49)	20, 20	140.47 (0.58)
20	20	20	40	40	0	63, 63	18, 19 or 19, 18	138.13 (0.35)	11, 11	128.97 (1.08)	20, 20	140.47 (0.58)

Table 5.3: Cost increase when using the wrong optimal order up to levels.

Problem no	Variables					Optimal order up to levels		Zero lead time			No transshipments		
	λ_1	λ_2	b_1	b_2	τ	S_1^*, S_2^*	Cost	S_1, S_2	Cost	Cost increase	S_1, S_2	Cost	Cost increase
1	1	1	20	20	0	4, 4	6.32	4, 4	-	-	5, 5	6.68	5.66 %
2	1	1	20	20	2	4, 4	6.49	4, 4	-	-	5, 5	6.73	3.79 %
3	1	1	20	40	0	4, 5	6.78	4, 4	7.42	9.44 %	5, 5	7.01	3.39 %
4	1	1	20	40	2	4, 5	6.88	4, 4	7.59	10.32 %	5, 5	7.08	2.90 %
5	1	1	40	40	0	5, 5	7.32	4, 5 or 5, 4	7.73	5.60 %	5, 5	-	-
6	1	1	40	40	2	5, 5	7.37	4, 5 or 5, 4	7.88	6.92 %	5, 5	-	-
7	1	2	20	20	0	4, 7	7.58	5, 6	8.22	8.44 %	5, 8	7.89	4.05 %
8	1	2	20	20	2	4, 7	7.67	4, 7	-	-	5, 8	7.95	3.63 %
9	1	2	20	40	0	4, 8	8.15	4, 7	8.86	8.71 %	5, 8	8.43	3.42 %
10	1	2	20	40	2	4, 8	8.36	4, 8	-	-	5, 8	8.50	1.78 %
11	1	2	40	20	0	5, 7	8.06	5, 6	8.63	7.08 %	5, 8	8.22	1.98 %
12	1	2	40	20	2	5, 7	8.16	4, 7	8.71	6.75 %	5, 8	8.25	1.13 %
13	1	2	20	20	0	5, 8	8.74	4, 8	9.14	4.58 %	5, 8	-	-
14	1	2	20	20	2	5, 8	8.83	5, 7	9.39	6.34 %	5, 8	-	-
15	2	2	20	20	0	7, 7	8.76	6, 7 or 7, 6	9.51	8.56 %	8, 8	9.13	4.22 %
16	2	2	20	20	2	7, 7	8.91	6, 7 or 7, 6	9.73	9.30 %	8, 8	9.16	2.80 %
17	2	2	20	20	0	7, 8	9.46	5, 9	11.43	20.82 %	8, 8	9.62	1.70 %
18	2	2	20	20	2	7, 8	9.66	7, 7	10.28	6.41 %	8, 8	9.73	0.76 %
19	2	2	20	20	0	8, 8	10.24	7, 7	11.46	11.91 %	8, 8	-	-
20	2	2	20	20	2	8, 8	10.18	7, 8 or 8, 7	10.86	6.68 %	8, 8	-	-

Table 5.4: Cost increase when using the wrong optimal reorderpoints.

No	Variables						Optimal reorder points		Zero lead time			No transshipments		
	λ_1	λ_2	b_1	b_2	τ	Q_1, Q_2	R_1^*, R_2^*	Cost	R_1, R_2	Cost	Cost increase	R_1, R_2	Cost	Cost increase
1	10	10	20	20	30	45, 45	11, 11	100.69	14, 14	103.82	3.11 %	8, 8	105.87	5.15 %
2	10	10	20	20	0	45, 45	5, 5	92.73	4, 4	93.46	0.79 %	8, 8	94.17	1.55 %
3	10	10	20	40	30	45, 45	11, 11	100.83	14, 14	103.91	3.05 %	8, 10	104.13	3.27 %
4	10	10	20	40	0	45, 45	4, 9	94.59	6, 6	94.96	0.39 %	8, 10	95.89	1.37 %
5	10	10	40	40	30	45, 45	11, 11	101.30	13, 14 or 14, 13	103.28	1.95 %	10, 10	101.85	0.54 %
6	10	10	40	40	0	45, 45	7, 8 or 8, 7	96.07	6, 6	96.71	0.66 %	10, 10	97.73	1.73 %
7	10	20	20	20	30	45, 63	11, 21	120.71	13, 26	124.80	3.39 %	8, 17	128.02	6.06 %
8	10	20	20	20	0	45, 63	0, 19	111.61	4, 14	112.81	1.08 %	8, 17	114.16	2.28 %
9	10	20	20	40	30	45, 63	11, 23	121.63	14, 26	125.69	3.34 %	8, 20	125.09	2.84 %
10	10	20	20	40	0	45, 63	3, 19	114.44	6 ,6	121.10	5.82 %	8, 20	116.56	1.85 %
11	10	20	40	20	30	45, 63	11, 20	120.89	14, 26	125.64	3.93 %	10, 18	123.76	2.38 %
12	10	20	40	20	0	45, 63	1, 21	114.33	6, 7	118.52	3.66 %	10, 18	116.27	1.70 %
13	10	20	40	40	30	45, 63	11, 21	121.82	14, 26	125.61	3.11 %	10, 20	122.58	0.62 %
14	10	20	40	40	0	45, 63	1, 22	115.69	6, 16	117.93	1.93 %	10, 20	118.29	2.25 %
15	20	20	20	20	30	63, 63	20, 20	140.96	26, 27 or 27, 26	148.01	5.00 %	17, 18 or 18, 17	147.50	4.64%
16	20	20	20	20	0	63, 63	12,15 or 15, 12	132.40	9, 9	137.43	3.80 %	17, 18 or 18, 17	134.12	1.29 %
17	20	20	20	40	30	63, 63	21, 21	141.30	26, 26	147.08	4.09 %	17, 20	146.17	3.45 %
18	20	20	20	40	0	63, 63	10, 21	135.09	10, 10	144.00	6.60 %	17, 20	136.69	1.19 %
19	20	20	40	40	30	63, 63	21, 22 or 22, 21	142.40	27, 27	149.22	4.79 %	20, 20	143.52	0.79 %
20	20	20	40	40	0	63, 63	18, 19 or 19, 18	138.13	11, 11	148.28	7.34 %	20, 20	138.98	0.61 %

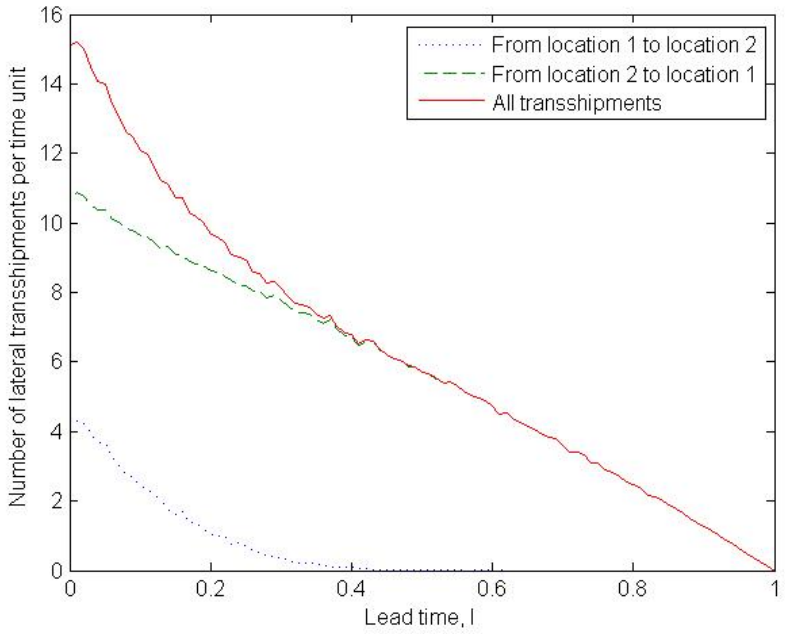

Figure 5.3: The number of lateral transshipments as a function of the transshipment lead time for problem no 8 of the (R,Q) policy system, $L_1 = L_2 = 1$.

