KADIR HAS UNIVERSITY
SCHOOL OF GRADUATE STUDIES
DEPARTMENT OF ENGINEERING AND NATURAL SCIENCES

OPTIMUM SPARE PARTS INVENTORY CONTROL IN EXISTENCE OF A NON-STATIONARY INSTALLED BASE

ALİ KÖK
ASST. PROF. DR. MUSTAFA HEKİMOĞLU

MASTER’S DEGREE THESIS

ISTANBUL, JULY, 2021

Ali KÖK | Master's Degree Thesis | 2021

OPTIMUM SPARE PARTS INVENTORY CONTROL IN EXISTENCE OF A NON-STATIONARY INSTALLED BASE

ALİ KÖK

MASTER'S THESIS

Submitted to the School of Graduate Studies of
Kadir Has University in partial fulfillment of the requirements for the degree of
Master of Science in Industrial Engineering

İSTANBUL, JULY, 2021

DECLARATION OF RESEARCH ETHICS /
METHODS OF DISSEMINATION

I, ALİ KÖK, hereby declare that;

- this master's thesis is my own original work and that due references have been appropriately provided on all supporting literature and resources;
- this master's thesis contains no material that has been submitted or accepted for a degree or diploma in any other educational institution;
- I have followed *Kadir Has University Academic Ethics Principles prepared in accordance with The Council of Higher Education's Ethical Conduct Principles.*

In addition, I understand that any false claim in respect of this work will result in disciplinary action in accordance with University regulations.

Furthermore, both printed and electronic copies of my work will be kept in Kadir Has Information Center under the following condition as indicated below:

☐ The full content of my thesis will be accessible from everywhere by all means.

ALİ KÖK
29.07.2021

KADİR HAS UNIVERSITY

SCHOOL OF GRADUATE STUDIES

ACCEPTANCE AND APPROVAL

This work entitled OPTIMUM SPARE PARTS INVENTORY CONTROL IN EXISTENCE OF A NON-STATIONARY INSTALLED BASE prepared by ALİ KÖK has been judged to be successful at the defense exam on 29.07.2021 and accepted by our jury as master's thesis.

APPROVED BY:

Asst. Prof. Dr. Mustafa Hekimoğlu (Advisor) ....
Kadir Has University

Prof. Dr. Ahmet Deniz Yücekaya ....
Kadir Has University

Assoc. Prof. Dr. Deniz Karlı ....
Işık University

I certify that the above signatures belong to the faculty members named above.

....

Prof. Dr. Mehmet Timur Aydemir

Dean of School of Graduate Studies

DATE OF APPROVAL: 29.07.2021

TABLE OF CONTENTS

OPTIMUM SPARE PARTS INVENTORY CONTROL IN EXISTENCE OF A NON-STATIONARY INSTALLED BASE

ABSTRACT

In spare parts supply chains, demand is profoundly dependent on the life cycle of the product. Thus, MROs should incorporate installed base information in demand forecasting to prevent production/service interruptions and high holding costs. MROs also try to exploit secondary markets as a cheap and expedited source of spare parts apart from the OEM. However, the secondary markets are not reliable since they have a limited and stochastic spare parts capacity. Therefore, MROs need to determine when and how much to order from two supply sources. Under the assumption of stationary demand, a mathematical model is developed for an inventory control model in a dual sourcing setup. Then, this model is extended by assuming a non-stationary demand by employing Hekimoğlu and Karlı (2021)'s demand model. Optimal ordering policies are derived when the lead time difference of suppliers is one period, under both stationarity assumptions. Heuristics policies are utilized when the lead time difference is more than one period. It is found that the Dual Index policy outperforms other considered heuristics, resulting in a satisfactory cost deviation from the optimum cost. The value of higher moment information in demand forecasting is measured by simulation studies. Information of the first two and three moments are found to be superior over the other for declining and growing installed bases, respectively. The same simulation study is conducted by presenting an estimation error to the first moment. Results showed that the information of higher moments could save costs up to 14.2% and 9.26% for growth and decline phases, respectively. Finally, empirical analyses are conducted on a company from the Turkish automotive sector by performing statistical tests. It is concluded that Hekimoğlu and Karlı (2021)'s demand model could be practical to model spare parts demand of automobiles in the growth phase.

Keywords: optimum spare parts inventory control, non-stationary demand, installed base, secondary markets, Markovian capacity

DURAĞAN OLMAYAN KURULU SİSTEMLERİN VARLIĞINDA OPTİMUM YEDEK PARÇA ENVANTER KONTROLÜ

ÖZET

Yedek parça tedarik zincirlerinde talep ürünün yaşam döngüsüne ciddi oranda bağlılık göstermektedir. Bu nedenle Bakım Onarım Firmaları (BOF) üretim/hizmet kesintilerini ve yüksek elde tutma maliyetlerini önlemek için talep tahmin modellerinde kurulu sistem bilgisini de kullanmalıdır. BOF'lar, Orijinal Parça Üreticileri dışında ucuz ve hızlı bir yedek parça kaynağı olarak ikincil marketlerden de yararlanmaya çalışırlar. Ancak ikincil pazarlar sınırlı ve rassal bir yedek parça ka-pasitesine sahip oldukları için güvenilir değildir. Bu nedenle BOF'ların iki tedarik kaynağından ne zaman ve ne kadar sipariş vereceğini en iyi şekilde belirlemesi gerekir. Durağan talep varsayımı altında, iki farklı tedarik seçeneğinin bulunduğu bir envanter kontrol modeli için matematiksel bir model geliştirilmiştir. Daha sonra bu model Hekimoğlu ve Karlı (2021)'in talep modeli kullanılarak durağan olmayan bir talep varsayımıyla modellenmiştir. Optimum sipariş politikaları, her iki durağanlık varsayımı altında da kanıtlanmıştır. Termin süresi farkı bir periyottan fazla olduğunda sezgisel yöntemler kullanılmıştır. İkili indeks politikasının dikkate alınan diğer sezgisel yöntemlerden daha iyi performans gösterdiği ve optimum maliyetten sapmasının tatmin edici bir düzeyde olduğu bulunmuştur. Talep tahminlemesinde daha yüksek moment bilgisinin değeri simülasyon çalışmaları ile ölçülmüştür. İlk iki ve ilk üç moment sırasıyla azalan ve artan kurulu sistemler için diğerine göre daha üstün bulunmuştur. Aynı simülasyon çalışması birinci momente bir tahmin hatası eklenerek yapılmıştır. Elde edilen sonuçlar daha yüksek moment bilgisinin artan ve azalan kurulu sistemler için sırasıyla %14,2 ve %9,26'ya varan maliyet tasarrufu sağlayabileceğini göstermiştir. Son olarak Türk otomotiv sektöründen bir firmadan elde edilen veriler üzerinde istatistiksel testler aracılığıyla ampirik analizler yapılmıştır. Hekimoğlu ve Karlı (2021)'in talep modelinin büyüme aşamasındaki otomobillerin yedek parça talebinin modellenmesinde kullanılabileceği sonucuna varılmıştır.

Anahtar Sözcükler: optimum yedek parça envanter kontrolü, durağan olmayan talep, kurulu sistem, ikincil market, Markov kapasite

ACKNOWLEDGEMENTS

First of all, I would like to express my sincerest appreciation to my advisor Asst. Prof. Dr. Mustafa Hekimoğlu for his guidance and patience. I have learned a lot from his experiences. He has always devoted his time to discuss my research results and provided constructive comments. Thanks to him, I have learned how to approach a problem scientifically and to do research.

Second, I would like to thank Assoc. Prof. Deniz Karlı for taking his time whenever I need assistance with his profession, mathematical analysis, throughout our research project. He spent a whole semester teaching our research group real analysis. Thanks to him, I improved my mathematical analysis skills.

Third, I would like to express my heartfelt gratitude to Prof. Dr. Ahmet Deniz Yücekaya for guiding me to an academic career. I have conducted my undergraduate thesis under his supervision and started to grow an interest in doing research.

Last but not least, I would like to express my deepest gratitude to my one only Ece and my family. I would not be able to come through the hectic periods without their support and encouragement. They have always stood by me regardless of any situation and given me morale.

To my one only Ece and my dearest family...

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS/ABBREVIATIONS

b	Backlog cost per unit
c^r	Unit ordering cost of regular supplier
c_0	Base price for a spare part in secondary market
D	Stationary demand
D_t	Non-sationary demand at time t
$\mathbb{E}$	Expectation operator
h	Holding cost per unit
K	Capacity of the secondary market in the decision period
K_+	Capacity of the secondary market in the following period
l^r	Lead time of regular supplier
N_t	Size of the installed base at time t
N_0	Number of capital products at the beginning of decline phase
q^r	Quantity of order placed on regular supplier
q^s	Quantity of order placed on secondary supplier
y_t	Inventory on hand at time t
P	Markov transition probability matrix
$\mathbb{R}$	Set of real numbers
T	Length of planning horizon
α	Rate of Poisson process representing product failures
ϵ	Error coefficient
γ	Discount factor
Λ	Rate of the stationary Poisson distribution
λ	Rate of Poisson process representing product installements/retirements
Ω	State space of Markov chain
ρ	Error level
CDF	Cumulative distribution function
MRO	Maintenace Repair Organization

OEM	Original Equipment Manufacturer
PMF	Probability mass function

1. INTRODUCTION

Capital products are the tools/equipment used for the production of goods and services. Hence, they pose great importance for the uninterrupted continuity of the production of goods and services. For this reason, capital products are subject to maintenance. These maintenance activities may be held regularly or randomly. In both cases, the users of the capital products expect the maintenance firms to keep sufficient spare parts inventory to prevent manufacturing or service interruptions. Spare parts supply chains differ from other supply chains due to the installed base dependency and the non-stationary, intermittent, and slow-moving nature of the demand.

Longman (2007) defines the installed base as the entire collection of systems or products a firm has sold and is still in use. Dekker et al. (2013) adapted this definition to spare parts logistics as the whole set of systems/products for which a company provides after-sales services by stressing that the original equipment manufacturer (OEM) is not required to be the organization that provides maintenance services. These organizations are called Maintenance Repair Organization (MRO) in the literature. In spare parts supply chains, demand is heavily dependent on the number of capital goods in use by the customers (the size of the installed base). For this reason, producers of capital goods or maintenance providers try to monitor the number of capital products used and their utilization rates closely.

E-commerce is widely used by MROs for business-to-business (B2B) exchanges of used products in spare parts supply chains (Hekimoğlu, 2015). These marketplaces are called secondary (or gray) markets in the literature. Parties on e-commerce platforms generally sell their second-hand inventory cheaper than their original supplier.

In addition, since they do not need time for production, gray markets typically deliver faster (Hekimoğlu, 2015). For a company that provides maintenance service to capital products, secondary markets are not only sources of spare parts but also are used to sell excess inventory to get extra income and reduce holding costs. Recent studies show that secondary markets allow retailers to make bulk purchases from their suppliers and exploit quantity discounts while selling the excess inventory when needed (Hu, Pavlin and Shi, 2013). As a result, purchasers benefit from trading with gray markets in various industries.

Purchasing from secondary markets is an advantageous option for maintenance firms' inventory control policy as they are cheaper and faster. On the other hand, the number of spare parts in secondary markets is limited and changes randomly over time. The random spare parts availability on secondary markets prevents the secondary market from being a reliable supply source. Hence, MROs consider secondary markets and original equipment manufacturers simultaneously and need to seek methods to calculate when and how much to order from both sources. The studies that focus on this problem -sourcing from two different supply modes- are called dual sourcing in the literature.

As mentioned earlier, spare parts demand depends on the size of the active installed base. The introduction of the new capital products to the market or the retirement of the capital products are determinants of the active installed base size. These factors change during different phases of a product's life cycle. According to their observations on installed base data, Dekker et al. (2013) classified this life cycle into three phases: initial phase, maturity phase and end-of-life phase (see Figure 1.1). Throughout this thesis, the initial phase is called the growth phase or growing installed base. Similarly, the end-of-life phase is called the decline phase or declining installed base. The visualization of the life cycle of a capital product taken from Dekker et al. (2013) is given in Figure 1.1.

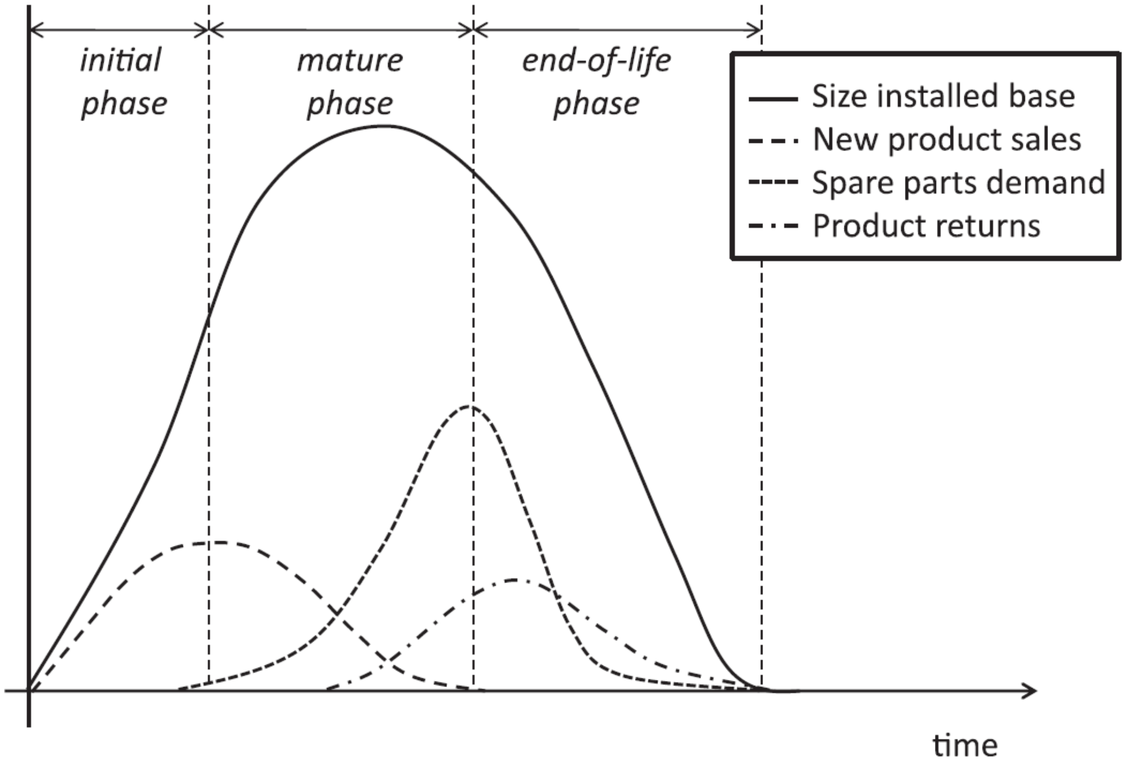


Figure 1.1 Life cycle of a capital product (Dekker et al., 2013).

As seen in Figure 1.1, spare parts demand varies significantly at each stage. The initial phase starts with the introduction of the capital product to the market. First spare parts demand is seen in this period. New product sales make a peak during the transition to the maturity stage. In the maturity phase, new product sales occur with a downwards trend, but product returns also start to take place in this phase. Therefore, it is relatively easier to forecast demand in the maturity phase since product returns and product sales balance each other. Most of the studies in the literature make the assumption that capital products and parts are in their maturity phase; hence the spare part demand is stationary. Finally, the end-of-life phase starts with the product being withdrawn from the market. During this phase, there is no new product sale but only product returns or retirements. Hence, shrinking installed base size reduces the demand for spare parts.

Note that the dynamics of this life cycle may change depending on the (design) characteristics of the capital product. For instance, a technical failure may cause spare part demand to occur at the beginning of the initial phase. Furthermore, planned maintenance or regulations may lead spare parts to be changed without any malfunction. For example, airplanes are subject to scheduled maintenance based on fly hour due to the strict regulations on the aviation industry (Dekker et al., 2013). Similarly, some parts of automobiles are changed regularly based on the distance traveled by the vehicle. Also, note that conditions in which the product is used may alter the spare parts demand along with the installed base size and (design) characteristics, e.g., environments such as deserts could make products or parts deteriorate faster (Dekker et al., 2013).

Initial and end-of-life are the phases where the spare parts demand depends on the installed base most. During these phases, spare parts demand follows a non-stationary distribution with different characteristics. Therefore, demand forecasting is challenging, and MROs should employ different inventory control policies for these phases. To incorporate the non-stationarity of spare parts demand into inventory control model, demand model developed by Hekimoğlu and Karlı (2021) is utilized. Structure of this model explained in Section 4.2. As mentioned before, the dependency of spare parts demand in the end-of-life phase stems from capital product retirements and failures. On the other hand, the retired capital products may be dismantled and used as a source of spare parts. Hence, there is a dependency between available spare parts in the secondary markets and different phases of the installed base.

To reflect this dependency, the secondary market's capacity is assumed to emerge following a discrete-time Markov chain. It is assumed that the number of available spare parts is known in the decision period and random in the following period. It is pretty complex to model dependency between the capacity of the secondary markets and installed base directly (see Hekimoğlu and Karlı, 2021); that is why the capacity of the secondary markets is assumed to evolve Markovian. The expected capacity of the secondary markets may show a tendency to increase or decrease based on the phase of the installed base. These market scenarios are modeled using stochastically increasing and decreasing transition probability matrices. It is also possible for capacity to have equal probabilities of growing and of declining. This market scenario is modeled using a symmetric transition probability matrix. Explicit forms of transition probability matrices are given in Sections 3.3 and 4.3.

Another result of the fluid nature of secondary markets is dynamic pricing. Since the capacity of the secondary markets changes over time depending on the phase of the installed base, the suppliers in the secondary markets employ different pricing policies based on their capacity and size of the order placed by customers. As a spare part gets scarce in the secondary market, its price go up. Similarly, when the order size increases, the suppliers makes the unit spare part price higher. In other words, as the number of available parts in the secondary markets rises, spare part prices decline and vice versa. As the amount of orders placed on the secondary market grows, spare part prices rise and vice versa. Therefore, there is an inverse proportion between the capacity of the secondary markets and the spare part prices while the order quantity and the spare part prices are directly proportional. In the conventional dual sourcing problem setup, the slow supplier is cheap, and the fast supplier is expensive. Therefore, customers have to opt for speed or price. This setup is not suitable for the problem structure in this thesis since secondary markets sell second-hand spare parts without a need to manufacture them (Hekimoğlu, 2015). Thus, the regular supplier is slower and usually expensive in our problem settings. The secondary supplier, however, is slower and usually cheaper. Thus, a dynamic acquisition cost function that changes according to market capacity and order quantity is presented for gray markets. Furthermore, it is assumed that the orders placed on secondary market delivered within the same period while regular supplier delivers after a certain lead time.

The organization of the thesis is as follows: the related works from the literature are presented in Chapter 2. In Chapter 3, an inventory model is developed under the assumption of stationary demand, and optimum purchasing policy is derived when lead time of OEM is one. Heuristics policies are utilized for general lead time, and their performances are presented. In Chapter 4, inventory control model presented in Chapter 3 is extended by assuming a non-stationary demand. The optimum ordering policy for lead time of one is derived. A simulation based approach is proposed for current heuristic policies for general lead time. Lastly, performances of different distribution selection algorithms compared via simulation studies. In Chapter 5, an empirical study conducted on Turkish automotive industry is presented. The Poisson process assumption on capital product installments and performance of the non-stationary demand model is tested for growing installed bases by performing statistical tests on empirical automobile sales and spare parts replacement data. Finally, the conclusions are presented in Chapter 6.

2. LITERATURE REVIEW

The literature review is introduced as two streams of studies. The first stream of studies is about optimum inventory control policies in general. The second stream of studies is about the installed bases, which is a supplementary concept for spare parts inventory control.

Scarf (1959) considers a single source dynamic inventory control problem and shows the optimal policy to be an (S,s) type when the inventory-related costs are convex. Namely, (S,s) policy stands for ordering up to order-up-to level S when the inventory position (inventory on hand plus outstanding orders) falls under reorder point s. He also uses the concept of K-convexity when there is a reordering cost-can also be assumed as fixed ordering or setup cost-. Fukuda (1964) studies an inventory control problem with two and three delivery modes and lead time differences of one, where ordering costs are linear and different for each supplier. This study constructs the fundamental cost function using dynamic programming used by most subsequent studies in inventory management literature and proves the optimal policy as a two-level base stock policy. Specifically, the base stock policy orders up to a certain level when the inventory position is less than that level. Whittemore and Saunder (1977) prove the optimal inventory management policy under stochastic demand with two supply options and consecutive lead times under more general conditions than the prior studies. They also extend their case to non-consecutive lead times and indicate that the optimal policy is in a complex form and challenging to achieve. Federgruen and Zipkin (1986) show the optimal policy of a single-source inventory system under a limited production capacity. Their contribution to the literature is the existence of a boundary on the production capacity. This capacity may be treated as the supplier's capacity depending on the problem setup as it is in this study. Yazlalı and Erhun (2009) study a dual-source inventory system where both suppliers have minimum and maximum supply limits and lead time differences of one. They show that the optimal policy is a modified two-level base stock policy. They also show state-dependency of the optimal policy using a new functional property, namely, bounded increasing (or decreasing) differences, the modified version of supermodularity (or submodularity). Tan, Feng and Chen (2016) study an inventory control problem with two delivery modes with random capacities and consecutive lead times and show the optimal policy. Their study is the first one that considers unreliable suppliers with different delivery times. One main difference between their work and ours is that they assume that the capacity is deterministic in the former period and random in the current period. In our case, the capacity is deterministic in the current period and stochastic in the following period. All the studies mentioned earlier, except for Tan, Feng and Chen (2016), assume that the fast supplier is expensive, and the slow supplier is cheap. Tan, Feng and Chen (2016) consider the case in which fast supplier may be cheaper or more expensive than slow supplier. We made the same assumption in this study. However, we provide an explicit formulation for the dynamic acquisition cost of the slow supplier so that the cost of the slow supplier changes over time based on different parameters while they assume different constant acquisition cost parameters. Hekimoğlu (2015) is the first study that considers secondary markets for spare parts logistics, yet he assumes that the demand is stationary. To the best of our knowledge, Yang, Qi and Xia (2005) is the closest study to our work. They consider a production inventory system where the random demand may be satisfied via in-house production (fast supplier) or outsourcing (slow supplier). They assume the fast supplier-regular supplier in our case- to be cheap and slow supplier -secondary markets in our case- to be expensive. Nevertheless, they assume fixed acquisition costs for both suppliers, similar to Tan, Feng and Chen (2016). Moreover, the fast supplier has a Markovian capacity while the slow supplier has no capacity constraint, as it is in our case. The main difference of our study from the all aforementioned studies is that we assume a non-stationary demand which stems from the dynamics of the growing and declining phases of the spare parts installed bases.

For the second stream of studies, Dekker et al. (2013) explain the install base concept for spare parts logistics and make a significant contribution to the literature. In this study, they review the advantages of using installed base information for different scenarios giving examples from the industry. They conclude that time series forecasting techniques that depend on historical data are not effective in the presence of non-stationary installed bases. Thus, installed base information should be used to make reliable forecasts. Van der Auweraer, Zhu and Boute (2021) study the benefits of installed base information in spare part inventory management. Parallel to Dekker et al. (2013), their results show that the forecast methods using installed base information reduce the inventory-related cost compared to classical time series forecasting methods. Their main contribution is approaching the problem from a different perspective by questioning what kind of installed base information is most valuable in spare part inventory control. They conclude that the knowledge of the active installed base size is the most valuable installed base information, especially in the existence of a non-stationary installed base. Pinçe and Dekker (2011) study a continuous review inventory control system where the demand for spare parts falls on a previously known date aiming to reduce obsolescence cost. They suggest that information about this kind of demand drop could be extracted by tracking the installed base. They provide a solution that makes a policy change in advance to prevent obsolescence cost. Jin and Liao (2009) propose a model for the total repair demand of a product that is in the growing phase of the installed base. They assume that new product sales follow a homogeneous Poisson process, and arrival times of product failures (maintenance demand) follow an Exponential distribution. They derive explicit formulas of aggregate maintenance demand's first two moments (mean and variance). Hekimoğlu and Karlı (2021) extend the model of Jin and Liao (2009) for growing installed bases by deriving explicit formula of the third moment. Their main contribution is constructing a similar model for declining installed bases. They employ two homogeneous Poisson processes representing product retirements and spare part breakdowns to model non-stationary spare parts demand. Again, they derive closed-form formulas of the first three central moments to characterize demand distribution. For both phases of the installed base, they use a distribution selection algorithm that uses the calculated moments to determine a probability distribution for each period. Throughout this thesis, the models developed by Hekimoğlu and Karlı (2021) for growing and installed bases are utilized to model non-stationary demand in respective phases of the installed base. The detailed explanation of their model is given in Section 4.2.

3. INVENTORY CONTROL MODEL WITH A STATIONARY DEMAND DISTRIBUTION

In order to have preliminary information about the dynamics of the system, the inventory control model is first modeled under the assumption of stationary demand distribution. In other words, the demand distribution stays same over whole planning horizon with fixed parameters. A mathematical model that minimizes the total inventory cost-the sum of purchasing, holding, and backlog costs- is developed for a periodic review inventory system. An optimum inventory control policy is determined in the presence of stationary spare parts demand and the Markovian secondary market capacity.

3.1 Mathematical Model

The inventory control model is first developed assuming that the lead time is one period, then this model is extended to the case where the lead time is a general integer. The inventory control model is derived using stochastic dynamic programming.

A T-period planning problem where periodic review is implemented is considered. Throughout this thesis, it is assumed that the OEM, regular supplier, delivers after a positive lead time, $l^r > 0$, and orders placed in the secondary markets are delivered within the same period. Deliveries from both suppliers are included in the same inventory, as there is no difference in quality between the parts from the secondary market and the regular supplier. Spare parts demand, if sufficient, is met from the existing stock. In case of lacking stock, the demand is backlogged to be met at the next delivery. The dynamics of the system are as follows:

1. At the beginning of period t, the inventory on hand, y_t, is observed after receiving incoming orders placed l^r periods before on regular supplier.
2. The capacity of the secondary market is observed.
3. Orders on regular and secondary suppliers are placed.
4. Orders placed on the secondary market are delivered.
5. Random demand, D, occurs (demand realization).
6. Holding, backlog and ordering costs are calculated and added to the total cost with a certain discount rate.

3.1.1 Inventory control model with a lead time of one period

Assuming that the lead time is one period, $l^r = 1$, and following the dynamics given in the former section, the single period cost minimization function is:

$$V(K, y) = \min_{\substack{q^r \geq 0, \\ K \geq q^s \geq 0,}} \{c^r q^r + c(q^s, K) + L(y + q^s)\}. \tag{3.1}$$

In Equation (3.1), the first term in the minimization is the acquisition cost of the regular supplier, where c^r and q^r represent the unit ordering cost of regular supplier and quantity of order placed on regular supplier, respectively. The second term is the dynamic cost function representing the acquisition cost of the secondary supplier, where K is the capacity of the secondary market in the decision period, and q^s is order quantity, and defined as:

$$c(q, K) = c_0 K^{\eta} q^{\xi}, \ \eta \leq 0, \ \xi \geq 1. \tag{3.2}$$

As mentioned before, the number of available parts in the secondary markets changes over time depending on the phase of the installed base. The retailers in the secondary markets apply different pricing policies based on their capacity and size of the order placed by customers. Equation (3.2) provides an explicit formula to reflect this dependency.

In Equation (3.2), c_0 is the base price for a spare part, q is the order quantity, and K is the capacity of the secondary market. η and ξ are the constants used to model to what extent order quantity and capacity will affect the spare part price. When η is smaller than zero, the higher the market capacity, the lower the unit spare part price. When η is equal to zero, market capacity does not affect unit spare parts price, which is the common assumption in the literature. Similarly, since ξ is greater than or equal to one, the higher the order quantity, the higher the unit spare part price. The visualization of unit spare parts price, $(\frac{c(q,k)}{q})$, where $c_0 = 5$, $\eta = -0.5$, $\xi = 1.5$ is given in Figure 3.1. Also, the dynamic cost function is convex increasing in order quantity. The visualization of this property using the same parameters with previous example is given in Figure 3.2.

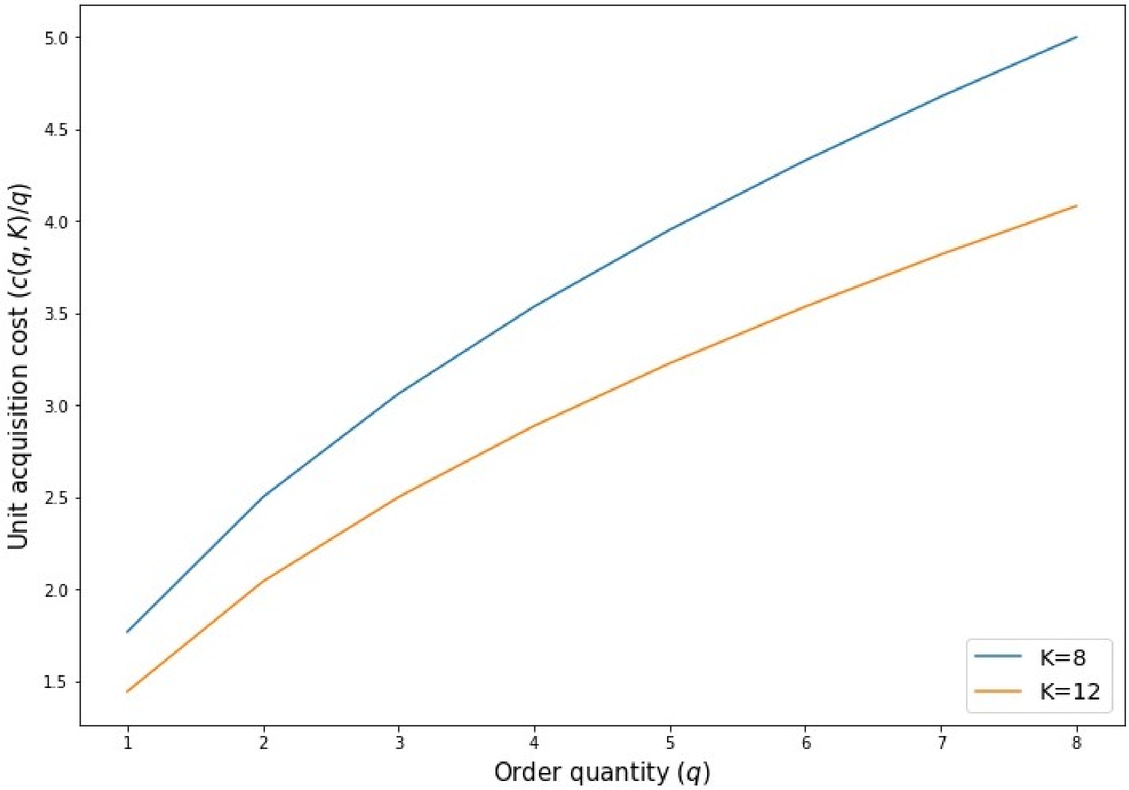


Figure 3.1 Example visualization of unit spare part price.

The dynamic acquisition cost function also allows using a linear procurement cost function, a common practice in the literature. To this end, η needs to be set as zero, and ξ needs to be set as one, which leads to the formulation: $c(q, K) = c_0 q,\ 0 \leq q \leq K$. In this case, the base spare part price of secondary markets, c_0, becomes the unit spare part price. In other words, the unit ordering cost does not change depending on the market capacity and order quantity.

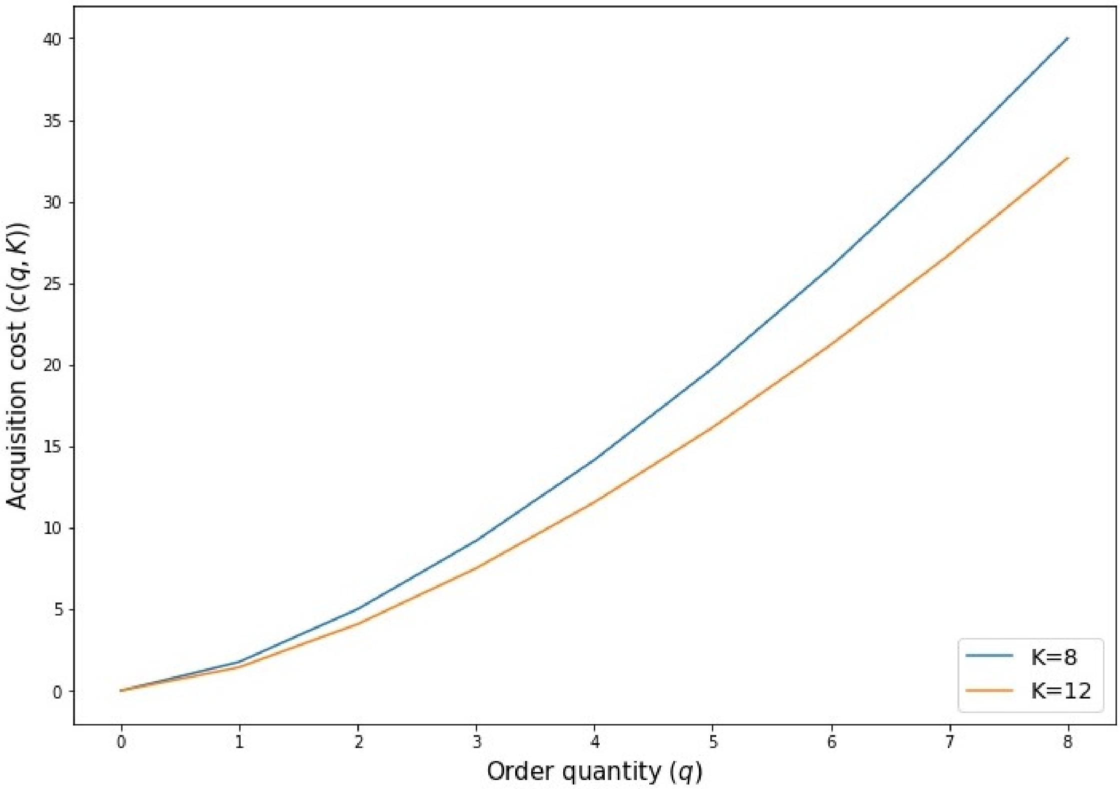


Figure 3.2 Example visualization of dynamic acquisition cost function.

It is assumed that the number of spare parts in the secondary markets emerges following a discrete-time Markov chain. Ω denotes the finite and countable space on which this Markov chain is defined. The states of the Markov chain are assumed to evolve between different points within this space with the transition probability matrix P.

In Equation (3.1), $L(y) := \mathbb{E}\left[h(y-D)^+ + b(D-y)^+\right]$ gives the expected inventory-related cost where $x^+ = \max(0, x)$, h is holding cost per unit for excess inventory, and b is penalty cost per unit for backlogged orders. It is known in the inventory control literature that $L(y)$ is convex (Fukuda, 1964; Porteus, 2009). The finite-horizon, multi-period and discounted total cost function becomes:

$$V_t(K, y_t) = \min_{\substack{q_t^r \geq 0, \\ K \geq q_t^s \geq 0,}} \{J_t(K, y_t, q_t^r, q_t^s)\}, \quad t = 1, 2, 3, \ldots, T, \tag{3.3}$$

$$J_t(K, y_t, q_t^r, q_t^s) = c^r q_t^r + c(q_t^s, K) + L(y_t + q_t^s) \\ + \gamma\, \mathbb{E}[V_{t+1}(K_+, y_t + q_t^r + q_t^s - D)]. \quad (3.4)$$

In Equation (3.4), the last term is the recursive term that represents the cost of the next period. K_+ is the random variable representing the number of items that will be available in the secondary market in the following period, and γ is the discount rate in each period. Also, note that the second term of the V_{t+1} is the inventory transferred to the next period, which corresponds to:

$$y_{t+1} = y_t + q_t^r + q_t^s - \mathbb{E}[D]. \quad (3.5)$$

The finite-horizon dynamic minimization problem is defined on the two-dimensional state space $(K, y) \in (\Omega \times \mathbb{R})$. The first variable of the state space is the number of parts available at the secondary market in a period, and the second one is the inventory on hand. The minimization problem has a two-dimensional action space such that $(q_t^r, q_t^s) \in (\mathbb{R} \times \mathbb{R})$.

Proposition 3.1.1. *The following statements are true for the multi-period, finite-horizon and discounted total cost function given in Equations* (3.3) *and* (3.4) *under the assumptions of stationary demand and* $l^r = 1$*:*

a. $J_t(K, y_t, q_t^r, q_t^s,)$ *is convex in* (q_t^r, q_t^s) *for given values of* K *and* y_t*.*
b. $V_t(K, y_t)$ *is convex in* y_t *for a given value of* K*.*
c. Optimum control policy is given by a base stock policy.

Proof. The proof of this proposition will be carried out by the method of proof by induction. Firstly, for $t = T$, $J_T(K, y_T, q_T^r, q_T^s)$ is a linearly increasing function in q_T^r. $J_T(K, y_T, q_T^r, q_T^s)$ is a convex function in (y_T, q_T^r, q_T^s) since $c(q_T^s, K)$ and $L(y_T + q_T^s)$ are convex functions in their parameters. Due to the preservation of convexity under minimization, $V_T(K, y_T)$ is a convex function in y_T. Thus, for period T, there is an optimum value of (q_T^r, q_T^s). Hence, induction is completed for $t = T$. Suppose that the expressions a, b, and c in the proposition are hold for $t + 1$. For period t, since $c(q_t^s, K)$ and $L(y_t + q_t^s)$ are convex functions, and $\mathbb{E}[V_{t+1}(K, y_t + q_t^s - D)]$ is convex,

$J_t(K, y_t, q_t^r, q_t^s)$ is a convex function in (y_t, q_t^r, q_t^s). As a result of minimization of $J_t(K, y_t, q_t^r, q_t^s)$ in (q_t^r, q_t^s), $V_t(K, y_t)$ is convex in y_t for a given K. Thus, proofs of statements a and b are completed. Statement b implies statement c. Hence, the proof is concluded. □

To prove the structure of optimum policy for $l^r = 1$ by using the mathematical model given in Equations (3.3) and (3.4), firstly, the following variable transformation is needed to be done:

$$w = y + q^s,$$
$$v = w + q^r.$$

Using this transformation, mathematical model given in Equations (3.3) and (3.4) could be expressed as below:

$$\tilde{V}_t(K, y) = \min_{\substack{v \geq w, \\ K+y \geq w \geq y}} \left\{ \tilde{J}_t(K, y, v, w) \right\}, \qquad t = 1, 2, 3, \ldots, T, \tag{3.6}$$

$$\tilde{J}_t(K, y, v, w) = c^r(v - w) + c_0(w - y)^\xi K^\eta + L(w) + \gamma\, \mathbb{E}[\tilde{V}_{t+1}(K_+, v - D)]. \tag{3.7}$$

Equation (3.7) could be expressed for the last period as below:

$$\tilde{J}_T(K, y, v, w) = c^r(v - w) + c_0(w - y)^\xi K^\eta + L(w). \tag{3.8}$$

It is assumed that $\eta = 0$, $\xi = 1$. The structure of the optimum policy and the value of the cost function are analyzed numerically for the general values of these parameters. For $\eta = 0$, $\xi = 1$, Equation (3.6) becomes:

$$\tilde{V}_t(K, y) = \min_{\substack{v \geq w, \\ K+y \geq w \geq y}} \left\{ c^r(v - y) + c_0(v - w) + L(w) + \gamma\, \mathbb{E}[\tilde{V}_{t+1}(K_+, v - D)] \right\}. \tag{3.9}$$

Let us define $c = -c^r + c_0$, then Equation (3.9) becomes:

$$\tilde{V}_t(K, y) = \min_{\substack{v \geq w, \\ K+y \geq w \geq y}} \left\{ c^r(v - y) + c(w - y) + L(w) + \gamma\, \mathbb{E}[\tilde{V}_{t+1}(K_+, v - D)] \right\}. \tag{3.10}$$

By using the second and the third terms in minimization, we could define:

$$\tilde{L}(K, y) = \min_{K+y \geq w \geq y} \{c(w - y) + L(w)\}. \tag{3.11}$$

The value that minimizes the unconstrained version of the mathematical model in Equation (3.11) is denoted as w^*. From this point of view, the optimum function can be characterized by the following proposition. To prove this proposition, the method developed by Fukuda (1964) is adapted to the case where the expedited supplier has a limited capacity.

Proposition 3.1.2. *The optimum policy that minimizes the mathematical model given in Equations* (3.3) *and* (3.4) *could be expressed using two parameters,* $v^*(K)$ *and* $w^*(K) = (w^* \vee (y + K)) \wedge y$, *that are functions of the market capacity in a period:* $q_t^s = (w_t^*(K) - y_t)$ *and* $q_t^r = (v_t^*(K) - w_t^*(K))$.

Proof. For the optimization of the function given in Equation (3.7), it will be analyzed where w^*, which is the point that minimizes $\tilde{L}(K, y)$, falls within the $[y, y+K]$ interval. Such that, if $w^* < y < y+K$, then $\tilde{L}(K, y) = L(y)$. Thus, the minimization in Equation (3.10) becomes:

$$\tilde{V}_t(K, y) = \tilde{L}(K, y) + \min_{v \geq y} \left\{ c^r(v - y) + \gamma\, \mathbb{E}[\tilde{V}_{t+1}(K_+, v - D)] \right\}. \tag{3.12}$$

If $y < w^* < y + K$, then $\tilde{L}(K, y) = c(w^* - y) + L(w^*)$. For $v > w^*$, the minimization in Equation (3.10) becomes:

$$\tilde{V}_t(K, y) = \tilde{L}(K, y) + \min_{v \geq w^*} \left\{ c^r(v - y) + \gamma\, \mathbb{E}[\tilde{V}_{t+1}(K_+, v - D)] \right\}. \tag{3.13}$$

For $v^* \leq w^*$,

$$\begin{aligned} \tilde{V}_t(K, y) = \tilde{L}(K, y) + \min_{w^* \geq v \geq y} \Big\{ & c^r(v - y) + c(v - y) \\ & + L(v) - \tilde{L}(K, y) + \gamma\, \mathbb{E}[\tilde{V}_{t+1}(K_+, v - D)] \Big\}, \end{aligned} \tag{3.14}$$

$$\begin{aligned} \tilde{V}_t(K, y) = \tilde{L}(K, y) + \min_{w^* \geq v \geq y} \Big\{ & c^r(v - y) + c(v - w^*) \\ & + L(v) - L(w^*) + \gamma\, \mathbb{E}[\tilde{V}_{t+1}(K_+, v - D)] \Big\}. \end{aligned} \tag{3.15}$$

If $y + K < w^*$, then $\tilde{L}(K, y) = c(K) + L(y + K)$. Thus, if $v > w^* > y + K$ or $w^* > v > y + K$, then the minimization in Equation (3.10) becomes:

$$\tilde{V}_t(K, y) = \tilde{L}(K, y) + \min_{v \geq y+K} \left\{ c^r(v - y) + \gamma\, \mathbb{E}[\tilde{V}_{t+1}(K_+, v - D)] \right\}, \tag{3.16}$$

$$\tilde{V}_t(K, y) = cK + L(y + K) + \min_{v \geq y+K} \left\{ c^r(v - y) + \gamma\, \mathbb{E}[\tilde{V}_{t+1}(K_+, v - D)] \right\}. \tag{3.17}$$

If $w^* > y + K > v \geq y$, then

$$\begin{aligned} \tilde{V}_t(K, y) = cK + L(y + K) + \min_{v \geq y+K} \Big\{ & c^r(v - y) + c(v - y - K) \\ & + L(v) - L(y - K) + \gamma\, \mathbb{E}[\tilde{V}_{t+1}(K_+, v - D)] \Big\}. \end{aligned} \tag{3.18}$$

To organize Equations (3.12)-(3.18) in a single model, let us define $z = \min(y + K, w^*)$ and

$$\Lambda(v) = \begin{cases} 0, & \text{if } v \geq z, \\ L(v) + L(z) + c(v - z), & \text{if } v < z. \end{cases} \tag{3.19}$$

Since $\Lambda(v)$ is a continuous and convex function, Equations (3.12)-(3.18) could be expressed as:

$$\tilde{V}_t(K, y) = \tilde{L}(K, y) + \min_{v \geq y} \left\{ c^r(v - y) + \Lambda(v) + \gamma\, \mathbb{E}[\tilde{V}_{t+1}(K_+, v - D)] \right\}. \tag{3.20}$$

If the optimum value of the minimization model given in Equation (3.20) is defined as v^*, optimum policy is derived as: if $v^* > z$, then $q_t^r = (v^* - z)$ and $q_t^s = (z - y)$; if $v^* \leq z$ and $w^* > y$, then $q_t^r = 0$ and $q_t^s = (z - y)$; if $v^* \leq z$ and $w^* \leq y$, then $q_t^r = 0$ and $q_t^s = 0$. □

The optimum control policy, whose analytical characterization is given in Proposition 3.1.2, consists of order-up-to levels depending on the number of parts observed in the secondary market in any period.

3.1.2 Inventory control model with a general lead time

By relaxing the constraint $l^r = 1$ of Section 3.1.1 through setting it to $l^r > 0$, the inventory control model is extended for a general lead time. Therefore, while making the ordering decision, the orders placed on the regular supplier and still on the way must also be considered. These incoming orders should be added to the state variables. To this end, the outstanding orders vector, $\overline{q}_t^r$, with a dimension of $l^r - 1$ where $\overline{q}_t^r = (q_{t-l^r+1}^r, q_{t-l^r+2}^r, \ldots, q_{t-2}^r, q_{t-1}^r)$ is defined. After placing an order to the regular supplier, the outstanding orders vector is updated by adding the order quantity, q_t^r. Thus, the pipeline orders vector becomes $\overline{q}_{t+1}^r = (q_{t-l^r+2}^r, q_{t-l^r+3}^r, \ldots, q_{t-1}^r, q_t^r)$ for period $t+1$.

Following the dynamics given in Section 3.1, the finite-horizon, multi-period and discounted total cost function is derived as below:

$$V_t(K, \overline{q}_t^r, y_t) = \min_{\substack{q_t^r \geq 0, \\ K \geq q_t^s \geq 0,}} \{J_t(K, \overline{q}_t^r, y_t, q_t^r, q_t^s)\}, \quad t = 1, 2, 3, \ldots, T, \tag{3.21}$$

$$\begin{aligned} J_t(K, \overline{q}_t^r, y_t, q_t^r, q_t^s) = c^r q_t^r + c(q_t^s, K) + L(y_t + q_t^s) \\ + \gamma\, \mathbb{E}[V_{t+1}(K_+, \overline{q}_{t+1}^r, y_t + q_{t-l^r+1}^r + q_t^s - D)]. \end{aligned} \tag{3.22}$$

Also, note that the last term of the V_{t+1} is the inventory transferred to the next period, which corresponds to:

$$y_{t+1} = y_t + q_{t-l^r+1}^r + q_t^s - \mathbb{E}[D]. \tag{3.23}$$

The finite-horizon dynamic minimization problem is defined on the state space $(K, \overline{q}^r, y) \in (\Omega \times \mathbb{R}^{l^r-1} \times \mathbb{R})$ with a dimension of $(l^r + 1)$. The first variable of the state space is the number of parts available at the secondary market in a period, the second one is the outstanding orders vector, and the last one is the inventory on hand. The minimization problem has a two-dimensional action space such that $(q_t^r, q_t^s) \in (\mathbb{R} \times \mathbb{R})$.

The convexity of the inventory control model given in Equations (3.21) and (3.22) is investigated; however, it is concluded that the model is not convex in the decision variables q_t^r and q_t^s. This result is in parallel with the result obtained for simpler models in the literature (Whittemore and Saunders, 1977). Consequently, it is determined that the analytical characterization of the multi-dimensional mathematical model depends on the state space, which is quite large. Therefore, heuristic policies are tested for control of the inventory model. Numerical experiments performed for this purpose and considered heuristic methods are explained in subsequent sections.

3.2 Heuristic Methods

The following heuristics in the literature are used in numerical experiments:

1. Dual index policy: This heuristic policy employs two order-up-to levels for slow supplier and expedited supplier. In the problem setup of this thesis, OEM is the slow supplier, and secondary market is the expedited supplier. Fukuda (1964) is proved that the dual index policy is optimum when the lead time difference of suppliers is one period under the assumption of a stationary demand. The method proposed by Veeraraghavan and Scheller-Wolf (2008) is utilized to optimize the policy parameters.
2. Tailored base-surge policy: This policy consists of two policy parameters for regular supplier and expedited supplier. The first parameter represents the fixed order amount placed on the regular supplier, while the second parameter is the order up to the level of the expedited supplier (Allon and Van Mieghem, 2010). In the problem setup of this thesis, OEM is the slow supplier, and secondary market is the expedited supplier.
3. Capped dual index policy: This policy is developed by Sun and Van Mieghem (2019), and it is an extended version of the dual index policy by presenting a new policy parameter. This new policy parameter is used to limit the maximum number of orders placed on the slow supplier.

3.3 Numerical Experiments

In Proposition 3.1.2, the optimum inventory control policy for $l^r = 1$ is derived, which is similar to the dual index policy. However, since the mathematical model for $l^r > 1$ cannot be characterized analytically and the optimum policy is state-dependent, heuristic policies are needed to control the system and obtain a practical solution. Then, an optimization model is developed and solved using the value iteration algorithm. The average deviation of the heuristics' cost from the optimum cost is calculated as a performance measure. In the numeric experiments, the parameters given in Table 3.1 are considered.

Table 3.1 Problem parameters used in numerical experiments for the stationary demand distribution.

Λ	h	b	c^r	η	ξ	l^r	T	Market Scenario
0.05	1	15	5	0	1	1	120	1
0.25	3	55		-0.1	1.1	2	360	
	5	95		-0.5	2	3		
				-0.9	5			

In Table 3.1, h is the holding cost per unit, and b represents the cost to be applied per unit for unsatisfied demand. In all numerical experiments carried out within the scope of this chapter, it is assumed that the demand distribution is stationary and follows a Poisson distribution. The parameter of the Poisson distribution (demand rate) is expressed with Λ. l^r represents the lead time of the OEM, and c^r is its ordering cost per unit. T is length of planning horizon. η and ξ values are the parameters of the secondary market's purchasing cost function. The value of c_0 in this function is assumed to be equal to c^r.

In numerical experiments, a subspace of the parameter space consisting of the values given in Table 3.1 is used. Most of the studies in the literature assume that the cost of purchasing from secondary markets is linear. The values $\eta = 0$ and $xi = 1$ are used to represent this assumption. In this way, it is aimed to compare the performance of heuristics in cases where the cost of purchasing from the secondary market is linear and non-linear.

When the lead time of the regular supplier is 3, and the planning horizon is 360 periods, the median values $h = 3$, $b = 55$, $\eta = -0.5$ and $\xi = 2$ are not taken into consideration; only the remaining extreme values are considered. This is because when the lead time of the OEM is 3 and the planning horizon is 360 periods, the size of the state space becomes too large. As a result, the numerical solution of the optimization models takes a very long time. It is aimed to observe the behavior of the system in extreme situations by considering only the extreme values. Similarly, when the size of the planning horizon is greater than 360, it is technologically infeasible to solve the optimization model numerically as the size of the state space grows too much. For a given Λ value, the total dimensions of the state spaces according to planning horizon and lead time variables are shown in Tables 3.2 and 3.3.

Table 3.2 Total dimensions of the state space for $\Lambda = 0.05$.

	Planning Horizon (T)			
	120	360	600	1800
l^r=1	496,840	4,514,440	12,564,040	113,292,040
l^r=2	4,450,030	40,565,230	112,968,430	1,019,304,430
l^r=3	39,761,600	364,215,200	1,015,260,800	9,169,368,800

Table 3.3 Total dimensions of the state space for $\Lambda = 0.25$.

	Planning Horizon (T)			
	120	360	600	1800
l^r=1	571,800	5,171,400	14,379,000	129,537,000
l^r=2	5,103,320	46,413,320	129,195,320	1,165,185,320
l^r=3	45,544,280	416,556,680	1,160,817,080	10,480,839,080

While solving the optimization models, an artificial upper limit is set on the capacity of the OEM in order to truncate the state space. This upper limit is chosen as 8 because the maximum demand that can be observed in any period is 4. Maximum demand is obtained by calculating the 0.9999th quantile of the Poisson distribution with a rate of 0.25. Note that the state space sizes given in Tables 3.2 and 3.3 are calculated using this upper bound. The maximum capacity of secondary markets is determined as 4. As mentioned earlier, it is assumed that the number of available spare parts in the secondary markets is random in the following period and follows a discrete-time Markov chain. The finite and countable space on which this Markov chain is defined is expressed as Ω. In this case, it is defined as $\Omega = \{0, 1, 2, 3, 4\}$. In all numerical experiments carried out within the scope of this chapter, market scenario 1 (symmetrical) is used, as indicated in Table 3.1. The Markov transition probability matrix for this scenario is as follows:

$$P = \begin{bmatrix} 0.5 & 0.5 & 0 & 0 & 0 \\ 0.25 & 0.5 & 0.25 & 0 & 0 \\ 0 & 0.25 & 0.5 & 0.25 & 0 \\ 0 & 0 & 0.25 & 0.5 & 0.25 \\ 0 & 0 & 0 & 0.5 & 0.5 \end{bmatrix}. \tag{3.24}$$

In numerical experiments performed to calculate the deviations of heuristics from the optimum policy, parameter optimization of heuristics is carried out through simulations with 1000 replications. The average percentage deviations of heuristic policies from the optimum policy for all problem parameters and when the acquisition cost of the secondary supplier is linear or non-linear are given in Table 3.4.

Table 3.4 Deviations of the heuristic policies from the optimum policy for stationary demand distribution.

Acquisition Cost	Dual Index	Tailored Base-Surge	Capped Dual Index
Linear	1.5%	32%	3.6%
Non-Linear	4.5%	46.4%	6.9%
Overall	**4.2%**	**44.9%**	**6.5%**

As seen in Table 3.4, deviations of heuristic policies from the optimal policy are lower when the cost of purchasing from secondary markets is linear. This is an anticipated result since these heuristic policies assume that the fast supplier is more expensive than the slow supplier. The dynamic procurement cost function contradicts this assumption as the secondary supplier may be cheaper or more expensive depending on the problem parameters. Furthermore, in all cases, the dual index policy outperforms tailored base-surge and capped dual index policies.

4. AN INVENTORY CONTROL MODEL WITH A NON-STATIONARY DEMAND DISTRIBUTION

In this chapter, a mathematical model that minimizes the total inventory cost-the sum of purchasing, holding, and backlog costs- is developed for a periodic review inventory system. An optimum inventory control policy is determined in the existence of non-stationary spare parts demand and Markovian secondary market capacity. The non-stationarity of the demand distribution means that the demand distribution follows different probability distributions or it follows the same probability distribution with different parameters over time.

The model developed by (Hekimoğlu and Karlı, 2021) is used for non-stationary spare parts demand. They employ two separate models: one for the growing installed base and one for the declining installed base. They use two independent and homogeneous Poisson process to model capital product sales/retirements and spare part breakdowns of each capital product. Product sales are used in the growth phase to represent the newly introduced capital products, while product retirements are utilized in the decline phase to represent capital product withdraws.

4.1 A Mathematical Model

The inventory control model is first developed, assuming a general lead time for the regular supplier. Then a special case of this model where the lead time is one period is analyzed. The inventory control model is derived using stochastic dynamic programming.

A T-period, periodic review inventory control problem is considered. As in Chapter 3, it is assumed that the OEM, the regular supplier, delivers after a positive lead time, $l^r > 0$, and orders placed in the secondary markets are delivered within the same period. Deliveries from both suppliers are included in the same inventory, as there is no difference in quality between the parts from the secondary market and the regular supplier. Spare parts demand, if sufficient, is met from the existing stock. In case of lacking stock, the demand is backlogged to be met at the next delivery. The dynamics of the system are as follows:

1. At the beginning of period t, the inventory on hand, y_t, is observed after receiving incoming orders placed l^r periods before on regular supplier.
2. The capacity of the secondary market is observed.
3. Orders on regular and secondary suppliers are placed.
4. Orders placed on the secondary market are delivered.
5. Non-stationary demand, D_t, occurs (demand realization).
6. Holding, backlog and ordering costs are calculated and added to the total cost with a certain discount rate.

Following these dynamics the single period cost minimization function is:

$$V(K, y) = \min_{\substack{q^r \geq 0, \\ K \geq q^s \geq 0,}} \{c^r q^r + c(q^s, K) + L(y + q^s)\}. \tag{4.1}$$

In Equation (3.1), the first term in the minimization is the purchasing cost of the regular supplier, where c^r and q^r are the unit ordering cost of the regular supplier and quantity of order placed on the regular supplier, respectively. The second term is the dynamic cost function representing the acquisition cost of the secondary supplier, where K is the capacity of the secondary market in the decision period, and q^s is order quantity. Total purchasing cost from secondary markets is calculated in the same way as in the previous Chapter 3, such that $c(q, K) = c_0 K^\eta q^\xi,\ \eta \leq 0,\ \xi \geq 1$. Thanks to this cost function, the dependency between the procurement cost of the secondary markets and the secondary market's capacity and order quantity is reflected in the model.

Similar to the previous chapter, the number of spare parts in the secondary markets is assumed to emerge following a discrete-time Markov chain. Ω denotes the finite and countable space on which this Markov chain is defined. It is assumed that the states of the Markov chain evolve between different points of this space with the transition probability matrix P. Unlike the model with the stationary demand distribution, not only the symmetrical but also stochastically increasing and stochastically decreasing versions of the P matrix are taken into account in numerical experiments.

In Equation (4.1), $L(y) = \mathbb{E}\left[h(y-D_t)^+ + b(D_t-y)^+\right]$ gives the expected inventory-related cost where $x^+ = \max(0,x)$, h is holding cost per unit for excess inventory, and b is penalty cost per unit for backlogged orders. It is known in the inventory control literature that $L(y)$ is convex. The finite-horizon, multi-period and discounted total cost function becomes:

$$V_t(K, \overline{q}_t^r, y_t) = \min_{\substack{q_t^r \geq 0, \\ K \geq q_t^s \geq 0,}} \{J_t(K, \overline{q}_t^r, y_t, q_t^r, q_t^s)\}, \quad t = 1, 2, 3, \ldots, T, \tag{4.2}$$

$$\begin{aligned} J_t(K, \overline{q}_t^r, y_t, q_t^r, q_t^s) = c^r q_t^r + c(q_t^s, K) + L(y_t + q_t^s) \\ + \gamma\, \mathbb{E}[V_{t+1}(K_+, \overline{q}_{t+1}^r, y_t + q_{t-l^r+1}^r + q_t^s - D_t)]. \end{aligned} \tag{4.3}$$

$\overline{q}_t^r$ is the outstanding orders vector with a dimension of $l^r - 1$ and defined as $\overline{q}_t^r = (q_{t-l^r+1}^r, q_{t-l^r+2}^r, \ldots, q_{t-2}^r, q_{t-1}^r)$. In Equation (4.3), the last term is the recursive term that represents the cost of the next period. K_+ is the random variable representing the number of items that will be available in the secondary market in the following period. D_t is the non-stationary demand in period t, and γ is the discount rate in each period. Also, note that the second term of the V_{t+1} is the inventory transferred to the next period, which corresponds to:

$$y_{t+1} = y_t + q_{t-l^r+1}^r + q_t^s - \mathbb{E}[D_t]. \tag{4.4}$$

The finite-horizon dynamic minimization problem is defined on the state space $(K, \overline{q}^r, y) \in (\Omega \times \mathbb{R}^{l^r-1} \times \mathbb{R})$ with a dimension of $(l^r + 1)$. The first variable of the state space is the number of parts available at the secondary market in a period, the second one is the outstanding orders vector, and the last one is the inventory on hand. The minimization problem has a two-dimensional action space such that $(q_t^r, q_t^s) \in (\mathbb{R} \times \mathbb{R})$.

When the lead time of the regular supplier is equal to 1, the cost function for the multi-period planning horizon is as follows:

$$V_t(K, y_t) = \min_{\substack{q_t^r \geq 0, \\ K \geq q_t^s \geq 0,}} \{J_t(K, y_t, q_t^r, q_t^s)\}, \quad t = 1, 2, 3, \ldots, T, \tag{4.5}$$

$$\begin{aligned} J_t(K, y_t, q_t^r, q_t^s) = c^r q_t^r + c(q_t^s, K) + L(y_t + q_t^s) \\ + \gamma\, \mathbb{E}[V_{t+1}(K_+, y_t + q_t^r + q_t^s - D_t)]. \end{aligned} \tag{4.6}$$

When $l^r = 1$, the finite-horizon dynamic minimization problem is defined on the two-dimensional state space $(K, y) \in (\Omega \times \mathbb{R})$. The first variable of the state space is the number of parts available at the secondary market in a period, and the second one is the inventory on hand. The minimization problem has a two-dimensional action space such that $(q_t^r, q_t^s) \in (\mathbb{R} \times \mathbb{R})$.

Proposition 4.1.1. *The following statements are true for the multi-period, finite-horizon and discounted total cost function given in Equations* (4.5) *and* (4.6) *under the assumptions of non-stationary demand and* $l^r = 1$*:*

a. $J_t(K, y_t, q_t^r, q_t^s,)$ *is convex in* (q_t^r, q_t^s) *for given values of* K *and* y_t*.*

b. $V_t(K, y_t)$ *is convex in* y_t *for a given value of* K*.*

c. Optimum control policy is given by a base stock policy.

Proof. The proof of this proposition will be carried out by the method of proof by induction. Firstly, for $t = T$, $J_T(K, y_T, q_T^r, q_T^s)$ is a linearly increasing function in q_T^r. $J_T(K, y_T, q_T^r, q_T^s)$ is a convex function in (y_T, q_T^r, q_T^s) since $c(q_T^s, K)$ and $L(y_T + q_T^s)$ are convex functions in their parameters. Due to the preservation of convexity under minimization, $V_T(K, y_T)$ is a convex function in y_T. Thus, for period T, there is an optimum value of (q_T^r, q_T^s). Hence, induction is completed for $t = T$. Suppose that the expressions a, b, and c in the proposition are hold for $t + 1$. For period t, since

$c(q_t^s, K)$ and $L(y_t + q_t^s)$ are convex functions, and $\mathbb{E}[V_{t+1}(K, y_t + q_t^s - D_t)]$ is convex, $J_t(K, y_t, q_t^r, q_t^s)$ is a convex function in (y_t, q_t^r, q_t^s). As a result of minimization of $J_t(K, y_t, q_t^r, q_t^s)$ in (q_t^r, q_t^s), $V_t(K, y_t)$ is convex in y_t for a given K. Thus, proofs of statements a and b are completed. Statement b implies statement c. Hence, the proof is concluded. □

To prove the structure of optimum policy for $l^r = 1$ by using the mathematical model given in Equations (4.5) and (4.6), firstly, the following variable transformation is needed to be done:

$$w = y + q^s,$$
$$v = w + q^r.$$

Using this transformation, mathematical model given in Equations (4.5) and (4.6) could be expressed as below:

$$\tilde{V}_t(K, y_t) = \min_{\substack{v \geq w, \\ K + y_t \geq w \geq y_t}} \left\{ \tilde{J}_t(K, y_t, v, w) \right\}, \qquad t = 1, 2, 3, \ldots, T, \tag{4.7}$$

$$\tilde{J}_t(K, y_t, v, w) = c^r(v - w) + c_0(w - y_t)^{\xi} K^{\eta} + L(w) + \gamma\, \mathbb{E}[\tilde{V}_{t+1}(K_+, v - D_t)]. \tag{4.8}$$

Equation (4.8) could be expressed for the last period as below:

$$\tilde{J}_T(K, y_T, v, w) = c^r(v - w) + c_0(w - y_T)^{\xi} K^{\eta} + L(w). \tag{4.9}$$

Similar to the model with stationary demand distribution, it is assumed that $\eta = 0$, $\xi = 1$. The structure of the optimum policy and the value of the cost function are analyzed numerically for the general values of these parameters. For $\eta = 0$, $\xi = 1$, Equation (4.7) becomes:

$$\tilde{V}_t(K, y_t) = \min_{\substack{v \geq w, \\ K + y_t \geq w \geq y_t}} \left\{ c^r(v - y_t) + c_0(v - w) + L(w) + \gamma\, \mathbb{E}[\tilde{V}_{t+1}(K_+, v - D_t)] \right\}. \tag{4.10}$$

Let us define $c = -c^r + c_0$, then Equation (4.10) becomes:

$$\tilde{V}_t(K, y_t) = \min_{\substack{v \geq w, \\ K + y_t \geq w \geq y_t}} \left\{ c^r(v - y_t) + c(w - y_t) + L(w) + \gamma\, \mathbb{E}[\tilde{V}_{t+1}(K_+, v - D_t)] \right\}. \tag{4.11}$$

By using the second and the third terms in minimization, we could define:

$$\tilde{L}(K, y_t) = \min_{K+y_t \geq w \geq y_t} \{c(w - y_t) + L(w)\}. \tag{4.12}$$

The value that minimizes the unconstrained version of the mathematical model in Equation (4.12) is denoted as w^*. From this point of view, the optimum function can be characterized by the following proposition. To prove this proposition, the method developed by Fukuda (1964) is adapted to the case where the expedited supplier has a limited capacity.

Proposition 4.1.2. *The optimum policy that minimizes the mathematical model given in Equations* (4.5) *and* (4.6) *could be expressed using two parameters,* $v^*(K)$ *and* $w^*(K) = (w^* \vee (y+K)) \wedge y$*, that are functions of the market capacity in a given period:* $q_t^s = (w_t^*(K) - y_t)$ *and* $q_t^r = (v_t^*(K) - w_t^*(K))$*.*

Proof. For the optimization of the function given in Equation (4.8), it will be analyzed where w^*, which is the point that minimizes $\tilde{L}(K, y)$, falls within the $[y, y+K]$ interval. Such that, if $w^* < y < y+K$, then $\tilde{L}(K, y) = L(y)$. Thus, the minimization in Equation (4.11) becomes:

$$\tilde{V}_t(K, y) = \tilde{L}(K, y) + \min_{v \geq y} \left\{ c^r(v - y) + \gamma\, \mathbb{E}[\tilde{V}_{t+1}(K_+, v - D_t)] \right\}. \tag{4.13}$$

If $y < w^* < y+K$, then $\tilde{L}(K, y) = c(w^* - y) + L(w^*)$. For $v > w^*$, the minimization in Equation (4.11) becomes:

$$\tilde{V}_t(K, y) = \tilde{L}(K, y) + \min_{v \geq w^*} \left\{ c^r(v - y) + \gamma\, \mathbb{E}[\tilde{V}_{t+1}(K_+, v - D_t)] \right\}. \tag{4.14}$$

For $v^* \leq w^*$,

$$\begin{aligned} \tilde{V}_t(K, y) = \tilde{L}(K, y) + \min_{w^* \geq v \geq y} \Big\{ & c^r(v - y) + c(v - y) \\ & + L(v) - \tilde{L}(K, y) + \gamma\, \mathbb{E}[\tilde{V}_{t+1}(K_+, v - D_t)] \Big\}, \end{aligned} \tag{4.15}$$

$$\begin{aligned} \tilde{V}_t(K, y) = \tilde{L}(K, y) + \min_{w^* \geq v \geq y} \Big\{ & c^r(v - y) + c(v - w^*) \\ & + L(v) - L(w^*) + \gamma\, \mathbb{E}[\tilde{V}_{t+1}(K_+, v - D_t)] \Big\}. \end{aligned} \tag{4.16}$$

If $y + K < w^*$, then $\tilde{L}(K, y) = c(K) + L(y + K)$. Thus, if $v > w^* > y + K$ or $w^* > v > y + K$, then the minimization in Equation (4.11) becomes:

$$\tilde{V}_t(K, y) = \tilde{L}(K, y) + \min_{v \geq y+K} \left\{ c^r(v - y) + \gamma\, \mathbb{E}[\tilde{V}_{t+1}(K_+, v - D_t)] \right\}, \tag{4.17}$$

$$\tilde{V}_t(K, y) = cK + L(y + K) + \min_{v \geq y+K} \left\{ c^r(v - y) + \gamma\, \mathbb{E}[\tilde{V}_{t+1}(K_+, v - D_t)] \right\}. \tag{4.18}$$

If $w^* > y + K > v \geq y$, then

$$\begin{aligned} \tilde{V}_t(K, y) = cK + L(y + K) + \min_{v \geq y+K} \Big\{ & c^r(v - y) + c(v - y - K) \\ & + L(v) - L(y - K) + \gamma\, \mathbb{E}[\tilde{V}_{t+1}(K_+, v - D_t)] \Big\}, \end{aligned} \tag{4.19}$$

To organize Equations (4.13)-(4.19) in a single model, let us define $z = \min(y + K, w^*)$ and

$$\Lambda(v) = \begin{cases} 0, & \text{if } v \geq z, \\ L(v) + L(z) + c(v - z), & \text{if } v < z. \end{cases} \tag{4.20}$$

Since $\Lambda(v)$ is a continuous and convex function, Equations (4.13)-(4.19) could be expressed as:

$$\tilde{V}_t(K, y) = \tilde{L}(K, y) + \min_{v \geq y} \left\{ c^r(v - y) + \Lambda(v) + \gamma\, \mathbb{E}[\tilde{V}_{t+1}(K_+, v - D_t)] \right\}. \tag{4.21}$$

If the optimum value of the minimization model given in Equation (4.21) is defined as v^*, optimum policy is derived as: if $v^* > z$, then $q_t^r = (v^* - z)$ and $q_t^s = (z - y)$; if $v^* \leq z$ and $w^* > y$, then $q_t^r = 0$ and $q_t^s = (z - y)$; if $v^* \leq z$ and $w^* \leq y$, then $q_t^r = 0$ and $q_t^s = 0$. □

The optimum control policy, whose analytical characterization is given in Proposition 4.1.2, consists of order-up-to levels depending on the number of parts observed in the secondary market in any period.

4.2 The Non-Stationary Demand Model

Throughout this thesis, the model developed by Hekimoğlu and Karlı (2021) is utilized to reflect non-stationary demand that stems from the dynamics of the growing and declining installed bases.

The model for growing installed bases employs two homogeneous Poisson processes representing capital product sales and spare parts failures of each product. The rates of these processes are λ and α, respectively. Hence, the interarrival times of capital product sales and individual spare part breakdowns of these capital products follow an exponential distribution with a rate of λ and α, respectively. The general structure of the model is given in Figure 4.1.

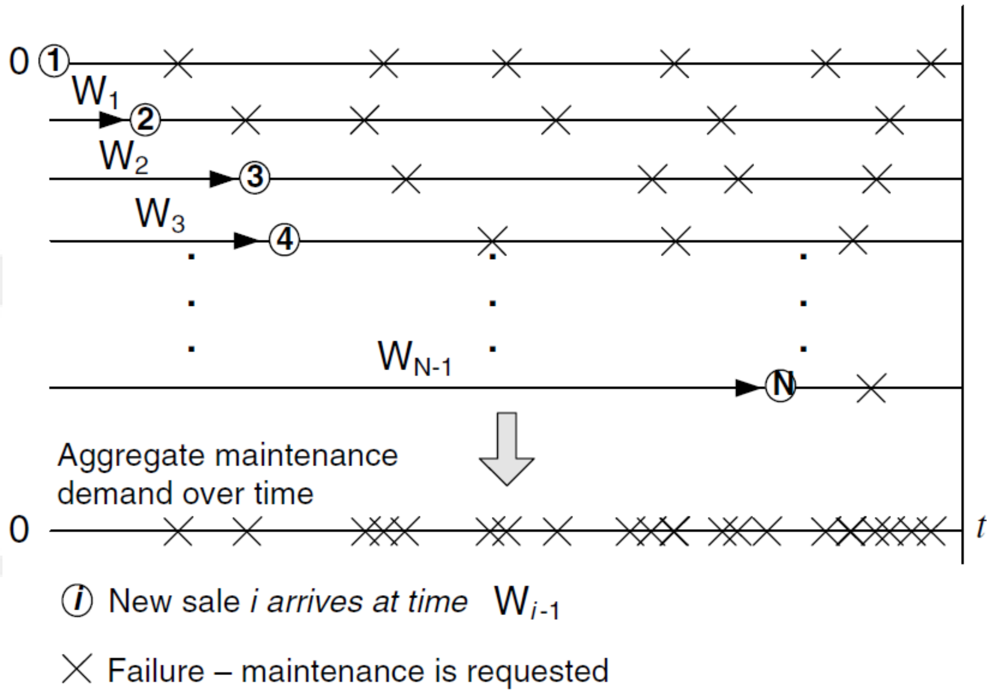


Figure 4.1 Dynamics of growing installed base (Jin and Liao, 2009).

In Figure 4.1, each horizontal line represents a capital product installation. $\times$ symbols on horizontal lines represent the random spare part failure of the respective capital product. Thus, total maintenance demand is obtained by taking the superposition of each spare part breakdown. As mentioned before, Hekimoğlu and Karlı (2021) uses this model structure and extend it by calculating the third central moment along with the first two moments. They utilize these moments to choose probability distributions for each period. To do so, they use Ord (1967)'s distribution selection algorithm for the hypergeometric distribution family. This algorithm includes Poisson, Negative Binomial, Beta-Binomial and Beta-Pascal distributions. After determining a probability distribution for each period, parameters of respective distributions are estimated using the moments of corresponding periods.

Like the growing installed base, the model for declining installed bases employs two homogeneous Poisson processes representing capital product retirements and spare parts failures of each product. The rates of these processes are represented by λ and α, respectively. Hence, the interarrival times of capital product retirements and individual spare part breakdowns of these capital products follow an exponential distribution with a rate of λ and α, respectively. The general structure of the model is given in Figure 4.2.

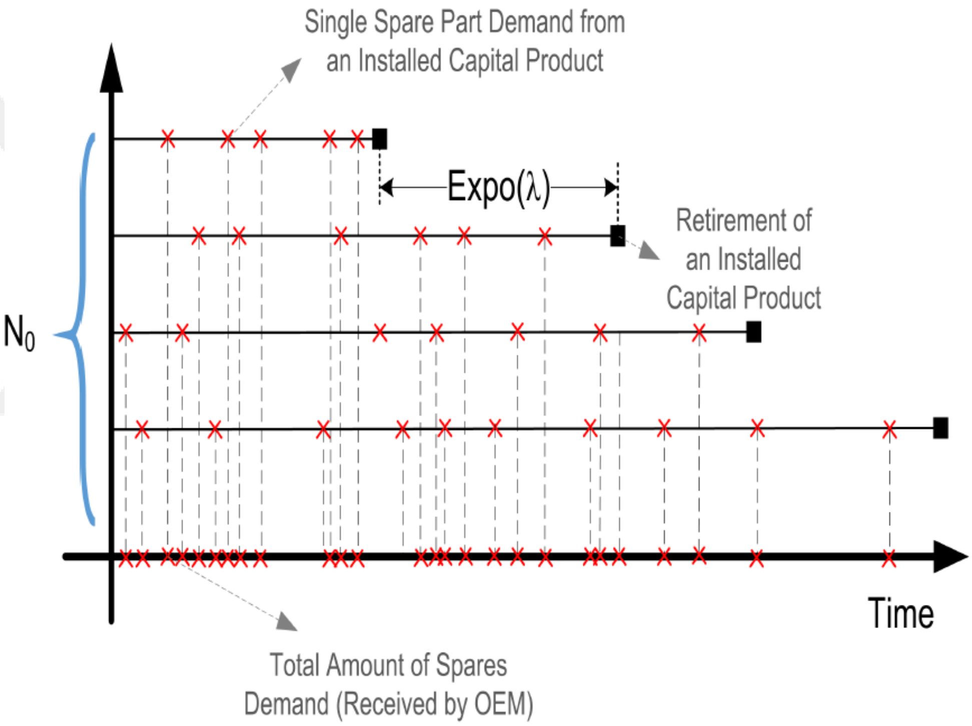


Figure 4.2 Dynamics of declining installed base (Hekimoğlu and Karlı, 2021).

In Figure 4.2, N_0 is the active installed base size at the beginning, and each horizontal line represents a capital product. $\times$ symbols on horizontal lines represent the random spare part failure of the respective capital product. Thus, total repair demand is obtained by taking the superposition of each spare part breakdown. Retirement of a capital product is represented by $\blacksquare$. Similar to growing installed base, the first three moments central moments are calculated for each period. Then, these moments are utilized to select a probability distribution for each period. To do so, Ord (1967)'s distribution selection algorithm for the hypergeometric distribution family is used. After determining a probability distribution for each period, parameters of respective distributions are estimated using the moments of corresponding periods.

4.3 Numerical Experiments

Since the dynamics of the growing and declining installed bases differ entirely from each other, numerical experiments are conducted separately for each phase of the installed base.

In Proposition 4.1.2, the optimum inventory control policy for $l^r = 1$ is derived. However, since the mathematical model for $l^r > 1$ cannot be characterized analytically and the optimum policy is state-dependent, heuristic policies are required to control the system and obtain an applicable solution. In this chapter, the same heuristic policies used for the stationary demand given in Section 3.3 are considered. However, these heuristic policies assume a stationary demand. Only capped dual index policy provides an implementation for the non-stationary demand under certain conditions. Nevertheless, it is not applicable for the problem setup of this thesis. Therefore, in this section, an optimization model is developed and solved using the value iteration algorithm, and heuristic policies are evaluated for the non-stationary demand in Section 4.3.3.

4.3.1 Growing installed base

The same structure with the optimization model presented in Section 3.3 is employed. The main difference of the optimization model used in this section is the demand model. While a stationary Poisson distribution is employed in Section 3.3, the non-stationary demand model developed for growing installed bases by (Hekimoğlu and Karlı, 2021) is utilized in this section.