Chapter 6

Conclusions

In Table 5.1 we can see that the three models in general gives different value of S_1^* and S_2^*. The differences are not that large, but for the model without lateral transshipment the optima order up to levels are a bit higher than the optimal levels for the case with a positive transshipment lead time. The optimal levels for the case where the lead time is zero, are a bit lower. This implies that using the optimal value for a model with no lateral transshipments or for a model with negligible lead time to optimize a model where there in fact are lateral transshipment or the lead time is not zero, may be incorrect. As seen in Table 5.3, this do lead to an increased cost for many problems. The largest cost increase happens for problem number 17, where wrongly assuming a zero transshipment lead time means a cost increase of almost 21%. This means that for a system with lateral transshipments with non-zero lead time, it is of importance to use reorder points that have been optimized for that system.

From Table 5.2 and 5.4 we experience similar results as for the S policy models. The optimal reorder points differ between the three models. But here, the variety of R_1^* and R_2^* is bigger and no general trend is easily seen. In Table 5.4 we can see that there is a slight increase of costs when using the wrong optimal reorder points, with the highest increase being 6.6 %.

When trying different pairs of R_1 and R_2 during simulation, some pairs give a cost very near the cost resulting from a simulation with another pair used as input. This means that the optimal value of R_1^* and R_2^* can be a bit hard to determine and that several pairs of reorder points can be equally good. This may have an effect on the results in Table 5.2 and 5.4.

The simulation results for problem number 8, 10, 12 and 14 for the inventory system with the (R,Q) policy are especially interesting. In these four cases, the difference between the R_1^* and R_2^* are greater than in the other cases and the optimal value of R_1 is low. For problem number 8 the value of R_1 is zero. For this group of problems, there is a bigger difference between the demand for the two locations. The location with the lowest value of the reordering point is location number one, which has the lowest demand rate. The low reorder point means that this location has little or no safety stock.

From the Figures 5.1 and 5.2 showing the number of lateral transshipments as a function of the transshipment lead time, we can see that even a small change in l can have a big impact of the number of transshipments. Decreasing the transshipment lead time may be a good idea to make the inventory system more effective, since the number of lateral transshipments increases significantly for a decrease of the lateral transshipment lead time. From these figures we can also conclude that the lateral transshipments are made more or less equally in the two directions. The decrease of l and thereby an increase of number of transshipments are, of course, only of interest when lateral transshipments leads to a lower system cost.

Figure 5.3 shows a bit different scenario than the other two figures regarding the number of lateral transshipments. This figure shows the number as function of the lead time for problem number 8, where the optimal policy is $(R_1^*, R_2^*) = (0, 19)$. We can see that the greater part of the lateral transshipments goes in one direction, namely from location two to location one. The number of transshipments made are also significantly more than for problem number 1 in Figure 5.2. This means that location one has no stock on hand when a new batch is ordered and that location two supplies location one through lateral transshipments.

Many things could be done to extend the work of this master thesis. Firstly, an analytical model (based on the ideas in Yang and Dekker (2010) and Olsson (2011)) could be made. The results from this model could then be compared with the ones of the simulations.

The simulation results may be improved due to the problem already mentioned that the best and second best value of the policy parameter cannot be separated well enough and the result of the optimal policy might be indecisive.

Of course, a wider range of simulation problems could be evaluated to see the impact of variations of the problem characteristics. The model has not been evaluated with a setting where the two locations have different lead times for normal replenishment. This could be interesting to evaluate to see the impact on

the lateral transshipments.

Variants of the lateral transshipment policy could also be tested. The policy that is used triggers a lateral transshipment when it is a faster solution than waiting for an incoming order (when the receiving location faces stock out and the delivering location has stock on hand). This policy does not take other aspects of the situation at the delivering location into concern, for example the inventory level. The delivering location might only have a few items left and maybe shipping these to the receiving location, and thereby be more likely to face stock out, is not the best option. This might not be a good transshipping policy if there were differences between the locations, such as one of them having a higher backorder cost. Then maybe the lower cost location rather go out of stock or backorder demand, compared to risking that the high cost location would do the same.

Chapter 7

References

Alfredsson, P. & Verrijdt, J. (1999) *Modeling Emergency Supply Flexibility in a Two-Echelon Inventory System*, Management Science, Vol. 45, pp.1416 - 1431.

Axsäter, S. (1990) *Modelling Emergency Lateral Transshipments in Inventory Systems*, Management Science, Vol. 36, pp.1329 - 1338.

Axsäter, S. (1991) *Lagerstyrning*, Studentlitteratur, Lund.

Axsäter, S. (2003) *A New Decision Rule for Lateral Transshipments in Inventory Systems*, Management Science, Vol. 49, pp. 1168 - 1179.

Axsäter, S. (2006) *Inventory Control*, Springer.

Banks, J. et al. (2005) *Discrete-Event System Simulation*, Pearson Prentice Hall.

Blom, G. et al. (2005) *Sannolikhetsteori och statistikteori med tillämpningar*, Studentlitteratur, Lund.

Howard, C. (2010) *Allocation Decisions and Emergency Shipments in Multi-Echelon Inventory Control*, Licentiate Thesis, Lund University, Faculty of engineering LTH.

Kukreja, A. et al. (2001) *Stocking Decisions for Low-Usage Items in a Multi-*

location Inventory System, Management Science, Vol. 47, pp. 1371 - 1383.

Laguna, M & Marklund, J. (n/a) *Business Process Modeling, Simulation and Design*, Prentice Hall.

Law, A. & Kelton, D. (2000) *Simulation Modeling and Analysis*, McGraw-Hill, Singapore.

Nahmias, S. (2009) *Production and Operations Analysis*, McGraw-Hill, Singapore.

Olsson, F. (2009) *Optimal Policies for inventory systems with lateral transshipments*, International Journal of Production Economics, Vol. 118, pp. 175 - 184.

Olsson, F. (2010) *An inventory model with unidirectional lateral transshipments*, European Journal of Operational Research, Vol. 200, pp. 725 - 732.

Olsson, F. (2011) *Emergency lateral transshipments in inventory systems with positive transshipment leadtimes*, Working paper, Lund University.

Paterson, C. et al. (2010) *Inventory models with lateral transshipments: A review*, European Journal of Operational Research, doi:10.1016/j.ejor.2010.05.048.

Wong, H. et al. (2006) *Multi-item spare parts system with lateral transshipments and waiting time constraints*, European Journal of Operational Research, Vol 171, pp. 1071 - 1093.

Yang, D. & Dekker, R. (2010) *Service Parts Inventory Control with Lateral Transshipment that Takes Time*, Working paper, Erasmus University Rotterdam.

Appendices

Appendix A

Notations

In this project, the following notations have been used.

S	Order up to level
R	Reorder point
Q	Batch quantity
L	Lead time for normal replenishment
l	Transshipment lead time
λ	Demand intensity
h	Holding cost per unit and time unit
b	Backorder cost per unit and time unit
A	Order cost
τ	Transshipment cost

Indexes are used to denote for which location, 1 or 2, the parameter is connected to. The optimal values for S, R and Q are marked with a star.

Appendix B

MATLAB code

Here are some examples of simulation models used.

B.1 Models for the $(S-1,S)$ policy inventory system

B.1.1 Without lateral transshipments

```
% function [meanTot, stdTot] = Spolicy_no(S1, S2)
% Inventory system with two retailers applying (S-1,S) policy.
% No lateral transshipments.
% Given order up to levels, the function returns mean total cost
% and standard deviation.

function [meanTot, stdTot] = Spolicy_no(S1, S2)

n = 5;                      % Number of runs

lambda1 = 1; lambda2 = 1;   % Demand rates
L1 = 2; L2 =2;              % Lead times

b1 = 20; b2 = 20;           % Backorder cost
h1 = 1; h2 = 1;             % Holding cost

costVector = [];    % Vector with total cost for the runs
summa = 0;          % Sum of total costs for all runs

for j = 1:n,
    % Creates points in time when demand will occur for each warehouse
    % Times between customers are exp(lambda) distributed
```

```
nbrOfEvents = 30000;
u1 = rand(1,nbrOfEvents); u2 = rand(1,nbrOfEvents);
y1 = -log(1-u1)/lambda1; y2 = -log(1-u2)/lambda2;

T1 = cumsum(y1); T2 = cumsum(y2);
stopTime = min([T1(nbrOfEvents) T2(nbrOfEvents)]);
time = 0;

% A random share of the units are located in inventory and the rest is
% outstanding orders at time zero
y1 = floor(S1*rand); y2 = floor(S2*rand);
% Arrival times for units in inventory
invTime1 = zeros([1,y1]); invTime2 = zeros([1,y2]);
% Arrival times for incoming items
incom1 = L1*ones([1,S1-y1]); incom2 = L2*ones([1,S2-y2]);
% Times when backorders have occurred
backTime1 = []; backTime2 = [];

timeInStock1 = 0; timeInStock2 = 0;         % Accumulated time in stock
timeInBackorder1 = 0; timeInBackorder2 = 0; % Acc. time in backorder
timeWithNoStock1 = 0; timeWithNoStock2 = 0; % Acc. time with zero stock

while time < stopTime,
    stock1 = length(invTime1);
    stock2 = length(invTime2);
    oldtime = time;

    % A trick to determine the next event
    if isempty(incom1),
        incom1 = realmax;
    end
    if isempty(incom2),
        incom2 = realmax;
    end

    % Determines the next event and the time when it happens
    [time, event] = min([T1(1) T2(1) incom1(1) incom2(1)]);

    if incom1(1) == realmax,
        incom1 = [];
    end
    if incom2(1) == realmax,
        incom2 = [];
    end
```

```
% A customer arrives at retailer 1
 if event == 1,
     % If there is stock on hand
     if length(invTime1) >= 1,
         timeInStock1 = timeInStock1 + time - invTime1(1);
         invTime1(1) = [];
         incom1 = [incom1 time+L1];
     % If there are no items in stock, the demanded item is backordered
     else
         backTime1 = [backTime1 time];
         incom1 = [incom1 time+L1];
     end
     T1(1) = [];

 % A customer arrives at retailer 2
 elseif event == 2,
     % If there is stock on hand
     if length(invTime2) >= 1,
         timeInStock2 = timeInStock2 + time - invTime2(1);
         invTime2(1) = [];
         incom2 = [incom2 time+L2];
     % If there are no items in stock, the demanded item is backordered
     else
         backTime2 = [backTime2 time];
         incom2 = [incom2 time+L2];
     end
     T2(1) = [];

 % An item arrives at retailer 1
 elseif event == 3,
     incom1(1) = [];
     % If there are backorders, the item is given to a waiting customer
     if ~isempty(backTime1),
         timeInBackorder1 = timeInBackorder1 + time - backTime1(1);
         backTime1(1)=[];
     % If not, the item is put in inventory
     else
         invTime1 = [invTime1 time];
     end

 % An item arrives at retailer 2
 elseif event == 4,
     incom2(1) = [];
     if ~isempty(backTime2),
         timeInBackorder2 = timeInBackorder2 + time - backTime2(1);
```

```
            backTime2(1)=[];
        % If not, the item is put in inventory
        else
            invTime2 = [invTime2 time];
        end
    end

    % Accumulates time with zero stock on hand
    if stock1 == 0,
        timeWithNoStock1 = timeWithNoStock1 + time - oldtime;
    end
    if stock2 == 0,
        timeWithNoStock2 = timeWithNoStock2 + time - oldtime;
    end
end

% After the time is up, remaining items in inventory and in backorder
% are accounted for in the accumulated times
if ~isempty(invTime1),
    for k = 1:length(invTime1),
        timeInStock1 = timeInStock1 + time - invTime1(1);
        invTime1(1) = [];
    end
end
if ~isempty(invTime2),
    for k = 1:length(invTime2),
        timeInStock2 = timeInStock2 + time - invTime2(1);
        invTime2(1) = [];
    end
end
if ~isempty(backTime1),
    for k = 1:length(backTime1),
        timeInBackorder1 = timeInBackorder1 + time - backTime1(1);
        backTime1(1) = [];
    end
end
if ~isempty(backTime2),
    for k = 1:length(backTime2),
        timeInBackorder2 = timeInBackorder2 + time - backTime2(1);
        backTime2(1) = [];
    end
end

%     fillrate1 = 1 - timeWithNoStock1/stopTime;
%     fillrate2 = 1 - timeWithNoStock2/stopTime;
```

```
    % The costs are calculated

    holdCost1 = timeInStock1*h1/stopTime;
    holdCost2 = timeInStock2*h2/stopTime;
    backCost1 = timeInBackorder1*b1/stopTime;
    backCost2 = timeInBackorder2*b2/stopTime;

    totCost = holdCost1 + backCost1 + holdCost2 + backCost2;
    costVector = [costVector totCost];
    summa = summa + totCost;

%     fillVector1 = [fillVector1 fillrate1];
%     fillVector2 = [fillVector2 fillrate2];
end

meanTot = 1/n*summa;

sumTot = 0;
for i = 1:n,
    sumTot = sumTot + (costVector(i)-meanTot)^2;
end
stdTot = sqrt(1/(n-1)*sumTot)/sqrt(n);

%     meanfill1 = mean(fillVector1);
%     meanfill2 = mean(fillVector2);
```