To this end, the moment functions of growing installed bases derived by (Hekimoğlu and Karlı, 2021) are coded. Then, the distribution selection algorithm of (Ord, 1967) is coded to choose a proper distribution for each period depending on the first three moments as suggested by (Hekimoğlu and Karlı, 2021). Subsequently, the parameters of the selected demand distributions are estimated using the calculated moments. Codes developed for the optimization models within the scope of this section are given in Section A.1. The parameters used in the optimization models are given in Table 4.1.

Table 4.1 Problem parameters used in numerical experiments for the growing installed bases.

λ	α	h	b	c^r	η	ξ	l^r	γ	T	Market Scenario
0.05	0.05	1	15	5	0	1	1	0.99	60	1
0.25	0.25	5	95		-0.1	1.1	2			2
					-0.9	2	3			3

In Table 4.1, h is the holding cost per unit, and b is the backlog cost per unit. λ is the rate of the Poisson process that represents capital product sales, and α is the rate of the Poisson process representing spare parts breakdowns of each capital product. l^r represents the lead time of the OEM, and c^r is its ordering cost per unit. T is the length of the planning horizon, and γ is the discount factor per period. η and ξ values are the parameters of the secondary market's purchasing cost function. The value of c_0 in this function is assumed to be equal to c^r.

In numerical experiments performed within the scope of this section, a subspace of space consisting of parameters given in Tables 4.1 is used. The maximum value of ξ is taken as 2 instead of 5, unlike Section 3.3. The reason for this is to prevent the overgrowth of the cost function and make it more realistic as the demand and, therefore, the number of orders placed in secondary markets increases. Also, the case where $\lambda = 0.25$, $\alpha = 0.25$ is not taken into consideration, and the planning horizon is shortened to 60 due to the overgrowth of the state space as in Section 3.3. When $\lambda = 0.25$, $\alpha = 0.25$ and $T > 60$, it is technologically infeasible to solve the optimization model numerically as the size of the state space grows too much. The underlying reason for this is the growth of the maximum observable demand in a period. Since capital products are in the growth phase, the size of the installed base hence, spare parts demand increases over time. Total dimensions of the state space according to T and l^r parameters for given λ and α values are given in Tables 4.2 - 4.5.

Table 4.2 Total dimensions of the state space for $\lambda = 0.05, \alpha = 0.05$ - Growing Installed Base.

	Planning Horizon (T)			
	60	120	360	600
l^r=1	425,919	1,824,459	18,756,023	56,778,800 3
l^r=2	2,269,813	12,026,833	171,752,763	632,363,706
l^r=3	12,189,762	80,259,582	1,603,027,062	7,216,196,351

Table 4.3 Total dimensions of the state space for $\lambda = 0.25, \alpha = 0.05$ - Growing Installed Base.

	Planning Horizon (T)			
	60	120	360	600
l^r=1	503,776	2,265,406	26,546,689	87,172,501
l^r=2	4,314,063	25,101,843	493,580,334	2,159,372,059
l^r=3	37,465,792	283,966,527	9,477,465,816	55,537,489,761

Table 4.4 Total dimensions of the state space for $\lambda = 0.05, \alpha = 0.25$ - Growing Installed Base.

	Planning Horizon (T)			
	60	120	360	600
l^r=1	586,820	2,617,706	30,079,023	97,579,820
l^r=2	6,289,296	35,420,931	642,280,054	2,707,524,547
l^r=3	67,737,540	486,689,099	14,083,844,398	77,612,117,037

Table 4.5 Total dimensions of the state space for $\lambda = 0.25, \alpha = 0.25$ - Growing Installed Base.

	Planning Horizon (T)			
	60	120	360	600
l^r=1	793,754	3,905,265	55,142,997	200,395,400
l^r=2	15,384,278	107,749,837	2,991,335,867	15,609,830,000
l^r=3	303,264,936	3,057,637,245	169,294,407,308	1,275,114,000,000

While solving optimization models, an artificial upper limit is set on the capacity of the OEM in order to reduce the size of the state space. This limit is determined as 1.5 times the maximum demand that may be observed in the relevant period. Maximum demand is obtained by calculating the 0.9999th quantile of the chosen distribution using its estimated parameters. Note that the state space sizes given in Tables 4.2 - 4.5 are calculated using this limit.

The maximum capacity of secondary markets is determined as 12. As mentioned earlier, it is assumed that the capacity of the secondary markets is random in the following period and follows a discrete Markov chain. The finite and countable space on which this Markov chain is defined is expressed as Ω. In this case, it is defined as $\Omega = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$. As seen in Table 4.1, market scenarios 1, 2 and 3 are considered in the numerical experiments of growing installed systems. These scenarios represent symmetric, stochastically increasing and stochastically decreasing transition probability matrices, respectively. The Markov transition probability matrices for these scenarios are given below as P_1, P_2 and P_3, respectively:

$$P_1 = \begin{bmatrix} 0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.25 & 0.5 & 0.25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.25 & 0.5 & 0.25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.25 & 0.5 & 0.25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.25 & 0.5 & 0.25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.25 & 0.5 & 0.25 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.25 & 0.5 & 0.25 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0.5 & 0.25 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0.5 & 0.25 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0.5 & 0.25 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0.5 & 0.25 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0.5 & 0.25 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 \end{bmatrix},$$

$$P_2 = \begin{bmatrix} 0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.05 & 0.5 & 0.45 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.05 & 0.5 & 0.45 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.05 & 0.5 & 0.45 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.05 & 0.5 & 0.45 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.05 & 0.5 & 0.45 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.05 & 0.5 & 0.45 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.05 & 0.5 & 0.45 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.05 & 0.5 & 0.45 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.05 & 0.5 & 0.45 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.05 & 0.5 & 0.45 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.05 & 0.5 & 0.45 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 \end{bmatrix},$$

$$P_3 = \begin{bmatrix} 0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.45 & 0.5 & 0.05 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.45 & 0.5 & 0.05 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.45 & 0.5 & 0.05 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.45 & 0.5 & 0.05 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.45 & 0.5 & 0.05 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.45 & 0.5 & 0.05 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.45 & 0.5 & 0.05 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.45 & 0.5 & 0.05 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.45 & 0.5 & 0.05 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.45 & 0.5 & 0.05 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.45 & 0.5 & 0.05 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 \end{bmatrix}.$$

4.3.2 Declining installed base

As in Section 4.3.1, the same structure with the optimization model developed for stationary demand is employed. The main difference of the optimization model used in this section is the demand model. While a stationary Poisson distribution is employed in Section 3.3, the non-stationary demand model developed for declining installed bases by (Hekimoğlu and Karlı, 2021) is utilized in this section.

To this end, the moment functions of declining installed bases derived by (Hekimoğlu and Karlı, 2021) are coded. Then, the distribution selection algorithm of (Ord, 1967) is coded to choose a proper distribution for each period depending on the first three moments as suggested by (Hekimoğlu and Karlı, 2021). Subsequently, the parameters of the selected demand distributions are estimated using the calculated moments. Codes developed for the optimization models within the scope of this section are given in Section A.1. The parameters used in the optimization models are shown in Table 4.6.

Table 4.6 Problem parameters used in numerical experiments for the declining installed bases.

λ	α	h	b	c^r	η	ξ	l^r	γ	T	Market Scenario	Strategy
0.05	0.05	1	15	5	0	1	1	0.99	60	1	1
0.25	0.25	5	95		-0.1	1.1	2			2	2
					-0.9	2	3				

In Table 4.6, h is the holding cost per unit, and b is the backlog cost per unit. λ is the rate of the Poisson process that represents capital product retirements, and α is the rate of the Poisson process representing spare parts malfunction of each capital product. l^r represents the lead time of the regular supplier, and c^r is its purchasing cost per unit. T is the length of the planning horizon, and γ is the discount factor per period. η and ξ values are the parameters of the secondary market's acquisition cost function. The value of c_0 in this function is assumed to be equal to c^r.

In numerical experiments carried out within the scope of this section, a subspace of space consisting of parameters given in Tables 4.6 is used. Similar to Section 4.3.1, the maximum value of ξ is taken as 2 instead of 5. The reason for this is to prevent the overgrowth of the cost function and make it more realistic. Furthermore, the case where $\lambda = 0.05$, $\alpha = 0.25$ is not taken into consideration, and the planning horizon is shortened to 60 due to the overgrowth of the state space as in Section 4.1. When $lambda = 0.05$, $\alpha = 0.25$ and $T > 60$, it is technologically infeasible to solve the optimization model numerically as the size of the state space grows too much. Total dimensions of the state space according to T and l^r parameters for given λ and α values are given in Tables 4.7 - 4.10.

Table 4.7 Total dimensions of the state space for $\lambda = 0.25, \alpha = 0.05$ - Declining Installed Base.

	Planning Horizon (T)			
	60	120	360	600
l^r=1	494,520	1,772,680	12,549,160	32,311,240
l^r=2	2,607,046	4,726,150	15,502,630	35,264,710
l^r=3	14,501,513	19,291,467	30,067,947	49,830,027

Table 4.8 Total dimensions of the state space for $\lambda = 0.25, \alpha = 0.25$ - Declining Installed Base.

	Planning Horizon (T)			
	60	120	360	600
l^r=1	764,127	2,629,549	15,858,349	38,072,749
l^r=2	8,587,683	14,113,840	27,342,640	49,557,040
l^r=3	100,240,855	126,599,369	139,828,169	162,042,569

Table 4.9 Total dimensions of the state space for $\lambda = 0.05, \alpha = 0.05$ - Declining Installed Base.

	Planning Horizon (T)			
	60	120	360	600
l^r=1	533,754	2,103,114	17,734,132	44,799,976
l^r=2	3,772,509	14,725,269	85,123,116	120,578,796
l^r=3	26,742,274	103,182,274	441,386,361	493,727,117

Table 4.10 Total dimensions of the state space for $\lambda = 0.05, \alpha = 0.25$ - Declining Installed Base.

	Planning Horizon (T)			
	60	120	360	600
l^r=1	844,740	3,300,778	27,274,858	66,523,535
l^r=2	12,446,317	44,961,111	275,614,131	361,155,132
l^r=3	184,524,925	616,938,270	2,986,782,240	3,259,442,485

While solving optimization models, an artificial upper bound is set on the capacity of the regular supplier similar to Section 4.3.1. This limit is determined as 1.5 times the maximum demand that may be observed in the respective period. Maximum demand is obtained by calculating the 0.9999th quantile of the chosen distribution using its estimated parameters. Note that the state space sizes given in Tables 4.7 - 4.10 are calculated using this limit.

The maximum capacity of secondary markets is determined as 12. As mentioned earlier, it is assumed that the capacity of the secondary markets is random in the following period and follows a discrete Markov chain. The finite and countable space on which this Markov chain is defined is expressed as Ω. In this case, it is defined as $\Omega = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$. As seen in Table 4.6, market scenarios 1 and 2 are considered in the numerical experiments of declining installed bases. These scenarios represent symmetric and stochastically increasing transition probability matrices, respectively. The P_1 and P_2 Markov transition probability matrices given in Section 4.3.1 are used for these two scenarios, respectively. In the decline phase of the installed base, the number of retired capital products increases over time. These capital products are dismantled and used as a source of spare parts. Therefore, the stochastically decreasing transition probability matrix, which is market scenario 3, is not considered for declining install bases. For the strategies specified in Table 4.6, the strategies given in Section 4.3.1 are used. Finally, the N_0 value, which represents the number of capital products at the beginning of the planning horizon, is taken as 10 in numerical experiments.

4.3.3 Policy Gradient heuristic

Since the optimum control policy of the inventory control model under the non-stationary demand distribution cannot be mathematically characterized when $lr > 1$, a heuristic search routine called policy gradient is used. In this approach, a simulation-based gradient search is performed on the policy parameters of the existing heuristics in the literature. While doing so, policy parameters are updated at specific points of the planning horizon since the demand distribution is not stationary. For example, consider the dual index policy. Dual index policy has two policy parameters; let us denote them by Sr and Ss. Suppose the planning horizon is six per iods, and the policy parameters are updated once during the planning horizon. In this case, a gradient search will be performed on four different policy parameters: Sr_1 and Ss_1 for the first three periods, Sr_2 and Ss_2 for the last three periods. An example visualization is given in Figure 4.3 to demonstrate the dynamics of the heuristic.

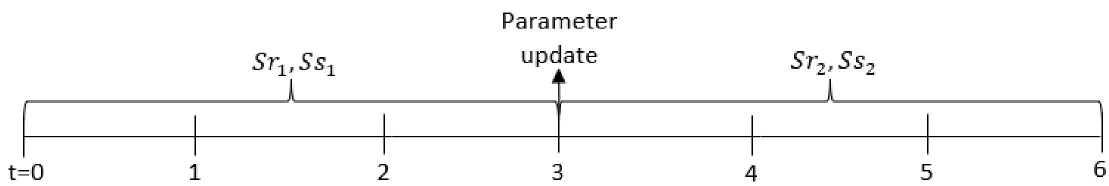


Figure 4.3 Example visualization of Policy Gradient heuristic.

Let z be the number of policy parameters to be subjected to the gradient search, and u be the number of times the policy parameters will be updated throughout the planning horizon. In this case, the size of the vector of policy parameters on which gradient search will be carried out is given by $z(u+1)$. For the example given above, $z = 1$ and $u = 1$; therefore, the size of the vector on which gradient search will be performed is 4.

For the gradient search, a starting point must be selected first. Moreover, since it is not known whether the cost function to be calculated through heuristic policies is convex in the relevant policy parameters, separate searches with more than one starting point are done. The search with the minimum cost is chosen as the optimum result. To this end, a zero vector with a dimension of $z(u+1)$ is chosen as the initial starting point. For the dual index policy example given above, the initial starting point would be $(Sr_1, Ss_1, Sr_2, Ss_2) = (0, 0, 0, 0)$.

For the second starting point, firstly, the maximum demand quantities that can be observed in each period are calculated using the previously calculated parameters of each period's demand distribution. Then, the maximum demand quantity that can be seen in each update period is selected. Let these demand quantities be represented by the $u+1$ dimensional vector $(dmax_1, \ldots, dmax_{u+1})$. This vector is chosen as the second starting point. In this vector, all the policy parameters of the first update period are chosen to be equal to $dmax_1$, and the policy parameters of the u+1st update period to $dmax_{u+1}$. To take the example of the dual index policy given above, the first update period covers the first three periods, and the second update period covers the last three periods. Assuming that the maximum quantity of demand that can be observed in the first three periods is $dmax_1 = 4$ and that the maximum quantity of demand that can be observed in the last three periods is $dmax_2 = 8$, the maximum demand vector that can be seen in each update period is $(4, 8)$. Hence, the second starting point is $(Sr_1, Ss_1, Sr_2, Ss_2) = (4, 4, 8, 8)$.

In this way, two different starting points for the gradient search are selected from the two extreme points of the $z(u+1)$ dimensional policy parameters space. In addition to these two starting points, experiments are also conducted with randomly selected starting points to improve the performance of the policy gradient heuristic. However, since it does not provide significant improvements and prolongs the working time of the heuristic, it is deemed appropriate to work with the two starting points explained earlier.

As the first step in the gradient search, since the policy parameters are integers, each parameter at the starting point is increased and decreased by one so that the gradient in every possible direction is calculated. After each increase and decrease, a simulation-based cost calculation is made, and whichever direction reduces the cost most, that parameter vector is chosen as the optimum. The individual costs are calculated again by increasing and decreasing each point on the new optimum parameter vector by one. In this step, the direction that reduces the cost most is chosen as the new optimum parameter vector. This process is repeated until all of the separately calculated costs in a step lead to an increase. When there is an increase in all directions, the search is terminated, and the previous parameter vector, in other words, the last parameter vector that reduces or does not change the cost, is chosen as the optimum parameter vector. The gradient search described here is performed for both starting points. Whichever has the minimum cost, the vector of policy parameters for that cost is accepted as the final optimum result.

Suppose a gradient search is performed on the first starting vector, $(Sr_1, Ss_1, Sr_2, Ss_2) = (0, 0, 0, 0)$, within the dual index policy example given above. In the first step, vector $(Sr_1, Ss_1, Sr_2, Ss_2) = (0, 0, 0, 0)$ will be increased or decreased by one so that separate cost calculations will be made for each of the following vectors: (1,0,0,0), (0,1,0,0), (0,0,1,0), (0,0,1,0), (0,0,0,1), (-1,0,0,0), (0,-1,0,0), (0,0,-1,0) and (0,0,0,-1). Assuming that the direction where the most improvement, cost reduction, occurs is $(0, 0, 0, 1)$, cost calculations will be held for each of the following vectors: (1,0,0,1), (0,1,0,1), (0,0,1,1), (0,0,0,2), (-1,0,0,1) (0,-1,0,1), (0,0,-1,1) and (0,0,0,0). Again, the direction with the most improvement will be chosen as the new optimum vector. These steps will be repeated until there is no improvement, and the parameter vector in the previous step will be chosen as the optimum parameter vector. Then, a gradient search will be performed on the second starting point, $(Sr_1, Ss_1, Sr_2, Ss_2) = (4, 4, 8, 8)$, following the same steps as the first starting point. As a result of the gradient searches on two starting points, the parameter vector that gives the minimum cost will be chosen as the final optimum policy parameters vector.

Gradient searches are performed through simulations with 1000 replications. Firstly, the moments of the system are calculated depending on whether the installed base is growing or declining and using the α and λ parameters of the respective system. Then, the distribution of each period is determined according to (Ord, 1967)'s geometric distribution class selection criteria. Next, the parameters of the distributions determined by the calculated moments are estimated, and 1000 demand vectors in the size of the planning horizon are produced with these parameters. At the same time, 1000 capacity vectors in the size of the planning horizon are produced using the Markov transition probability matrices of the relevant market scenarios in which the search is done. The inventory control system is managed through the policy parameters utilized in the simulation, using the demand and capacity vectors of each replication, and the cost calculation is made. Codes developed for the Policy Gradient heuristic are given in Section A.2.

The policy gradient heuristic described in this section is implemented separately for growing and declining installed bases. At the same time, an optimization model is developed, and it is solved using the value iteration algorithm. The deviation of the heuristic's cost, calculated using optimum parameters found as a result of the Policy Gradient, from the optimum cost found as a result of the optimization model is calculated. These deviations are evaluated as a measure of the performance of the heuristics. The problem parameters used for growing and declining installed bases are given in Table 4.11 and Table 4.12, respectively.

Table 4.11 Problem parameters used in numerical experiments of the Policy Gradient heuristic for the growing installed bases.

λ	α	h	b	c^r	η	ξ	l^r	γ	T	Market Scenario	u
0.05	0.05	1	15	5	0	1	1	0.99	20	1	0
0.25	0.25	5	95		-0.1	1.1	2			2	1
					-0.9	2	3			3	2
											4

Table 4.12 Problem parameters used in numerical experiments of the Policy Gradient heuristic for the declining installed bases.

λ	α	h	b	c^r	η	ξ	l^r	γ	T	Market Scenario	u
0.05	0.05	1	15	5	0	1	1	0.99	20	1	0
0.25	0.25	5	95		-0.1	1.1	2			2	1
					-0.9	2	3				2
											4

In Tables 4.11 and 4.12, h is the holding cost per unit, and b is the backlog cost per unit. λ is the rate of the Poisson process that represents capital product sales for growing installed base, the rate of the Poisson process that represents capital product retirements for declining installed base. α is the rate of the Poisson process representing spare parts failures of each capital product. l^r represents the lead time of the OEM, and c^r is its ordering cost per unit. T is the length of the planning horizon, and γ is the discount factor per period. η and ξ values are the parameters of the secondary market's purchasing cost function. The value of c_0 in this function is assumed to be equal to c^r. u is the number of times the policy parameters will be updated throughout the planning horizon. The P_1, P_2 and P_3 Markov transition probability matrices, whose explicit forms are given in Section 4.3.1, are used for market scenarios 1, 2 and 3, respectively. The value of the N_0 is equal to 10 for declining installed base.

The policy gradient heuristic routine is implemented on the Dual Index and Tailored Base-Surge policies and not implemented on the Capped Dual Index policy. The reason for this is that while the number of parameters of the other two policies is 2 ($z = 2$), the number of policy parameters of the Capped Dual Index policy is 3 ($z = 3$). Thus, it requires more calculation at each step, and the optimum parameters cannot be found for a very long time in numerical experiments. The average percentage deviations of the heuristic's cost from the optimum cost according to the number of updates on the policy parameters are given in Table 4.13 and Table 4.14 for growing and declining installed bases, respectively.

Table 4.13 Deviations of the Policy Gradient heuristic from the optimum cost for growing installed base.

Number of Updates (u)	Dual Index Policy	Tailored Base-Surge Policy
0	7.27%	49.60%
1	6.18%	39.38%
3	4.60%	32.58%
4	4.32%	28.83%

Table 4.14 Deviations of the Policy Gradient heuristic from the optimum cost for declining installed base.

Number of Updates (u)	Dual Index Policy	Tailored Base-Surge Policy
0	6.09%	65.81%
1	3.93%	42.18%
3	2.74%	27.12%
4	2.53%	23.93%

As seen in Table 4.13, the Dual Index policy showed the best performance with an average deviation of 4.32% in the case of 4 updates in growing installed bases. As expected, as the number of updates increases, policy parameters adapt better to non-stationary demand, and the performance of both policies improves. Likewise, for declining installed bases, the Dual Index policy shows the best performance with an average deviation of 2.53% when the policy parameters are updated four times. The performance of both policies increases as the number of updates increases for declining installed bases too. Note that, as the planning horizon get longer, policy parameters should be updated more. Therefore, Policy Gradient approach is not feasible for long planning horizons but applicable for mid-term planning.

4.4 Comparison of Different Distribution Selection Algorithms

As explained in Section 4.2, the non-stationary demand model uses the first three moments to characterize the maintenance demand. These moments are used to choose probability distributions according to Ord (1967)'s algorithm. As suggested by Hekimoğlu and Karlı (2021), another distribution selection algorithm developed by Adan, Van Eenige and Resing (1995) is used as a benchmark. Unlike Ord (1967)'s algorithm, Adan, Van Eenige and Resing (1995) utilize the first two moments in their algorithm to choose a probability distribution. Also, their algorithm comprises Binomial mixtures, Poisson, Negative Binomial Mixtures and Geometric mixtures distributions different than Ord (1967)'s algorithm. Since these two algorithms employ the first two and three moments, it is intended to give insights into the value of using the third moment by comparing their performance. In order to include the first moment in this comparison, the situation where the demand always follows the Poisson distribution is also taken into consideration. Similar to the two distribution selection algorithms presented before, the parameter of Poisson distribution is estimated using the calculated moments of each period. In this way, three different strategies are created as follows:

- Strategy 1 (Pure Poisson): assuming a Poisson distribution for the entire planning horizon.
- Strategy 2: Choosing each period's probability distribution according to Ord (1967)'s algorithm.
- Strategy 3: Choosing each period's probability distribution according to Adan, Van Eenige and Resing (1995)'s algorithm.

An optimization model is developed for the three strategies presented above. These optimization models have the same structure as the optimization model in Section 4.3 except for consideration of a single sourcing setup. Optimization models are solved using the value iteration algorithm for the parameters given in Table 4.15. Also, the value of N_0 parameter is taken as 25. Then, the state spaces used in the optimization model and the corresponding optimum actions (order quantities) of each state variable are recorded into a CSV file.

Table 4.15 Problem parameters used in numerical experiments for comparison of distribution selection algorithms.

λ	α	h	b	c^r	η	ξ	l^r	γ	T	Strategy
0.05	0.01	1	15	5	0	1	1	0.99	52	1
1	0.25	5	90				3			2
			95							3
			99							

After calculating the optimum actions of each strategy, a simulation model is developed to simulate the demand following dynamics of growing and declining installed base models explained in Section 4.3. So, demand sample paths are generated using two Poisson processes representing capital product sales or retirements and spare part failures of each product. One thousand demand sample paths with a length of 52 are generated for each member of the parameter space consisting of parameter values given in Table 4.15. These demand sample paths are also recorded.

Finally, a simulation model is developed to measure the performance of each strategy. Recorded demand sample paths are fed into the simulation model. Then, the optimum actions of the respective strategy are fed into the simulation model. Assuming a beginning inventory level of zero in the first period, the optimum action is found from the action space fed into the simulation model. The optimum action is found by matching the state variable and choosing the corresponding optimum action. Since a single sourcing setup is considered, the sole state variable is inventory level. For instance, in the first step, optimum action corresponding to the inventory level of zero in the first period is matched. Then, the first member of the demand sample path, demand for the first period, is subtracted from the inventory level after receiving the incoming orders, if any. Finally, the cost of the first period is calculated. The resulting inventory level becomes the state variable of the next period. Hence, it is matched with its corresponding optimum action in the next period. The same steps as the first period are followed, and the cost of the second period is calculated and added to the total cost. The code blocks developed for the aforementioned procedure are given in Section A.3.

After running simulation models with 1000 replications for each strategy, the deviations of Strategy 2 and Strategy 3 from Strategy 1 are calculated as follows:

$$\frac{(Cost\ of\ Strategy\ i) - (Cost\ of\ Strategy\ 1)}{Cost\ of\ Strategy\ 1}, \quad i \in \{2, 3\}. \tag{4.22}$$

The average and maximum improvements of Strategy 2 and Strategy 3 for growing and declining installed bases with respect to λ and α parameters are given in Table 4.16 and Table 4.17, respectively. Note that all the values given in Table 4.16 and Table 4.17 are negative; that is why they are referred to as improvements.

Table 4.16 The average and maximum improvements of Strategy 2 and Strategy 3 for growing installed base.

	Strategy 2		Strategy 3	
	Average	Max	Average	Max
$\lambda = 0.05$	1.37%	7.13%	1.37%	7.13%
$\lambda = 1$	1.19%	6.24%	1.18%	6.24%
α=0.01	0%	0%	0.01%	0.22%
α=0.25	2.56%	7.13%	2.56%	7.13%
Overall	**1.28%**	**7.13%**	**1.28%**	**7.13%**

As seen in Table 4.16, improvements made by both strategies for the growing installed base are very close. However, Strategy 2 is slightly better than Strategy 3 on average. Therefore, the use of the third moment creates an advantage over first two moments. Both strategies made an overall average cost reduction of nearly 1.28%, and they can reduce the cost up to 7.13% for specific parameters values. This implies the value of the non-stationary demand model comparing to consideration of a pure Poisson distribution over the entire planning horizon.

Table 4.17 The average and maximum improvements of Strategy 2 and Strategy 3 for declining installed base.

	Strategy 2		Strategy 3	
	Average	Max	Average	Max
$\lambda = 0.05$	0%	0.03%	0.01%	0.19%
$\lambda = 1$	1.18%	4.86%	1.2%	4.68%
α=0.01	0%	0.06%	0.09%	1.57%
α=0.25	1.18%	4.86%	1.11%	4.68%
Overall	**0.59%**	**4.86%**	**0.6%**	**4.68%**

As seen in Table 4.17, improvements made by both strategies for the declining installed base are close to each other. However, Strategy 3 is slightly better than Strategy 2. Therefore, using the first two moments is more effective than the use of the first three moments for the declining installed base. Strategy 2 and Strategy 3 made an average cost reduction of 0.59% and 0.6%, and they can reduce the cost up to 4.86% and 4.68% for specific parameters values, respectively. This implies the non-stationary demand model's importance compared to considering a pure Poisson distribution over the entire planning horizon, as it is in growing installed base.

The simulation study presented above assumes that the λ and α values are perfectly estimated. In real life, however, the observed demand may differ from the expected ones due to the errors that occurred during the calculation of λ and α values. These errors may stem from the data quality or any other human-caused errors. For instance, in the empirical study given in Chapter 5, empirical vehicle sales and spare part demand data of automobiles are analyzed. In that study, the source of the spare parts demand data is authorized MROs. However, the users of automobiles do not always get service from the official MROs. They may sometimes prefer unofficial MROs since they are cheaper than the authorized ones. This situation led to a lower α rate.

To test these kinds of scenarios, uniformly distributed error coefficients, ϵ, based on error levels, $\rho \in \{0.05, 0.25\}$, are incorporated into the simulation model. Hence, the error coefficients becomes $\epsilon \sim Uniform(-\lambda\rho, \lambda\rho)$ for λ, $\epsilon \sim Uniform(-\alpha\rho, \alpha\rho)$ for α. The error coefficients are added to the λ and α values before generating demand sample paths, the same method explained above is followed for the remaining parts. In other words, while calculating optimum actions through optimization models, no error is presented. In this way, it is to aimed test fault tolerance of the distribution selection algorithms. The code blocks developed for the aforementioned procedure are given in Section A.3. The simulation study is conducted using the parameters given in Table 4.15. After running simulation models with 1000 replications for each strategy, the deviations of Strategy 2 and Strategy 3 from Strategy 1 are calculated following Equation(4.22). The average and maximum improvements of Strategy 2 and Strategy 3 for growing and declining installed bases with respect to error level are given in Table 4.18 and Table 4.19, respectively. Note that all the values given in Table 4.18 and Table 4.19 are negative; that is why they are referred to as improvements.

Table 4.18 The average and maximum improvements of Strategy 2 and Strategy 3 for growing installed base when estimation error is presented.

	Strategy 2		Strategy 3	
	Average	Max	Average	Max
$\rho = 0$	1.28%	7.13%	1.28%	7.13%
$\rho = 0.5$	1.33%	7.73%	1.32%	7.73%
$\rho = 0.25$	1.61%	14.2%	1.6%	14.2%
Overall	**1.41%**	**14.2%**	**1.4%**	**14.2%**

As seen in Table 4.18, improvements made by both strategies for the growing installed base are very close to each other, but Strategy 2 showed slightly better performance similar to the result in Table 4.16. Thus, the use of the third moment is advantageous when estimation error is presented. Both strategies made an overall average cost reduction of 1.4%, and they can reduce the cost up to 14.2% for specific parameters values. This implies the value of the non-stationary demand model comparing to consideration of a pure Poisson distribution over the entire planning horizon when there is an estimation error.

Table 4.19 The average and maximum improvements of Strategy 2 and Strategy 3 for declining installed base when estimation error is presented.

	Strategy 2		Strategy 3	
	Average	Max	Average	Max
$\rho = 0$	0.59%	4.86%	0.6%	4.68%
$\rho = 0.5$	0.64%	5.53%	0.66%	4.96%
$\rho = 0.25$	0.93%	8.82%	1.08%	9.26%
Overall	**0.72%**	**8.82%**	**0.78%**	**9.26%**

As seen in Table 4.19, improvements made by both strategies for the growing installed base are close to each other. However, Strategy 3 is slightly better than Strategy 2 similar to the results given in Table 4.17. Therefore, using the first two moments is more effective than the use of the first three moments for the declining installed base when estimation error is presented. Strategy 2 and Strategy 3 made an average cost reduction of 0.72% and 0.78%, and they can reduce the cost up to 8.82% and 9.26% for specific parameters values, respectively. This implies the non-stationary demand model's importance compared to considering a pure Poisson distribution over the entire planning horizon, when there is an estimation error.

To sum up, the non-stationary demand model leads to improvements for both stages of the installed base. The performances of the Strategy 2 and Strategy 3 are very close in growing installed base, but Strategy 2 is slightly better. In the declining installed base, Strategy 3 is slightly better than the Strategy 2. The importance of the non-stationary demand model is revealed when there is an estimation error. In case of estimation error, both strategies make significant cost reductions up to 14.2% for growing installed base and up to 9.26% for declining installed bases.

5. AN EMPIRICAL ANALYSIS OF NON-STATIONARY SPARE PARTS DEMAND IN THE AUTOMOTIVE INDUSTRY

In this chapter, an empirical study is conducted on a Turkish automotive importer to test the non-stationary demand model explained in Section 4.2. To this end, automobile sales data and spare part replacement data of these automobiles are collected. Spare part replacement data represent the repair demand of cars. Analyses are held for one of the top seller vehicles of the Turkish automotive industry. From now on, this selected automobile model will be called Model 1 due to confidentiality.

Firstly, an exploratory data analysis is conducted for Model 1 to get insights into the dynamics of how spare parts demand is generated. The in-depth examinations of automobile sales and spare part replacement data are presented separately in Section 5.1 and Section 5.2, respectively.

Sales quantities of Model 1 with respect to given years are given in Table 5.1.

Table 5.1 Quantities of Model 1s sold for given periods.

Year	Sales Quantity
Before 2018	223,057
2018	9,922
2019	5,938
2020	7,682
Total	**246,599**

If we consider the three years 2018, 2019 and 2020, the total number of Model 1 sold is 23,452, which means the yearly average sales of 7,847.33 and monthly average sales of 653.94.

After sales data, the spare part replacement data set is filtered based on the vehicle IDs of the cars sold. The number of unique cars involved in a part change is calculated and combined with Table 5.1 as given in Table 5.2. Note that the spare part replacement data set consists of data starting from 2018.

Table 5.2 Number of Model 1 sold for a given period and involvement of those cars in part changes.

Year	Quantity of Vehicles Sold	Number of Unique Vehicles Involved in a Part Change	Percentage of Unique Vehicles Involved in a Part Change
Before 2013	111,131	13,954	12.6%
2013	22,449	6,955	31%
2014	19,748	9,368	47.4%
2015	21,718	13,753	63.3%
2016	27,580	22,772	82.6%
2017	20,431	19,442	95.2%
Before 2018	223,057	86,244	38.7%
2018	9,922	9,595	96.7%
2019	5,938	5,415	91.2%
2020	7,682	1,689	22%
2018 and Later	27,972	16,803	60.1%
Total	**246,599**	**102,943**	**41.7%**

The source of the spare parts change data is the authorized MROs for respective automobile models in Turkey. Since Model 1 is a low-mid segment car, the users of the vehicles may prefer unofficial MROs after the end of the warranty period due to their lower service fees and cheap unofficial spare parts they provide. The guarantee period for Model 1 is two years. As seen in Table 5.2, vehicles sold in 2016, 2017, 2018, 2019 and 2020 most participated in part changes that occurred in 2018 and later since the data period covers their warranty period. This implies the hypothesis that the users of the vehicles may prefer unofficial MROs after the end of guarantee period.

To see the impact of the warranty period, what percent of the total spare part demand for given years is generated by Model 1s sold in 2013 is calculated and given in Table 5.3. To see the same impact from another perspective, what percent of Model 1s sold in 2013 got involved in a part change for given years is calculated and given in Table 5.4.

Table 5.3 Quantity of spare parts demand driven by Model 1s sold in 2013 for given years.

Year	Total Demand	Demand Driven by Model 1s Sold in 2013	Percentage of Demand Driven by Model 1s Sold in 2013
2018	1,884,983	120,823	6.41%
2019	1,685,805	87,919	5.22%
2020	1,355,817	72,273	5.33%

Table 5.4 Model 1s sold in 2013 and got involved in a part change for given years.

Year	Model 1s Participated in a Part Change	Model 1s Sold in 2013 and Participated in a Part Change	Percentage of Model 1s Sold in 2013 and Participated in a Part Change
2018	73,683	4,639	6.3%
2019	64,069	3,554	5.5%
2020	52,414	2,876	5.5%

Since the spare part replacement data starts from 2018 and as a result of analyses explained above, it is decided to work with Model 1s sold in 2018, 2019 and 2020. Thus, the spare part demand data set is filtered for the vehicles sold in 2018, 2019 and 2020. This filtered data set will be referred to as spare part replacement/demand data set from now on. In this way, a growing installed base will be obtained. Recall that the warranty period for Model 1 is two years.

An analysis is also held for selected spare parts of Model 1. To this end, four spare parts are chosen for further examination. Replacement frequencies of selected spare parts are given in Table 5.5.

Table 5.5 Replacement frequencies of selected parts.

Part Number	Frequency
1	3,146
2	2,163
3	24
4	19

Part 1 and Part 2 are chosen to represent high-frequency spare parts, and Part 3 and Part 4 are selected to represent low-frequency parts. To reveal how the demand quantity emerges over time, frequencies of selected parts with respect to years are calculated and given in Table 5.6.

Table 5.6 Replacement frequencies of selected parts by years.

Year	Part 1	Part 2	Part 3	Part 4
2018	166	51	2	7
2019	1,220	592	5	8
2020	1,760	1,520	17	4
Total	3,146	2,163	24	19

As seen in Table 5.6, the observed spare part demand increases over time as the number of vehicles in use rise, hence indicates a growing installed base. The only exception is Part 4 in 2020, but this could be tolerable since Part 4 is an extremely slow-moving part. Further investigation is conducted by visualizing the spare parts demand of each part by years as given in Figure 5.1 (a) and Figure 5.1 (b), where $+$ stands for a spare part replacement. These visualizations are done by plotting demand generated by cars sold in 2018, 2019 and 2020 separately and taking their superposition as overall demand. The reason to plot spare parts demand by years is that it is not feasible to plot it by each unique vehicle by a horizontal line as in

Figure 4.1 since there are thousands of them. Figures 5.1 (a) and 5.1 (b) are pretty similar to Figure 4.1. Hence, it could be qualitatively said that the demand of Part 1 and Part 2 follows similar dynamics to the growing installed base explained by Jin and Liao (2009) and Hekimoğlu and Karlı (2021).

5.1 Testing Vehicle Sales Data

Before proceeding to the testing stage, the sales process of Model 1 is investigated by visualizations. The daily sales of Model 1 ($N_{(t,\Delta t)}$) over time (t) throughout the analysis period are given in Figure 5.2.