B.1.2 With lateral transshipments with zero lead time

```
% function [meanTot, stdTot] = Spolicy_lt_0(S1, S2)
% Inventory system with two retailers applying (S-1,S) policy.
% Lateral transshipments allowed. Zero lead time.
% Given order up to levels, the function returns mean total cost
% and standard deviation.

function [meanTot, stdTot] = Spolicy_lt_0(S1, S2)

n = 5;                      % Number of runs

lambda1 = 1; lambda2 = 1;   % Demand rates
L1 = 2; L2 =2;              % Lead times

b1 = 20; b2 = 20;           % Backorder cost
h1 = 1; h2 = 1;             % Holding cost
tau = 0;                    % Transshipment cost
```

```
costVector = [];    % Vector with total cost for the runs
summa = 0;          % Sum of total costs for all runs

for j = 1:n,
    % Creates points in time when demand will occur for each warehouse
    % Times between customers are exp(lambda) distributed
    nbrOfEvents = 30000;
    u1 = rand(1,nbrOfEvents); u2 = rand(1,nbrOfEvents);
    y1 = -log(1-u1)/lambda1; y2 = -log(1-u2)/lambda2;

    T1 = cumsum(y1); T2 = cumsum(y2);
    stopTime = min([T1(nbrOfEvents) T2(nbrOfEvents)]);
    time = 0;

    % A random share of the units are located in inventory and the rest is
    % outstanding orders at time zero
    y1 = floor(S1*rand); y2 = floor(S2*rand);
    % Arrival times for units in inventory
    invTime1 = zeros([1,y1]); invTime2 = zeros([1,y2]);
    % Arrival times for incoming items
    incom1 = L1*ones([1,S1-y1]); incom2 = L2*ones([1,S2-y2]);
    % Times when backorders have occurred
    backTime1 = []; backTime2 = [];

    timeInStock1 = 0; timeInStock2 = 0;         % Accumulated time in stock
    timeInBackorder1 = 0; timeInBackorder2 = 0; % Acc. time in backorder
    timeWithNoStock1 = 0; timeWithNoStock2 = 0; % Acc. time with zero stock

    transCost21 = 0; transCost12 = 0;   % Cost for all transshipments during sim.

    while time < stopTime,
        stock1 = length(invTime1);
        stock2 = length(invTime2);
        oldtime = time;

        % A trick to determine the next event
        if isempty(incom1),
            incom1 = realmax;
        end
        if isempty(incom2),
            incom2 = realmax;
        end

        % Determines the next event and the time when it happens
```

```
[time, event] = min([T1(1) T2(1) incom1(1) incom2(1)]);

if incom1(1) == realmax,
    incom1 = [];
end
if incom2(1) == realmax,
    incom2 = [];
end

% A customer arrives at location 1
if event == 1,
    % If there is stock on hand
    if length(invTime1) >= 1,
        timeInStock1 = timeInStock1 + time - invTime1(1);
        invTime1(1) = [];
        incom1 = [incom1 time+L1];
    % If the other location has stock on hand, a lateral transshipment
    % is made
    elseif length(invTime2) >= 1,
        timeInStock2 = timeInStock2 + time - invTime2(1);
        invTime2(1) = [];
        transCost21 = transCost21 + tau;
        incom2 = [incom2 time+L2];  % The other retailer orders a new item
    % If there are no items at either location, the item is backordered
    else
        backTime1 = [backTime1 time];
        incom1 = [incom1 time+L1];
    end
    T1(1) = [];

% A customer arrives at location 2
elseif event == 2,
    % If there is stock on hand
    if length(invTime2) >= 1,
        timeInStock2 = timeInStock2 + time - invTime2(1);
        invTime2(1) = [];
        incom2 = [incom2 time+L2];
    % If the other location has stock on hand, a lateral transshipment
    % is made
    elseif length(invTime1) >= 1,
        timeInStock1 = timeInStock1 + time - invTime1(1);
        invTime1(1) = [];
        transCost12 = transCost12 + tau;
        incom1 = [incom1 time+L1];  % The other retailer orders a new item
    % If there are no items at either location, the item is backordered
```

```
        else
            backTime2 = [backTime2 time];
            incom2 = [incom2 time+L2];
        end
        T2(1) = [];

    % An item arrives at location 1
    elseif event == 3,
        incom1(1) = [];
        % If there are backorders, the item are given to a waiting customer
        if ~isempty(backTime1),
            timeInBackorder1 = timeInBackorder1 + time - backTime1(1);
            backTime1(1) = [];
        % If not, the item is put in inventory
        else
            invTime1 = [invTime1 time];
        end

    % An item arrives at location 2
    elseif event == 4,
        incom2(1) = [];
        % If there are backorders, the item are given to a waiting customer
        if ~isempty(backTime2),
            timeInBackorder2 = timeInBackorder2 + time - backTime2(1);
            backTime2(1)=[];
        % If not, the item is put in inventory
        else
            invTime2 = [invTime2 time];
        end
    end

    % Accumulates time with zero stock on hand
    if stock1 == 0,
        timeWithNoStock1 = timeWithNoStock1 + time - oldtime;
    end
    if stock2 == 0,
        timeWithNoStock2 = timeWithNoStock2 + time - oldtime;
    end
end

% After the time is up, remaining items in inventory and in backorder
% are accounted for in the accumulated times
if ~isempty(invTime1),
    for k = 1:length(invTime1),
        timeInStock1 = timeInStock1 + time - invTime1(1);
```

```
            invTime1(1) = [];
        end
    end
    if ~isempty(invTime2),
        for k = 1:length(invTime2),
            timeInStock2 = timeInStock2 + time - invTime2(1);
            invTime2(1) = [];
        end
    end
    if ~isempty(backTime1),
        for k = 1:length(backTime1),
            timeInBackorder1 = timeInBackorder1 + time - backTime1(1);
            backTime1(1) = [];
        end
    end
    if ~isempty(backTime2),
        for k = 1:length(backTime2),
            timeInBackorder2 = timeInBackorder2 + time - backTime2(1);
            backTime2(1) = [];
        end
    end

%     fillrate1 = 1 - timeWithNoStock1/stopTime;
%     fillrate2 = 1 - timeWithNoStock2/stopTime;

    % The costs are calculated

    transCost21 = transCost21/stopTime;
    transCost12 = transCost12/stopTime;

    holdCost1 = timeInStock1*h1/stopTime;
    holdCost2 = timeInStock2*h2/stopTime;
    backCost1 = timeInBackorder1*b1/stopTime;
    backCost2 = timeInBackorder2*b2/stopTime;

    totCost = holdCost1 + backCost1 + transCost21 ...
        + holdCost2 + backCost2 + transCost12;
    costVector = [costVector totCost];
    summa = summa + totCost;

%     fillVector1 = [fillVector1 fillrate1];
%     fillVector2 = [fillVector2 fillrate2];
end

meanTot = 1/n*summa;
```

```
sumTot = 0;
for i = 1:n,
    sumTot = sumTot + (costVector(i)-meanTot)^2;
end
stdTot = sqrt(1/(n-1)*sumTot)/sqrt(n);

%     meanfill1 = mean(fillVector1);
%     meanfill2 = mean(fillVector2);
```