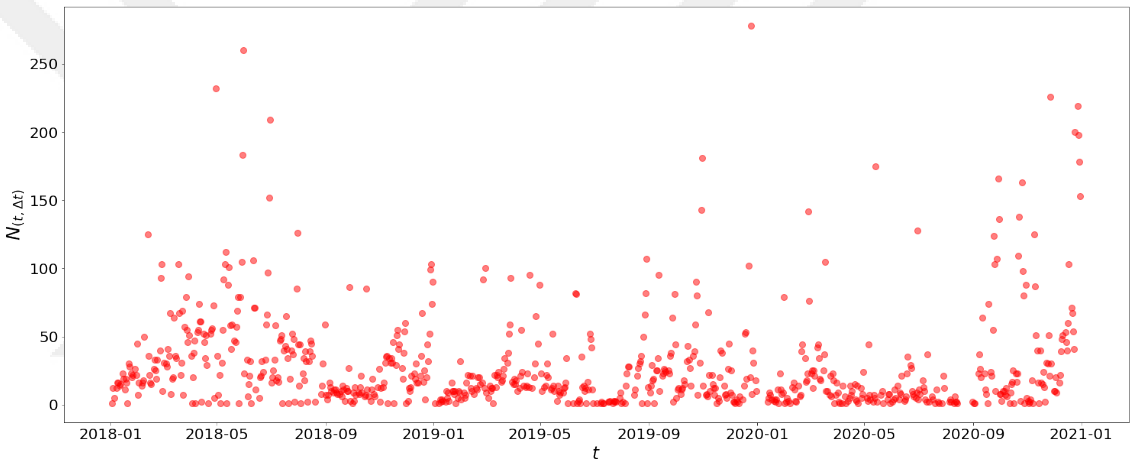
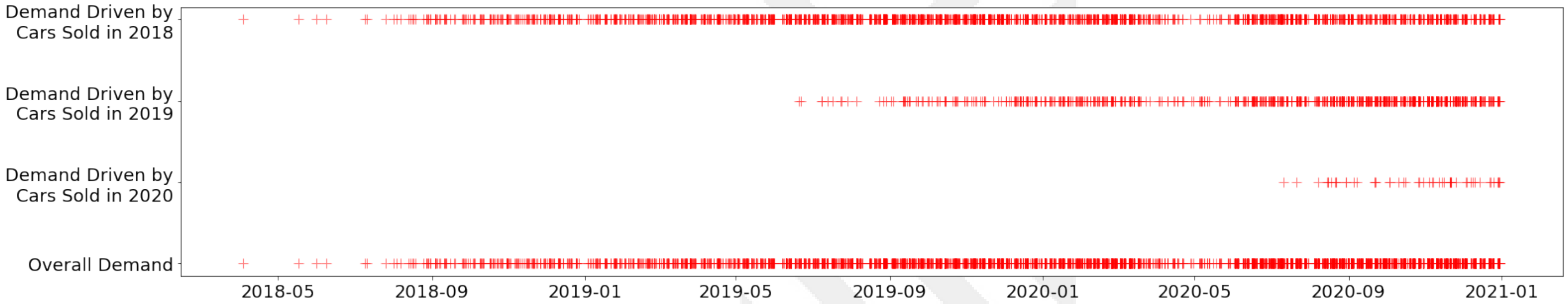
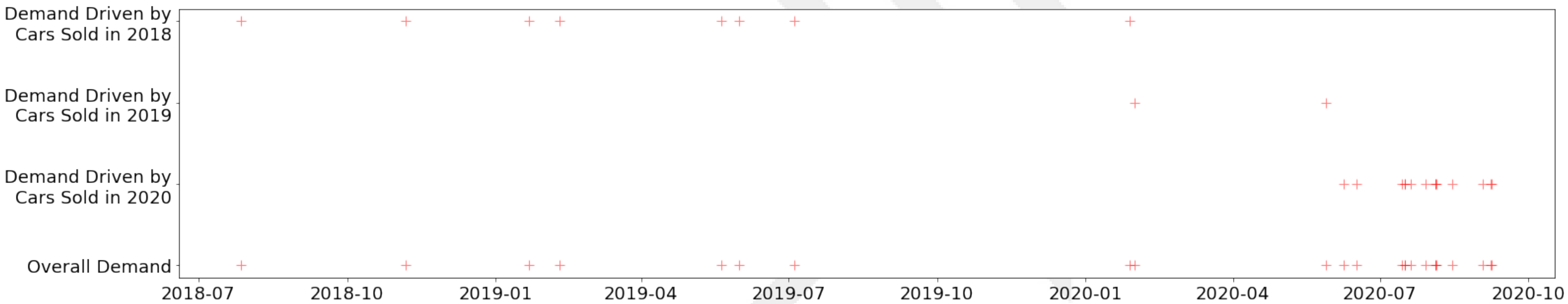


Figure 5.2 Daily sales of Model 1 in 2018-2020.

The daily sales process intensity of Model 1 ($\frac{N_t}{t}$) by time during the analysis period is plotted in Figure 5.3 to see how the sales process intensity changes over time.



(a) Part 1.



(b) Part 4.

Figure 5.1 Visualization of spare parts demand by vehicles sold in 2018, 2019 and 2020.

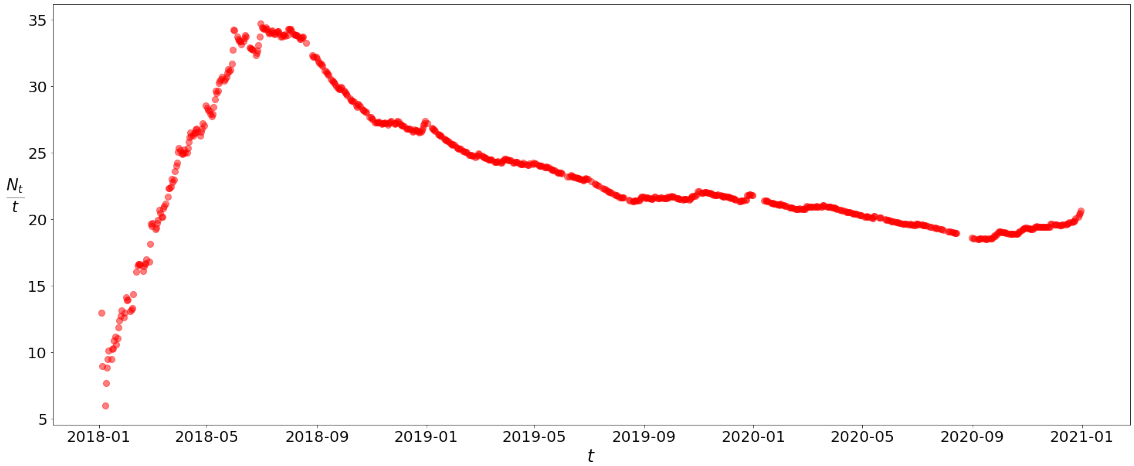


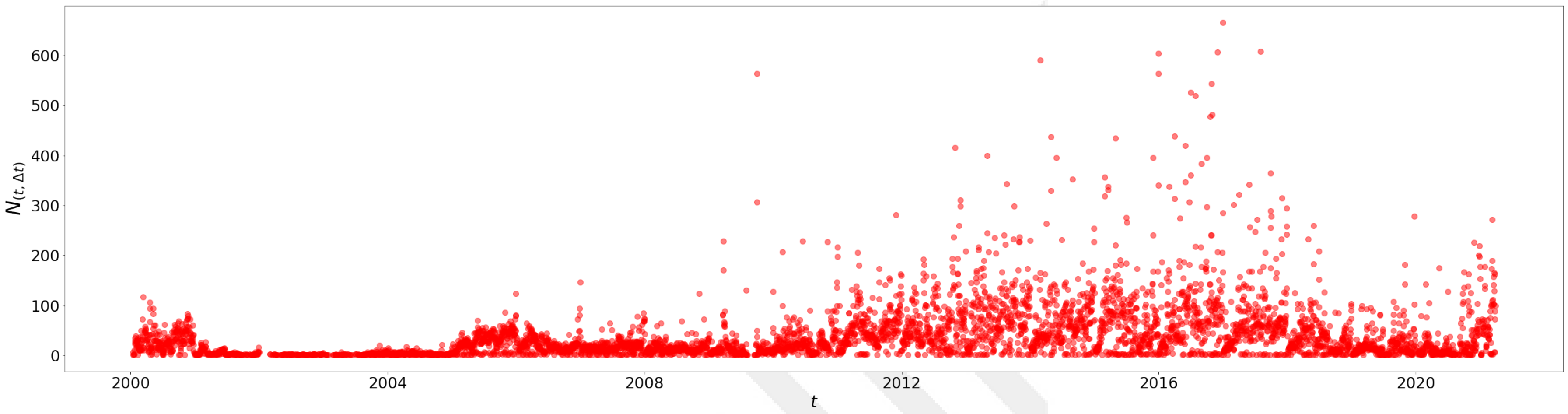
Figure 5.3 Daily sales process intensity of Model 1 in 2018-2020.

As seen in Figure 5.3, the sales process intensity of Model 1 does not converge to a constant value in contrast with the assumption of a homogeneous Poisson process by Jin and Liao (2009) and Hekimoğlu and Karlı (2021). The same visualizations are also made for a more extensive time interval to detect if the sales process intensity is still non-stationary. Figure 5.4 (a) shows the daily sales of Model 1 over the period 2000-2020. The daily sales process intensity of Model 1 from 2000 to 2020 is given in Figure 5.4 (b) to demonstrate how the sales process density changes over time. Model 1's sales process intensity does not converge to a fixed value, as seen in Figure 5.4 (b), in contradiction to assumption of a homogeneous Poisson process.

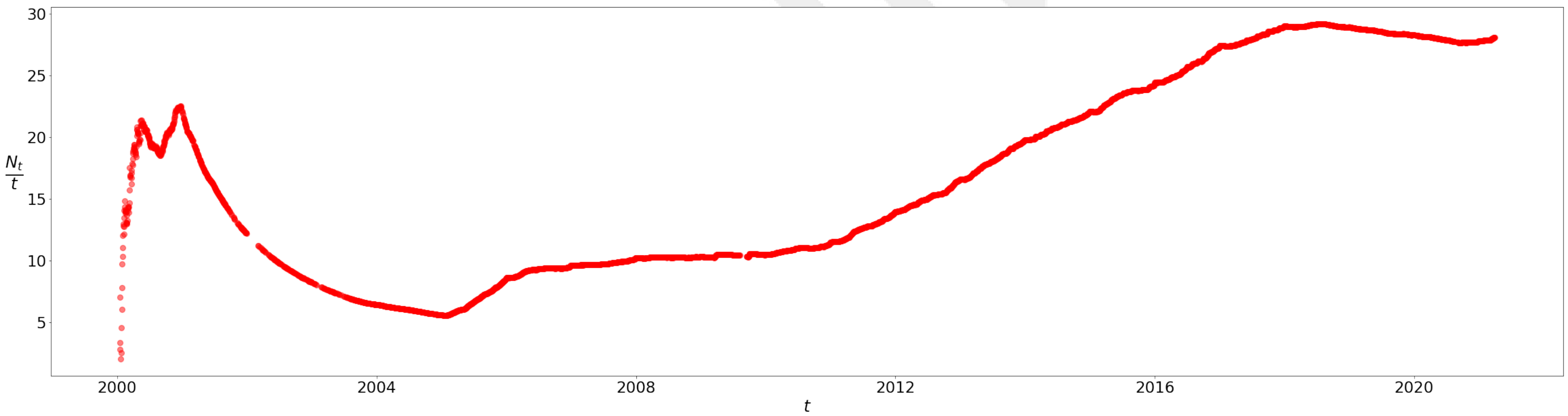
Several statistical tests are conducted in the following subsections to test the Poisson process assumption on the sales process.

5.1.1 Kolmogorov-Smirnov test

In this section, a Kolmogorov-Smirnov test is performed on the interarrival times of the vehicle sales. We test the hypothesis of car sales following a homogeneous Poisson Process. The test hypothesis implies that interarrival times are distributed exponentially. We test the null hypothesis that interarrival times of Model 1 follow an exponential distribution using Kolmogorov-Smirnov (KS) test. Since the time unit of the interarrival times affects the test results, all KS tests are conducted



(a) Daily sales.



(b) Intensity of daily sales.

Figure 5.4 Daily sales process of Model 1 during 2000-2020.

in three different instances by converting the time units into hours, minutes and seconds.

In our data set, the sales data before 2005 have the date information in day-month-year format only. Since there are many sales events within the same day, using this format leads to interarrival times of zero. Hence, the KS test rejects the null hypothesis because the interarrival times mainly consist of zeros. However, the sales data starting from 2005 have the date information in day-month-year-hour-minute-second format. Therefore, the vehicle sales data starting from 2005 is employed in the KS tests.

The inter-sales times between 2005 and 2020 are calculated. Then, the KS test is conducted on these inter-sales times, and the null hypothesis is rejected, resulting a p-value of zero. This is an expected result since the test period is very long and car sales are affected by economic factors and customers' future expectations. For this reason, the periods subjected to the KS test are split into smaller intervals. In this way, we test the hypothesis that car sales follow a non-homogeneous Poisson Process for given time intervals. Our test method followed to obtain test intervals is given below:

Weekdays: The sales dates are filtered based on the weekdays, and interarrivals times within each day are calculated. For instance, let us assume that the test is performed on Mondays. Then, sales dates are filtered for the Monday of the first week, and their interarrival times are calculated within themselves. These interarrival times are then recorded in a vector. Afterward, the sales dates are filtered for the Monday of the second week, and the inter-sales times are calculated. The calculated inter-sales times are added to the vector in which the interarrival times of the first week are recorded. This process is repeated until the end of 2020. In this way, we test a hypothesis that interarrival times of sales on Mondays follow an exponential distribution. By following this method, seven groups representing weekdays are obtained, and separate KS tests are held for each group.

Weeks of the year: The selling dates are grouped by week of a year. Then, inside each week, interarrival times are computed. For instance, selling dates are filtered by the first week of 2005. Inter-sales times are calculated within this week and saved in a vector. Subsequently, sales dates in the first week of 2006 are selected, and inter-sale times are calculated then saved in the vector containing the inter-sale times of 2005's first week. The same steps are repeated until 2020. In this manner, it is intended to evaluate if the time between sales in the first week of a year follows an exponential distribution. As a result, 52 groups are formed, and individual KS tests are administered to each group. To formulate the test hypothesis, let c be the index for smaller cyclic periods such as months, weeks, quarters et cetera. Then, the hypotheses become:

$$H_0 : D_c \sim Poisson(\lambda_c).$$
$$H_1 : D_c \nsim Poisson(\lambda_c).$$

The same methodology as explained in weekdays and weeks of the year is utilized for all cyclic time intervals. Table 5.7 shows periodic time intervals and the resulting number of groups to be tested. KS tests are performed for all these periods. All tests rejected the null hypothesis that inter-sales times of Model 1 follow an exponential distribution, resulting in a p-value equal to zero.

Table 5.7 Cyclic periods and their resulting groups to be tested by KS test.

Cyclic Period	Resulting Groups
Weekdays	7
Weeks of the Year	53
Months	12
Quarters	4
Seasons	4

After cyclical periods, another approach is employed. First, a time length is determined, then KS tests are performed for fixed and consecutive intervals of a predetermined time length. For instance, if the selected time interval is a month, the

KS test is conducted for sales in January 2005. In the next step, the KS test is implemented for sales in February 2005, and identical steps are repeated until the end of 2020. To formulate the test hypothesis, let t be year index and j be the index for smaller successive periods such as months, weeks, quarters et cetera. Then, the hypotheses become:

$$H_0 : S_{t,j} \sim Poisson(\lambda_{t,j}).$$
$$H_1 : S_{t,j} \nsim Poisson(\lambda_{t,j}).$$

The time intervals considered in the tests, and the corresponding number of groups to be tested are given in Table 5.8.

Table 5.8 Consecutive time intervals and their resulting groups to be tested by KS test.

Time Interval	Resulting Groups
Months	192
Quarters	64
Seasons	63
Half years	32
Years	16

KS tests are conducted for the successive periods shown in Table 5.8. The hypothesis that the inter-sales times Model 1 are distributed exponentially is rejected for all periods and resulted in a p-value of zero. The code blocks used to perform KS test and the test result are given in Section B.1.

5.1.2 Böhning's test

In this section, the test procedure presented by Böhning (1994) is implemented. The null hypothesis is that the sales process of Model 1 follows a non-homogeneous Poisson process.

Böhning's test could be classified as a variance-related test among the numerous procedures developed to test the Poisson assumption (Karlis and Xekalaki, 2000). These kinds of tests use the fact that variance is equal to the mean for Poisson distribution, hence make use of the coefficient of dispersion (Karlis and Xekalaki, 2000). Böhning (1994) utilizes this fact and defines the test statistics, O, as follows:

$$O = \sqrt{\frac{n-1}{2}} \left(\frac{S^2}{\overline{X}} - 1 \right), \tag{5.1}$$

where n is the sample size, S is the sample standard deviation, and $\overline{X}$ is the sample mean. He derives this statistic by using the fact that $S^2 - \overline{X}$ should follow a standard normal distribution. Karlis and Xekalaki (2000) state that Böhning's test is the best one in its class with respect to the power of statistical tests; also, it is easy to implement. We choose Böhning's test to test the Poisson process assumption on vehicle sales data. To this end, the test procedure is coded and implemented on Model 1's sales data, as explained below.

Böhning's test is conducted on the number of daily arrivals (vehicle sales) for a given period. Hence, unlike the KS test, precise sales date information is not needed. In our analysis, the vehicle sales data set which starts from 1996 is used. The same method explained in Section 5.1.1 is used to determine test periods, with one difference: this time test is performed on daily sales quantities in the selected time interval.

First, Böhning's test is conducted on cyclic periods. For instance, the number of arrivals on respective weekdays is grouped as samples for weekdays, resulting in 7 groups. Then, Böhning's test is implemented separately on these samples. If the weekday subjected to test is Monday, it is tested whether the sales that occurred on Mondays follow a Poisson distribution. Considered periodic time intervals and the resulting number of groups to be tested are same as given in Table 5.7. Böhning's test rejected the null hypothesis that the sales process of Model 1 follows a Poisson process for all cyclic periods.

Next, successive time intervals are considered following the same approach explained in Section 5.1.1. The time intervals considered in the tests, the corresponding number of groups subjected to the test, and the acceptance rates are given in Table 5.9.

Table 5.9 Successive time intervals, their resulting groups subjected to test, and acceptance rates by Böhning's test.

Time Interval	Resulting Groups	Acceptance Rate
Fortnight	632	9.34%
Month	292	4.81%
Quarter	98	1.03%
Season	98	0%

As seen in Table 5.9, acceptance rates decline as time intervals to be tested get longer. Moreover, a closer look at the results reveals that acceptance density is high in 2002 and 2003. Thus, the vehicle sales during 2002 and 2003 could be modeled using a Poisson process. Table 1 shows the monthly test results for 2002 and 2003, along with test statistics, p-values, and sample means. Note that N/A values in the test statistics mean no sales record in the respective period; hence p-values of zero are assigned to them.

Table 5.10 Monthly results of Böhning's test in 2002 and 2003.

Year	Month	O	P-value	$\overline{X}$
2002	1	N/A	0.00	0.00
2002	2	N/A	0.00	0.00
2002	3	3.07	0.00	0.35
2002	4	-0.53	0.70	0.17
2002	5	1.07	0.14	0.52
2002	6	0.95	0.17	0.63
2002	7	4.03	0.00	0.58
2002	8	-1.16	0.88	0.32
2002	9	0.53	0.30	0.40
2002	10	-0.36	0.64	0.32
2002	11	3.02	0.00	0.60
2002	12	2.66	0.00	1.06
2003	1	-0.52	0.70	0.16
2003	2	-0.14	0.55	0.07
2003	3	0.10	0.46	0.26
2003	4	5.10	0.00	0.63
2003	5	2.15	0.02	0.42
2003	6	-0.26	0.60	0.10
2003	7	3.92	0.00	0.39
2003	8	-0.94	0.83	0.52
2003	9	3.01	0.00	1.70
2003	10	9.73	0.00	2.48
2003	11	25.79	0.00	2.07
2003	12	5.32	0.00	4.16

In Figure 5.5, the daily sales of Model 1 are plotted and marked for 2002 and 2003 since these years differ from others in acceptance density. In this way, it is aimed to demonstrate how 2002 and 2003 differ from others.

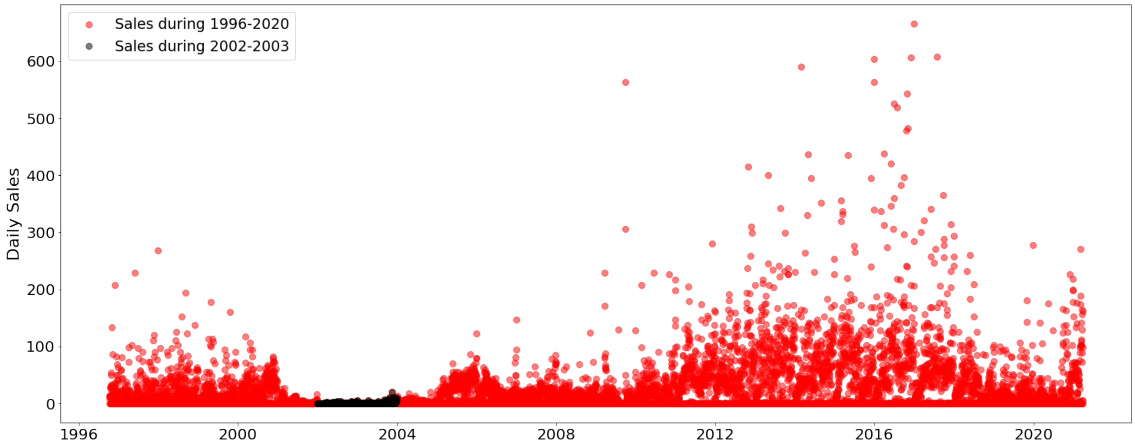


Figure 5.5 Daily sales of Model 1 marked for 2002 and 2003.

As seen in Figure 5.5, the daily sales throughout 2002 and 2003 are steady, while other year are highly over-dispersed. This prevents the coefficient of dispersion to be equal to one. Therefore, Böhnings's test rejects the null hypothesis for those intervals. The code blocks used to perform Böhning's test and the test result are given in Section B.2.

5.1.3 Brown's test

In this section, the test procedure developed by Brown et al. (2005) is implemented on the sales data of Model 1.

Brown et al. (2005) develop the test procedure for a call center. They utilize the a commonly used assumption for call centers that is arrival rate remains constant for fixed time interval. Then they extend this assumption by proposing a non-homogeneous Poisson process with piecewise arrival rates. They construct the test procedure as explained below.

$$R_{ij} = (J(i) + 1 - j)\left(-\log\left(\frac{L - T_{ij}}{L - T_{i,j-1}}\right)\right), \quad j = 1, \dots, J(i). \tag{5.2}$$

In Equation (5.2), T_{ij} is the jth ordered arrival time within the ith group where $i = 1, \dots, I$. $J(i)$ represents quantity of total arrivals within the ith group, hence $T_{i1} \leq \cdots \leq T_{iJ(i)}$. Note that $T_{i0} = 0$. Brown et al. (2005) show that $\{R_{ij}\}$

consist of standard exponential variables. They reach this conclusion by utilizing the null hypothesis that arrival rate is fixed within each group. Finally, they suggest performing a KS test on R_{ij}.

Since test procedure uses arrival times, the vehicle sales data starting from 2005 is subjected to test because of the data format as explained in Section 5.1.1. Brown's test first performed for the null hypothesis that automobile sales data of Model 1 follows a homogeneous Poisson process throughout 2005-2020. The test resulted in a p-value equal to zero, thus the null hypothesis is rejected. Then, the test is conducted for shorter time intervals, and these intervals are obtained following the method given in Section 5.1.1.

For shorter periods, successive intervals are considered. In these test, the null hypothesis that the vehicle sales of Model 1 follows a homogeneous Poisson process for respective time interval tested. The time intervals, the resulting number of groups to be tested, and the acceptance rates are shown in Table 5.11.

Table 5.11 Successive time intervals, resulting number of groups subjected to test, and their acceptance rates by Brown's test.

Time Interval	Resulting Groups	Acceptance Rate
Fortnight	422	0.24%
Month	196	0.52%
Quarter	64	0%
Half-year	32	0%

The acceptance rates are very low for biweekly and monthly intervals and zero for quarterly and half-yearly intervals. Therefore, it is concluded that the sales process of Model 1 is not a homogeneous Poisson process for considered periods. The code blocks used to perform Brown's test and the test result are given in Section B.3.

5.2 Testing Repair Demand Data

The repair demand process of Model 1 is investigated by visualizations before proceeding to the testing stage. Please remind that the analysis period of the growing installed base is determined as 2018, 2019 and 2020. Therefore, the spare part demand data set is filtered for demand generated by automobiles sold in 2018, 2019 and 2020.

In addition to Part 1 and Part 2, four more new parts are incorporated into the further investigation. The mean daily maintenance demand rate of these spare parts, ($\frac{S(t)}{t}$), over time is shown in Figure 5.6.

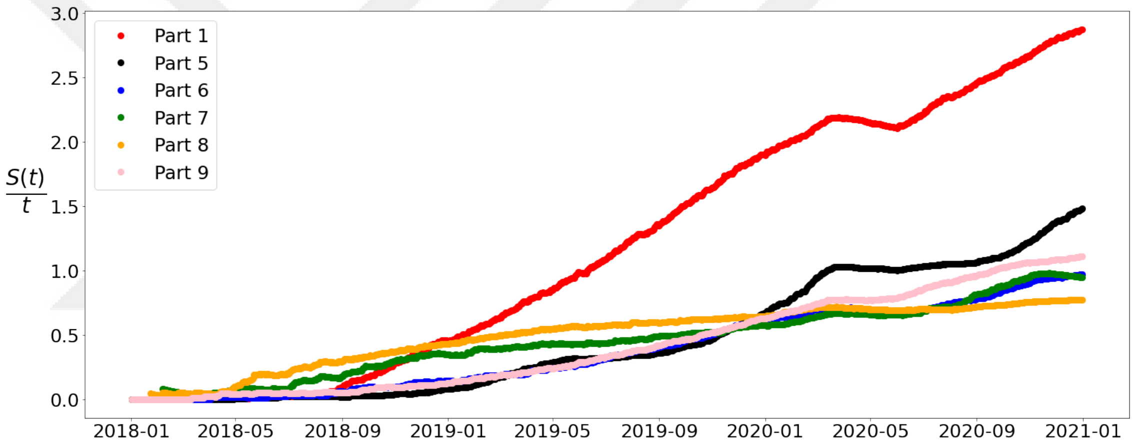


Figure 5.6 Mean daily maintenance demand of selected parts.

As seen in Figure 5.6, average daily demand rate increases over time as expected for a growing installed base. Daily repair demand per capital product, $\frac{S(t)}{N_t}$, is plotted in Figure 5.7. Figure 5.7 implies that daily repair demand per vehicle is increasing by time, hence indicating a growing installed base. To examine individual parts in detail, arrivals of daily demand for Part 1 and Part 8 visualized in Figure 5.8 and Figure 5.9, respectively. Daily demand arrivals of the both parts show an increasing demand, which implies the assumption of a growing installed base.

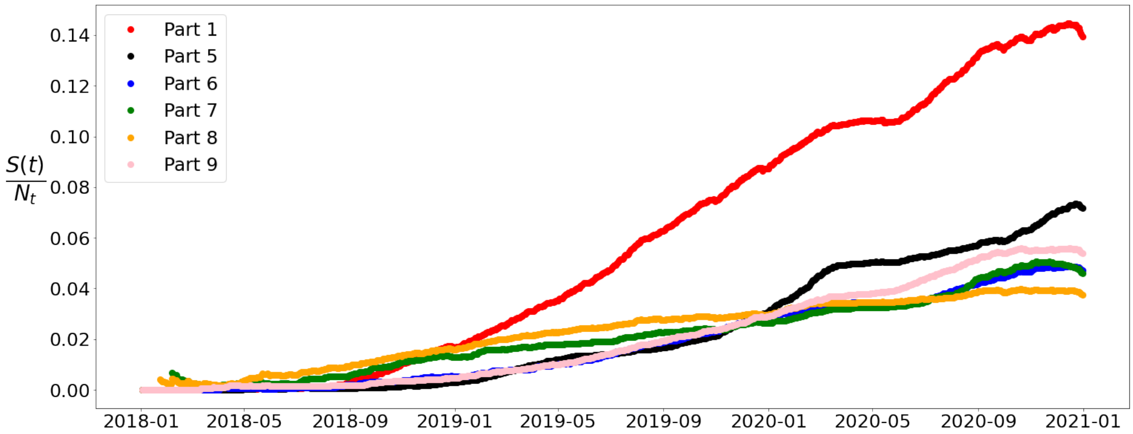


Figure 5.7 Daily repair demand per capital product for selected parts.

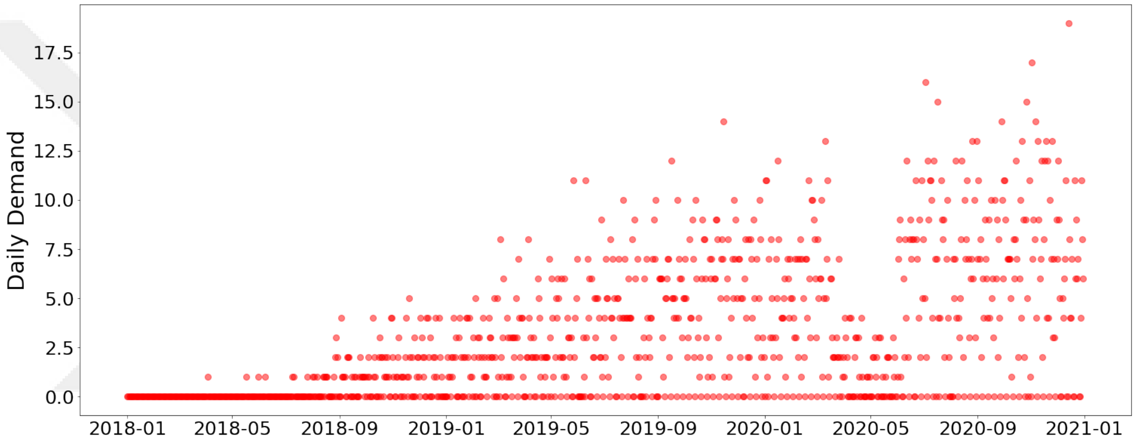


Figure 5.8 Daily repair demand of Part 1.

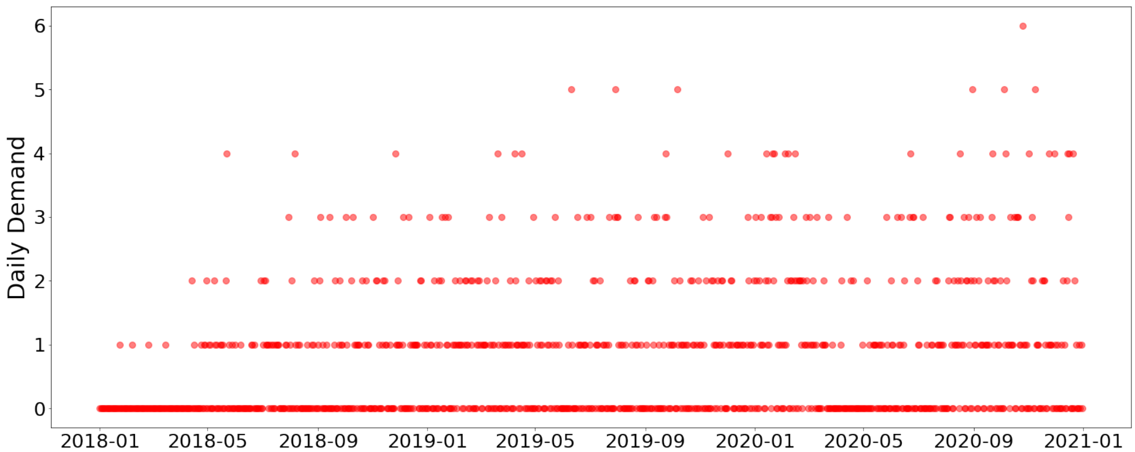


Figure 5.9 Daily repair demand of Part 8.

5.2.1 Chi-square goodness of fit test

In this section, It is intended to see how well empirical data fit chosen distributions. Moreover, tests are separately conducted utilizing Ord's and Adan's distribution selection algorithms in an effort to compare their performances on empirical maintenance demand data.

To do so, the interval of 2018-2020 is divided into 30 days periods, and tests are performed on daily demand arrival within these periods. The resulting number of groups to be tested is 36. Then, the α rate of the spare part to be tested is calculated. The α rates are obtained by calculating the α rate of individual capital products then taking the average of them. Then, the λ rate is calculated as 21.54, which is the same for each spare part since they belong to the same automobile model. Next, the moments are calculated using the α and λ rates for a period of 1095. Probability distributions are determined depending on the distribution selection algorithm, and their parameters are estimated for each period by utilizing these moments. A probability distribution must be determined to implement the chi-square goodness of fit test on intervals of 30 days. To this end, the probability distributions calculated from the moments are also divided into 30-day periods, and the median probability distribution is used in the test. Let say the first 30 days period is to be tested, then the 15th probability distribution among 1095 is employed in the chi-square test. For the second 30 days interval, the 45th probability distribution is utilized. Then, the daily demand data is consolidated so as to obtain minimum expected frequencies of two, as suggested by Rayner, Thas and Best (2009). Expected frequencies are calculated using the selected distribution's probability mass functions (PMF) and cumulative distribution functions (CDF). Consequently, the chi-square statistic, critical value and p-value are calculated. The code blocks used to perform chi-square test are given in Section B.4.

The chi-square goodness of fit test is implemented for 46 different spare parts. These parts are selected among the most frequent ones because the low-frequent parts are lead to the sole class with an expected frequency of at least two, which is zero. Also, it is attempted to extract spare parts used for periodic maintenance as possible since those parts are usually ordered in batch quantities, hence breaking down the Poisson process assumption. The spare parts subjected to the chi-square test, their α values along with the acceptance rates of Ord's and Adan's algorithm are given in Table 5.12.

Table 5.12 Results of the chi-square goodness of fit test.

		Acceptance Rate		
Part Number	α	Ord	Adan	Superiority
1	2.09E-04	16.67%	13.89%	Ord
2	1.97E-04	8.33%	8.33%	None
5	5.74E-03	0.00%	0.00%	None
6	1.96E-03	0.00%	0.00%	None
7	1.75E-03	0.00%	0.00%	None
8	1.30E-03	0.00%	0.00%	None
9	9.19E-04	5.56%	5.56%	None
10	9.29E-04	5.56%	5.56%	None
11	8.62E-04	2.78%	2.78%	None
12	8.17E-04	5.56%	5.56%	None
13	9.58E-04	5.56%	5.56%	None
14	9.04E-04	5.56%	5.56%	None
15	9.02E-04	8.33%	8.33%	None
16	5.37E-04	0.00%	0.00%	None
17	5.37E-04	0.00%	0.00%	None
18	6.33E-04	19.44%	19.44%	None
19	3.96E-04	2.78%	2.78%	None
20	3.76E-04	0.00%	0.00%	None
21	5.11E-04	0.00%	2.78%	Adan
22	3.61E-04	2.78%	2.78%	None
23	5.22E-04	2.78%	2.78%	None

Table 5.12 continued from previous page

24	3.13E-04	8.33%	5.56%	Ord
25	1.76E-04	0.00%	0.00%	None
26	1.46E-04	0.00%	0.00%	None
27	1.78E-04	2.78%	2.78%	None
28	1.77E-04	0.00%	0.00%	None
29	2.08E-04	2.78%	0.00%	Ord
30	1.26E-04	19.44%	19.44%	None
31	9.35E-05	2.78%	0.00%	Ord
32	7.59E-05	33.33%	30.56%	Ord
33	9.29E-05	5.56%	2.78%	Ord
34	9.29E-05	11.11%	8.33%	Ord
35	7.04E-05	47.22%	44.44%	Ord
36	7.62E-05	2.78%	0.00%	Ord
37	1.35E-04	11.11%	11.11%	None
38	9.54E-05	16.67%	19.44%	Adan
39	5.77E-05	13.89%	13.89%	None
40	6.95E-05	11.11%	11.11%	None
41	4.15E-04	0.00%	0.00%	None
42	7.98E-05	5.56%	2.78%	Ord
43	1.50E-04	0.00%	0.00%	None
44	5.12E-05	11.11%	11.11%	None
45	6.64E-05	36.11%	38.89%	Adan
46	1.07E-04	52.78%	55.56%	Adan
47	5.59E-05	11.11%	11.11%	None
48	4.38E-05	16.67%	16.67%	None

As seen in Table 5.12, the acceptance rates vary among spare parts, reaching maximum acceptance rates of 52.75% and 55.56% for Ord's and Adan's algorithms, respectively. The superiority of these algorithms is also presented in Table 5.12. It is found that for 23% of the parts Ord's algorithm and 9% of the parts Adan's algorithm has superiority over the other. This result is in parallel to theoretical results presented in Section 4.4.

6. CONCLUSIONS

MROs are critical in spare parts supply chains since the continuity of production/services depends on the quality of the service they provide. Thus, MROs should keep sufficient inventory to meet their customers' needs on time while preventing high holding costs. The spare parts demand is profoundly dependent on the installed base, so the installed base information should be incorporated in demand forecasting methods for effective inventory control. Moreover, the dynamics of the installed base entirely differ in growth and decline phases; hence different inventory control policies should be employed for these phases. Another factor that needs to be considered by MROs is secondary markets since they can be exploited as a cheap and expedited source of spare parts and an alternative where excess inventory can be sold. However, secondary markets have a finite capacity, making them unreliable. Therefore, MROs need a procurement policy to utilize secondary markets and OEM at the same time optimally.

In this thesis, a finite-horizon discounted cost function is derived using dynamic programming for an inventory control model in a dual sourcing setup where the capacity of the secondary supplier is modeled with a discrete-time Markov chain. Transition probability matrices are used to realize the dependency of the secondary market's capacity on different phases of products' life cycle. Also, a dynamic acquisition cost function is proposed for secondary markets to represent dynamic pricing policies. An assumption of stationary demand is made initially then this assumption is extended to non-stationary demand. Under both assumptions, the optimal control policy is derived when the lead time of OEM is one, and the acquisition cost function of the secondary supplier is linear. Heuristics policies are employed since the optimum policy could not be characterized mathematically for a general lead time.

To be able to implement these policies under non-stationary demand, a simulation-based gradient search is performed on the policy parameters. The dual index policy outperformed the tailored base-surge and capped dual index policies by resulting in slight deviations from the optimum cost in both stationary and non-stationary demand cases.

To reflect the dynamics of installed bases on spare parts demand, Hekimoğlu and Karlı (2021)'s non-stationary demand model is utilized throughout the thesis. They separately derive mathematical models for growing and declining installed bases by assuming homogeneous Poisson processes for product installments/retirements and spare part failures. Moreover, they present the closed-form formulas of the first three moments and choose probability distributions based on these moments using Ord (1967)'s and Adan, Van Eenige and Resing (1995)'s distribution selection algorithms. Hekimoğlu and Karlı (2021)'s model is comprehensively tested throughout the thesis. In multiple steps, performances of Ord's and Adan's algorithms are tested through a simulation study along with a pure Poisson distribution assumption over the whole planning horizon. This simulation study aims to test the value of greater degrees moments since Adan's and Ord's algorithms utilize the first two and three moments while Poisson's sole parameter is estimated by the first moment. As a result of the experiments, it is found that Adan's and Ord's algorithms beat pure Poisson distribution leading to lower costs. Although cost reductions led by the information of the first two and three moments are pretty close, the former make slightly more improvements for the decline phase while the same situation is valid for the latter in the growth phase. The same simulation study is performed by introducing an estimation error to the first moment, which is highly likely to happen in real life. The results showed that the information of higher moments could save costs up to 14.2% and 9.26% for growing and declining installed bases, respectively. As in the first simulation study, information of the first two and three moments showed better performances for growth and decline phases, respectively.

Apart from the theoretical work, an empirical study is conducted on the data set of a company from the Turkish automotive industry to test Hekimoğlu and Karlı (2021)'s non-stationary demand model. One of the top seller automobile models of Turkey is analyzed. First, statistical tests are conducted on sales data. KS tests are implemented to test the null hypothesis that interarrival times of the sales process are distributed exponentially. All the tests performed on numerous periodic and successive time blocks rejected the null hypothesis; hence it is concluded that the inter-sales times do not follow an exponential distribution. Next, Brown et al. (2005)'s test procedure is utilized to test the null hypothesis that the sales process is a homogenous Poisson process. The test is implemented for the entire period initially, and the null hypothesis is rejected, then proceeded to the consideration of smaller consecutive intervals. A meager proportion of the tests resulted in acceptance; hence it is concluded that the sales process does not follow a homogenous Poisson process. Finally, the test procedure presented by Böhning (1994) is performed on the arrival process of vehicle sales to test the null hypothesis that arrivals of automobile sales follow a Poisson distribution. Firstly, Böhning's test is implemented for periodic time blocks, and all tests are rejected the null hypothesis. Then, successive periods are tested, and it is found that shorter intervals produce higher acceptance rates. Consequently, it is concluded that biweekly or monthly time intervals should be used to model arrival process automobiles of automobile sales. Furthermore, the years 2002 and 2003 are found to have a high acceptance rate. A closer look at this interval revealed that arrivals are way more steady than the other years, hence resulting in a coefficient of dispersion close to one. Therefore, the automobile sales process could be modeled using a Poisson process during 2002 and 2003.

Finally, the chi-square goodness of fit test is implemented on the repair demand to test to what extent the probability distributions chosen by utilizing Hekimoğlu and Karlı (2021)'s non-stationary demand model for growing installed bases fit empirical data. The spare part demand in 2018, 2019 and 2020, driven by vehicles sold in the same period, is used to obtain a growing installed base. The in-depth data analyses imply that growing installed base assumption hold for this approach. Moreover, chi-squared tests resulted in up to 55.56% acceptance rate over 36 groups consist of 30 days intervals over the analysis period. The spare parts with low acceptance rates are mostly found to be parts that are used for periodic maintenance. Besides, Ord's distribution selection algorithm showed superiority over Adan's algorithm in parallel with the theoretical results. Thus, it is concluded that Hekimoğlu and Karlı (2021)'s demand model could be used to model spare parts repair demand of automobiles in the growth phase employing the Ord's distribution selection algorithm, although their Poisson process assumption on the sales vehicle process is mainly rejected.

APPENDIX A: OPTIMIZATION AND SIMULATION CODES

The optimization and simulation codes developed using R and RCPP (integration of R and C++) for inventory control model with non-stationary demand distribution are given in the following subsections.

A.1 Optimization Codes

The code block given below is developed using R to control code runs and record logs.

```
setwd("C:\\Users\\suser\\Desktop\\OrdvsAdan\\Declining_IB")
getwd()

parameters=read.csv("optimization_parameters.csv")

Declining_IB=T
capacity=F #if false capacity of secondary supplier=0(like single
   sourcing problem)

strategy=1
stepsize=1
market_sc=1 #not used when capacity=F
N0=25 #for Declining_IB(# of capital products at the beginning)

source("DynamicDistSelect_Optimization_Ord&Adan.R")
if(Declining_IB)
{
  source("DynamicDistSelect_DecliningIB_CalcStats_Ord&Adan.R")
}else
{
  source("DynamicDistSelect_GrowingIB_CalcStats_Ord&Adan.R")
}
library(Rcpp)
sourceCpp("RowMatchAndCostCalc.cpp")


for(dist_sel_alg in c("Ord","Adan"))
{
for (i in c(1:100))
{
  param_num=i
```

```
  if(Declining_IB)
  {
    params=sprintf("DecliningIB dist_sel_alg:%s lambda:%s alpha:%s holding:%s backlog:%s eta:%s xi:%s t:%s disc_fact:%s LT:%s Stepsize:%s Strategy:%s Capacity:%s",dist_sel_alg,parameters[i,1],parameters[i,2],parameters[i,3],parameters[i,4],parameters[i,5],parameters[i,6],parameters[i,7],parameters[i,8],parameters[i,9],stepsize,strategy,capacity)
    logname=sprintf("C:\\Users\\suser\\Desktop\\OrdvsAdan\\Declining_IB\\OptimizationResultFiles\\Log_%s_DecliningIB_Cap(%s)_Strtgy(%s)_%s.txt",i,capacity,strategy,dist_sel_alg)
  }else
  {
    params=sprintf("GrowingIB dist_sel_alg:%s lambda:%s alpha:%s holding:%s backlog:%s eta:%s xi:%s t:%s disc_fact:%s LT:%s Stepsize:%s Strategy:%s Capacity:%s",dist_sel_alg,parameters[i,1],parameters[i,2],parameters[i,3],parameters[i,4],parameters[i,5],parameters[i,6],parameters[i,7],parameters[i,8],parameters[i,9],stepsize,strategy,capacity)
    logname=sprintf("C:\\Users\\suser\\Desktop\\OrdvsAdan\\Growing_IB\\OptimizationResultFiles\\Log_%s_GrowingIB_Cap(%s)_Strtgy(%s)_%s.txt",i,capacity,strategy,dist_sel_alg)
  }
  start=timestamp()
  start=rbind(params,sprintf("Start time: %s",start))
  write.table(start,file=logname, col.names=FALSE, row.names=FALSE)
  optimal(parameters[i,1],parameters[i,2],parameters[i,3],parameters[i,4],parameters[i,5],parameters[i,6],parameters[i,7],parameters[i,8],parameters[i,9],param_num,market_sc,stepsize,strategy,capacity,Declining_IB,N0,dist_sel_alg)
  finish=timestamp()
  finish=sprintf("End time: %s",finish)
  write.table(finish, file=logname, col.names=FALSE, row.names=FALSE, append=TRUE)
}
}
```

The following code block is developed using R to perform optimization.

```
#Name of this file is: DynamicDistSelect_Optimization_Ord&Adan.R
optimal=function(lambd,alph,hold,back,eta,xi,plan_hor,disc_fac,leadtime,param_num,market_scenario,stepsize,strategy,capacity,Declining_IB,N0,dist_sel_alg)
{
  lambda=lambd #parameter of poisson process that represents capital product sales
  alpha=alph #parameter of poisson process that represents
  t=plan_hor

  if(Declining_IB)
  {
    list=DynamicDistStats(lambda,alpha,plan_hor,stepsize,strategy,N0,dist_sel_alg)
  }else
  {
    list=DynamicDistStats(lambda,alpha,plan_hor,stepsize,strategy,dist_sel_alg)
  }

```

```
dmax=list$dmaxs
demandmatrix=list$prob
distributions=list$dists

if(capacity)
{
  q_s_max=12 #4 #max order size for secondary supp.
  K_prime_max=12 #4 #same as q_s_max
  if (market_scenario==1)
  {
    P=as.matrix(read.csv("Keq12_Symmetric_TPM.csv",sep = ";", header = F))
  }else if(market_scenario==2)
  {
    P=as.matrix(read.csv("Keq12_Increasing_TPM.csv",sep = ";", header = F))
  }else if(market_scenario==3)
  {
    P=as.matrix(read.csv("Keq12_Decreasing_TPM.csv",sep = ";", header = F))
  }
}else if(capacity==F)
{
  q_s_max=0 #4 #max order size for secondary supp.
  K_prime_max=0 #4 #same as q_s_max
  P=matrix(1,1,1)
}

Xmin=0 #min inv. level
Xmax=0 #max inv. level
q_r_max=0
q_r_max[1]=floor(dmax[1]*(1.5))
c_r=5 #price of regular
h=hold#H*c_r #unit holding cost
b=back#h*SL/(1-SL) #unit backlog cost

row_size=(q_s_max+1)
mininv=0
maxinv=0

for (aa in 2:leadtime) ##aa is period
{
  if (aa==2)
  {
    Xmint=Xmin-dmax[aa-1]
    Xmaxt=Xmax+q_s_max
    mininv[aa]=Xmint
    maxinv[aa]=Xmaxt
    q_r_max[aa]=floor(dmax[aa]*(1.5))
    row_size[aa]=(-Xmint+Xmaxt+1)*(q_s_max+1)*(q_r_max[1]+1)
  }
  if (aa==3)
  {
    Xmint=Xmint-dmax[aa-1]
    Xmaxt=Xmaxt+q_s_max
    mininv[aa]=Xmint
    maxinv[aa]=Xmaxt
    q_r_max[aa]=floor(dmax[aa]*(1.5))
    row_size[aa]=(-Xmint+Xmaxt+1)*(q_s_max+1)*(q_r_max[1]+1)*(q_r_max[2]+1)
```

```
    }
    if (aa==4)
    {
      Xmint=Xmint-dmax[aa-1]
      Xmaxt=Xmaxt+q_s_max
      mininv[aa]=Xmint
      maxinv[aa]=Xmaxt
      q_r_max[aa]=floor(dmax[aa]*(1.5))
      row_size[aa]=(-Xmint+Xmaxt+1)*(q_s_max+1)*(q_r_max[1]+1)*(q_r
   _max[2]+1)*(q_r_max[3]+1)
    }
    if (aa==5)
    {
      Xmint=Xmint-dmax[aa-1]
      Xmaxt=Xmaxt+q_s_max
      mininv[aa]=Xmint
      maxinv[aa]=Xmaxt
      q_r_max[aa]=floor(dmax[aa]*(1.5))
      row_size[aa]=(-Xmint+Xmaxt+1)*(q_s_max+1)*(q_r_max[1]+1)*(q_r
   _max[2]+1)*(q_r_max[3]+1)*(q_r_max[3]+1)
    }
  }

  if (leadtime==1)
  {
    Xmint=0
    Xmaxt=0
    mininv=0
    maxinv=0
    q_r_max=floor(dmax[1]*(1.5))
    row_size=(q_s_max+1)
  }

  for (bb in (leadtime+1):t) ##bb is period
  {
    if (leadtime==1)
    {
      Xmint=Xmint-dmax[bb-1]
      Xmaxt=Xmaxt+q_s_max+q_r_max[bb-leadtime]
      mininv[bb]=Xmint
      maxinv[bb]=Xmaxt
      q_r_max[bb]=floor(dmax[bb]*(1.5))
      row_size[bb]=(-Xmint+Xmaxt+1)*(q_s_max+1)
    }
    if (leadtime==2)
    {
      Xmint=Xmint-dmax[bb-1]
      Xmaxt=Xmaxt+q_s_max+q_r_max[bb-leadtime]
      mininv[bb]=Xmint
      maxinv[bb]=Xmaxt
      q_r_max[bb]=floor(dmax[bb]*(1.5))
      row_size[bb]=(-Xmint+Xmaxt+1)*(q_s_max+1)*c(q_r_max[bb-1]+1)
    }
    if (leadtime==3)
    {
      Xmint=Xmint-dmax[bb-1]
      Xmaxt=Xmaxt+q_s_max+q_r_max[bb-leadtime]
      mininv[bb]=Xmint
      maxinv[bb]=Xmaxt
      q_r_max[bb]=floor(dmax[bb]*(1.5))
```

```
      row_size[bb]=(-Xmint+Xmaxt+1)*(q_s_max+1)*c(q_r_max[bb-2]+1)*c(q_r_max[bb-1]+1)
    }
    if (leadtime==4)
    {
      Xmint=Xmint-dmax[bb-1]
      Xmaxt=Xmaxt+q_s_max+q_r_max[bb-leadtime]
      mininv[bb]=Xmint
      maxinv[bb]=Xmaxt
      q_r_max[bb]=floor(dmax[bb]*(1.5))
      row_size[bb]=(-Xmint+Xmaxt+1)*(q_s_max+1)*c(q_r_max[bb-3]+1)*c(q_r_max[bb-2]+1)*c(q_r_max[bb-1]+1)
    }
    if (leadtime==5)
    {
      Xmint=Xmint-dmax[bb-1]
      Xmaxt=Xmaxt+q_s_max+q_r_max[bb-leadtime]
      mininv[bb]=Xmint
      maxinv[bb]=Xmaxt
      q_r_max[bb]=floor(dmax[bb]*(1.5))
      row_size[bb]=(-Xmint+Xmaxt+1)*(q_s_max+1)*c(q_r_max[bb-4]+1)*c(q_r_max[bb-3]+1)*c(q_r_max[bb-2]+1)*c(q_r_max[bb-1]+1)
    }
  }

  state_number=cumsum(row_size)

  if (leadtime==1)
  {
    maxpipe=0
  }else
  {
    maxpipe=matrix(0, nrow=t, ncol=(leadtime-1))
    for (l in 2:leadtime)
    {
      if (l==2)
      {
        maxpipe[1,leadtime-1]=q_r_max[1]
      }
      if (l==3)
      {
        maxpipe[1,leadtime-2]=q_r_max[1]
        maxpipe[1,leadtime-1]=q_r_max[2]
      }
      if (l==4)
      {
        maxpipe[1,leadtime-3]=q_r_max[1]
        maxpipe[1,leadtime-2]=q_r_max[2]
        maxpipe[1,leadtime-1]=q_r_max[3]
      }
      if (l==5)
      {
        maxpipe[1,leadtime-4]=q_r_max[1]
        maxpipe[1,leadtime-3]=q_r_max[2]
        maxpipe[1,leadtime-2]=q_r_max[3]
        maxpipe[1,leadtime-1]=q_r_max[4]
      }
    }

    transpose.qrmax=as.matrix(q_r_max)
```

```
    maxpipe=cbind(maxpipe,transpose.qrmax)

    for (m in (leadtime+1):t)
    {
      for (n in 1:(leadtime-1))
      {
        maxpipe[m,n]=maxpipe[m-1,n+1]
      }
    }
    maxpipe=maxpipe[,-leadtime]
  }

  generate.stspace=function(t,lt,mininv,maxinv,maxpipe)
  {
    if (lt==1)
    {
      f=expand.grid(t,c(mininv[t]:maxinv[t]),c(0:q_s_max),0,0,0)#last three columns are: optqr,optqs,optcost
    }
    if (lt==2)
    {
      f=expand.grid(t,c(mininv[t]:maxinv[t]),c(0:q_s_max),c(0:maxpipe[t]),0,0,0)#last three columns are: optqr,optqs,optcost
    }
    if (lt==3)
    {
      f=expand.grid(t,c(mininv[t]:maxinv[t]),c(0:q_s_max),c(0:maxpipe[t,1]),c(0:maxpipe[t,2]),0,0,0)
    }
    if (lt==4)
    {
      f=expand.grid(t,c(mininv[t]:maxinv[t]),c(0:q_s_max),c(0:maxpipe[t,1]),c(0:maxpipe[t,2]),c(0:maxpipe[t,3]),0,0,0)
    }
    if (lt==5)
    {
      f=expand.grid(t,c(mininv[t]:maxinv[t]),c(0:q_s_max),c(0:maxpipe[t,1]),c(0:maxpipe[t,2]),c(0:maxpipe[t,3]),c(0:maxpipe[t,4]),0,0,0)
    }
    return(f)
  }

  f_prev=generate.stspace(t,leadtime,mininv,maxinv,maxpipe)
  ######### expected cost for period T-1 #########
  f_prev=as.data.frame(calc_fprev(t, row_size, as.matrix(f_prev), q_r_max, dmax, c_r, eta, xi, h, b, demandmatrix, leadtime))
  t=t-1
  delete_index=c(-c(1:(3+leadtime-1)),-(3+leadtime-1+3))
  row_ind=nrow(f_prev)
  if(Declining_IB)
  {
    filename=sprintf("C:\\Users\\suser\\Desktop\\OrdvsAdan\\Declining_IB\\OptimizationResultFiles\\Result_%s_DecliningIB_Cap(%s)_Strtgy(%s)_%s.csv",param_num,capacity,strategy,dist_sel_alg)
  }else
  {
    filename=sprintf("C:\\Users\\suser\\Desktop\\OrdvsAdan\\Growing_IB\\OptimizationResultFiles\\Result_%s_GrowingIB_Cap(%s)_Strtgy
```

```
(%s)_%s.csv",param_num,capacity,strategy,dist_sel_alg)
}
write.table(cbind(c(1:row_ind),f_prev[delete_index]), file=filename, sep = ",", col.names=FALSE, row.names=FALSE)

######### expected cost for period t #########
while (t >= 1)
{
  f=generate.stspace(t,leadtime,mininv,maxinv,maxpipe)
  f=costfunc(as.matrix(f),t,c_r,eta,xi,h,b,dmax,K_prime_max,as.matrix(f_prev),as.matrix(P),demandmatrix, mininv, maxinv, disc_fac, leadtime, as.matrix(maxpipe), row_size, q_r_max)
  f_prev=f
  if (t==1)
  {
    del_index=-c(1:(3+leadtime-1+2))
    filename2="All_Results.txt"
    if(Declining_IB)
    {
      if(capacity)
      {
        costs=cbind(sprintf("%s_DecliningIB_Cap(%s)_Strtgy(%s)_%s",param_num,capacity,strategy,dist_sel_alg),f[1,del_index],f[2,del_index],f[3,del_index],f[4,del_index],f[5,del_index],f[6,del_index],f[7,del_index],f[8,del_index],f[9,del_index],f[10,del_index],f[11,del_index],f[12,del_index],f[13,del_index])
      }else
      {
        costs=cbind(sprintf("%s_DecliningIB_Cap(%s)_Strtgy(%s)_%s",param_num,capacity,strategy,dist_sel_alg),f[1,del_index])
      }
    }else
    {
      if(capacity)
      {
        costs=cbind(sprintf("%s_GrowingIB_Cap(%s)_Strtgy(%s)_%s",param_num,capacity,strategy,dist_sel_alg),f[1,del_index],f[2,del_index],f[3,del_index],f[4,del_index],f[5,del_index],f[6,del_index],f[7,del_index],f[8,del_index],f[9,del_index],f[10,del_index],f[11,del_index],f[12,del_index],f[13,del_index])
      }else
      {
        costs=cbind(sprintf("%s_GrowingIB_Cap(%s)_Strtgy(%s)_%s",param_num,capacity,strategy,dist_sel_alg),f[1,del_index])
      }
    }
    #colnames(costs)=c("param num", "cost of K=0","cost of K=1","cost of K=2")
    write.table(costs, file=filename2, sep = "\t", col.names=F, row.names=F, append=TRUE)
  }
  if(capacity==FALSE&t==1)
  {
    f=as.data.frame(matrix(f[,delete_index],1,2))
  }else
  {
    f=f[,delete_index]
  }
  f=cbind(c((row_ind+1):(row_ind+nrow(f))),f)
  row_ind=row_ind+nrow(f)
```

```
    write.table(f, file=filename, sep = ",", col.names=FALSE, row.
    names=FALSE, append=TRUE)
    print(t)
    t=t-1
  }
}
```

The code blocks given below are developed using R to calculate moments, choose probability distributions and calculate the statistics for declining and growing installed bases respectively.

```
#Name of this file is:DynamicDistSelect_DecliningIB_CalcStats_Ord&
    Adan.R
library(rmutil) #for betabinom func.
#library(extraDistr) #for bnbinom func.
library(brr) #for beta_nbinom func.(bnbinom doesn't have a quantile
     funct.)
source("ParamEstFuncs_v3.R")

DynamicDistStats=function(lambd,alph,plan_hor,stepsize,strategy,N0,
    dist_sel_alg)
{
  lambda = lambd
  alpha = alph
  step_size = stepsize
  tmax = plan_hor
  N0=N0
  tvect=c(1:(tmax*step_size))/step_size

  ########  PARAMETERS  ########
  alphat=alpha*tvect;
  lambdat=lambda*tvect

  h=1/step_size
  alphah=alpha*h
  lambdah=lambda*h

  ########  FUNCTIONS  ########
  qt2<- function(x,lambda,t)
  {
    lambdat=t*lambda
    ppois(x,lambdat)
  }

  beta2<- function(k,N0)
  {
    res=0;
    kvect=c(0:k)
    1/(k+1)*prod(N0+kvect)
  }

  ########  PARAMTERS  ########
  alphat=alpha*tvect;
  lambdat=lambda*tvect

  ########  NON-CENTRAL CUMULATIVE MOMENTS  ########
  #FIRST MOMENT
```

```
NonCentralMom1<-function(a,L,tvector,N)
{
  at=a*tvector;
  Lt=L*tvector
  at*(N*qt2(N,L,tvector)-(1/2)*Lt*qt2(N-1,L,tvector))+(a/L)*beta2(1,N)*(1-qt2(N+1,L,tvector))
}

#SECOND MOMENT
NonCentralMom2<-function(a,L,tvector,N)
{
  at=a*tvector;
  Lt=L*tvector
  Lt* qt2(N-1,L,tvector)*(at^2*((1/12)-N)-0.5*at) + (Lt^2* qt2(N-2,L,tvector)+Lt* qt2(N-1,L,tvector))*0.25*at^2 + qt2(N,L,tvector)*(N*at+N^2*at^2) + (a/L)*beta2(1,N)*(1-qt2(N+1,L,tvector)) + 0.25* (a/L)^2*beta2(2,N)*(3*N+1)*(1-qt2(N+2,L,tvector))
}

#THIRD MOMENT
NonCentralMom3<-function(a,L,tvector,N)
{
  at=a*tvector;
  Lt=L*tvector
  term1=-1/8*at^3*(Lt^3 * qt2(N-3,L,tvector) +3*Lt^2 * qt2(N-2,L,tvector) +Lt * qt2(N-1,L,tvector) )
  term2=(1/8)*at^2*( at*(6*N-1)+6)*(Lt^2 * qt2(N-2,L,tvector) +Lt * qt2(N-1,L,tvector)  )
  term3=-(1/4)*at*(at^2*N*(6*N-1)-at*(1-12*N)+2)*Lt * qt2(N-1,L,tvector)
  term4=(at^3*(N^3)+at^2*(3*N^2)+N*at)*qt2(N,L,tvector)
  term5=(at/Lt)*beta2(1,N)*(1-qt2(N+1,L,tvector))
  + (3/4)*(at/Lt)^2*beta2(2,N)*(3*N+1)*(1-qt2(N+2,L,tvector))
  + (at/Lt)^3*beta2(1,N)*beta2(3,N)*(1-qt2(N+3,L,tvector))

  term1+term2+term3+term4+term5
}

########  CENTRAL CUMULATIVE MOMENTS  ########
centralmoment1=NonCentralMom1(alpha,lambda,tvect,N0)
centralmoment2=NonCentralMom2(alpha,lambda,tvect,N0) - (NonCentralMom1(alpha,lambda,tvect,N0))^2
centralmoment3=(NonCentralMom3(alpha,lambda,tvect,N0) -3*centralmoment1*centralmoment2 - centralmoment1^3)
##############################################

########  CENTRAL MARGINAL MOMENTS  ########
########  PARAMETERS  ########
h=1/step_size
alphah=alpha*h
lambdah=lambda*h
epsilon=0.1;
epsilon2=0.05

NonCM1=0 #FIRST NON-CENTRAL MOMENT OF MARGINAL
NonCM2=0 #SECOND NON-CENTRAL MOMENT OF MARGINAL
NonCM3=0 #THIRD NON-CENTRAL MOMENT OF MARGINAL

for (n in 0:N0){
  NonCM1=NonCM1+NonCentralMom1(alpha,lambda,h,N0-n)*dpois(n,
```

```
  lambda*tvect)
   NonCM2=NonCM2+NonCentralMom2(alpha,lambda,h,N0-n)*dpois(n,
  lambda*tvect)
   NonCM3=NonCM3+NonCentralMom3(alpha,lambda,h,N0-n)*dpois(n,
  lambda*tvect)
}

#FIRST CENTRAL MOMENT OF MARGINAL
Dmom1=NonCM1

#SECOND CENTRAL MOMENT OF MARGINAL
Dmom2=NonCM2-NonCM1^2

#THIRD CENTRAL MOMENT OF MARGINAL
Dmom3=(NonCM3 -3*NonCM1*Dmom2 - NonCM1^3)

#CV MARGINAL
DmomCV=Dmom2/Dmom1

#ZERO PROBABILITY
part1.1=lambda*exp(alphah)*tvect+lambda/alpha*(exp(alphah)-1);
part1.2=(lambdat*exp(alphah))^{N0}/factorial(N0);

D.Zero.Prob=((1-qt2(N0-1,lambda,tvect)) + exp(-N0*alphah-lambdat-
  lambdah)*(exp(part1.1+log(ppois(N0,part1.1)))-part1.2))

param.Pois = rep(0,length(Dmom3))
param1 = rep(0,length(Dmom3)) #parameter of NegBin
param2 = rep(0,length(Dmom3)) #parameter of NegBin

#epsilon=0.001;
dist.vect=0

S_Index.vect<- 0;
I_Index.vect<- 0;

dmaxs = rep(NA,length(Dmom3))
dmax1 = rep(NA,length(Dmom3))
dmax2 = rep(NA,length(Dmom3))
probabilities = matrix(NA,length(Dmom3),100)

if (dist_sel_alg=="Ord")
{
  print("Ord")
  for(v in 1:tmax)
  {
    S_Index = Dmom3[v] / Dmom2[v];
    I_Index = Dmom2[v] / Dmom1[v];

    S_Index.vect[v] = S_Index;
    I_Index.vect[v] = I_Index;

    #Poisson Dist.
    cond1 = ((abs(S_Index-1) + abs(I_Index-1))<epsilon2)
    if(cond1)
    {
      dist.vect[v]="Pois"
      param.Pois[v] = Dmom1[v];

      dmaxs[v]=qpois(1-10^-4,param.Pois[v])
```

```
      probabilities[v,c(1:(dmaxs[v]+1))]=dpois((0:dmaxs[v]),param.Pois[v])
    }

    #Negative Binomial Dist.
    cond2 = (abs(S_Index - 2*I_Index+1)<epsilon)
    if(!cond1 & cond2)
    {
      dist.vect[v]="NBinom"
      var_v = Dmom2[v]
      param2[v] = Dmom1[v] / var_v;  ##prob
      param1[v] = Dmom1[v] * param2[v] / (1-param2[v]);

      if(!is.finite(param1[v]))
        param1[v]=10000;

      dmaxs[v]=qnbinom(1-10^-4, size = param1[v], prob=param2[v])

      probabilities[v,c(1:(dmaxs[v]+1))] = dnbinom((0:dmaxs[v]), size = param1[v], prob=param2[v])
    }

    ###BETA BINOMIAL DISTRIBUTION!!
    cond3 = (S_Index - 2*I_Index+1 < (-epsilon))
    if((!cond1)&(!cond2) & cond3)
    {
      dist.vect[v]="Beta_Binom";
      Param.BetaBinom = beta.binom.parameterv4_2(Dmom1[v],Dmom2[v],Dmom3[v],D.Zero.Prob[v]);

      dmaxs[v]=qbetabinom(1-10^-4, size=Param.BetaBinom$size.est, m=Param.BetaBinom$m.est, s=Param.BetaBinom$s.est)#size,m,s

      probabilities[v,c(1:(dmaxs[v]+1))] = dbetabinom((0:dmaxs[v]), size=Param.BetaBinom$size.est, m=Param.BetaBinom$m.est, s=Param.BetaBinom$s.est)
    }

    ###BETA PASCAL DISTRIBUTION!!
    cond4 = (S_Index - 2*I_Index+1 > epsilon)
    if((!cond1)&(!cond2) & cond4)
    {
      dist.vect[v]="Beta_Pascal"

      Param.BetaPascal <- beta.pascal.parameter(Dmom1[v],Dmom2[v],Dmom3[v])
      if(Param.BetaPascal$r.apx<0)
      {
        Param.BetaPascal$r.apx=NA;Param.BetaPascal$b.apx=NA;
      }

      dmaxs[v]=qbeta_nbinom(1-10^-4, size=Param.BetaPascal$r.apx, alpha=Param.BetaPascal$a, beta=Param.BetaPascal$b.apx)

      probabilities[v,c(1:(dmaxs[v]+1))] = dbeta_nbinom((0:dmaxs[v]), size=Param.BetaPascal$r.apx, alpha=Param.BetaPascal$a, beta=Param.BetaPascal$b.apx)
    }
    if(cond1 + (!cond1 & cond2) + ((!cond1)&(!cond2) & cond3) +
```

```
    ((!cond1)&(!cond2) & cond4) >1)
      {
        print("Multiple True Conditions!");break();
      }
    }
  }else if(dist_sel_alg=="Adan")
  {
    print("Adan")
    for(v in 1:length(Dmom3))
    {
      cx.vect=0
      a.vect=0

      cx=sqrt(Dmom2[v])/Dmom1[v]
      a=cx^2-1/Dmom1[v]

      cx.vect[v] = cx;
      a.vect[v] = a;

      #Binomial Dist.
      if((a<0) & (a>=-1))
      {
        dist.vect[v]="Binom"

        k=0
        while(-1/(k+1)<a)
          k=k+1

        k.star=k

        q=(1+a*(1+k.star)+sqrt(-a*k.star*(1+k.star)-k.star))/(1+a)

        p=Dmom1[v]/(k.star+1-q)

        dmax1[v]=qbinom(1-10^-4,k.star,p)
        dmax2[v]=qbinom(1-10^-4,k.star+1,p)
        dmaxs[v]=max(dmax1[v],dmax2[v])

        probabilities[v,c(1:(dmaxs[v]+1))]=dbinom(0:dmaxs[v],k.star
    ,p)*q + (1-q)*dbinom(0:dmaxs[v],k.star+1,p)
      }

      #Poisson Dist.
      if(a==0)
      {
        dist.vect[v]="Pois";
        param.Pois[v] = Dmom1[v];

        dmaxs[v]=qpois(1-10^-4,param.Pois[v])

        probabilities[v,c(1:(dmaxs[v]+1))]=dpois((0:dmaxs[v]),param
    .Pois[v])
      }

      #Negative Binomial Dist.
      if((a<1) & (a>0))
      {
        dist.vect[v]="NBinom"
        #print(sprintf("Calculating k. v=%s",v))

```

```
        k= ceiling(1/a)*10
        while((1/k)<a)
          k=k-1
        #print(sprintf("Calculated k. v=%s",v))

        k.star<- k

        q<- ((1+k.star)*a - sqrt((1+k.star)*(1-a*k.star)))/(1+a)
        p<- Dmom1[v]/(k.star+1-q+Dmom1[v])

        dmax1[v]=qnbinom(1-10^-4, size = k.star, prob = 1-p)
        dmax2[v]=qnbinom(1-10^-4, size = k.star+1, prob = 1-p)
        dmaxs[v]=max(dmax1[v],dmax2[v])

        probabilities[v,c(1:(dmaxs[v]+1))] = dnbinom(0:dmaxs[v],k.star,1-p) * q + (1-q) * dnbinom(0:dmaxs[v],k.star+1,1-p)
      }

      #Geometric Dist.
      if(a>=1)
      {
        dist.vect[v]="Geom"

        r1=(1+a + sqrt(a^2-1));
        r2=  (1+a - sqrt(a^2-1));
        p1= Dmom1[v] * r1 / (2+Dmom1[v]*r1)
        p2= Dmom1[v] * r2 / (2+Dmom1[v]*r2)
        q1= 1 / r1
        q2= 1 / r2

        dmax1[v]=qgeom(1-10^-4,1-p1)
        dmax2[v]=qgeom(1-10^-4,1-p2)
        dmaxs[v]=max(dmax1[v],dmax2[v])

        probabilities[v,c(1:(dmaxs[v]+1))] = dgeom(0:dmaxs[v],1-p1) * q1 + q2 * dgeom(0:dmaxs[v],1-p2)
      }
    }
  }
  return(list(dmaxs=dmaxs, prob=probabilities, dists=dist.vect))
}
```

```
#Name of this file is:DynamicDistSelect_DecliningIB_CalcStats_Ord&Adan.R
source("ParamEstFuncs_v3.R")
#install.packages("rmutil")
#install.packages("extraDistr")
#install.packages("brr")
library(rmutil) #for betabinom func.
#library(extraDistr) #for bnbinom func.
library(brr) #for beta_nbinom func.(bnbinom doesn't have a quantile funct.)

DynamicDistStats=function(lambd,alph,plan_hor,stepsize,strategy,dist_sel_alg)
{
  ########  PARAMETERS  ########
  lambda=lambd#parameter of poisson process that represents capital product sales
  alpha=alph#parameter of poisson process that represents part failures
```

```
tmax=plan_hor #total time period
t=plan_hor
step_size=stepsize
tvect=c(1:(tmax*step_size))/step_size
alphat=alpha*tvect;
lambdat=lambda*tvect

h=1/step_size
alphah=alpha*h
lambdah=lambda*h

######## h - CUMULATIVE MOMENTS ########
#FIRST MOMENT
moment1_h = alphah*(0.5*lambdah+1)

#SECOND MOMENT
moment2_h = alphah^2 * (0.25*lambdah^2 + 4/3*lambdah+1) + moment1
  _h

#THIRD MOMENT
moment3_h = alphah^3*(1/8*lambdah^3 + 5/4*lambdah^2+11/4*lambdah
  +1) +
   3*moment2_h -2*moment1_h

######## NON-CENTRAL CUMULATIVE MOMENTS ########
#FIRST MOMENT
Dmoment1_noncentral = alphah*(lambdat+0.5*lambdah+1)

#SECOND MOMENT
Dmoment2_noncentral = alphah^2*(lambdat^2+lambdat)+alphah*lambdat
  +alphah^2 * (0.25*lambdah^2 + 4/3*lambdah+1) + alphah*(0.5*
  lambdah+1)+2*lambdat*alphah*alphah*(0.5*lambdah+1)

#THIRD MOMENT
Dmoment3_noncentral = alphah^3*(lambdat^3+3*lambdat^2+lambdat)+3*
  alphah^2*(lambdat^2+lambdat)+alphah*lambdat+3*moment1_h*(alphah
  ^2*(lambdat^2+lambdat)+alphah*lambdat)+3*moment2_h*alphah*
  lambdat+moment3_h

######## CENTRAL CUMULATIVE MOMENTS ########
#FIRST MOMENT
Dmom1=Dmoment1_noncentral

#SECOND MOMENT
Dmom2=Dmoment2_noncentral-Dmoment1_noncentral^2

#THIRD MOMENT
Dmom3=Dmoment3_noncentral - 3*Dmoment1_noncentral*Dmoment2_
  noncentral + 2*Dmoment1_noncentral^3

#CV MARGINAL
DmomCV=Dmom2/Dmom1

#ZERO PROBABILITY
P0 = exp(lambdat*(exp(-alphah)-1))* exp(-h*(lambda+alpha))
D.Zero.Prob = P0*exp(-lambda/alpha*(exp(-alphah)-1))

param.Pois = rep(NA,length(Dmom3))
Param.NegBinom = matrix(NA,length(Dmom3),2)
Param.BetaBinom = matrix(NA,length(Dmom3),3)
```

```
Param.BetaPascal = matrix(NA,length(Dmom3),3)
dmaxs = rep(NA,length(Dmom3))
dmax1 = rep(NA,length(Dmom3))
dmax2 = rep(NA,length(Dmom3))
probabilities = matrix(NA,length(Dmom3),100)
dist.vect=0

if (dist_sel_alg=="Ord")
{
  print("Ord")
  epsilon=0.1;
  epsilon2=0.05

  S_Index.vect<- 0;
  I_Index.vect<- 0;

  for(v in 1:length(Dmom3))
  {
    S_Index = Dmom3[v] / Dmom2[v];
    I_Index = Dmom2[v] / Dmom1[v];

    S_Index.vect[v] = S_Index;
    I_Index.vect[v] = I_Index;

    #Poisson Dist.
    cond1 = ((abs(S_Index-1) + abs(I_Index-1))<epsilon2)
    if(cond1)
    {
      dist.vect[v]="Pois";
      param.Pois[v] = Dmom1[v];

      dmaxs[v]=qpois(1-10^-4,param.Pois[v])

      probabilities[v,c(1:(dmaxs[v]+1))]=dpois((0:dmaxs[v]),param
.Pois[v])
    }

    #Negative Binomial Dist.
    cond2 = (abs(S_Index - 2*I_Index+1)<epsilon)
    if(!cond1 & cond2)
    {
      dist.vect[v]="NBinom"
      var_v = Dmom2[v]
      Param.NegBinom[v,1] = Dmom1[v] / var_v;  #prob(p)
      Param.NegBinom[v,2] = Dmom1[v] * Param.NegBinom[v,1] / (1-
Param.NegBinom[v,1]);#N

      dmaxs[v]=qnbinom(1-10^-4, size = Param.NegBinom[v,2], prob
= Param.NegBinom[v,1])

      probabilities[v,c(1:(dmaxs[v]+1))] = dnbinom((0:dmaxs[v]),
size = Param.NegBinom[v,2], prob = Param.NegBinom[v,1])
    }

    #BETA BINOMIAL DISTRIBUTION!!
    cond3 = (S_Index - 2*I_Index+1 < (-epsilon))
    if(!cond2 & cond3)
    {
      dist.vect[v]="Beta_Binom"
      params = beta.binom.parameterv4_4(Dmom1[v],Dmom2[v],Dmom3[v
```

```
],D.Zero.Prob[v]);
      Param.BetaBinom[v,c(1:3)]=c(params$size.est,params$m.est,params$s.est)#size,m,s

      dmaxs[v]=qbetabinom(1-10^-4, size=Param.BetaBinom[v,1],m=Param.BetaBinom[v,2],
                                    s=Param.BetaBinom[v,3])#size,m,s

      probabilities[v,c(1:(dmaxs[v]+1))] = dbetabinom((0:dmaxs[v]),size=Param.BetaBinom[v,1],m=Param.BetaBinom[v,2],
                                                                    s=Param.BetaBinom[v,3])
    }

    ###BETA PASCAL DISTRIBUTION!!
    cond4 = (S_Index - 2*I_Index+1 > epsilon)
    if(!cond2 & cond4)
    {
      dist.vect[v]="Beta_Pascal"
      params <- beta.pascal.parameter(Dmom1[v],Dmom2[v],Dmom3[v])

      if(params$r.apx<0)
      {
        params$r.apx=NA;params$b.apx=NA;
      }
      Param.BetaPascal[v,c(1:3)]=c(params$r.apx,params$a,params$b.apx)#r(size),a(alpha),b(beta)
      dmaxs[v]=qbeta_nbinom(1-10^-4, Param.BetaPascal[v,1], Param.BetaPascal[v,2],
                                      Param.BetaPascal[v,3])
      probabilities[v,c(1:(dmaxs[v]+1))] = dbeta_nbinom((0:dmaxs[v]),Param.BetaPascal[v,1], Param.BetaPascal[v,2],
                                                                      Param.BetaPascal[v,3])
    }

    if(cond1 + (!cond1 & cond2) + (!cond2 & cond3) + (!cond2 & cond4) >1)
    {
      print("Multiple True Conditions!");break();
    }
  }
}else if(dist_sel_alg=="Adan")
{
  print("Adan")
  for(v in 1:length(Dmom3))
  {
    cx.vect=0
    a.vect=0

    cx=sqrt(Dmom2[v])/Dmom1[v]
    a=cx^2-1/Dmom1[v]

    cx.vect[v] = cx;
    a.vect[v] = a;

    #Binomial Dist.
    if((a<0) & (a>=-1))
    {
      dist.vect[v]="Binom"
```

```

    k=0
    while(-1/(k+1)<a)
      k=k+1

    k.star=k

    q=(1+a*(1+k.star)+sqrt(-a*k.star*(1+k.star)-k.star))/(1+a)

    p=Dmom1[v]/(k.star+1-q)

    dmax1[v]=qbinom(1-10^-4,k.star,p)
    dmax2[v]=qbinom(1-10^-4,k.star+1,p)
    dmaxs[v]=max(dmax1[v],dmax2[v])

    probabilities[v,c(1:(dmaxs[v]+1))]=dbinom(0:dmaxs[v],k.star,p)*q + (1-q)*dbinom(0:dmaxs[v],k.star+1,p)
  }

  #Poisson Dist.
  if(a==0)
  {
    dist.vect[v]="Pois";
    param.Pois[v] = Dmom1[v];

    dmaxs[v]=qpois(1-10^-4,param.Pois[v])

    probabilities[v,c(1:(dmaxs[v]+1))]=dpois((0:dmaxs[v]),param.Pois[v])
  }

  #Negative Binomial Dist.
  if((a<1) & (a>0))
  {
    dist.vect[v]="NBinom"
    #print(sprintf("Calculating k. v=%s",v))

    k= ceiling(1/a)*10
    while((1/k)<a)
      k=k-1
    #print(sprintf("Calculated k. v=%s",v))

    k.star<- k

    q<- ((1+k.star)*a - sqrt((1+k.star)*(1-a*k.star)))/(1+a)
    p<- Dmom1[v]/(k.star+1-q+Dmom1[v])

    dmax1[v]=qnbinom(1-10^-4, size = k.star, prob = 1-p)
    dmax2[v]=qnbinom(1-10^-4, size = k.star+1, prob = 1-p)
    dmaxs[v]=max(dmax1[v],dmax2[v])

    probabilities[v,c(1:(dmaxs[v]+1))] = dnbinom(0:dmaxs[v],k.star,1-p) * q + (1-q) * dnbinom(0:dmaxs[v],k.star+1,1-p)
  }

  #Geometric Dist.
  if(a>=1)
  {
    dist.vect[v]="Geom"