B.1.3 With lateral transshipments with positive lead time

```
% function [meanTot, stdTot] = Spolicy_lt(S1, S2)
% Inventory system with two retailers applying (S-1,S) policy.
% Lateral transshipments allowed. Positive lead time.
% Given order up to levels, the function returns mean total cost
% and standard deviation.

function [meanTot, stdTot] = Spolicy_lt(S1, S2)

n = 5;                      % Number of runs

lambda1 = 1; lambda2 = 1;   % Demand rates
L1 = 2; L2 =2;              % Lead times
l = 0.5;                    % Transshipment lead time

b1 = 20; b2 = 20;           % Backorder cost
h1 = 1; h2 = 1;             % Holding cost
tau = 0;                    % Transshipment cost

costVector = [];    % Vector with total cost for the runs
summa = 0;          % Sum of total costs for all runs

for j = 1:n,
    % Creates points in time when demand will occur for each warehouse
    % Times between customers are exp(lambda) distributed
    nbrOfEvents = 30000;
    u1 = rand(1,nbrOfEvents); u2 = rand(1,nbrOfEvents);
    y1 = -log(1-u1)/lambda1; y2 = -log(1-u2)/lambda2;

    T1 = cumsum(y1); T2 = cumsum(y2);
    stopTime = min([T1(nbrOfEvents) T2(nbrOfEvents)]);
    time = 0;

    % A random share of the units are located in inventory and the rest is
```

```
% outstanding orders at time zero
y1 = floor(S1*rand); y2 = floor(S2*rand);
% Arrival times for units in inventory
invTime1 = zeros([1,y1]); invTime2 = zeros([1,y2]);
% Arrival times for incoming items
incom1 = L1*ones([1,S1-y1]); incom2 = L2*ones([1,S2-y2]);
% Arrival times for incoming items from lateral transshipments
incomTrans1=[]; incomTrans2=[];
% Times when backorders have occurred
backTime1 = []; backTime2 = [];
% Times when backorders have occurred and a lateral transshipment is made
backTimeTrans1 = []; backTimeTrans2 = [];

timeInStock1 = 0; timeInStock2 = 0;         % Accumulated time in stock
timeInBackorder1 = 0; timeInBackorder2 = 0; % Acc. time in backorder
timeInBackTrans1 = 0; timeInBackTrans2 = 0; % Acc. time in backorder
timeWithNoStock1 = 0; timeWithNoStock2 = 0; % Acc. time with zero stock

transCost21 = 0; transCost12 = 0;   % Cost for all transshipments during sim.

while time < stopTime,
    stock1 = length(invTime1);
    stock2 = length(invTime2);
    oldtime = time;

    % A trick to determine the next event
    if isempty(incom1),
        incom1 = realmax;
    end
    if isempty(incom2),
        incom2 = realmax;
    end
    if isempty(incomTrans1),
        incomTrans1 = realmax;
    end
    if isempty(incomTrans2),
        incomTrans2 = realmax;
    end

    % Determines the next event and the time when it happens
    [time, event] = min([T1(1) T2(1) incom1(1) incom2(1) ...
        incomTrans1(1) incomTrans2(1)]);

    if incom1(1) == realmax,
        incom1 = [];
```

```
end
if incom2(1) == realmax,
    incom2 = [];
end
if incomTrans1(1) == realmax,
    incomTrans1 = [];
end
if incomTrans2(1) == realmax,
    incomTrans2 = [];
end

% A customer arrives at location 1
if event == 1,
    Nback = length(backTime1);
    % If there is stock on hand
    if length(invTime1) >= 1,
        timeInStock1 = timeInStock1 + time - invTime1(1);
        invTime1(1) = [];
        incom1 = [incom1 time+L1];
    % If the other location has stock on hand and the waiting time is
    % longer than the transshipment lead time, a lateral transshipment
    % is made (incom1(Nback+1)-time is the residual leadtime)
    elseif length(invTime2) >= 1 && incom1(Nback+1)-time > l,
        timeInStock2 = timeInStock2 + time - invTime2(1);
        invTime2(1) = [];
        transCost21 = transCost21 + tau;
        incom2 = [incom2 time+L2];  % The other retailer orders a new item
        incomTrans1 = [incomTrans1 time+l];
        backTimeTrans1 = [backTimeTrans1 time];
    % If there are no items at either location, the item is backordered
    else
        backTime1 = [backTime1 time];
        incom1 = [incom1 time+L1];
    end
    T1(1) = [];

% A customer arrives at location 2
elseif event == 2,
    Nback = length(backTime2);
    % If there is stock on hand
    if length(invTime2) >= 1,
        timeInStock2 = timeInStock2 + time - invTime2(1);
        invTime2(1) = [];
        incom2 = [incom2 time+L2];
    % If the other location has stock on hand and the waiting time is
```

```
    % longer than the transshipment lead time, a lateral transshipment
    % is made (incom2(Nback+1)-time is the residual leadtime)
    elseif length(invTime1) >= 1 && incom2(Nback+1)-time > l,
        timeInStock1 = timeInStock1 + time - invTime1(1);
        invTime1(1) = [];
        transCost12 = transCost12 + tau;
        incom1 = [incom1 time+L1];  % The other retailer orders a new item
        incomTrans2 = [incomTrans2 time+l];
        backTimeTrans2 = [backTimeTrans2 time];
    % If there are no items at either location, the item is backordered
    else
        backTime2 = [backTime2 time];
        incom2 = [incom2 time+L2];
    end
    T2(1) = [];

% An item arrives at location 1
elseif event == 3,
    incom1(1) = [];
    % If there are backorders, the item are given to a waiting customer
    if ~isempty(backTime1),
        timeInBackorder1 = timeInBackorder1 + time - backTime1(1);
        backTime1(1) = [];
    % If not, the item is put in inventory
    else
        invTime1 = [invTime1 time];
    end

% An item arrives at location 2
elseif event == 4,
    incom2(1) = [];
    % If there are backorders, the item are given to a waiting customer
    if ~isempty(backTime2),
        timeInBackorder2 = timeInBackorder2 + time - backTime2(1);
        backTime2(1)=[];
    % If not, the item is put in inventory
    else
        invTime2 = [invTime2 time];
    end

% An item arrives at location 1 by a lateral transshipment from location 2
elseif event == 5,
    incomTrans1(1) = [];
    timeInBackTrans1 = timeInBackTrans1 + time - backTimeTrans1(1);
    backTimeTrans1(1) = [];
```

```
    % An item arrives at location 2 by a lateral transshipment from location 1
    elseif event == 6,
        incomTrans2(1) = [];
        timeInBackTrans2 = timeInBackTrans2 + time - backTimeTrans2(1);
        backTimeTrans2(1) = [];
    end

    % Accumulates time with zero stock on hand
    if stock1 == 0,
        timeWithNoStock1 = timeWithNoStock1 + time - oldtime;
    end
    if stock2 == 0,
        timeWithNoStock2 = timeWithNoStock2 + time - oldtime;
    end
end

% After the time is up, remaining items in inventory and in backorder
% are accounted for in the accumulated times
if ~isempty(invTime1),
    for k = 1:length(invTime1),
        timeInStock1 = timeInStock1 + time - invTime1(1);
        invTime1(1) = [];
    end
end
if ~isempty(invTime2),
    for k = 1:length(invTime2),
        timeInStock2 = timeInStock2 + time - invTime2(1);
        invTime2(1) = [];
    end
end
if ~isempty(backTime1),
    for k = 1:length(backTime1),
        timeInBackorder1 = timeInBackorder1 + time - backTime1(1);
        backTime1(1) = [];
    end
end
if ~isempty(backTime2),
    for k = 1:length(backTime2),
        timeInBackorder2 = timeInBackorder2 + time - backTime2(1);
        backTime2(1) = [];
    end
end
if ~isempty(backTimeTrans1),
    for k = 1:length(backTimeTrans1),
```

```
            timeInBackTrans1 = timeInBackTrans1 + time - backTimeTrans1(1);
            backTimeTrans1(1) = [];
        end
    end
    if ~isempty(backTimeTrans2),
        for k = 1:length(backTimeTrans2),
            timeInBackTrans2 = timeInBackTrans2 + time - backTimeTrans2(1);
            backTimeTrans2(1) = [];
        end
    end

%     fillrate1 = 1 - timeWithNoStock1/stopTime;
%     fillrate2 = 1 - timeWithNoStock2/stopTime;

    % The costs are calculated

    transCost21 = transCost21/stopTime;
    transCost12 = transCost12/stopTime;

    holdCost1 = timeInStock1*h1/stopTime;
    holdCost2 = timeInStock2*h2/stopTime;
    backCost1 = (timeInBackorder1+timeInBackTrans1)*b1/stopTime;
    backCost2 = (timeInBackorder2+timeInBackTrans2)*b2/stopTime;

    totCost = holdCost1 + backCost1 + transCost21 ...
        + holdCost2 + backCost2 + transCost12;
    costVector = [costVector totCost];
    summa = summa + totCost;

%     fillVector1 = [fillVector1 fillrate1];
%     fillVector2 = [fillVector2 fillrate2];
end

meanTot = 1/n*summa;

sumTot = 0;
for i = 1:n,
    sumTot = sumTot + (costVector(i)-meanTot)^2;
end
stdTot = sqrt(1/(n-1)*sumTot)/sqrt(n);

%     meanfill1 = mean(fillVector1);
%     meanfill2 = mean(fillVector2);
```