```

```
        r1=(1+a + sqrt(a^2-1));
        r2=  (1+a - sqrt(a^2-1));
        p1= Dmom1[v] * r1 / (2+Dmom1[v]*r1)
        p2= Dmom1[v] * r2 / (2+Dmom1[v]*r2)
        q1= 1 / r1
        q2= 1 / r2

        dmax1[v]=qgeom(1-10^-4,1-p1)
        dmax2[v]=qgeom(1-10^-4,1-p2)
        dmaxs[v]=max(dmax1[v],dmax2[v])

        probabilities[v,c(1:(dmaxs[v]+1))] = dgeom(0:dmaxs[v],1-p1)
    * q1 + q2 * dgeom(0:dmaxs[v],1-p2)
      }
    }
  }
  return(list(dmaxs=dmaxs, prob=probabilities, dists=dist.vect))
}
```

The following code block is developed using R to estimate parameters of the probability distributions.

```
#Name of this file is:ParamEstFuncs_v3.R
beta.binom.parameterv4_4<- function(mu1,mu2,mu3,D.0.Prob,n.val=NA,
    plotflg=FALSE)
{
  rho1 = mu2/mu1
  rho2 = mu3/mu2

  a<- function(n)
  {
    mu1 * (n-mu1-rho1)/(rho1*n-n+mu1)
  }

  b<- function(n)
  {
    (n-mu1)*(n-mu1-rho1)/(rho1*n-n+mu1)
  }

  f<- function(n)
  {
    rho2*(a(n)+b(n))^2 + rho2*n*(a(n)+b(n)) + a(n)*(b(n)-a(n))*((a(
   n)+b(n))^2 + n*(3*(a(n)+b(n))+2*n))
  }

  ff<- function(mm)
  {
    D.0.Prob * beta(a(mm) , b(mm))-beta(a(mm) ,mm + b(mm))
  }

  tt<- seq(0,max(1000,3*mu3),length.out=10000);
  if(sum(f(tt)>=0)<10000)
    nlow=floor(tt[max(which(f(tt)<0))]);  #min(mu1,)
  if(sum(f(tt)>=0) == 10000)
    nlow=0;
  nhigh=max(1000,abs(mu1*mu2*mu3))
  nhigh.org=nhigh;

```

```
if(plotflg)
{
  plot(tt,f(tt),type="l",ylim=c(-100,500))
  abline(h=0,col="red")
}

y.mid=100;iteration=1;

countleft=0

while(abs(y.mid)>0.001)
{
  n.mid= (nlow + nhigh)/2
  y.mid = (-a(n.mid) + b(n.mid))*(a(n.mid) + b(n.mid)+2*n.mid)/(a(n.mid) + b(n.mid)) - rho2 #f2(n.mid); #f(n.mid);
  if(y.mid>0)
    nhigh=n.mid
  if(y.mid<0)
    nlow=n.mid
  if(plotflg)
    print(c(n.mid,y.mid))

  iteration=iteration+1;
  if(nlow-n.mid)
countleft = countleft+1;
  if((countleft>3)&(abs(y.mid)>0.01))
  {
  nlow=nlow-1;countleft=0;
  }
  if(iteration>500)
    break;
}

n.low = ceiling(n.mid);

n.mid.vect = c(n.low:(n.low+10000)) #c(n.low:1000)
diff.abs=abs(D.0.Prob * beta(a(n.mid.vect) , b(n.mid.vect))-beta(a(n.mid.vect) ,n.mid.vect+ b(n.mid.vect)))
DIFF1= D.0.Prob * beta(a(n.mid.vect) , b(n.mid.vect))-beta(a(n.mid.vect) ,n.mid.vect+ b(n.mid.vect))
if(sum(is.na(DIFF1))>0)
{
  n.mid.vect=n.mid.vect[!is.na(DIFF1)]
  DIFF1= D.0.Prob * beta(a(n.mid.vect) , b(n.mid.vect))-beta(a(n.mid.vect) ,n.mid.vect+ b(n.mid.vect))
}

DIFF1.org = DIFF1
if((DIFF1.org[1]<0)&(max(DIFF1.org[DIFF1.org!=0])<0))
{
  if(0 %in% DIFF1.org)
  {
    n.est.ind = min(which(DIFF1.org==0))
    n.est=n.mid.vect[n.est.ind]
  }
  if(!(0 %in% DIFF1.org))
  {
    n.est.ind=which(abs(DIFF1.org)<0.00001)[1];
    n.est = n.mid.vect[n.est.ind];
    if(is.na(n.est))
```

```
        n.est = max(n.mid.vect)
    }
  }

  if((DIFF1.org[1]<0)&(max(DIFF1.org)>0))
  {
    if((0 %in% DIFF1.org))
    {
      n.est.ind = min(which(DIFF1.org==0))
      n.est=n.mid.vect[n.est.ind]
    }
    if(!(0 %in% DIFF1.org))
    {
      ind.after.max = which(DIFF1.org == max(DIFF1.org)):length(
  DIFF1.org)
      if( DIFF1.org[length(DIFF1.org)]<0.00001)
        n.est = n.mid.vect[ind.after.max][which(DIFF1.org[ind.after
  .max]<0.00001)[1]];
      if( DIFF1.org[length(DIFF1.org)]>0.00001)
        n.est = n.mid.vect[length(DIFF1.org)]
    }
  }

  if(DIFF1.org[1]>0)
  {
    diff.abs = abs(DIFF1.org);
    n.est = min(n.mid.vect[(diff.abs<0.00001)]);

    if(!is.finite(n.est))
    {
      end=10000;n.low.cycle =n.low;
      min.vect=0;k=1;
      while(!is.finite(n.est))
      {
        n.mid.vect = c(n.low.cycle:(n.low.cycle+end))
        diff.abs=abs(D.0.Prob * beta(a(n.mid.vect) , b(n.mid.vect))
  -beta(a(n.mid.vect) ,n.mid.vect+ b(n.mid.vect)))

    n.mid.vect=n.mid.vect[!is.nan(diff.abs)]
    diff.abs=diff.abs[!is.nan(diff.abs)]

        n.est = min(n.mid.vect[(diff.abs<0.00001)]);
        rate.diff = (max(diff.abs)-min(diff.abs))/10000
        n.low.cycle = (n.low.cycle+end);
        end=max(10000,ceiling((min(diff.abs)-0.00001)/rate.diff));
  #             print(min(diff.abs))
        min.vect[k]=min(diff.abs);k=k+1

        if(k>3)
        {
          n.est= n.low.cycle+end;
        }
      }
    }
  }

  if(length(DIFF1.org[DIFF1.org!=0])==0)
  {
  n.est = n.low
  }
```

```
  alpha.est = a(n.est)
  beta.est = b(n.est)

  s.est = alpha.est + beta.est
  m.est = alpha.est / s.est
  size.est = n.est

  res=list(s.est = s.est,m.est= m.est,size.est = n.est,
           n.est = n.est,alpha.est = a(n.est),beta.est = b(n.est))

  return(res)
}

beta.pascal.parameter<- function(mu1,mu2,mu3,plotflg=FALSE)
{
  rho1 = mu2/mu1;
  rho2 = mu3/mu2;

  flgvect=rep(0,5);

  cond1 = ((rho1>1)&(rho2>2*rho1-1));
  if(cond1)  ## ||((rho2<2*rho1-1)&(rho1<1)) -- WE DONT NEED THE SECOND HALF
  {
    a = (2*mu1+3*rho2 + 1 - 4*rho1)/(rho2 - 2*rho1+1)
    flgvect[1]=1;

    Delta1 = (a-1+mu1-rho1*(a-2))^2 - 4*mu1*(a-1)
    Delta2 = (a-1+4*mu1-rho2*(a-3))^2 - 4*mu1*(a-1)

    if((Delta1 >= 0)&(Delta2 >= 0))
    {
      b = (-(a-1+mu1-rho1*(a-2)) - sqrt(Delta1))/2;
      r = mu1*(a-1)/b;
      flgvect[2]=1;
      b.approx = b;
      r.approx = r;

    }

    if((Delta1 >= 0)&(Delta2 < 0))
    {
      b = (-(a-1+mu1-rho1*(a-2)) - sqrt(Delta1))/2;
      r = mu1*(a-1)/b;
      flgvect[3]=1;
    }

    if((Delta1 < 0)&(Delta2 >= 0))
    {
      b.approx = (-(a-1+4*mu1-rho2*(a-3)) + sqrt(Delta2))/4;
      r.approx = mu1*(a-1)/b.approx;
      b=NA
      r=NA
      #b.approx = -(a-1+mu1-rho1*(a-2))/2;
      flgvect[4]=1
    }

    if((Delta1 < 0)&(Delta2 < 0))
    {
```

```
      b.approx = -(a-1+mu1-rho1*(a-2))/2;
      if(b.approx<0)
        b.approx = -(4*mu1 - rho2*(a-3)+a-1)/4
      r.approx = mu1*(a-1)/b.approx;
      b=NA
      r=NA

      flgvect[5]=1;
    }

    rvect = c(1:100)
    a = (2*mu1+3*rho2 + 1 - 4*rho1)/(rho2 - 2*rho1+1)
    bvect = mu1*(a-1)/rvect

    devv = abs(mu2-rvect * bvect*(a+rvect-1)*(a+bvect-1)/(a-2)/(a
   -1)^2)
    index = which(devv==min(devv))
    r.est = rvect[index[1]]
    b.est = bvect[index[1]]

    res<- list(a=a,r=r,b=b,b.apx = b.approx, r.apx = r.approx ,b.
   est = b.est, r.est= r.est,flgvect= flgvect);
  }

  if(!cond1)
  {
    print(sprintf("(%.3f,%.3f,%.3f) are not suitable for Beta-
   Pascal!",mu1,mu2,mu3));
    res<- list(a=NA,r=NA,b=NA,b.apx = NA, r.apx = NA ,b.est = NA, r
   .est= NA,flgvect= rep(0,5));
  }
  #print(c(r,b,a))
  #print(c(r.est,b.est,a ))
  return(res)
}
```

The following code block is developed using RCPP to implement the value iteration algorithm and calculate the costs.

```
//Name of this file is: RowMatchAndCostCalc.cpp
#include<Rcpp.h>
#include <math.h>
#include <vector>
using namespace Rcpp;

double L(int z, int q_r, int q_s,Rcpp::NumericVector dmax, int c_r,
     double q_s_max, double eta, double xi, double h, double b, Rcpp
   ::NumericMatrix demandmatrix,int t)
{
  double expected_cost=0;
  int backlogged_inventory;
  int excess_inventory;
  double costmultiplier1;
  double costmultiple2;
  if (q_s_max==0)
  {
    costmultiplier1= pow((double)q_s_max+0.01,eta);
    costmultiple2=0;
```

```
  }
  if(q_s==0)
  {
    costmultiple2=0;
  }
  if (q_s_max>0)
  {
    costmultiplier1=pow((double)q_s_max,eta);
    costmultiple2=pow((double) q_s,xi);
  }

  expected_cost=(double)q_r*(double)c_r+ (double) c_r*2*costmultiple2*costmultiplier1;

  for(int d=0;d <= dmax(t-1);d++)
  {
    excess_inventory= std::max(z-d,0);
    backlogged_inventory= std::max(0,d-z);
    expected_cost=expected_cost+(excess_inventory*h+b*backlogged_inventory)*demandmatrix(t-1,d);
  }
  return expected_cost;
}


int match_cpp(int t, int y, int K_prime, int q_r, int K_prime_max, Rcpp::NumericVector mininv, Rcpp::NumericVector maxinv, Rcpp::NumericMatrix maxpipe, Rcpp::NumericVector pipevect, int leadtime)
{
  int index=0;
  if (leadtime==1)
  {
    index=index+(-1*mininv(t)+maxinv(t)+1)*K_prime;
    index=index+(-1*mininv(t)+y);
  }
  if (leadtime==2)
  {
    index=(-1*mininv(t)+maxinv(t)+1)*(K_prime_max+1)*q_r;
    index=index+(-1*mininv(t)+maxinv(t)+1)*K_prime;
    index=index+(-1*mininv(t)+y);
  }
  if (leadtime==3)
  {
    index=(-1*mininv(t)+maxinv(t)+1)*(K_prime_max+1)*(maxpipe(t,0)+1)*q_r;
    index=index+(-1*mininv(t)+maxinv(t)+1)*(K_prime_max+1)*pipevect(1);
    index=index+(-1*mininv(t)+maxinv(t)+1)*K_prime;
    index=index+(-1*mininv(t)+y);
  }
  if (leadtime==4)
  {
    index=(-1*mininv(t)+maxinv(t)+1)*(K_prime_max+1)*(maxpipe(t,0)+1)*(maxpipe(t,1)+1)*q_r;
    index=index+(-1*mininv(t)+maxinv(t)+1)*(K_prime_max+1)*(maxpipe(t,0)+1)*pipevect(2);
    index=index+(-1*mininv(t)+maxinv(t)+1)*(K_prime_max+1)*pipevect(1);
    index=index+(-1*mininv(t)+maxinv(t)+1)*K_prime;
```

```
    index=index+(-1*mininv(t)+y);
  }
  return(index);
}


// [[Rcpp::export]]
Rcpp::NumericMatrix calc_fprev(int t, Rcpp::NumericVector row_size, Rcpp::NumericMatrix f_prev, Rcpp::NumericVector q_r_max, Rcpp::NumericVector dmax, int c_r, double eta, double xi, double h, double b, Rcpp::NumericMatrix demandmatrix, int leadtime)
{
  for (int i=0 ; i<row_size(t-1) ; i++)
  {
    double mincost=10000000;
    int y=f_prev(i,1);
    double q_s_max=f_prev(i,2);
    double expected_cost;
    int optqr=0;
    int optqs=0;
    int q_r,q_s;
    for (q_r=0; q_r<=q_r_max(t-1); q_r++)
    {
      for (q_s=0; q_s<=q_s_max; q_s++)
      {
        expected_cost=L((y+q_s),q_r,q_s,dmax,c_r,q_s_max,eta,xi,h,b,demandmatrix,t);
        if (expected_cost<mincost)
        {
          mincost=expected_cost;
          optqr=q_r;
          optqs=q_s;
        }
      }
    }
    f_prev(i,(3+leadtime-1))=optqr;
    f_prev(i,(4+leadtime-1))=optqs;
    f_prev(i,(5+leadtime-1))=mincost;
  }
  return f_prev;
}


// [[Rcpp::export]]
Rcpp::NumericMatrix costfunc(Rcpp::NumericMatrix f, int t, int c_r, double eta, double xi, double h, double b, Rcpp::NumericVector dmax, int K_prime_max, Rcpp::NumericMatrix f_prev, Rcpp::NumericMatrix P, Rcpp::NumericMatrix demandmatrix, Rcpp::NumericVector mininv, Rcpp::NumericVector maxinv, double alpha, int leadtime, Rcpp::NumericMatrix maxpipe, Rcpp::NumericVector row_size, Rcpp::NumericVector q_r_max)
{
  int optqs=0;
  int optqr;
  int y;
  double q_s_max;
  int q_r;
  int q_s;
  double expected_cost;
  double mincostqs;
```

```
int d;
int K_prime;
int index;
int m;
int i=0;
int startind = 0;
int endind = row_size(t-1);

for (i=startind ; i<endind ; i++)
{
  Rcpp::NumericVector pipevect;

  y=f(i,1);
  q_s_max=f(i,2);

  for (int cc=0 ; cc<(leadtime-1);cc++)
  {
    pipevect.push_back(f(i,cc+3));
  }

  mincostqs=10000000;

  for(q_r=0;q_r<=q_r_max(t-1);q_r++)
  {
    for(q_s=0; q_s<=(int)q_s_max; q_s++)
    {
      expected_cost=L((y+q_s),q_r,q_s,dmax,c_r,q_s_max,eta,xi,h,b,demandmatrix,t);
      for(d=0; d <= dmax(t-1); d++)
      {
        if(leadtime==1)
        {
          //Rcout<<"d:"<<d<<" dmax:"<<dmax(t-1)<<std::endl;
          m=y+q_r+q_s-d;
        }
        else
        {
          m=y+q_s+pipevect(0)-d;
        }
        for(K_prime=0; K_prime<=K_prime_max;K_prime++)
        {
          index=match_cpp(t,m,K_prime,q_r,K_prime_max,mininv,maxinv,maxpipe,pipevect,leadtime);
          expected_cost=expected_cost+alpha*f_prev(index,(5+leadtime-1))*demandmatrix(t-1,d)*P(q_s_max,K_prime);
        }
      }
      if(expected_cost<mincostqs)
      {
        mincostqs=expected_cost;
        optqs=q_s;
        optqr=q_r;
      }
    }
  }
  f(i,(3+leadtime-1))=optqr;
  f(i,(4+leadtime-1))=optqs;
  f(i,(5+leadtime-1))=mincostqs;
}
return f;
```

```
}
```

A.2 Policy Gradient

The code block given below is developed using R to control code runs and record logs.

```
setwd("C:\\Users\\suser\\Desktop\\3501 Project\\GrowingIB\\Simulations")
getwd()

Declining_IB=T
N0=10
strategy=1
capacity=T #if false capacity of secondary supplier=0(like single sourcing problem)
stepsize=1

library(rmutil) #for betabinom func.
#library(extraDistr) #for bnbinom func.
library(brr) #for beta_nbinom func.(bnbinom doesn't have a quantile funct.)
library(Rcpp)
if (Declining_IB)
{
  source("DynamicDistSelect_DecliningIB_CalcStats.R")
}else
{
  source("DynamicDistSelect_GrowingIB_CalcStats.R")
}

sourceCpp("PolicyGradient_v2.cpp")

range=109
for(U in c(1))#number of updates
{
  for(NN in c(2))#number of starting points
  {
    cr=5 #price of regular
    epsilon=0.0001


    n=NN #how many different points to try for policy gradient search
    update=U #how many times heurisitics' paramaters will be updated

    ######naming parameters table#####
    if (Declining_IB)
    {
      parameters <- read.csv("SD_PolGrad_ProjectParameters.csv")
    }else
    {
      parameters <- read.csv("SG_PolGrad_ProjectParameters.csv")
    }

```

```
  for (ind in 0:update)
  {
    parameters = cbind(parameters,NA,NA,NA,NA)
  }

  dualnames=c("DualSs1","DualSr1")
  tailorednames=c("TailoredSs1","Tailored1Sr1")
  if (update>0)
  {
    for (i in 2:(update+1))
    {
      Ssname=sprintf("Ss%s",i)
      Srname=sprintf("Sr%s",i)
      dualnames=c(dualnames,Ssname,Srname)
    }
    for (i in 2:(update+1))
    {
      Ssname=sprintf("Ss%s",i)
      Srname=sprintf("Sr%s",i)
      tailorednames=c(tailorednames,Ssname,Srname)
    }
  }
  parameters=cbind(parameters,NA,NA,NA,NA,NA,NA)
  names(parameters)[c(11:(14+(4*update)+6))]=c(dualnames,tailorednames,"optcostindexDual","costofN0Dual",
                                    "costofN1Dual","optcostindexTail","costofN0Tail","costofN1Tail")

  ######Main Loop#####
  replength = 1000
  for (cc in range)
  {
    starttime=proc.time()
    start_time = Sys.time()

    lambda=parameters[cc,1]
    alpha=parameters[cc,2]
    h=parameters[cc,3]
    b=parameters[cc,4]
    eta=parameters[cc,5]
    xi=parameters[cc,6]
    simlength=parameters[cc,7]
    discountrate=parameters[cc,8]
    market_scenario=parameters[cc,9]
    LT=parameters[cc,10]

    if(Declining_IB)
    {
      list=DynamicDistStats(lambda,alpha,simlength,stepsize,strategy,N0)
    }else
    {
      list=DynamicDistStats(lambda,alpha,simlength,stepsize,strategy)
    }

    dmax=list$dmaxs
    dists=list$dists
    param.Pois=list$param.Pois
    Param.NegBinom=list$Param.NegBinom
```

```
    Param.BetaBinom=list$Param.BetaBinom
    Param.BetaPascal=list$Param.BetaPascal

    #calculating dmaxs for each update period
    #calculating boundary of each update period to send into c++ function
    remainder=simlength%%(update+1)
    periods=0
    maxparams=rep(NA,update+1)
    if(remainder==0)
    {
      updatelength=simlength/(update+1)
      for (i in 1:(update+1))
      {
        maxparams[i]=dmax[updatelength*i]
        print(updatelength*i)
        periods[i]=updatelength*i
      }
    }else
    {
      updatelength=(simlength-remainder)/(update+1)
      for (i in 1:(update))
      {
        maxparams[i]=dmax[updatelength*i]
        print(updatelength*i)
        periods[i]=updatelength*i
      }
      maxparams[update+1]=dmax[updatelength*(update+1)+remainder]
      print(updatelength*(update+1)+remainder)
      periods[update+1]=updatelength*(update+1)+remainder
    }
    periods=c(0,periods)

    if (market_scenario==1)
    {
      P=as.matrix(read.csv("Keq12_Symmetric_TPM.csv",sep = ";", header = F))
    }else if(market_scenario==2)
    {
      P=as.matrix(read.csv("Keq12_Increasing_TPM.csv",sep = ";", header = F))
    }else if(market_scenario==3)
    {
      P=as.matrix(read.csv("Keq12_Decreasing_TPM.csv",sep = ";", header = F))
    }

    WW=upper.tri(matrix(1,13,13))*matrix(1,13,13)+diag(13)#upper.tri(matrix(1,5,5))*matrix(1,5,5)+diag(5)
    P.cum=P%*%WW

    #####Capacity Generation(each row represents a replication)####
    set.seed(100)
    Kmatrix=matrix(NA,replength,simlength)
    for (k in 1:replength)
    {
      K_vect=0
      r.num=runif(simlength)
      for(i in 1:simlength-1)
```

```
      {
        K_vect[i+1]=sum(P.cum[K_vect[i]+1,]<r.num[i])
      }
      Kmatrix[k,]=K_vect
    }

    #####Demand Generation(each row represents a replication)####
    Demandmatrix=matrix(NA,replength,simlength)
    for (i in 1:simlength)
    {
      if(dists[i]=="Pois")
      {
        Demandmatrix[c(1:replength),i]=rpois(replength,param.Pois[i])
      }else if(dists[i]=="NBinom")
      {
        Demandmatrix[c(1:replength),i]=rnbinom(replength,size = Param.NegBinom[i,2], prob = Param.NegBinom[i,1])
      }else if(dists[i]=="Beta_Binom")
      {
        Demandmatrix[c(1:replength),i]=rbetabinom(replength,size=Param.BetaBinom[i,1],m=Param.BetaBinom[i,2],
                                                   s=Param.BetaBinom[i,3])
      }else if(dists[i]=="Beta_Pascal")
      {
        Demandmatrix[c(1:replength),i]=rbeta_nbinom(replength,Param.BetaPascal[i,1], Param.BetaPascal[i,2],
                                                      Param.BetaPascal[i,3])
      }
    }

    set.seed(NULL)
    dualsize=1; tailoredsize=1;

    ########### calculating policy gradient of dual index ############
    #generating initial parameters for policy graident search
    if(n==1)#just look for policy gradient of a zero vector (0,...,0) that is a corner point
    {
      paramsmatrix=matrix(0,1,2*(update+1))
    }else if(n==2)#look for policy gradient of a zero vector (0,...,0)
    {      #and a vector of (maxparams[1],maxparams[1],...,maxparams[update+1],maxparams[update+1])
      paramsmatrix=matrix(NA,2,2*(update+1))
      paramsmatrix[1,]=rep(0,2*(update+1))
      for (k in 0:update)
      {
        paramsmatrix[2,2*k+1]=maxparams[k+1]
        paramsmatrix[2,2*k+2]=maxparams[k+1]
      }
    }else
    {
      paramsmatrix=matrix(NA,n,2*(update+1))
      paramsmatrix[1,]=rep(0,2*(update+1))
      for (k in 0:update)
      {
```

```
      paramsmatrix[2,2*k+1]=maxparams[k+1]
      paramsmatrix[2,2*k+2]=maxparams[k+1]
    }
    for (i in 3:n)
    {
      for (k in 0:update)
      {
        paramsmatrix[i,2*k+1]=sample(c(1:maxparams[k+1]),1)
        paramsmatrix[i,2*k+2]=sample(c(1:maxparams[k+1]),1)
      }
    }
  }

  optparamsdual=CalPolGradDual(nrow(paramsmatrix),paramsmatrix, update, dualsize, Demandmatrix, Kmatrix, simlength,
                                LT, h, b, cr, xi, eta, discountrate, replength, periods)

  ######## calculating policy gradient of tailored base surge ########
  #generating initial parameters for policy graident search
  if(n==1)#just look for policy gradient of a zero vector (0,...,0) that is a corner point
  {
    paramsmatrix=matrix(0,1,2*(update+1))
  }else if(n==2)#look for policy gradient of a zero vector (0,...,0)
  {      #and a vector of (maxparams[1],maxparams[1],..., maxparams[update+1],maxparams[update+1])
    paramsmatrix=matrix(NA,2,2*(update+1))
    paramsmatrix[1,]=rep(0,2*(update+1))
    for (k in 0:update)
    {
      paramsmatrix[2,2*k+1]=maxparams[k+1]
      paramsmatrix[2,2*k+2]=maxparams[k+1]
    }
  }else
  {
    paramsmatrix=matrix(NA,n,2*(update+1))
    paramsmatrix[1,]=rep(0,2*(update+1))
    for (k in 0:update)
    {
      paramsmatrix[2,2*k+1]=maxparams[k+1]
      paramsmatrix[2,2*k+2]=maxparams[k+1]
    }
    for (i in 3:n)
    {
      for (k in 0:update)
      {
        paramsmatrix[i,2*k+1]=sample(c(1:maxparams[k+1]),1)
        paramsmatrix[i,2*k+2]=sample(c(1:maxparams[k+1]),1)
      }
    }
  }

  optparamstailored=CalPolGradTailored(nrow(paramsmatrix), paramsmatrix, update, tailoredsize, Demandmatrix, Kmatrix, simlength,
                                         LT, h, b, cr, xi, eta, discountrate, replength, periods)
```

```
      ld=length(optparamsdual)
      parameters[cc,c(11:(14+(4*update)+6))]=c(optparamsdual[1:(ld-3)],optparamstailored[1:(ld-3)],
                                                      optparamsdual[(ld-3+1):ld],optparamstailored[(ld-3+1):ld])

      end_time = Sys.time()
      endtime=proc.time()
      timep=end_time-start_time
      time=endtime-starttime
      timespent=sprintf("parameter:%s lambda:%s alpha:%s simlength:%s update:%s user:%s system:%s elapsed:%s time:%s",cc,lambda,alpha,simlength,update,time[[1]],time[[2]],time[[3]],timep)
      write.table(timespent,"times.txt",append = T,row.names = F,col.names = F)

      XX=sprintf("opt_parameters_costs(%s)_update(%s)_n(%s).txt",cc,U,NN)
      write.table(parameters[cc,],file=XX, sep="\t",row.names = F)
    }
  }
}
```

Following code block is developed using RCPP to implement Policy Gradient heuristic.

```
//Name of this file is: PolicyGradient_v2.cpp
#include<Rcpp.h>
#include <math.h>
#include <vector>
#include "ParamOptim.h"
using namespace Rcpp;

// [[Rcpp::export]]
NumericVector CalPolGradDual(int n, NumericMatrix paramsmatrix, int update, int dualsize, NumericMatrix Demandmatrix, NumericMatrix Kmatrix, int simlength, int LT, double h, double b, double cr, double xi, double eta, double discountrate, int replength, NumericVector periods)
{                              //n is the row size of paramsmatrix
  int k; NumericVector params; int j; int paramnum=2*(update+1); NumericMatrix Ss(1,update+1);
  NumericMatrix Sr(1,update+1); double cost; double mincost; int i; NumericVector updatedparams; int index;
  NumericVector prevparams; NumericMatrix optparamsmatrix_dual(n,paramnum); NumericVector optcostsvectdual(n);
  NumericVector res;

  for(k=0; k<n; k++)
  {
    params=paramsmatrix(k,_);
    for(j=0; j<(update+1); j++)
    {
      Ss(0,j)=params(2*j);
      Sr(0,j)=params(2*j+1);
    }
    cost=optimizedual(dualsize, Ss, Sr, Demandmatrix, Kmatrix,
```

```
    simlength, LT, h, b, cr, xi, eta, discountrate, replength,
    periods)(0);

     NumericVector costvect(paramnum*2);
     NumericMatrix updatedparamsmatrix(paramnum*2,paramnum);
     mincost=99999999;
     while(cost<mincost)
     {
       mincost=cost;
       for(i=0; i<paramnum; i++)
       {
         updatedparams=clone(params);
         updatedparams(i)=updatedparams(i)+1;

         for(j=0; j<(update+1); j++)
         {
           Ss(0,j)=updatedparams(2*j);
           Sr(0,j)=updatedparams(2*j+1);
         }

         costvect(i)=optimizedual(dualsize, Ss, Sr, Demandmatrix,
    Kmatrix, simlength, LT, h, b, cr, xi, eta, discountrate,
    replength, periods)(0);
         updatedparamsmatrix(i,_)=updatedparams;

         updatedparams=clone(params);
         updatedparams(i)=updatedparams(i)-1;

         for(j=0; j<(update+1); j++)
         {
           Ss(0,j)=updatedparams(2*j);
           Sr(0,j)=updatedparams(2*j+1);
         }
         costvect(paramnum+i)=optimizedual(dualsize, Ss, Sr,
    Demandmatrix, Kmatrix, simlength, LT, h, b, cr, xi, eta,
    discountrate, replength, periods)(0);
         updatedparamsmatrix(paramnum+i,_)=updatedparams;
       }
       index=which_min(costvect);
       cost=costvect(index);
       prevparams=clone(params);
       params=updatedparamsmatrix(index,_);
     }
     optparamsmatrix_dual(k,_)=prevparams;
     optcostsvectdual(k)=mincost;
   }
   index=which_min(optcostsvectdual);
   Rcout<<optcostsvectdual(0)<<" "<<optcostsvectdual(1)<<std::endl;
   Rcout<<index<<std::endl;
   NumericVector optparamsdual=optparamsmatrix_dual(index,_);
   res=clone(optparamsdual);
   res.push_back(index);
   res.push_back(optcostsvectdual(0));
   res.push_back(optcostsvectdual(1));
   return res;
}

// [[Rcpp::export]]
NumericVector CalPolGradTailored(int n, NumericMatrix paramsmatrix,
     int update, int tailoredsize, NumericMatrix Demandmatrix,
```

```
    NumericMatrix Kmatrix, int simlength, int LT, double h, double b
    , double cr, double xi, double eta, double discountrate, int
    replength, NumericVector periods)
{                                   //n is the row size of paramsmatrix
  int k; NumericVector params; int j; int paramnum=2*(update+1);
    NumericMatrix Ss(1,update+1);
  NumericMatrix Sr(1,update+1); double cost; double mincost; int i;
     NumericVector updatedparams; int index;
  NumericVector prevparams; NumericMatrix optparamsmatrix_tailored(
    n,paramnum); NumericVector optcostsvecttailored(n);
  NumericVector res;

  for(k=0; k<n; k++)
  {
    params=paramsmatrix(k,_);
    for(j=0; j<(update+1); j++)
    {
      Ss(0,j)=params(2*j);
      Sr(0,j)=params(2*j+1);
    }
    cost=optimizetailored(tailoredsize, Ss, Sr, Demandmatrix,
    Kmatrix, simlength, LT, h, b, cr, xi, eta, discountrate,
    replength, periods)(0);

    NumericVector costvect(paramnum*2);
    NumericMatrix updatedparamsmatrix(paramnum*2,paramnum);
    mincost=99999999;
    while(cost<mincost)
    {
      mincost=cost;
      for(i=0; i<paramnum; i++)
      {
        updatedparams=clone(params);
        updatedparams(i)=updatedparams(i)+1;

        for(j=0; j<(update+1); j++)
        {
          Ss(0,j)=updatedparams(2*j);
          Sr(0,j)=updatedparams(2*j+1);
        }

        costvect(i)=optimizetailored(tailoredsize, Ss, Sr,
    Demandmatrix, Kmatrix, simlength, LT, h, b, cr, xi, eta,
    discountrate, replength, periods)(0);
        updatedparamsmatrix(i,_)=updatedparams;

        updatedparams=clone(params);
        updatedparams(i)=updatedparams(i)-1;

        for(j=0; j<(update+1); j++)
        {
          Ss(0,j)=updatedparams(2*j);
          Sr(0,j)=updatedparams(2*j+1);
        }

        costvect(paramnum+i)=optimizetailored(tailoredsize, Ss, Sr,
     Demandmatrix, Kmatrix, simlength, LT, h, b, cr, xi, eta,
    discountrate, replength, periods)(0);
        updatedparamsmatrix(paramnum+i,_)=updatedparams;
      }
```

```
      index=which_min(costvect);
      cost=costvect(index);
      prevparams=clone(params);
      params=updatedparamsmatrix(index,_);
    }
    optparamsmatrix_tailored(k,_)=prevparams;
    optcostsvecttailored(k)=mincost;
  }
  index=which_min(optcostsvecttailored);
  Rcout<<optcostsvecttailored(0)<<" "<<optcostsvecttailored(1)<<std::endl;
  Rcout<<index<<std::endl;
  NumericVector optparamstailored=optparamsmatrix_tailored(index,_);
  res=clone(optparamstailored);
  res.push_back(index);
  res.push_back(optcostsvecttailored(0));
  res.push_back(optcostsvecttailored(1));
  return res;
  }
```

Following code block is developed using RCPP to optimize heuristics' parameters.

```
// ParamOptim.h
#include<Rcpp.h>
#include <math.h>
#include <vector>
#include "Heuristics.h"
using namespace Rcpp;

NumericVector optimizedual(int dualsize, NumericMatrix dualSs, NumericMatrix dualSr, NumericMatrix Demandmatrix, NumericMatrix Kmatrix, int simlength, int LT, double h, double b, double cr, double xi, double eta, double discountrate, int replength, NumericVector periods)
{
  NumericVector Ss; NumericVector optSs; NumericVector demandvect; NumericVector Kvect;
  NumericVector Sr; NumericVector optSr;
  NumericMatrix sonuc; int rep; int i;
  NumericVector avgcost(dualsize); double dualcost;
  for(i=0; i<dualsize; i++)//each row of action space
  {
    for(rep=1; rep<=replength; rep++)//each replication
    {
      Ss=dualSs(i,_);
      Sr=dualSr(i,_);
      demandvect=Demandmatrix(rep-1,_);
      Kvect=Kmatrix(rep-1,_);

      dualcost=dualindex(demandvect,Kvect,simlength,Ss,Sr,LT,h,b,cr,xi,eta,discountrate,periods)(5);
      avgcost[i]=avgcost[i]+dualcost/replength;
    }
  }
  return avgcost;
}
```

```
NumericVector optimizetailored(int tailoredsize, NumericMatrix
    tailoredSs, NumericMatrix tailoredSr, NumericMatrix Demandmatrix
    , NumericMatrix Kmatrix, int simlength, int LT, double h, double
     b, double cr, double xi, double eta, double discountrate, int
    replength, NumericVector periods)
{
  NumericVector Ss; NumericVector optSs; NumericVector demandvect;
   NumericVector Kvect;
  NumericVector Sr; NumericVector optSr;
  NumericMatrix sonuc; int rep; int i;
  NumericVector avgcost(tailoredsize); double tailoredcost;
  for(i=0; i<tailoredsize; i++)
  {
    for(rep=1; rep<=replength; rep++)
    {
      Ss=tailoredSs(i,_);
      Sr=tailoredSr(i,_);
      demandvect=Demandmatrix(rep-1,_);
      Kvect=Kmatrix(rep-1,_);

      tailoredcost=tailoredbase(demandvect,Kvect,simlength,Ss,Sr,LT
    ,h,b,cr,xi,eta,discountrate,periods)(5);
      avgcost[i]=avgcost[i]+tailoredcost/replength;
    }
  }
  return avgcost;
}
```

Following code block is developed using RCPP for heuristics and their cost calculations.

```
// heuristics.h

#include<Rcpp.h>
#include <math.h>
#include <vector>
using namespace Rcpp;

NumericVector costcalc(NumericVector Demandvect, NumericVector
    Kvect, double h, double b, double cr, double xi, double eta,
    double discountrate, NumericVector I, NumericVector IP,
    NumericVector Qs, NumericVector Qr, int simlength)
{
  NumericVector acq_cost_reg; NumericVector acq_cost_sec;
   NumericVector acq_cost_tot;
  NumericVector hold_cost; NumericVector backl_cost; NumericVector
   cost;
  double cs_t; double costmultiplier1; double costmultiplier2;
  double acqr; double acqs; double acqtot; double hold; double
   backl;
  double periodic_cost; double acquisition_cost_reg; double
   acquisition_cost_sec;
  double acquisition_cost_tot; double holding_cost; double
   backlog_cost; double totalcost;
  NumericVector result; int excess; int backlogged; int tt;

  for(tt=1; tt<=simlength; tt++)
  {
```

```
    if(Kvect(tt-1)==0)
    {
      costmultiplier1=0;
    }
    if (Kvect(tt-1)>0)
    {
      costmultiplier1=pow((double)Kvect(tt-1),eta);
      costmultiplier2=pow((double)Qs(tt-1),xi);
    }

    cs_t = cr*2*costmultiplier1*costmultiplier2;
    acqs=pow(discountrate,tt-1) * cs_t;
    acq_cost_sec.push_back(acqs);

    acqr=pow(discountrate,tt-1) * cr*Qr(tt-1);
    acq_cost_reg.push_back(acqr);

    acqtot=acq_cost_reg(tt-1) + acq_cost_sec(tt-1);
    acq_cost_tot.push_back(acqtot);

    excess=I(tt-1)+Qs(tt-1)-Demandvect(tt-1);
    hold=pow(discountrate,tt-1) * h* std::max(excess,0);
    hold_cost.push_back(hold);

    backlogged=Demandvect(tt-1)-I(tt-1)-Qs(tt-1);
    backl=pow(discountrate,tt-1) * b*std::max(backlogged,0);
    backl_cost.push_back(backl);

    periodic_cost=acq_cost_tot(tt-1)+hold_cost(tt-1)+backl_cost(tt-1);
    cost.push_back(periodic_cost);
  }
  acquisition_cost_reg=sum(acq_cost_reg);
  acquisition_cost_sec=sum(acq_cost_sec);
  acquisition_cost_tot=sum(acq_cost_tot);
  holding_cost=sum(hold_cost);
  backlog_cost=sum(backl_cost);
  totalcost=sum(cost);
  result.push_back(acquisition_cost_reg); result.push_back(acquisition_cost_sec); result.push_back(acquisition_cost_tot);
  result.push_back(holding_cost); result.push_back(backlog_cost); result.push_back(totalcost);
  return result;
}

//size is number of updates+1(if there is no update size is 1)
//if there is no update, periods is c(0,simlength)
NumericVector dualindex(NumericVector Demandvect, NumericVector Kvect, int simlength, NumericVector Ss, NumericVector Sr, int LT, double h, double b, double cr, double xi, double eta, double discountrate, NumericVector periods)
{
  NumericVector I; I.push_back(0);
  NumericVector IP; IP.push_back(0);
  int size=Ss.size();
  NumericVector Qr;
  NumericVector Qs;
  int qr; int qs; NumericVector res; int inv; int invpos;
  int sorder; int rorder; int k; int tt;

```

```
  for(k=0; k<size; k++)
  {
    for(tt=periods(k)+1; tt<=periods(k+1); tt++)
    {
      sorder=Ss(k) - I(tt-1);
      qs=std::min(std::max(sorder,0),(int)Kvect(tt-1));
      rorder=Sr(k)-IP(tt-1)-qs;
      qr=std::max(rorder,0);
      Qs.push_back(qs);
      Qr.push_back(qr);

      if(tt<LT)
      {
        inv = I(tt-1) - Demandvect(tt-1) + Qs(tt-1);
        I.push_back(inv);
        invpos = IP(tt-1)+ Qr(tt-1) + Qs(tt-1) - Demandvect(tt-1);
        IP.push_back(invpos);
      }
      if (tt>=LT)
      {
        inv = I(tt-1) + Qr[tt-LT] - Demandvect(tt-1) + Qs(tt-1);
        I.push_back(inv);
        invpos = IP(tt-1) + Qr(tt-1) + Qs(tt-1) - Demandvect(tt-1);
        IP.push_back(invpos);
      }
    }
  }
  res=costcalc(Demandvect, Kvect, h, b, cr, xi, eta, discountrate, I, IP, Qs, Qr, simlength);
  return res;
}

NumericVector tailoredbase(NumericVector Demandvect, NumericVector Kvect, int simlength, NumericVector Ss, NumericVector Sr, int LT, double h, double b, double cr, double xi, double eta, double discountrate, NumericVector periods)
{
  NumericVector I; I.push_back(0);
  NumericVector IP; IP.push_back(0);
  int size=Ss.size();
  NumericVector Qr;
  NumericVector Qs;
  int qr; int qs; NumericVector res; int inv; int invpos;
  int sorder; int rorder; int k; int tt;


  for(k=0; k<size; k++)
  {
    for(tt=periods(k)+1; tt<=periods(k+1); tt++)
    {
      sorder=Ss(k) - I(tt-1);
      qs=std::min(std::max(sorder,0),(int)Kvect(tt-1));
      rorder=Sr(k);
      qr=std::max(rorder,0);
      Qs.push_back(qs);
      Qr.push_back(qr);

      if(tt<LT)
      {
        inv = I(tt-1) - Demandvect(tt-1) + Qs(tt-1);
```

```
          I.push_back(inv);
          invpos = IP(tt-1)+ Qr(tt-1) + Qs(tt-1) - Demandvect(tt-1);
          IP.push_back(invpos);
        }
        if (tt>=LT)
        {
          inv = I(tt-1) + Qr[tt-LT] - Demandvect(tt-1) + Qs(tt-1);
          I.push_back(inv);
          invpos = IP(tt-1) + Qr(tt-1) + Qs(tt-1) - Demandvect(tt-1);
          IP.push_back(invpos);
        }
      }
    }
    res=costcalc(Demandvect, Kvect, h, b, cr, xi, eta, discountrate,
     I, IP, Qs, Qr, simlength);
    return res;
}
```

A.3 Simulation Codes

The code block given below is developed using R to simulate and save demand sample paths following the dynamics of Hekimoğlu and Karlı (2021)'s model.

```
setwd("C:\\Users\\suser\\Desktop\\OrdvsAdan\\DecliningRepo")
getwd()

Declining_IB=T
N0=25
#strategy=1
capacity=F #if false capacity of secondary supplier=0(like single
    sourcing problem)
stepsize=1
seedvalue=NULL
replication=1000

library(rmutil)
library(Rcpp)
source("C:\\Users\\suser\\Desktop\\OrdvsAdan\\Declining_IB\\
    DynamicDistSelect_OptActsStats_Ord&Adan.R")
if (Declining_IB)
{
  sourceCpp("C:\\Users\\suser\\Desktop\\OrdvsAdan\\Declining_IB\\
    RcppFunction_v2.cpp")
}

parameters=read.csv("C:\\Users\\suser\\Desktop\\OrdvsAdan\\
    Declining_IB\\optimization_parameters_v2.csv")
for (sensivity_level in c(0,0.05,0.25))
{
  replicsize=replication
  for (cc in 47)#33:48
  {
    h=parameters[cc,3]
    b=parameters [cc,4]
    eta=parameters [cc,5]
    xi=parameters[cc,6]
```

```
    simlength=parameters[cc,7]
    discountrate=parameters[cc,8]
    LT=parameters[cc,9]

    step_size = stepsize

    if(Declining_IB)
    {
      filename=sprintf("Demand_%s_DecliningIB_SensLevel_%s.Rdata",cc,sensivity_level)
    }else
    {
      filename=sprintf("Demand_%s_GrowingIB_SensLevel_%s.Rdata",cc,sensivity_level)
    }

    if(Declining_IB==F)
    {
      lambda=parameters[cc,1]
      lambda=lambda+runif(1,-lambda*sensivity_level,lambda*sensivity_level)
      alpha=parameters[cc,2]
      alpha=alpha+runif(1,-alpha*sensivity_level,alpha*sensivity_level)

      tmax = simlength+1
      Tvect=c(0:(tmax*step_size))/step_size
      tvect= Tvect
      simpartarrive= 3000

      #######    SIMULATION  -  BEGIN    #######
      cumsparesdemand=matrix(0,replicsize,length(Tvect))
      marginaldemand = matrix(0,replicsize,length(Tvect)-1)

      for(r in 1:replicsize)
      {
        interarrivals=rexp(2000,lambda)
        arrivaltimes=c(0,cumsum(interarrivals));
        ntot=sum(arrivaltimes<=tmax);

        for(i in 1:ntot)
        {
          interarrive=rexp(simpartarrive,rate=alpha)
          sparesarrive=cumsum(interarrive)+arrivaltimes[i]
          if(i==1)
            sparearrvalsall=sparesarrive[sparesarrive<=tmax]

          if(i>1)
            sparearrvalsall = c(sparearrvalsall,sparesarrive[sparesarrive<=tmax])
        }

        sparearrivalsall =sort(sparearrvalsall )

        for(indt in 1:(step_size*tmax))
        {
          tt=Tvect[indt];
          cumsparesdemand[r,indt]=sum(sparearrivalsall <=tt)
        }
        demandvect = cumsparesdemand[r,]
```

```
      marginaldemand[r,] = demandvect[-1] - demandvect[-length(demandvect)]
    }
    #######    SIMULATION  -  END    #######
  }else if(Declining_IB)
  {
    tmax = simlength+1
    Tvect=c(1:(tmax*step_size))/step_size
    tvect= Tvect
    cumsparesdemand= matrix(0,replicsize,length(Tvect))
    marginaldemand = matrix(0,replicsize,length(Tvect)-1)
    simpartarrive=simpartarrive.main=3000;

    for(r in 1:replicsize)
    {
      lambda=parameters[cc,1]
      lambda=lambda+runif(1,-lambda*sensivity_level,lambda*sensivity_level)
      alpha=parameters[cc,2]
      alpha=alpha+runif(1,-alpha*sensivity_level,alpha*sensivity_level)

      interarrivals=rexp(N0,lambda)
      arrivaltimes=c(cumsum(interarrivals));

      for(i in 1:N0)
      {
        nn=0;flg=FALSE;
        sparesarrive<- func(arrivaltimes, nn, i, simpartarrive.main, alpha, simpartarrive, flg)

        if(length(sparesarrive)>simpartarrive)
          simpartarrive.main = length(sparesarrive);

        if(i==1)
          sparearrvalsall=sparesarrive

        if(i>1)
          sparearrvalsall = c(sparearrvalsall,sparesarrive)
      }

      sparearrivalsall =sort(sparearrvalsall )
      if(length(sparearrivalsall)>0)
        demandvect = demand_counterv3(sparearrivalsall ,tvect)  ##CPP FUNCTION!
      if(length(sparearrivalsall)==0)
        demandvect = rep(0,length(tvect))

      cumsparesdemand[r,] = demandvect
      marginaldemand[r,] = demandvect[-1] - demandvect[-length(demandvect)]
    }
  }
  save(marginaldemand,file = filename)
  print(sprintf("Parameter:%s Sensivity Level:%s demand generation succesfully completed!",cc,sensivity_level))
 }
}
```

The following code block is developed using R to calculate necessary statistics to generate demand sample paths and match optimum actions with state variables.

```
#Name of this file is: DynamicDistSelect_OptActsStats_Ord&Adan.R
ReturnList=function(lambd,alph,plan_hor,leadtime,stepsize,strategy,
    capacity,Declining_IB,N0,dist_select_algrthm)
{
  lambda=lambd #parameter of poisson process that represents
    capital product sales
  alpha=alph #parameter of poisson process that represents  product
    failures
  t=plan_hor

  if(Declining_IB)
  {
    source("DynamicDistSelect_DecliningIB_CalcStats_Ord&Adan.R")
    list=DynamicDistStats(lambda,alpha,plan_hor,stepsize,strategy,
  N0,dist_select_algrthm)
  }else
  {
    source("DynamicDistSelect_GrowingIB_CalcStats_Ord&Adan.R")
    list=DynamicDistStats(lambda,alpha,plan_hor,stepsize,strategy,
  dist_select_algrthm)
  }

  dmax=list$dmaxs
  distributions=list$dists

  if(capacity)
  {
    q_s_max=12 #max order size for secondary supp.
    K_prime_max=12 #same as q_s_max
    if (market_scenario==1)
    {
      P=as.matrix(read.csv("Keq12_Symmetric_TPM.csv",sep = ";",
  header = F))
    }else if(market_scenario==2)
    {
      P=as.matrix(read.csv("Keq12_Increasing_TPM.csv",sep = ";",
  header = F))
    }else if(market_scenario==3)
    {
      P=as.matrix(read.csv("Keq12_Decreasing_TPM.csv",sep = ";",
  header = F))
    }
  }else if(capacity==F)
  {
    q_s_max=0
    K_prime_max=0
    P=matrix(1,1,1)
  }

  #epsilon=0.0001
  Xmin=0 #min inv. level
  Xmax=0 #max inv. level
  q_r_max=0
  q_r_max[1]=floor(dmax[1]*(1.5))

  row_size=(q_s_max+1)
  mininv=0
```

```
maxinv=0

for (aa in 2:leadtime) ##aa is period
{
  if (aa==2)
  {
    Xmint=Xmin-dmax[aa-1]
    Xmaxt=Xmax+q_s_max
    mininv[aa]=Xmint
    maxinv[aa]=Xmaxt
    q_r_max[aa]=floor(dmax[aa]*(1.5))
    row_size[aa]=(-Xmint+Xmaxt+1)*(q_s_max+1)*(q_r_max[1]+1)
  }
  if (aa==3)
  {
    Xmint=Xmint-dmax[aa-1]
    Xmaxt=Xmaxt+q_s_max
    mininv[aa]=Xmint
    maxinv[aa]=Xmaxt
    q_r_max[aa]=floor(dmax[aa]*(1.5))
    row_size[aa]=(-Xmint+Xmaxt+1)*(q_s_max+1)*(q_r_max[1]+1)*(q_r_max[2]+1)
  }
  if (aa==4)
  {
    Xmint=Xmint-dmax[aa-1]
    Xmaxt=Xmaxt+q_s_max
    mininv[aa]=Xmint
    maxinv[aa]=Xmaxt
    q_r_max[aa]=floor(dmax[aa]*(1.5))
    row_size[aa]=(-Xmint+Xmaxt+1)*(q_s_max+1)*(q_r_max[1]+1)*(q_r_max[2]+1)*(q_r_max[3]+1)
  }
  if (aa==5)
  {
    Xmint=Xmint-dmax[aa-1]
    Xmaxt=Xmaxt+q_s_max
    mininv[aa]=Xmint
    maxinv[aa]=Xmaxt
    q_r_max[aa]=floor(dmax[aa]*(1.5))
    row_size[aa]=(-Xmint+Xmaxt+1)*(q_s_max+1)*(q_r_max[1]+1)*(q_r_max[2]+1)*(q_r_max[3]+1)*(q_r_max[3]+1)
  }
}

if (leadtime==1)
{
  Xmint=0
  Xmaxt=0
  mininv=0
  maxinv=0
  q_r_max=floor(dmax[1]*(1.5))
  row_size=(q_s_max+1)
}

for (bb in (leadtime+1):t) ##bb is period
{
  if (leadtime==1)
  {
    Xmint=Xmint-dmax[bb-1]
```

```
    Xmaxt=Xmaxt+q_s_max+q_r_max[bb-leadtime]
    mininv[bb]=Xmint
    maxinv[bb]=Xmaxt
    q_r_max[bb]=floor(dmax[bb]*(1.5))
    row_size[bb]=(-Xmint+Xmaxt+1)*(q_s_max+1)
  }
  if (leadtime==2)
  {
    Xmint=Xmint-dmax[bb-1]
    Xmaxt=Xmaxt+q_s_max+q_r_max[bb-leadtime]
    mininv[bb]=Xmint
    maxinv[bb]=Xmaxt
    q_r_max[bb]=floor(dmax[bb]*(1.5))
    row_size[bb]=(-Xmint+Xmaxt+1)*(q_s_max+1)*c(q_r_max[bb-1]+1)
  }
  if (leadtime==3)
  {
    Xmint=Xmint-dmax[bb-1]
    Xmaxt=Xmaxt+q_s_max+q_r_max[bb-leadtime]
    mininv[bb]=Xmint
    maxinv[bb]=Xmaxt
    q_r_max[bb]=floor(dmax[bb]*(1.5))
    row_size[bb]=(-Xmint+Xmaxt+1)*(q_s_max+1)*c(q_r_max[bb-2]+1)*
 c(q_r_max[bb-1]+1)
  }
  if (leadtime==4)
  {
    Xmint=Xmint-dmax[bb-1]
    Xmaxt=Xmaxt+q_s_max+q_r_max[bb-leadtime]
    mininv[bb]=Xmint
    maxinv[bb]=Xmaxt
    q_r_max[bb]=floor(dmax[bb]*(1.5))
    row_size[bb]=(-Xmint+Xmaxt+1)*(q_s_max+1)*c(q_r_max[bb-3]+1)*
 c(q_r_max[bb-2]+1)*c(q_r_max[bb-1]+1)
  }
  if (leadtime==5)
  {
    Xmint=Xmint-dmax[bb-1]
    Xmaxt=Xmaxt+q_s_max+q_r_max[bb-leadtime]
    mininv[bb]=Xmint
    maxinv[bb]=Xmaxt
    q_r_max[bb]=floor(dmax[bb]*(1.5))
    row_size[bb]=(-Xmint+Xmaxt+1)*(q_s_max+1)*c(q_r_max[bb-4]+1)*
 c(q_r_max[bb-3]+1)*c(q_r_max[bb-2]+1)*c(q_r_max[bb-1]+1)
  }
}

state_number=cumsum(row_size)

if (leadtime==1)
{
  maxpipe=0
}else
{
  maxpipe=matrix(0, nrow=t, ncol=(leadtime-1))
  for (l in 2:leadtime)
  {
    if (l==2)
    {
      maxpipe[l,leadtime-1]=q_r_max[1]
```

```
      }
      if (l==3)
      {
        maxpipe[l,leadtime-2]=q_r_max[1]
        maxpipe[l,leadtime-1]=q_r_max[2]
      }
      if (l==4)
      {
        maxpipe[l,leadtime-3]=q_r_max[1]
        maxpipe[l,leadtime-2]=q_r_max[2]
        maxpipe[l,leadtime-1]=q_r_max[3]
      }
      if (l==5)
      {
        maxpipe[l,leadtime-4]=q_r_max[1]
        maxpipe[l,leadtime-3]=q_r_max[2]
        maxpipe[l,leadtime-2]=q_r_max[3]
        maxpipe[l,leadtime-1]=q_r_max[4]
      }
    }

    transpose.qrmax=as.matrix(q_r_max)
    maxpipe=cbind(maxpipe,transpose.qrmax)

    for (m in (leadtime+1):t)
    {
      for (n in 1:(leadtime-1))
      {
        maxpipe[m,n]=maxpipe[m-1,n+1]
      }
    }
    maxpipe=maxpipe[,-leadtime]
  }
  return(list(mininv=mininv, maxinv=maxinv, maxpipe=maxpipe,
   rowsize=row_size, statenum=state_number, dmaxs=dmax))
}
```

The code block given below is developed using RCPP to expedite demand simulation process for declining installed base.

```
//Name of this file: RcppFunction_v2.cpp
#include<Rcpp.h>
#include <numeric>    // for std::partial_sum
//#include"\Users\Mustafa Hekimoglu\source\Mylib\boncuk.h"
using namespace Rcpp;

template <class T> const T& minoftwo(const T& a, const T& b)
{
  //MAX OF TWO VARIABLES
  return (a > b) ? b : a;     // or: return comp(a,b)?b:a; for
   version (2)
}

NumericVector cumsum1(NumericVector x)
{
  double acc = 0; NumericVector res(x.size());

  for(int i = 0; i < x.size(); i++){
```

```
    acc += x[i];
    res[i] = acc;
  }
  return res;
}

// [[Rcpp::export]]
NumericVector func(NumericVector arrivaltimes,int nn, int i, int simpartarrive_main, double alpha, int simpartarrive, bool flg)
{
  NumericVector interarrive; NumericVector sparesarrive; int k; NumericVector randomnums;
  while(nn<arrivaltimes(i-1))
  {
    if(nn==0)
    {
      interarrive=Rcpp::rexp(simpartarrive_main,alpha);
      sparesarrive=cumsum1(interarrive);
      nn=sparesarrive(sparesarrive.length()-1);
    } else if(nn>0)
    {
      randomnums=Rcpp::rexp(simpartarrive,alpha);
      for(k=0; k<randomnums.length(); k++)
      {
        interarrive.push_back(randomnums(k));
      }
      sparesarrive=cumsum1(interarrive);
      nn=sparesarrive(sparesarrive.length()-1);
      flg=TRUE;
    }
  }
  if(flg==TRUE)
  {
    simpartarrive_main=sparesarrive.length();
  }
  return sparesarrive[sparesarrive<arrivaltimes(i-1)];
}

// [[Rcpp::export]]
Rcpp::NumericVector demand_counterv3(Rcpp::NumericVector sparearrivalsall , Rcpp::NumericVector Tvect)
{
  int i=0;
  Rcpp::NumericVector res=runif(Tvect.size(),0,0.001);
  int cnt=0;int k=0; bool flg=TRUE;

  int dimens= (sparearrivalsall.size()-1);

  for(i=0;i<Tvect.size();i++)
  {
    if(flg)
    {
      while((sparearrivalsall(k)<=Tvect(i)) && (k<=(sparearrivalsall.size()-1)))
      {
        k=minoftwo(k+1,dimens);
        cnt = minoftwo(cnt+1,dimens+1);

        if(k>=(sparearrivalsall.size()-1))
        {
```

```
          flg=TRUE;
          if(sparearrivalsall(k)<= Tvect(i))
            cnt = minoftwo(cnt+1,dimens+1);
          break;
        }
      }
    }
    res(i)=cnt;
  }
  return(res);
}
```

The code block given below is developed using R to read saved demand paths and optimum actions and calculate costs.

```
setwd("C:\\Users\\suser\\Desktop\\OrdvsAdan\\Declining_IB")
getwd()

Declining_IB=T
N0=25
stepsize=1
seedvalue=NULL
replication=1000
strategy=1

library(rmutil)
library(PEIP)
library(Rcpp)
source("DynamicDistSelect_OptActsStats_Ord&Adan.R")
if (Declining_IB)
{
  sourceCpp("RcppFunction_v2.cpp")
}

sourceCpp("Simulation_OptActs_v5.cpp")

parameters=read.csv("optimization_parameters_v2.csv")
range=c(43,47)
for (dist_select_algrthm in c("Ord","Adan","Pure Pois"))#
{
  replicsize=replication
  for (cc in range)
  {
    if(Declining_IB)
    {
      OptActs <- read.csv(sprintf("C:\\Users\\suser\\Desktop\\OrdvsAdan\\Declining_IB\\OptimizationResultFiles\\Result_%s_DecliningIB_Cap(%s)_Strtgy(%s)_%s.csv",cc,capacity,strategy,dist_select_algrthm),header = F)
    }else
    {
      OptActs <- read.csv(sprintf("C:\\Users\\suser\\Desktop\\OrdvsAdan\\Growing_IB\\OptimizationResultFiles\\Result_%s_GrowingIB_Cap(%s)_Strtgy(%s)_%s.csv",cc,capacity,strategy,dist_select_algrthm),header = F)
    }

    lambda=parameters[cc,1]
```

```
 alpha=parameters[cc,2]
 h=parameters[cc,3]
 b=parameters[cc,4]
 eta=parameters[cc,5]
 xi=parameters[cc,6]
 simlength=parameters[cc,7]
 discountrate=parameters[cc,8]
 LT=parameters[cc,9]

 step_size = stepsize
 cr=5

 if(capacity)
 {
   if (market_scenario==1)
   {
     q_s_max=12 #max order size for secondary supp.
     P=as.matrix(read.csv("Keq12_Symmetric_TPM.csv",sep = ";", header = F))
   }else if(market_scenario==2)
   {
     q_s_max=12 #max order size for secondary supp.
     P=as.matrix(read.csv("Keq12_Increasing_TPM.csv",sep = ";", header = F))
   }else if(market_scenario==3)
   {
     q_s_max=12 #max order size for secondary supp.
     P=as.matrix(read.csv("Keq12_Decreasing_TPM.csv",sep = ";", header = F))
   }

   WW=upper.tri(matrix(1,13,13))*matrix(1,13,13)+diag(13)#upper.tri(matrix(1,5,5))*matrix(1,5,5)+diag(5)
   P.cum=P%*%WW
 }else if(capacity==F)
 {
   P=matrix(1,1,1)
   WW=upper.tri(matrix(1,1,1))*matrix(1,1,1)+diag(1)#upper.tri(matrix(1,5,5))*matrix(1,5,5)+diag(5)
   P.cum=P%*%WW
   q_s_max=0
 }

 for (sensivity_level in c(0,0.05,0.25))
 {

 if (Declining_IB)
 {
   load(sprintf("C:\\Users\\suser\\Desktop\\OrdvsAdan\\DecliningRepo\\Demand_%s_DecliningIB_SensLevel_%s.Rdata",cc,sensivity_level))
 }else
 {
   load(sprintf("C:\\Users\\suser\\Desktop\\OrdvsAdan\\GrowingRepo\\Demand_%s_GrowingIB_SensLevel_%s.Rdata",cc,sensivity_level))
 }

 maxdemandperperiod=sapply(as.data.frame(marginaldemand),max)

```

```
  costmatrix=matrix(NA,replicsize,5)
  if(capacity)
  {
    #each row represents a replication
    set.seed(seedvalue)
    Kmatrix=matrix(NA,replicsize,simlength)
    for (k in 1:replicsize)
    {
      K_vect=0
      r.num=runif(simlength)
      for(i in 1:simlength-1)
      {
        K_vect[i+1]=sum(P.cum[K_vect[i]+1,]<r.num[i])
      }
      Kmatrix[k,]=K_vect
    }
  }else
  {
    Kmatrix=matrix(0,replicsize,simlength)
  }

  periods=c(0,simlength)

  List=ReturnList(lambda, alpha, simlength, LT, stepsize,
strategy, capacity, Declining_IB, N0, dist_select_algrthm)

  mininv=List$mininv
  maxinv=List$maxinv
  maxpipe=List$maxpipe
  rowsize=List$rowsize
  rowsize=rev(rowsize)
  statenum=List$statenum
  dmaxs=List$dmaxs

  costmatrix=simulateOptacts(marginaldemand, Kmatrix, simlength,
LT, h, b, cr, xi, eta, discountrate, replicsize, periods,
                              q_s_max, mininv, maxinv, as.matrix(
maxpipe), as.matrix(OptActs), rowsize)

  costmatrix=as.data.frame(costmatrix)
  res=c(mean(as.matrix(costmatrix[1,])),mean(as.matrix(costmatrix
[2,])),mean(as.matrix(costmatrix[3,])),mean(as.matrix(costmatrix
[4,])),mean(as.matrix(costmatrix[5,])))
  stddev=sd(as.matrix(costmatrix[5,]))
  hw=stddev*tinv(0.99,replication-1)/sqrt(replication)

  if(Declining_IB)
  {
    costs=cbind(sprintf("%s_DecliningIB_Cap(%s)_Strtgy(%s)_%s_
SensLevel_%s",cc,capacity,strategy,dist_select_algrthm,sensivity
_level),res[1],res[2],res[3],res[4],res[5],hw)
  }else
  {
    costs=cbind(sprintf("%s_GrowingIB_Cap(%s)_Strtgy(%s)_%s_
SensLevel_%s",cc,capacity,strategy,dist_select_algrthm,sensivity
_level),res[1],res[2],res[3],res[4],res[5],hw)
  }
  write.table(costs,"OptActs_Results.txt",sep = "\t",col.names =
F,row.names = F,append = T)
  print(sprintf("Parameter:%s %s Sensivity Level:%s completed!",
```

```
cc,dist_select_algrthm,sensivity_level))
  }
}
}
```

The following code block is developed using RCPP to expedite calculations.

```
//Name of this file: Simulation_OptActs_v5
#include<Rcpp.h>
#include <math.h>
#include <vector>
#include "Heuristics.h"
using namespace Rcpp;

// [[Rcpp::export]]
NumericMatrix simulateOptacts(NumericMatrix Demandmatrix, NumericMatrix Kmatrix, int simlength, int LT,
                              double h, double b, double cr, double xi, double eta, double discountrate,
                              int replength, NumericVector periods, int q_s_max, NumericVector mininv,
                              NumericVector maxinv, NumericMatrix maxpipe, NumericMatrix optacts, NumericVector rowsize)
{
  int rep; NumericVector costvect; NumericMatrix result(5,replength);
  NumericVector costrep=0; NumericVector costrep_acqreg=0;
  NumericVector costrep_acqsec=0; NumericVector costrep_hold=0;
  NumericVector costrep_backl=0; NumericVector Kvect; NumericVector Demandvect;
  NumericVector IP;
  int qr; int qs; int index; int i; int indexold; int zz; int kk; int k; int n;
  NumericVector res; int inv; int tt;

   for(rep=0; rep<replength; rep++)
   {
     NumericVector Qr; NumericVector Qs;
     NumericVector I; I.push_back(0);

     kk=std::max(1,LT-1);
     NumericVector pipeline(kk);//To avoid pipeline to become a vector of zero when LT=1

     Demandvect=Demandmatrix(rep,_);
     Kvect=Kmatrix(rep,_);
     for(tt=1; tt<=simlength; tt++)
     {
       if(I(tt-1)<mininv(tt-1) || I(tt-1)>maxinv(tt-1))
       {
         Rcout<<"ERROR:Inventory is out of bounds!"<<std::endl;//Inventory level is out of state space of optimization
         Rcout<<"Replication:"<<rep+1<<" Period:"<<tt<<" Demand:"<<Demandvect(tt-1)<<" Inventory:"<<I(tt-1)<<std::endl;
         break;
       }
       index= match_cppOptActs_v2(simlength, tt-1, I(tt-1), Kvect(tt-1), q_s_max, mininv, maxinv, maxpipe, pipeline, LT, rowsize);
       qs=optacts(index,2);
```

```
        qr=optacts(index,1);
        Qs.push_back(qs);
        Qr.push_back(qr);
        zz=std::max(0,LT-2);//To avoid index from becoming negative
    when LT=1(1-2=-1)
        for(i=0; i<zz; i++)
        {
          pipeline(i)=pipeline(i+1);
        }
        pipeline(zz)=qr;
        if(tt<LT)
        {
          inv = I(tt-1) - Demandvect(tt-1) + Qs(tt-1);
          I.push_back(inv);
        }
        if (tt>=LT)
        {
          inv = I(tt-1) + Qr(tt-LT) - Demandvect(tt-1) + Qs(tt-1);
          I.push_back(inv);
        }
      }
      costvect=costcalc(Demandvect, Kvect, h, b, cr, xi, eta,
   discountrate, I, IP, Qs, Qr, simlength);
      costrep_acqreg.push_back(costvect(0));
      costrep_acqsec.push_back(costvect(1));
      costrep_hold.push_back(costvect(3));
      costrep_backl.push_back(costvect(4));
      costrep.push_back(costvect(5));
    }
  result(0,_)=costrep_acqreg;
  result(1,_)=costrep_acqsec;
  result(2,_)=costrep_hold;
  result(3,_)=costrep_backl;
  result(4,_)=costrep;

  return result;
}
```

APPENDIX B: CODES AND RESULTS OF THE STATISTICAL TESTS

In this chapter, the code blocks developed using Python for respective test procedures and the test results are provided. Each statistical test is presented in a separate section. Also, the results of each statistical test are given in different subsections for cyclic and successive intervals.

B.1 Kolmogorov-Smirnov Test

Codes blocks and results of the KS Tests for periodic and consecutive time intervals are given in the following subsections. The test results are presented for three instances by converting the inter-sales times into seconds, minutes and hours. Following code block is used for this conversion:

```
import datetime

def convert_to_hours(td):
  return (td.dt.days*24)+(td.dt.seconds/3600)

def convert_to_minutes(td):
  return (td.dt.days*24*60)+(td.dt.seconds/60)

def convert_to_seconds(td):
  return (td.dt.days*24*60*60)+(td.dt.seconds)
```

To implement the KS Test, the ks_test function of the scipy library's stats module is used. In addition, the libraries in the code block below are also imported for data manipulation.

```
  from scipy import stats
  import pandas as pd
  import numpy as np
```

B.1.1 Cyclic time intervals

The code block developed for KS Test using weekdays as cyclic time intervals is given below, and the test results are given in Table B.1.

```python
test_stat_sec=[]
test_stat_min=[]
test_stat_hour=[]
p_val_sec=[]
p_val_min=[]
p_val_hour=[]
days=["Mon","Tue","Wed","Thu","Fri","Sat","Sun"]
for day in range(7):
    intersales_time=[]
    a=sales_after2005.loc[sales_after2005.index.dayofweek==day]
    a.reset_index(drop=True, inplace=True)

    for i in range(a.index.max()):
        if a.selling_date.loc[i+1].day-a.selling_date.loc[i].day>0:
            continue
        else:
            intersales_time.append(a.invoice_date.loc[i+1]-a.invoice_date.loc[i])

    intersales_time=pd.Series(intersales_time)
    kstest_res_sec=kstest(convert_to_seconds(intersales_time), 'expon')
    kstest_res_hour=kstest(convert_to_hours(intersales_time), 'expon')
    kstest_res_min=kstest(convert_to_minutes(intersales_time), 'expon')

    test_stat_sec.append(kstest_res_sec[0])
    test_stat_min.append(kstest_res_min[0])
    test_stat_hour.append(kstest_res_hour[0])
    p_val_sec.append(kstest_res_sec[1])
    p_val_min.append(kstest_res_min[1])
    p_val_hour.append(kstest_res_hour[1])
pd.DataFrame(list(zip(days,test_stat_sec, test_stat_min, test_stat_hour, p_val_sec, p_val_min, p_val_hour)), columns=['days','test_stat_sec', 'test_stat_min', 'test_stat_hour', 'p_val_sec', 'p_val_min', 'p_val_hour'])
```

Table B.1 Results of the KS Test when using weekdays as cyclic time intervals.

	Test Statistic			P-value		
Weekdays	Seconds	Minutes	Hours	Seconds	Minutes	Hours
Mon	9.67E-01	4.09E-01	6.24E-01	0.00E+00	0.00E+00	0.00E+00
Tue	9.69E-01	4.24E-01	6.08E-01	0.00E+00	0.00E+00	0.00E+00
Wed	9.72E-01	4.22E-01	5.97E-01	0.00E+00	0.00E+00	0.00E+00
Thu	9.70E-01	4.31E-01	5.96E-01	0.00E+00	0.00E+00	0.00E+00
Fri	9.70E-01	4.17E-01	5.96E-01	0.00E+00	0.00E+00	0.00E+00
Sat	9.49E-01	3.00E-01	6.43E-01	0.00E+00	0.00E+00	0.00E+00
Sun	9.53E-01	3.18E-01	6.67E-01	4.51E-204	2.92E-14	1.11E-67

The code block developed for KS Test using weeks of the year as cyclic time intervals is given below, and the test results are given in Table B.2.

```
test_stat_sec=[]
test_stat_min=[]
test_stat_hour=[]
p_val_sec=[]
p_val_min=[]
p_val_hour=[]
weeks=range(1,54)
for week in range(1,54):
    intersales_time=[]
    a=sales_after2005.loc[sales_after2005.index.isocalendar().week
==week]
    a.reset_index(drop=True, inplace=True)

    for i in range(a.index.max()):
        if a.selling_date.loc[i+1].year-a.selling_date.loc[i].year
>0:
            continue
        else:
            intersales_time.append(a.invoice_date.loc[i+1]-a.
invoice_date.loc[i])

    intersales_time=pd.Series(intersales_time)
    kstest_res_sec=kstest(convert_to_seconds(intersales_time), '
expon')
    kstest_res_hour=kstest(convert_to_hours(intersales_time), '
expon')
    kstest_res_min=kstest(convert_to_minutes(intersales_time), '
expon')

    test_stat_sec.append(kstest_res_sec[0])
    test_stat_min.append(kstest_res_min[0])
    test_stat_hour.append(kstest_res_hour[0])
    p_val_sec.append(kstest_res_sec[1])
    p_val_min.append(kstest_res_min[1])
    p_val_hour.append(kstest_res_hour[1])
```

```python
pd.DataFrame(list(zip(weeks, test_stat_sec, test_stat_min, test_stat_hour, p_val_sec, p_val_min, p_val_hour)), columns=['weeks','test_stat_sec', 'test_stat_min', 'test_stat_hour', 'p_val_sec', 'p_val_min', 'p_val_hour'])
```

Table B.2 Results of the KS Test when using weeks of the year as cyclic time intervals.

	Test Statistic			P-value		
Weeks	Seconds	Minutes	Hours	Seconds	Minutes	Hours
1	9.77E-01	4.61E-01	5.75E-01	0.00E+00	0.00E+00	0.00E+00
2	9.86E-01	6.14E-01	4.42E-01	0.00E+00	0.00E+00	4.41E-279
3	9.88E-01	5.94E-01	4.13E-01	0.00E+00	0.00E+00	1.31E-272
4	9.86E-01	6.08E-01	4.06E-01	0.00E+00	0.00E+00	3.91E-281
5	9.86E-01	5.56E-01	4.56E-01	0.00E+00	0.00E+00	0.00E+00
6	9.90E-01	6.40E-01	4.38E-01	0.00E+00	0.00E+00	0.00E+00
7	9.75E-01	5.35E-01	5.16E-01	0.00E+00	0.00E+00	0.00E+00
8	9.80E-01	4.97E-01	5.47E-01	0.00E+00	0.00E+00	0.00E+00
9	9.48E-01	2.87E-01	6.75E-01	0.00E+00	0.00E+00	0.00E+00
10	9.82E-01	4.61E-01	5.59E-01	0.00E+00	0.00E+00	0.00E+00
11	9.79E-01	4.72E-01	5.91E-01	0.00E+00	0.00E+00	0.00E+00
12	9.74E-01	4.41E-01	5.95E-01	0.00E+00	0.00E+00	0.00E+00
13	9.77E-01	4.02E-01	6.36E-01	0.00E+00	0.00E+00	0.00E+00
14	9.76E-01	5.38E-01	5.14E-01	0.00E+00	0.00E+00	0.00E+00
15	9.82E-01	4.88E-01	5.75E-01	0.00E+00	0.00E+00	0.00E+00
16	9.73E-01	4.86E-01	5.70E-01	0.00E+00	0.00E+00	0.00E+00
17	9.70E-01	3.93E-01	6.18E-01	0.00E+00	0.00E+00	0.00E+00
18	9.67E-01	3.11E-01	6.75E-01	0.00E+00	0.00E+00	0.00E+00
19	9.73E-01	4.14E-01	6.09E-01	0.00E+00	0.00E+00	0.00E+00
20	9.71E-01	4.24E-01	5.96E-01	0.00E+00	0.00E+00	0.00E+00
21	9.80E-01	4.71E-01	5.86E-01	0.00E+00	0.00E+00	0.00E+00
22	9.59E-01	3.32E-01	6.62E-01	0.00E+00	0.00E+00	0.00E+00
23	9.89E-01	5.06E-01	5.29E-01	0.00E+00	0.00E+00	0.00E+00

Table B.2 continued from previous page

	Test Statistic			P-value		
24	9.68E-01	3.88E-01	6.08E-01	0.00E+00	0.00E+00	0.00E+00
25	9.74E-01	4.42E-01	5.80E-01	0.00E+00	0.00E+00	0.00E+00
26	9.66E-01	3.97E-01	6.13E-01	0.00E+00	0.00E+00	0.00E+00
27	9.77E-01	4.35E-01	5.78E-01	0.00E+00	0.00E+00	0.00E+00
28	9.72E-01	4.43E-01	5.79E-01	0.00E+00	0.00E+00	0.00E+00
29	9.83E-01	5.02E-01	5.63E-01	0.00E+00	0.00E+00	0.00E+00
30	9.62E-01	3.81E-01	6.44E-01	0.00E+00	0.00E+00	0.00E+00
31	9.64E-01	4.35E-01	5.77E-01	0.00E+00	0.00E+00	0.00E+00
32	9.81E-01	5.49E-01	5.12E-01	0.00E+00	0.00E+00	0.00E+00
33	9.77E-01	4.86E-01	5.75E-01	0.00E+00	0.00E+00	0.00E+00
34	9.78E-01	4.24E-01	6.10E-01	0.00E+00	0.00E+00	0.00E+00
35	9.76E-01	3.87E-01	6.30E-01	0.00E+00	0.00E+00	0.00E+00
36	9.90E-01	5.67E-01	4.69E-01	0.00E+00	0.00E+00	0.00E+00
37	9.87E-01	5.41E-01	4.92E-01	0.00E+00	0.00E+00	0.00E+00
38	9.83E-01	4.71E-01	5.57E-01	0.00E+00	0.00E+00	0.00E+00
39	9.51E-01	3.01E-01	6.66E-01	0.00E+00	0.00E+00	0.00E+00
40	9.67E-01	3.89E-01	6.24E-01	0.00E+00	0.00E+00	0.00E+00
41	9.80E-01	5.13E-01	5.45E-01	0.00E+00	0.00E+00	0.00E+00
42	9.58E-01	3.94E-01	6.22E-01	0.00E+00	0.00E+00	0.00E+00
43	9.63E-01	3.79E-01	6.40E-01	0.00E+00	0.00E+00	0.00E+00
44	9.66E-01	3.52E-01	6.62E-01	0.00E+00	0.00E+00	0.00E+00
45	9.88E-01	5.44E-01	5.26E-01	0.00E+00	0.00E+00	0.00E+00
46	9.74E-01	4.81E-01	5.72E-01	0.00E+00	0.00E+00	0.00E+00
47	9.82E-01	4.78E-01	5.89E-01	0.00E+00	0.00E+00	0.00E+00
48	9.44E-01	3.25E-01	6.72E-01	0.00E+00	0.00E+00	0.00E+00
49	9.42E-01	3.61E-01	6.22E-01	0.00E+00	0.00E+00	0.00E+00
50	9.85E-01	4.85E-01	5.95E-01	0.00E+00	0.00E+00	0.00E+00
51	9.79E-01	4.65E-01	6.10E-01	0.00E+00	0.00E+00	0.00E+00
52	9.51E-01	3.27E-01	6.82E-01	0.00E+00	0.00E+00	0.00E+00

Table B.2 continued from previous page

	Test Statistic			P-value		
53	9.44E-01	2.12E-01	7.88E-01	0.00E+00	1.15E-105	0.00E+00

The code block developed for KS Test using months as cyclic time intervals is given below, and the test results are given in Table B.3.

```
test_stat_sec=[]
test_stat_min=[]
test_stat_hour=[]
p_val_sec=[]
p_val_min=[]
p_val_hour=[]
months=["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"]
for month in range(1,13):
    intersales_time=[]
    a=sales_after2005.loc[sales_after2005.index.month==month]
    a.reset_index(drop=True, inplace=True)

    for i in range(a.index.max()):
        if a.selling_date.loc[i+1].year-a.selling_date.loc[i].year>0:
            continue
        else:
            intersales_time.append(a.invoice_date.loc[i+1]-a.invoice_date.loc[i])

    intersales_time=pd.Series(intersales_time)
    kstest_res_sec=kstest(convert_to_seconds(intersales_time), 'expon')
    kstest_res_hour=kstest(convert_to_hours(intersales_time), 'expon')
    kstest_res_min=kstest(convert_to_minutes(intersales_time), 'expon')

    test_stat_sec.append(kstest_res_sec[0])
    test_stat_min.append(kstest_res_min[0])
    test_stat_hour.append(kstest_res_hour[0])
    p_val_sec.append(kstest_res_sec[1])
    p_val_min.append(kstest_res_min[1])
    p_val_hour.append(kstest_res_hour[1])
pd.DataFrame(list(zip(months,test_stat_sec, test_stat_min, test_stat_hour, p_val_sec, p_val_min, p_val_hour)), columns=['months','test_stat_sec', 'test_stat_min', 'test_stat_hour', 'p_val_sec', 'p_val_min', 'p_val_hour'])
```

Table B.3 Results of the KS Test when using months as cyclic time intervals.