B.2 Models for the (R, Q) policy inventory system

B.2.1 Without lateral transshipments

```
% function [meanTot, stdTot] = RQpolicy_no(R1, R2)
% Inventory system with two retailers applying (R,Q) policy.
% No lateral transshipments.
% Given reorder points, the function returns mean total cost
% and standard deviation.

function [meanTot, stdTot] = RQpolicy_no(R1, R2)

n = 10;                     % Number of runs

lambda1 = 10; lambda2 = 10; % Demand rates
L1 = 1; L2 = 1;             % Lead times

b1 = 20; b2 = 20;           % Backorder cost
h1 = 1; h2 = 1;             % Holding cost
A = 100;                    % Order cost

% The batch quantities by the EOQ formula
Q1 = round(sqrt(2*A*lambda1/h1)); Q2 = round(sqrt(2*A*lambda2/h2));

costVector = [];    % Vector with total cost for the runs
summa = 0;          % Sum of total costs for all runs

for j = 1:n,
    % Creates points in time when demand will occur for each warehouse
    % Times between customers are exp(lambda) distributed
    nbrOfEvents = 30000;
    u1 = rand(1,nbrOfEvents); u2 = rand(1,nbrOfEvents);
    y1 = -log(1-u1)/lambda1; y2 = -log(1-u2)/lambda2;

    T1 = cumsum(y1); T2 = cumsum(y2);
    stopTime = min([T1(nbrOfEvents) T2(nbrOfEvents)]);
    time = 0;

    invLevel1 = 0; invLevel2 = 0;   % Inventory levels
    invPos1 = 0; invPos2 = 0;       % Inventory positions
    invTime1 = []; invTime2 = [];   % Arrival times for units in inventory
    incom1 = []; incom2 = [];       % Arrival times for incoming items
    backTime1 = []; backTime2 = []; % Times when backorders have occurred

    timeInStock1 = 0; timeInStock2 = 0;         % Accumulated time in stock
```

```
timeInBackorder1 = 0; timeInBackorder2 = 0; % Accumulated time in backorder
orderCost1 = 0; orderCost2 = 0; % Cost for all orders during simulation

N1tot = 0; N2tot = 0;           % Number of customers during simulation
N1 = 0; N2 = 0;                 % Number of backorders during simulation

% The first tenth of the time is start-up time
transientTime = stopTime/10;
while time < stopTime,
    if time < transientTime,
        timeInStock1 = 0; timeInStock2 = 0;
        timeInBackorder1 = 0; timeInBackorder2 = 0;
    end

    % A trick to determine the next event
    if isempty(incom1),
        incom1 = realmax;
    end
    if isempty(incom2),
        incom2 = realmax;
    end

    % Determines the next event and the time when it happens
    [time, event] = min([T1(1) T2(1) incom1(1) incom2(1)]);

    if incom1(1) == realmax,
        incom1 = [];
    end
    if incom2(1) == realmax,
        incom2 = [];
    end

    % A customer arrives at retailer 1
    if event == 1,
        N1tot = N1tot + 1;
        % If there is stock on hand
        if invLevel1 >= 1,
            timeInStock1 = timeInStock1 + time - invTime1(1);
            invTime1 = invTime1(2:length(invTime1));
            invLevel1 = invLevel1 - 1;
            invPos1 = invPos1 - 1;
            % Orders new items while the inventory position is less than R
            while invPos1 <= R1,
                orderCost1 = orderCost1 + A;
                invPos1 = invPos1 + Q1;
```

```
            for k = 1:Q1
                incom1 = [incom1 time+L1];
            end
        end
    % If there are no items in stock, the demanded item is backordered
    else
        N1 = N1 + 1;
        backTime1 = [backTime1 time];
        invLevel1 = invLevel1 - 1;
        invPos1 = invPos1 - 1;
        % Orders new items while the inventory position is less than R
        while invPos1 <= R1,
            orderCost1 = orderCost1 + A;
            invPos1 = invPos1 + Q1;
            for k = 1:Q1
                incom1 = [incom1 time+L1];
            end
        end
    end
    T1 = T1(2:length(T1));

% A customer arrives at retailer 2
elseif event == 2,
    N2tot = N2tot + 1;
    % If there is stock on hand
    if invLevel2 >= 1,
        timeInStock2 = timeInStock2 + time - invTime2(1);
        invTime2 = invTime2(2:length(invTime2));
        invLevel2 = invLevel2 - 1;
        invPos2 = invPos2 - 1;
        % Orders new items while the inventory position is less than R
        while invPos2 <= R2,
            orderCost2 = orderCost2 + A;
            invPos2 = invPos2 + Q2;
            for k = 1:Q2
                incom2 = [incom2 time+L2];
            end
        end
    % If there are no items in stock, the demanded item is backordered
    else
        N2 = N2 + 1;
        backTime2 = [backTime2 time];
        invLevel2 = invLevel2 - 1;
        invPos2 = invPos2 - 1;
        % Orders new items while the inventory position is less than R
```

```
            while invPos2 <= R2,
                orderCost2 = orderCost2 + A;
                invPos2 = invPos2 + Q2;
                for k = 1:Q2
                    incom2 = [incom2 time+L2];
                end
            end
        end
        T2 = T2(2:length(T2));

    % An item arrives at retailer 1
    elseif event == 3,
        invLevel1 = invLevel1 + 1;
        incom1 = incom1(2:length(incom1));
        % If there are backorders, the item is given to a waiting customer
        if ~isempty(backTime1),
            timeInBackorder1 = timeInBackorder1 + time - backTime1(1);
            backTime1 = backTime1(2:length(backTime1));
        % If not, the item is put in inventory
        else
            invTime1 = [invTime1 time];
        end

    % An item arrives at retailer 2
    elseif event == 4,
        invLevel2 = invLevel2 + 1;
        incom2 = incom2(2:length(incom2));
        % If there are backorders, the item is given to a waiting customer
        if ~isempty(backTime2),
            timeInBackorder2 = timeInBackorder2 + time - backTime2(1);
            backTime2 = backTime2(2:length(backTime2));
        % If not, the item is put in inventory
        else
            invTime2 = [invTime2 time];
        end
    end
end

% After the time is up, remaining items in inventory and in backorder
% are accounted for in the accumulated times
if ~isempty(invTime1),
    for k = 1:length(invTime1),
        timeInStock1 = timeInStock1 + time - invTime1(1);
        invTime1 = invTime1(2:length(invTime1));
    end
```

```
    end
    if ~isempty(invTime2),
        for k = 1:length(invTime2),
            timeInStock2 = timeInStock2 + time - invTime2(1);
            invTime2 = invTime2(2:length(invTime2));
        end
    end
    if ~isempty(backTime1),
        for k = 1:length(backTime1),
            timeInBackorder1 = timeInBackorder1 + time - backTime1(1);
            backTime1 = backTime1(2:length(backTime1));
        end
    end
    if ~isempty(backTime2),
        for k = 1:length(backTime2),
            timeInBackorder2 = timeInBackorder2 + time - backTime2(1);
            backTime2 = backTime2(2:length(backTime2));
        end
    end

    stopTime = stopTime - transientTime;

%     fillrate1 = 1 - N1/N1tot;
%     fillrate2 = 1 - N2/N2tot;

    % The costs are calculated

    orderCost1 = orderCost1/stopTime;
    orderCost2 = orderCost2/stopTime;

    holdCost1 = timeInStock1*h1/stopTime;
    holdCost2 = timeInStock2*h2/stopTime;
    backCost1 = timeInBackorder1*b1/stopTime;
    backCost2 = timeInBackorder2*b2/stopTime;

    totCost = orderCost1 + holdCost1 + backCost1 ...
        + orderCost2 + holdCost2 + backCost2;
    costVector = [costVector totCost];
    summa = summa + totCost;

%     fillVector1 = [fillVector1 fillrate1];
%     fillVector2 = [fillVector2 fillrate2];
end

meanTot = 1/n*summa;
```

```
stdTot = std(costVector);

%     meanfill1 = mean(fillVector1);
%     meanfill2 = mean(fillVector2);
```

B.2.2 With lateral transshipments with zero lead time

```
% function [meanTot, stdTot] = RQpolicy_lt_0(R1, R2)
% Inventory system with two locations applying (R,Q) policy.
% Lateral transshipments allowed. Zero lead time.
% Given reorder points, the function returns mean total cost
% and standard deviation.

function [meanTot, stdTot] = RQpolicy_lt_0(R1, R2)

n = 10;                     % Number of runs

lambda1 = 10; lambda2 = 10; % Demand rates
L1 = 1; L2 = 1;             % Lead times

b1 = 20; b2 = 20;           % Backorder cost
h1 = 1; h2 = 1;             % Holding cost
A = 100;                    % Order cost
tau = 30;                   % Transshipment cost

% The batch quantities by the EOQ formula
Q1 = round(sqrt(2*A*lambda1/h1)); Q2 = round(sqrt(2*A*lambda2/h2));

costVector = [];    % Vector with total cost for the runs
summa = 0;          % Sum of total costs for all runs

for j = 1:n,
    % Creates points in time when demand will occur for each location
    % Times between customers are exp(lambda) distributed
    nbrOfEvents = 30000;
    u1 = rand(1,nbrOfEvents); u2 = rand(1,nbrOfEvents);
    y1 = -log(1-u1)/lambda1; y2 = -log(1-u2)/lambda2;

    T1 = cumsum(y1); T2 = cumsum(y2);
    stopTime = min([T1(nbrOfEvents) T2(nbrOfEvents)]);
    time = 0;

    invLevel1 = 0; invLevel2 = 0;   % Inventory levels
    invPos1 = 0; invPos2 = 0;       % Inventory positions
    invTime1 = []; invTime2 = []; % Arrival times for units in inventory
```

```
incom1 = []; incom2 = [];       % Arrival times for incoming items
backTime1 = []; backTime2 = []; % Times when backorders have occurred

timeInStock1 = 0; timeInStock2 = 0;         % Accumulated time in stock
timeInBackorder1 = 0; timeInBackorder2 = 0; % Accumulated time in backorder
orderCost1 = 0; orderCost2 = 0;     % Cost for all orders during simulation
transCost21 = 0; transCost12 = 0;   % Cost for all transshipments during sim.

N1tot = 0; N2tot = 0;           % Number of customers during simulation
N1 = 0; N2 = 0;                 % Number of backorders during

% The first tenth of the time is start-up time
transientTime = stopTime/10;
while time < stopTime,
    if time < transientTime,
        timeInStock1 = 0; timeInStock2 = 0;
        timeInBackorder1 = 0; timeInBackorder2 = 0;
    end

    % A trick to determine the next event
    if isempty(incom1),
        incom1 = realmax;
    end
    if isempty(incom2),
        incom2 = realmax;
    end

    % Determines the next event and the time when it happens
    [time, event] = min([T1(1) T2(1) incom1(1) incom2(1)]);

    if incom1(1) == realmax,
        incom1 = [];
    end
    if incom2(1) == realmax,
        incom2 = [];
    end

    % A customer arrives at location 1
    if event == 1,
        N1tot = N1tot + 1;
        % If there is stock on hand
        if invLevel1 >= 1,
            timeInStock1 = timeInStock1 + time - invTime1(1);
            invTime1 = invTime1(2:length(invTime1));
            invLevel1 = invLevel1 - 1;
```

```
        invPos1 = invPos1 - 1;
        % Orders new items while the inventory position is less than R
        while invPos1 <= R1,
            orderCost1 = orderCost1 + A;
            invPos1 = invPos1 + Q1;
            for k = 1:Q1
                incom1 = [incom1 time+L1];
            end
        end
    % If the other location has stock on hand, a lateral transshipment
    % is made
    elseif invLevel2 >= 1,
        timeInStock2 = timeInStock2 + time - invTime2(1);
        invTime2 = invTime2(2:length(invTime2));
        invLevel2 = invLevel2 - 1;
        invPos2 = invPos2 - 1;
        transCost21 = transCost21 + tau;
        % Orders new items while the inventory position is less than R
        % (at the other location)
        while invPos2 <= R2,
            orderCost2 = orderCost2 + A;
            invPos2 = invPos2 + Q2;
            for k = 1:Q2
                incom2 = [incom2 time+L2];
            end
        end
    % If there are no items at either location, the item is backordered
    else
        N1 = N1 + 1;
        backTime1 = [backTime1 time];
        invLevel1 = invLevel1 - 1;
        invPos1 = invPos1 - 1;
        % Orders new items while the inventory position is less than R
        while invPos1 <= R1,
            orderCost1 = orderCost1 + A;
            invPos1 = invPos1 + Q1;
            for k = 1:Q1
                incom1 = [incom1 time+L1];
            end
        end
    end
    T1 = T1(2:length(T1));

% A customer arrives at location 2
elseif event == 2,
```

```
N2tot = N2tot + 1;
% If there is stock on hand
if invLevel2 >= 1,
    timeInStock2 = timeInStock2 + time - invTime2(1);
    invTime2 = invTime2(2:length(invTime2));
    invLevel2 = invLevel2 - 1;
    invPos2 = invPos2 - 1;
    % Orders new items while the inventory position is less than R
    while invPos2 <= R2,
        orderCost2 = orderCost2 + A;
        invPos2 = invPos2 + Q2;
        for k = 1:Q2
            incom2 = [incom2 time+L2];
        end
    end
% If the other location has stock on hand, a lateral transshipment
% is made
elseif invLevel1 >= 1,
    timeInStock1 = timeInStock1 + time - invTime1(1);
    invTime1 = invTime1(2:length(invTime1));
    invLevel1 = invLevel1 - 1;
    invPos1 = invPos1 - 1;
    transCost12 = transCost12 + tau;
    % Orders new items while the inventory position is less than R
    % (at the other location)
    while invPos1 <= R1,
        orderCost1 = orderCost1 + A;
        invPos1 = invPos1 + Q1;
        for k = 1:Q1
            incom1 = [incom1 time+L1];
        end
    end
% If there are no items at either location, the item is backorderded
else
    N2 = N2 + 1;
    backTime2 = [backTime2 time];
    invLevel2 = invLevel2 - 1;
    invPos2 = invPos2 - 1;
    % Orders new items while the inventory position is less than R
    while invPos2 <= R2,
        orderCost2 = orderCost2 + A;
        invPos2 = invPos2 + Q2;
        for k = 1:Q2
            incom2 = [incom2 time+L2];
        end
```

```
            end
        end
        T2 = T2(2:length(T2));

    % An item arrives at location 1
    elseif event == 3,
        invLevel1 = invLevel1 + 1;
        incom1 = incom1(2:length(incom1));
        % If there are backorders, the item are given to a waiting customer
        if ~isempty(backTime1)
            timeInBackorder1 = timeInBackorder1 + time - backTime1(1);
            backTime1 = backTime1(2:length(backTime1));
        % If not, the item is put in inventory
        else
            invTime1 = [invTime1 time];
        end

    % An item arrives at location 2
    elseif event == 4,
        invLevel2 = invLevel2 + 1;
        incom2 = incom2(2:length(incom2));
        % If there are backorders, the item is given to a waiting customer
        if ~isempty(backTime2)
            timeInBackorder2 = timeInBackorder2 + time - backTime2(1);
            backTime2 = backTime2(2:length(backTime2));
        % If not, the item is put in inventory
        else
            invTime2 = [invTime2 time];
        end
    end
end

% After the time is up, remaining items in inventory and in backorder
% are accounted for in the accumulated times
if ~isempty(invTime1),
    for k = 1:length(invTime1),
        timeInStock1 = timeInStock1 + time - invTime1(1);
        invTime1 = invTime1(2:length(invTime1));
    end
end
if ~isempty(invTime2),
    for k = 1:length(invTime2),
        timeInStock2 = timeInStock2 + time - invTime2(1);
        invTime2 = invTime2(2:length(invTime2));
    end
```

```
    end
    if ~isempty(backTime1),
        for k = 1:length(backTime1),
            timeInBackorder1 = timeInBackorder1 + time - backTime1(1);
            backTime1 = backTime1(2:length(backTime1));
        end
    end
    if ~isempty(backTime2),
        for k = 1:length(backTime2),
            timeInBackorder2 = timeInBackorder2 + time - backTime2(1);
            backTime2 = backTime2(2:length(backTime2));
        end
    end

    stopTime = stopTime - transientTime;
%
%     fillrate1 = 1 - N1/N1tot;
%     fillrate2 = 1 - N2/N2tot;
%
    % The costs are calculated

    orderCost1 = orderCost1/stopTime;
    orderCost2 = orderCost2/stopTime;
    transCost21 = transCost21/stopTime;
    transCost12 = transCost12/stopTime;

    holdCost1 = timeInStock1*h1/stopTime;
    holdCost2 = timeInStock2*h2/stopTime;
    backCost1 = timeInBackorder1*b1/stopTime;
    backCost2 = timeInBackorder2*b2/stopTime;

    totCost = orderCost1 + holdCost1 + backCost1 + transCost21 ...
        + orderCost2 + holdCost2 + backCost2 + transCost12;
    costVector = [costVector totCost];
    summa = summa + totCost;

%     fillVector1 = [fillVector1 fillrate1];
%     fillVector2 = [fillVector2 fillrate2];
end


meanTot = 1/n*summa;
stdTot = std(costVector);

%     meanfill1 = mean(fillVector1);
```

```
%     meanfill2 = mean(fillVector2);
```

B.2.3 With lateral transshipments with positive lead time

```
% function [meanTot, stdTot] = RQpolicy_lt(R1, R2)
% Inventory system with two retailers applying (R,Q) policy.
% Lateral transshipments allowed. Positive lead time.
% Given reorder points, the function returns mean total cost
% and standard deviation.

function [meanTot, stdTot] = RQpolicy_lt(R1, R2)

n = 10;                     % Number of runs

lambda1 = 10; lambda2 = 10; % Demand rates
L1 = 1; L2 = 1;             % Lead times
l = 0.25;                   % Transshipment lead time

b1 = 20; b2 = 20;           % Backorder cost
h1 = 1; h2 = 1;             % Holding cost
A = 100;                    % Order cost
tau = 30;                   % Transshipment cost

% The batch quantities by the EOQ formula
Q1 = round(sqrt(2*A*lambda1/h1)); Q2 = round(sqrt(2*A*lambda2/h2));

costVector = [];    % Vector with total cost for the runs
summa = 0;          % Sum of total costs for all runs

for j = 1:n,
    % Creates points in time when demand will occur for each warehouse
    % Times between customers are exp(lambda) distributed
    nbrOfEvents = 30000;
    u1 = rand(1,nbrOfEvents); u2 = rand(1,nbrOfEvents);
    y1 = -log(1-u1)/lambda1; y2 = -log(1-u2)/lambda2;

    T1 = cumsum(y1); T2 = cumsum(y2);
    stopTime = min([T1(nbrOfEvents) T2(nbrOfEvents)]);
    time = 0;

    invLevel1 = 0; invLevel2 = 0;   % Inventory levels
    invPos1 = 0; invPos2 = 0;       % Inventory positions
    invTime1 = []; invTime2 = [];   % Arrival times for units in inventory
    incom1 = []; incom2 = [];       % Arrival times for incoming items
    backTime1 = []; backTime2 = []; % Times when backorders have occurred
```

```
timeInStock1 = 0; timeInStock2 = 0;         % Accumulated time in stock
timeInBackorder1 = 0; timeInBackorder2 = 0; % Accumulated time in backorder
orderCost1 = 0; orderCost2 = 0;     % Cost for all orders during simulation
transCost21 = 0; transCost12 = 0;   % Cost for all transshipments during sim.

N1tot = 0; N2tot = 0;           % Number of customers during simulation
N1 = 0; N2 = 0;                 % Number of backorders during simulation

% The first tenth of the time is start-up time
transientTime = stopTime/10;
while time < stopTime,
    if time < transientTime,
        timeInStock1 = 0; timeInStock2 = 0;
        timeInBackorder1 = 0; timeInBackorder2 = 0;
    end

    % A trick to determine the next event
    if isempty(incom1),
        incom1 = realmax;
    end
    if isempty(incom2),
        incom2 = realmax;
    end

    % Determines the next event and the time when it happens
    [time, event] = min([T1(1) T2(1) incom1(1) incom2(1)]);

    if incom1(1) == realmax,
        incom1 = [];
    end
    if incom2(1) == realmax,
        incom2 = [];
    end

    % A customer arrives at retailer 1
    if event == 1,
        N1tot = N1tot + 1;
        % If there is stock on hand
        if invLevel1 >= 1,
            timeInStock1 = timeInStock1 + time - invTime1(1);
            invTime1 = invTime1(2:length(invTime1));
            invLevel1 = invLevel1 - 1;
            invPos1 = invPos1 - 1;
            % Orders new items while the inventory position is less than R
```

```
        while invPos1 <= R1,
            orderCost1 = orderCost1 + A;
            invPos1 = invPos1 + Q1;
            for k = 1:Q1
                incom1 = [incom1 time+L1];
            end
        end
    % If the other retailer has stock on hand and the waiting time is longer
    % than the transshipment lead time, a lateral transshipment is made
    elseif invLevel2 >= 1 && incom1(1+length(backTime1)) - time > l,
        timeInStock2 = timeInStock2 + time - invTime2(1);
        invTime2 = invTime2(2:length(invTime2));
        invLevel2 = invLevel2 - 1;
        invPos2 = invPos2 - 1;
        transCost21 = transCost21 + tau;
        % Orders new items while the inventory position is less than R
        % (at the other retailer)
        while invPos2 <= R2,
            orderCost2 = orderCost2 + A;
            invPos2 = invPos2 + Q2;
            for k = 1:Q2
                incom2 = [incom2 time+L2];
            end
        end
    % If there are no items at either retailer, the item is backordered
    else
        N1 = N1 + 1;
        backTime1 = [backTime1 time];
        invLevel1 = invLevel1 - 1;
        invPos1 = invPos1 - 1;
        % Orders new items while the inventory position is less than R
        while invPos1 <= R1,
            orderCost1 = orderCost1 + A;
            invPos1 = invPos1 + Q1;
            for k = 1:Q1
                incom1 = [incom1 time+L1];
            end
        end
    end
    T1 = T1(2:length(T1));

% A customer arrives at retailer 2
elseif event == 2,
    N2tot = N2tot + 1;
    % If there is stock on hand
```

```
if invLevel2 >= 1,
    timeInStock2 = timeInStock2 + time - invTime2(1);
    invTime2 = invTime2(2:length(invTime2));
    invLevel2 = invLevel2 - 1;
    invPos2 = invPos2 - 1;
    % Orders new items while the inventory position is less than R
    while invPos2 <= R2,
        orderCost2 = orderCost2 + A;
        invPos2 = invPos2 + Q2;
        for k = 1:Q2
            incom2 = [incom2 time+L2];
        end
    end
% If the other retailer has stock on hand and the waiting time is longer
% than the transshipment lead time, a lateral transshipment is made
elseif invLevel1 >= 1 && incom2(1+length(backTime2)) - time > l,
    timeInStock1 = timeInStock1 + time - invTime1(1);
    invTime1 = invTime1(2:length(invTime1));
    invLevel1 = invLevel1 - 1;
    invPos1 = invPos1 - 1;
    transCost12 = transCost12 + tau;
    % Orders new items while the inventory position is less than R
    % (at the other location)
    while invPos1 <= R1,
        orderCost1 = orderCost1 + A;
        invPos1 = invPos1 + Q1;
        for k = 1:Q1
            incom1 = [incom1 time+L1];
        end
    end
% If there are no items at either retailer, the item is backordered
else
    N2 = N2 + 1;
    backTime2 = [backTime2 time];
    invLevel2 = invLevel2 - 1;
    invPos2 = invPos2 - 1;
    % Orders new items while the inventory position is less than R
    while invPos2 <= R2,
        orderCost2 = orderCost2 + A;
        invPos2 = invPos2 + Q2;
        for k = 1:Q2
            incom2 = [incom2 time+L2];
        end
    end
end
```

```
        T2 = T2(2:length(T2));

    % An item arrives at retailer 1
    elseif event == 3,
        invLevel1 = invLevel1 + 1;
        incom1 = incom1(2:length(incom1));
        % If there are backorders, the item are given to a wating customer
        if ~isempty(backTime1)
            timeInBackorder1 = timeInBackorder1 + time - backTime1(1);
            backTime1 = backTime1(2:length(backTime1));
        % If not, the item is put in inventory
        else
            invTime1 = [invTime1 time];
        end

    % An item arrives at retailer 2
    elseif event == 4,
        invLevel2 = invLevel2 + 1;
        incom2 = incom2(2:length(incom2));
        % If there are backorders, the item is given to a wating customer
        if ~isempty(backTime2)
            timeInBackorder2 = timeInBackorder2 + time - backTime2(1);
            backTime2 = backTime2(2:length(backTime2));
        % If not, the item is put in inventory
        else
            invTime2 = [invTime2 time];
        end
    end
end

% After the time is up, remaining items in inventory and in backorder
% are accounted for in the accumulated times
if ~isempty(invTime1),
    for k = 1:length(invTime1),
        timeInStock1 = timeInStock1 + time - invTime1(1);
        invTime1 = invTime1(2:length(invTime1));
    end
end
if ~isempty(invTime2),
    for k = 1:length(invTime2),
        timeInStock2 = timeInStock2 + time - invTime2(1);
        invTime2 = invTime2(2:length(invTime2));
    end
end
if ~isempty(backTime1),
```

```
        for k = 1:length(backTime1),
            timeInBackorder1 = timeInBackorder1 + time - backTime1(1);
            backTime1 = backTime1(2:length(backTime1));
        end
    end
    if ~isempty(backTime2),
        for k = 1:length(backTime2),
            timeInBackorder2 = timeInBackorder2 + time - backTime2(1);
            backTime2 = backTime2(2:length(backTime2));
        end
    end

    stopTime = stopTime - transientTime;

%     fillrate1 = 1 - N1/N1tot;
%     fillrate2 = 1 - N2/N2tot;

    % The costs are calculated

    orderCost1 = orderCost1/stopTime;
    orderCost2 = orderCost2/stopTime;
    transCost21 = transCost21/stopTime;
    transCost12 = transCost12/stopTime;

    holdCost1 = timeInStock1*h1/stopTime;
    holdCost2 = timeInStock2*h2/stopTime;
    backCost1 = timeInBackorder1*b1/stopTime;
    backCost2 = timeInBackorder2*b2/stopTime;

    totCost = orderCost1 + holdCost1 + backCost1 + transCost21 ...
        + orderCost2 + holdCost2 + backCost2 + transCost12;
    costVector = [costVector totCost];
    summa = summa + totCost;

%     fillVector1 = [fillVector1 fillrate1];
%     fillVector2 = [fillVector2 fillrate2];
end


meanTot = 1/n*summa;
stdTot = std(costVector);

%     meanfill1 = mean(fillVector1);
%     meanfill2 = mean(fillVector2);
```