	Test Statistic			P-value		
Months	Seconds	Minutes	Hours	Seconds	Minutes	Hours
Jan	0.99	0.59	0.43	0	0	0
Feb	0.97	0.45	0.55	0	0	0
Mar	0.98	0.44	0.60	0	0	0
Apr	0.97	0.43	0.59	0	0	0
May	0.97	0.40	0.61	0	0	0
Jun	0.97	0.42	0.59	0	0	0
Jul	0.97	0.43	0.59	0	0	0
Aug	0.98	0.47	0.57	0	0	0
Sep	0.97	0.40	0.59	0	0	0
Oct	0.97	0.41	0.61	0	0	0
Nov	0.97	0.42	0.61	0	0	0
Dec	0.96	0.38	0.64	0	0	0

The code block developed for KS Test using quarters as cyclic time intervals is given below, and the test results are given in Table B.4.

```
test_stat_sec=[]
test_stat_min=[]
test_stat_hour=[]
p_val_sec=[]
p_val_min=[]
p_val_hour=[]
quarters=range(1,5)
for quarter in range(1,5):
    intersales_time=[]
    a=sales_after2005.loc[sales_after2005.index.quarter==quarter]
    a.reset_index(drop=True, inplace=True)

    for i in range(a.index.max()):
        if a.selling_date.loc[i+1].year-a.selling_date.loc[i].year>0:
            continue
        else:
            intersales_time.append(a.invoice_date.loc[i+1]-a.invoice_date.loc[i])

    intersales_time=pd.Series(intersales_time)
    kstest_res_sec=kstest(convert_to_seconds(intersales_time), 'expon')
    kstest_res_hour=kstest(convert_to_hours(intersales_time), 'expon')
```

```
    kstest_res_min=kstest(convert_to_minutes(intersales_time), '
expon')

    test_stat_sec.append(kstest_res_sec[0])
    test_stat_min.append(kstest_res_min[0])
    test_stat_hour.append(kstest_res_hour[0])
    p_val_sec.append(kstest_res_sec[1])
    p_val_min.append(kstest_res_min[1])
    p_val_hour.append(kstest_res_hour[1])
pd.DataFrame(list(zip(quarters, test_stat_sec, test_stat_min,
    test_stat_hour, p_val_sec, p_val_min, p_val_hour)), columns=['
    quarters','test_stat_sec', 'test_stat_min', 'test_stat_hour', '
    p_val_sec', 'p_val_min', 'p_val_hour'])
```

Table B.4 Results of the KS Test when using quarters as cyclic time intervals.

	Test Stat			P-value		
Quarters	Seconds	Minutes	Hours	Seconds	Minutes	Hours
1	0.98	0.47	0.55	0.00	0.00	0.00
2	0.97	0.42	0.60	0.00	0.00	0.00
3	0.97	0.43	0.59	0.00	0.00	0.00
4	0.96	0.40	0.62	0.00	0.00	0.00

The code block developed for KS Test using seasons as cyclic time intervals is given below, and the test results are given in Table B.5.

```
test_stat_sec=[]
test_stat_min=[]
test_stat_hour=[]
p_val_sec=[]
p_val_min=[]
p_val_hour=[]
seasons=["autumn","winter","spring","summer"]
for season in seasons:
    if season=='autumn':
        months=[9,10,11]
    elif season=="winter":
        months=[12,1,2]
    elif season=="spring":
        months=[3,4,5]
    elif season=="summer":
        months=[6,7,8]

    intersales_time=[]
    a=sales_after2005.loc[sales_after2005.index.month.isin(months)]
    a.reset_index(drop=True, inplace=True)

    for i in range(a.index.max()):
        if a.selling_date.loc[i+1].year-a.selling_date.loc[i].year
>0:
            continue
```

```
        else:
            intersales_time.append(a.invoice_date.loc[i+1]-a.invoice_date.loc[i])

    intersales_time=pd.Series(intersales_time)
    kstest_res_sec=kstest(convert_to_seconds(intersales_time), 'expon')
    kstest_res_hour=kstest(convert_to_hours(intersales_time), 'expon')
    kstest_res_min=kstest(convert_to_minutes(intersales_time), 'expon')

    test_stat_sec.append(kstest_res_sec[0])
    test_stat_min.append(kstest_res_min[0])
    test_stat_hour.append(kstest_res_hour[0])
    p_val_sec.append(kstest_res_sec[1])
    p_val_min.append(kstest_res_min[1])
    p_val_hour.append(kstest_res_hour[1])
pd.DataFrame(list(zip(seasons, test_stat_sec, test_stat_min, test_stat_hour, p_val_sec, p_val_min, p_val_hour)), columns=['seasons','test_stat_sec', 'test_stat_min', 'test_stat_hour', 'p_val_sec', 'p_val_min', 'p_val_hour'])
```

Table B.5 Results of the KS Test when using seasons as cyclic time intervals.

	Test Stat			P-value		
Seasons	Seconds	Minutes	Hours	Seconds	Minutes	Hours
Autumn	0.97	0.41	0.60	0.00	0.00	0.00
Winter	0.96	0.43	0.58	0.00	0.00	0.00
Spring	0.97	0.42	0.60	0.00	0.00	0.00
Summer	0.97	0.44	0.58	0.00	0.00	0.00

B.1.2 Successive time intervals

The code block developed for KS Test using quarters as consecutive time intervals is given below, and the test results are given in Table B.6.

```
test_stat_sec=[]
test_stat_min=[]
test_stat_hour=[]
p_val_sec=[]
p_val_min=[]
p_val_hour=[]
quarters=[1,2,3,4]*20
quarters.append(1)
years=[2005, 2005, 2005, 2005]
for year in range(2006,2021):
        for t in range(0,4):
            years.append(year)
years.append(2021)
```

```python
for year in range(2005,2022):
    for quarter in range(1,5):
        if (year==2021) & (quarter==2):
            break

        intersales_time=[]
        a=sales_after2005.loc[(sales_after2005.index.year==year) & (sales_after2005.index.quarter==quarter)]
        a.reset_index(drop=True, inplace=True)

        for i in range(a.index.max()):
            intersales_time.append(a.invoice_date.loc[i+1]-a.invoice_date.loc[i])

        intersales_time=pd.Series(intersales_time)
        kstest_res_sec=kstest(convert_to_seconds(intersales_time), 'expon')
        kstest_res_hour=kstest(convert_to_hours(intersales_time), 'expon')
        kstest_res_min=kstest(convert_to_minutes(intersales_time), 'expon')

        test_stat_sec.append(kstest_res_sec[0])
        test_stat_min.append(kstest_res_min[0])
        test_stat_hour.append(kstest_res_hour[0])
        p_val_sec.append(kstest_res_sec[1])
        p_val_min.append(kstest_res_min[1])
        p_val_hour.append(kstest_res_hour[1])

result=pd.DataFrame(list(zip(years, quarters, test_stat_sec, test_stat_min, test_stat_hour, p_val_sec, p_val_min, p_val_hour)), columns=['years', 'quarters','test_stat_sec', 'test_stat_min', 'test_stat_hour', 'p_val_sec', 'p_val_min', 'p_val_hour'])
pd.set_option('display.max_rows', result.shape[0]+1)
result
```

Table B.6 Results of the KS Test when using quarters as successive time intervals.

		Test Statistic			P-value		
Years	Quarters	Seconds	Minutes	Hours	Seconds	Minutes	Hours
2005	1	0.99	0.74	0.30	0.00	0.00	0.00
2005	2	0.99	0.69	0.42	0.00	0.00	0.00
2005	3	0.99	0.67	0.45	0.00	0.00	0.00
2005	4	0.99	0.63	0.51	0.00	0.00	0.00
2006	1	0.99	0.66	0.41	0.00	0.00	0.00
2006	2	0.99	0.69	0.37	0.00	0.00	0.00
2006	3	0.99	0.77	0.25	0.00	0.00	0.00
2006	4	0.99	0.62	0.41	0.00	0.00	0.00

Table B.6 continued from previous page

		Test Statistic			P-value		
2007	1	0.99	0.78	0.25	0.00	0.00	0.00
2007	2	0.99	0.68	0.29	0.00	0.00	0.00
2007	3	0.99	0.72	0.31	0.00	0.00	0.00
2007	4	0.99	0.57	0.45	0.00	0.00	0.00
2008	1	1.00	0.66	0.34	0.00	0.00	0.00
2008	2	0.99	0.75	0.24	0.00	0.00	0.00
2008	3	0.99	0.70	0.25	0.00	0.00	0.00
2008	4	0.99	0.61	0.36	0.00	0.00	0.00
2009	1	0.99	0.53	0.49	0.00	0.00	0.00
2009	2	0.99	0.71	0.31	0.00	0.00	0.00
2009	3	0.96	0.28	0.73	0.00	0.00	0.00
2009	4	0.99	0.71	0.29	0.00	0.00	0.00
2010	1	0.99	0.56	0.43	0.00	0.00	0.00
2010	2	0.98	0.49	0.49	0.00	0.00	0.00
2010	3	0.99	0.60	0.41	0.00	0.00	0.00
2010	4	0.98	0.45	0.56	0.00	0.00	0.00
2011	1	0.98	0.54	0.49	0.00	0.00	0.00
2011	2	0.97	0.45	0.61	0.00	0.00	0.00
2011	3	0.98	0.48	0.55	0.00	0.00	0.00
2011	4	0.98	0.40	0.63	0.00	0.00	0.00
2012	1	0.98	0.46	0.58	0.00	0.00	0.00
2012	2	0.97	0.38	0.62	0.00	0.00	0.00
2012	3	0.98	0.35	0.61	0.00	0.00	0.00
2012	4	0.96	0.33	0.70	0.00	0.00	0.00
2013	1	0.97	0.33	0.66	0.00	0.00	0.00
2013	2	0.98	0.34	0.67	0.00	0.00	0.00
2013	3	0.97	0.34	0.68	0.00	0.00	0.00
2013	4	0.97	0.37	0.67	0.00	0.00	0.00
2014	1	0.96	0.37	0.62	0.00	0.00	0.00

Table B.6 continued from previous page

		Test Statistic			P-value		
2014	2	0.97	0.34	0.65	0.00	0.00	0.00
2014	3	0.98	0.46	0.62	0.00	0.00	0.00
2014	4	0.98	0.44	0.66	0.00	0.00	0.00
2015	1	0.97	0.35	0.67	0.00	0.00	0.00
2015	2	0.97	0.37	0.68	0.00	0.00	0.00
2015	3	0.98	0.44	0.61	0.00	0.00	0.00
2015	4	0.96	0.34	0.67	0.00	0.00	0.00
2016	1	0.97	0.40	0.64	0.00	0.00	0.00
2016	2	0.97	0.31	0.71	0.00	0.00	0.00
2016	3	0.96	0.35	0.70	0.00	0.00	0.00
2016	4	0.93	0.26	0.72	0.00	0.00	0.00
2017	1	0.95	0.48	0.54	0.00	0.00	0.00
2017	2	0.95	0.40	0.64	0.00	0.00	0.00
2017	3	0.93	0.32	0.67	0.00	0.00	0.00
2017	4	0.96	0.43	0.64	0.00	0.00	0.00
2018	1	0.97	0.58	0.47	0.00	0.00	0.00
2018	2	0.98	0.45	0.60	0.00	0.00	0.00
2018	3	0.99	0.60	0.45	0.00	0.00	0.00
2018	4	0.98	0.56	0.45	0.00	0.00	0.00
2019	1	0.99	0.64	0.36	0.00	0.00	0.00
2019	2	0.98	0.53	0.40	0.00	0.00	0.00
2019	3	0.98	0.62	0.42	0.00	0.00	0.00
2019	4	0.97	0.39	0.55	0.00	0.00	0.00
2020	1	0.99	0.55	0.42	0.00	0.00	0.00
2020	2	0.99	0.46	0.47	0.00	0.00	0.00
2020	3	0.98	0.43	0.56	0.00	0.00	0.00
2020	4	0.94	0.33	0.67	0.00	0.00	0.00

The code block developed for KS Test using seasons as successive time intervals is given below, and the test results are given in Table B.7.

```
from dateutil.relativedelta import *
from datetime import datetime

test_stat_sec=[]
test_stat_min=[]
test_stat_hour=[]
p_val_sec=[]
p_val_min=[]
p_val_hour=[]

years=[2005, 2005, 2005]
for year in range(2006,2021):
    for t in range(0,4):
        years.append(year)
years.append(2021)

seasons=["spring","summer","autumn"]
for i in range(0,15):
    for season in ["winter","spring","summer","autumn"]:
        seasons.append(season)
seasons.append("winter")

start_date=pd.Timestamp(2005,3,1)

for iteration in range(0,64):
    fin_date = start_date + relativedelta(months=+3)

    intersales_time=[]
    a=sales_after2005[(sales_after2005.invoice_date>=start_date) &
    (sales_after2005.invoice_date<fin_date)]
    a.reset_index(drop=True, inplace=True)

    start_date = fin_date

    for i in range(a.index.max()):
        intersales_time.append(a.invoice_date.loc[i+1]-a.
    invoice_date.loc[i])

    intersales_time=pd.Series(intersales_time)
    kstest_res_sec=kstest(convert_to_seconds(intersales_time), '
    expon')
    kstest_res_hour=kstest(convert_to_hours(intersales_time), '
    expon')
    kstest_res_min=kstest(convert_to_minutes(intersales_time), '
    expon')

    test_stat_sec.append(kstest_res_sec[0])
    test_stat_min.append(kstest_res_min[0])
    test_stat_hour.append(kstest_res_hour[0])
    p_val_sec.append(kstest_res_sec[1])
    p_val_min.append(kstest_res_min[1])
    p_val_hour.append(kstest_res_hour[1])

result=pd.DataFrame(list(zip(years, seasons, test_stat_sec,
    test_stat_min, test_stat_hour, p_val_sec, p_val_min, p_val_hour)
    ), columns=['years', 'seasons','test_stat_sec', 'test_stat_min',
     'test_stat_hour', 'p_val_sec', 'p_val_min', 'p_val_hour'])
pd.set_option('display.max_rows', result.shape[0]+1)
result
```

Table B.7 Results of the KS Test when using seasons as successive time intervals.

		Test Statistic			P-value		
Years	Seasons	Seconds	Minutes	Hours	Seconds	Minutes	Hours
2005	spring	0.99	0.71	0.37	0.00	0.00	0.00
2005	summer	0.99	0.67	0.45	0.00	0.00	0.00
2005	autumn	0.99	0.64	0.48	0.00	0.00	0.00
2006	winter	0.99	0.65	0.46	0.00	0.00	0.00
2006	spring	0.99	0.66	0.41	0.00	0.00	0.00
2006	summer	0.99	0.74	0.31	0.00	0.00	0.00
2006	autumn	0.99	0.77	0.25	0.00	0.00	0.00
2007	winter	0.99	0.62	0.41	0.00	0.00	0.00
2007	spring	0.99	0.73	0.25	0.00	0.00	0.00
2007	summer	1.00	0.70	0.31	0.00	0.00	0.00
2007	autumn	0.99	0.63	0.39	0.00	0.00	0.00
2008	winter	0.99	0.60	0.40	0.00	0.00	0.00
2008	spring	1.00	0.69	0.32	0.00	0.00	0.00
2008	summer	0.99	0.72	0.25	0.00	0.00	0.00
2008	autumn	0.99	0.62	0.33	0.00	0.00	0.00
2009	winter	0.99	0.68	0.28	0.00	0.00	0.00
2009	spring	0.99	0.56	0.48	0.00	0.00	0.00
2009	summer	1.00	0.57	0.42	0.00	0.00	0.00
2009	autumn	0.96	0.36	0.60	0.00	0.00	0.00
2010	winter	0.98	0.57	0.41	0.00	0.00	0.00
2010	spring	0.99	0.56	0.46	0.00	0.00	0.00
2010	summer	0.98	0.50	0.47	0.00	0.00	0.00
2010	autumn	0.99	0.51	0.48	0.00	0.00	0.00
2011	winter	0.98	0.50	0.53	0.00	0.00	0.00
2011	spring	0.97	0.43	0.61	0.00	0.00	0.00
2011	summer	0.99	0.51	0.55	0.00	0.00	0.00

Table B.7 continued from previous page

		Test Statistic			P-value		
2011	autumn	0.98	0.43	0.60	0.00	0.00	0.00
2012	winter	0.98	0.43	0.60	0.00	0.00	0.00
2012	spring	0.98	0.39	0.62	0.00	0.00	0.00
2012	summer	0.98	0.38	0.61	0.00	0.00	0.00
2012	autumn	0.96	0.29	0.71	0.00	0.00	0.00
2013	winter	0.98	0.42	0.63	0.00	0.00	0.00
2013	spring	0.97	0.31	0.69	0.00	0.00	0.00
2013	summer	0.97	0.36	0.67	0.00	0.00	0.00
2013	autumn	0.98	0.35	0.67	0.00	0.00	0.00
2014	winter	0.96	0.35	0.65	0.00	0.00	0.00
2014	spring	0.97	0.34	0.66	0.00	0.00	0.00
2014	summer	0.98	0.42	0.63	0.00	0.00	0.00
2014	autumn	0.98	0.49	0.62	0.00	0.00	0.00
2015	winter	0.97	0.40	0.66	0.00	0.00	0.00
2015	spring	0.97	0.35	0.70	0.00	0.00	0.00
2015	summer	0.98	0.40	0.64	0.00	0.00	0.00
2015	autumn	0.98	0.44	0.59	0.00	0.00	0.00
2016	winter	0.96	0.35	0.67	0.00	0.00	0.00
2016	spring	0.97	0.34	0.68	0.00	0.00	0.00
2016	summer	0.96	0.33	0.70	0.00	0.00	0.00
2016	autumn	0.95	0.31	0.71	0.00	0.00	0.00
2017	winter	0.91	0.28	0.66	0.00	0.00	0.00
2017	spring	0.95	0.42	0.63	0.00	0.00	0.00
2017	summer	0.94	0.35	0.66	0.00	0.00	0.00
2017	autumn	0.94	0.39	0.64	0.00	0.00	0.00
2018	winter	0.96	0.45	0.59	0.00	0.00	0.00
2018	spring	0.98	0.48	0.58	0.00	0.00	0.00
2018	summer	0.98	0.54	0.51	0.00	0.00	0.00
2018	autumn	0.98	0.61	0.39	0.00	0.00	0.00

Table B.7 continued from previous page

		Test Statistic			P-value		
2019	winter	0.99	0.53	0.43	0.00	0.00	0.00
2019	spring	0.98	0.60	0.38	0.00	0.00	0.00
2019	summer	0.99	0.56	0.45	0.00	0.00	0.00
2019	autumn	0.98	0.54	0.45	0.00	0.00	0.00
2020	winter	0.96	0.33	0.59	0.00	0.00	0.00
2020	spring	0.99	0.51	0.43	0.00	0.00	0.00
2020	summer	0.99	0.55	0.38	0.00	0.00	0.00
2020	autumn	0.94	0.32	0.66	0.00	0.00	0.00

The code block developed for KS Test using half years as successive time intervals is given below, and the test results are given in Table B.8.

```
test_stat_sec=[]
test_stat_min=[]
test_stat_hour=[]
p_val_sec=[]
p_val_min=[]
p_val_hour=[]
half_years=["1st","2nd"]*20
years=[2005, 2005]
for year in range(2006,2021):
        for t in range(0,2):
            years.append(year)
for year in range(2005,2021):
    for half_year in range(1,3):
        intersales_time=[]
        a=sales_after2005.loc[(sales_after2005.index.year==year) &
    (sales_after2005.index.quarter.isin([half_year,half_year+1]))]
        a.reset_index(drop=True, inplace=True)

        for i in range(a.index.max()):
            intersales_time.append(a.invoice_date.loc[i+1]-a.
    invoice_date.loc[i])

        intersales_time=pd.Series(intersales_time)
        kstest_res_sec=kstest(convert_to_seconds(intersales_time),
    'expon')
        kstest_res_hour=kstest(convert_to_hours(intersales_time), '
    expon')
        kstest_res_min=kstest(convert_to_minutes(intersales_time),
    'expon')

        test_stat_sec.append(kstest_res_sec[0])
        test_stat_min.append(kstest_res_min[0])
        test_stat_hour.append(kstest_res_hour[0])
        p_val_sec.append(kstest_res_sec[1])
        p_val_min.append(kstest_res_min[1])
```

```
        p_val_hour.append(kstest_res_hour[1])

result=pd.DataFrame(list(zip(years, half_years, test_stat_sec,
    test_stat_min, test_stat_hour, p_val_sec, p_val_min, p_val_hour)
    ), columns=['years', 'half_of_year','test_stat_sec', '
    test_stat_min', 'test_stat_hour', 'p_val_sec', 'p_val_min', '
    p_val_hour'])
pd.set_option('display.max_rows', result.shape[0]+1)
result
```

Table B.8 Results of the KS Test when using half years as successive time intervals.

		Test Statistic			P-value		
Years	Half Year	Seconds	Minutes	Hours	Seconds	Minutes	Hours
2005	1st	0.99	0.70	0.37	0.00	0.00	0.00
2005	2nd	0.99	0.68	0.43	0.00	0.00	0.00
2006	1st	0.99	0.67	0.39	0.00	0.00	0.00
2006	2nd	0.99	0.72	0.33	0.00	0.00	0.00
2007	1st	0.99	0.72	0.27	0.00	0.00	0.00
2007	2nd	0.99	0.70	0.30	0.00	0.00	0.00
2008	1st	1.00	0.70	0.29	0.00	0.00	0.00
2008	2nd	0.99	0.73	0.24	0.00	0.00	0.00
2009	1st	0.99	0.60	0.42	0.00	0.00	0.00
2009	2nd	0.97	0.43	0.54	0.00	0.00	0.00
2010	1st	0.98	0.52	0.46	0.00	0.00	0.00
2010	2nd	0.99	0.53	0.46	0.00	0.00	0.00
2011	1st	0.98	0.47	0.57	0.00	0.00	0.00
2011	2nd	0.98	0.46	0.58	0.00	0.00	0.00
2012	1st	0.98	0.42	0.60	0.00	0.00	0.00
2012	2nd	0.98	0.37	0.61	0.00	0.00	0.00
2013	1st	0.97	0.34	0.67	0.00	0.00	0.00
2013	2nd	0.97	0.34	0.68	0.00	0.00	0.00
2014	1st	0.97	0.35	0.64	0.00	0.00	0.00
2014	2nd	0.97	0.39	0.64	0.00	0.00	0.00

Table B.8 continued from previous page

		Test Statistic			P-value		
2015	1st	0.97	0.36	0.68	0.00	0.00	0.00
2015	2nd	0.97	0.40	0.65	0.00	0.00	0.00
2016	1st	0.97	0.35	0.68	0.00	0.00	0.00
2016	2nd	0.97	0.33	0.70	0.00	0.00	0.00
2017	1st	0.95	0.43	0.60	0.00	0.00	0.00
2017	2nd	0.94	0.35	0.66	0.00	0.00	0.00
2018	1st	0.97	0.50	0.55	0.00	0.00	0.00
2018	2nd	0.98	0.50	0.55	0.00	0.00	0.00
2019	1st	0.98	0.57	0.38	0.00	0.00	0.00
2019	2nd	0.98	0.57	0.41	0.00	0.00	0.00
2020	1st	0.99	0.51	0.43	0.00	0.00	0.00
2020	2nd	0.98	0.44	0.52	0.00	0.00	0.00

The code block developed for KS Test using half years as successive time intervals is given below, and the test results are given in Table B.9.

```
test_stat_sec=[]
test_stat_min=[]
test_stat_hour=[]
p_val_sec=[]
p_val_min=[]
p_val_hour=[]
years=range(2005,2021)
for year in range(2005,2021):
    intersales_time=[]
    a=sales_after2005.loc[sales_after2005.index.year==year]
    a.reset_index(drop=True, inplace=True)

    for i in range(a.index.max()):
        intersales_time.append(a.invoice_date.loc[i+1]-a.invoice_date.loc[i])

    intersales_time=pd.Series(intersales_time)
    kstest_res_sec=kstest(convert_to_seconds(intersales_time),'expon')
    kstest_res_hour=kstest(convert_to_hours(intersales_time),'expon')
    kstest_res_min=kstest(convert_to_minutes(intersales_time),'expon')

    test_stat_sec.append(kstest_res_sec[0])
    test_stat_min.append(kstest_res_min[0])
    test_stat_hour.append(kstest_res_hour[0])
```

```
    p_val_sec.append(kstest_res_sec[1])
    p_val_min.append(kstest_res_min[1])
    p_val_hour.append(kstest_res_hour[1])

pd.DataFrame(list(zip(years, test_stat_sec, test_stat_min, test_stat_hour, p_val_sec, p_val_min, p_val_hour)), columns=['years', 'test_stat_sec', 'test_stat_min', 'test_stat_hour', 'p_val_sec', 'p_val_min', 'p_val_hour'])
```

Table B.9 Results of the KS Test when using years as successive time intervals.

	Test Statistic			P-value		
Years	Seconds	Minutes	Hours	Seconds	Minutes	Hours
2005	0.99	0.67	0.44	0.00	0.00	0.00
2006	0.99	0.67	0.37	0.00	0.00	0.00
2007	0.99	0.67	0.34	0.00	0.00	0.00
2008	0.99	0.67	0.30	0.00	0.00	0.00
2009	0.98	0.51	0.47	0.00	0.00	0.00
2010	0.98	0.50	0.49	0.00	0.00	0.00
2011	0.98	0.45	0.58	0.00	0.00	0.00
2012	0.97	0.37	0.64	0.00	0.00	0.00
2013	0.97	0.35	0.67	0.00	0.00	0.00
2014	0.97	0.40	0.64	0.00	0.00	0.00
2015	0.97	0.37	0.66	0.00	0.00	0.00
2016	0.95	0.32	0.69	0.00	0.00	0.00
2017	0.95	0.39	0.64	0.00	0.00	0.00
2018	0.98	0.53	0.51	0.00	0.00	0.00
2019	0.98	0.52	0.44	0.00	0.00	0.00
2020	0.96	0.40	0.58	0.00	0.00	0.00

B.2 Böhning's Test

Codes blocks and results of the Böhning's Tests for periodic and consecutive time intervals are given in the following subsections. Following libraries are imported for data manipulation and constructing the test procedure.

```
import math
```

```python
imports pandas as pd
import statistics
import scipy.stats as stats
from dateutil.relativedelta import *
from datetime import datetime
```

The code block given below is the function that implements Böhning's test procedure.

```python
def bohning_test(sample):
  n=sample.size
  test_stat=math.sqrt((n-1)/2)*(statistics.stdev(sample)**2/
 statistics.mean(sample)-1)
  res=1-stats.norm.cdf(test_stat,0,1)
  return test_stat, res
```

Before implementing Böhning's test, it is crucial to include the days with no sales record with a sales quantity of zero. To this end, the operations given in the code block below are performed.

```python
all_sales=pd.read_csv(r"C:\Users\ali.kok\OneDrive\Desktop\Spare
 Parts Project\Model1_Sales.csv", sep='|', parse_dates=['
 selling_date', 'invoice_date', 'guaranty_start_date', '
 guaranty_end_date', 'extended_guaranty_finish_date'])
#how many unique cars sold for each date (daily basis)
sample=all_sales.groupby("selling_date").count()["Vehicle_ID"].
 to_frame()
idx = pd.date_range('1996-10-08', '2020-12-31')
sample=sample.reindex(idx, fill_value=0)
```

B.2.1 Cyclic time intervals

The code block developed for Böhning's Test using weekdays as periodic time intervals is given below, and the test results are given in Table B.10.

```python
daily_sample=sample.groupby(by=[sample.index.weekday,sample.index.
    month,sample.index.year]).sum()
days=["Mon","Tue","Wed","Thu","Fri","Sat","Sun"]
test_stat=[]
p_val=[]
for i in range(0,7):
    samp=daily_sample[daily_sample.index.isin([i], level=0)]
    s=bohning_test(samp.chassis)
    test_stat.append(s[0])
    p_val.append(s[1])
pd.DataFrame(list(zip(days,test_stat, p_val)), columns=['days', '
    test_stat', 'p_val'])
```

Table B.10 Results of the Böhning's Test when using weekdays as cyclic time intervals.

Days	Test Statistic	P-value
Mon	2455.81	0.00
Tue	2242.92	0.00
Wed	1857.60	0.00
Thu	2096.02	0.00
Fri	1972.14	0.00
Sat	114603.52	0.00
Sun	479.90	0.00

The code block developed for Böhning's Test using weeks of the year as periodic time intervals is given below, and the test results are given in Table B.11.

```
weekly_sample=sample.groupby(by=[sample.index.isocalendar().week,
    sample.index.month,sample.index.year]).sum()
weeks=range(1,54)
test_stat=[]
p_val=[]
for i in range(1,54):
    samp=weekly_sample[weekly_sample.index.isin([i], level=0)]
    s=bohning_test(samp.chassis)
    test_stat.append(s[0])
    p_val.append(s[1])
result=pd.DataFrame(list(zip(weeks,test_stat, p_val)), columns=['
    weeks', 'test_stat', 'p_val'])
result
```

Table B.11 Results of the Böhning's Test when using weeks of the year as cyclic time intervals.

Weeks	Test Statistic	P-value
1	917.44	0.00
2	437.18	0.00
3	289.51	0.00
4	431.95	0.00
5	539.94	0.00
6	221.32	0.00
7	398.59	0.00

8	475.63	0.00
9	1741.78	0.00
10	720.45	0.00
11	631.58	0.00
12	764.16	0.00
13	1379.76	0.00
14	297.16	0.00
15	683.94	0.00
16	512.65	0.00
17	1121.90	0.00
18	1721.83	0.00
19	593.34	0.00
20	418.42	0.00
21	535.24	0.00
22	1712.39	0.00
23	426.92	0.00
24	586.00	0.00
25	684.47	0.00
26	1566.73	0.00
27	563.44	0.00
28	620.50	0.00
29	555.87	0.00
30	1312.93	0.00
31	832.22	0.00
32	340.50	0.00
33	750.05	0.00
34	897.53	0.00
35	1219.16	0.00
36	312.21	0.00
37	308.57	0.00

38	521.03	0.00
39	2140.34	0.00
40	969.65	0.00
41	515.95	0.00
42	1149.64	0.00
43	915.68	0.00
44	1093.74	0.00
45	390.04	0.00
46	620.56	0.00
47	542.69	0.00
48	1425.10	0.00
49	874.22	0.00
50	530.15	0.00
51	476.88	0.00
52	1662.12	0.00
53	24628.90	0.00

The code block developed for Böhning's Test using months as cyclic time intervals is given below, and the test results are given in Table B.12.

```
monthly_sample=sample.groupby(by=[sample.index.month,sample.index.
    year]).sum()
months=["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct"
    ,"Nov","Dec"]
test_stat=[]
p_val=[]
for i in range(1,13):
    samp=monthly_sample[monthly_sample.index.isin([i], level=0)]
    s=bohning_test(samp.chassis)
    test_stat.append(s[0])
    p_val.append(s[1])
pd.DataFrame(list(zip(months,test_stat, p_val)), columns=['months',
    'test_stat', 'p_val'])
```

Table B.12 Results of the Böhning's Test when using months as cyclic time intervals.

Months	Test Statistic	P-value
Jan	30350.05	0.00
Feb	1671.55	0.00
Mar	2235.60	0.00
Apr	2528.85	0.00
May	2443.81	0.00
Jun	2379.51	0.00
Jul	2512.30	0.00
Aug	2420.59	0.00
Sep	1956.09	0.00
Oct	2849.06	0.00
Nov	2129.73	0.00
Dec	2926.00	0.00

The code block developed for Böhning's Test using quarters as cyclic time intervals is given below, and the test results are given in Table B.13.

```
quarterly_sample=sample.groupby(by=[sample.index.quarter,sample.
    index.month,sample.index.year]).sum()
quarters=range(1,5)
test_stat=[]
p_val=[]
for i in range(1,5):
     samp=quarterly_sample[quarterly_sample.index.isin([i], level=0)
    ]
     s=bohning_test(samp.chassis)
     test_stat.append(s[0])
     p_val.append(s[1])
pd.DataFrame(list(zip(quarters,test_stat, p_val)), columns=['
    quarters', 'test_stat', 'p_val'])
```

Table B.13 Results of the Böhning's Test when using quarters as cyclic time intervals.

Quarters	Test Statistic	P-value
1	21282.13218	0
2	4204.684285	0
3	3925.918222	0
4	4687.105103	0

The code block developed for Böhning's Test using seasons as cyclic time intervals is given below, and the test results are given in Table B.14.

```
test_stat=[]
p_val=[]

seasons=["autumn","winter","spring","summer"]
for season in seasons:
    if season=='autumn':
        months=[9,10,11]
    elif season=="winter":
        months=[12,1,2]
    elif season=="spring":
        months=[3,4,5]
    elif season=="summer":
        months=[6,7,8]

    samp=sample.loc[sample.index.month.isin(months)]
    s=bohning_test(samp.chassis)
    test_stat.append(s[0])
    p_val.append(s[1])

pd.DataFrame(list(zip(seasons, test_stat, p_val)), columns=['
    seasons','test_stat', 'p_val'])
```

Table B.14 Results of the Böhning's Test when using seasons as cyclic time intervals.

Seasons	Test Statistic	P-value
autumn	3319.419899	0
winter	102410.1568	0
spring	2650.323744	0
summer	2611.646408	0

B.2.2 Successive time intervals

The code block developed for Böhning's Test using fortnights as consecutive time intervals is given below, and the test results are given in Table B.15.

```
test_stat=[]
p_val=[]
dates=[]
start_date=pd.Timestamp(1996,10,18)

while start_date<=pd.Timestamp(2020,12,17):
    fin_date = start_date + relativedelta(weeks=+2)

    df=sample[(sample.index>=start_date) & (sample.index<fin_date)]
    #print(start_date)
    date='%s & %s' % (start_date, fin_date)
    start_date = fin_date

    #some samples' sales records are consist of only zeros
    #those days are given a p-value of 0
    #to prevent errors arising due to division by zero
    if all([ v == 0 for v in df.sales_quant]):
        test_stat.append(None)
        p_val.append(0)
        continue

    s=bohning_test(df.chassis)
    test_stat.append(s[0])
    p_val.append(s[1])
    dates.append(date)

result=pd.DataFrame(list(zip(dates, test_stat, p_val)), columns=['
    dates', 'test_stat', 'p_val'])
pd.set_option('display.max_rows', result.shape[0]+1)
result
```

Table B.15 Results of the Böhning's Test when using bi-weekly periods as successive time intervals.

Dates	Test Statistic	P-value
18/10/1996-01/11/1996	1.20E+02	0.00E+00
01/11/1996-15/11/1996	8.85E+01	0.00E+00
15/11/1996-29/11/1996	2.67E+01	0.00E+00
29/11/1996-13/12/1996	2.64E+02	0.00E+00
13/12/1996-27/12/1996	3.40E+01	0.00E+00
27/12/1996-10/01/1997	1.89E+01	0.00E+00
10/01/1997-24/01/1997	4.31E-01	3.33E-01

Table B.15 continued from previous page

Dates	Test Statistic	P-value
24/01/1997-07/02/1997	5.55E+01	0.00E+00
07/02/1997-21/02/1997	1.70E+01	0.00E+00
21/02/1997-07/03/1997	3.19E+01	0.00E+00
07/03/1997-21/03/1997	3.71E+01	0.00E+00
21/03/1997-04/04/1997	1.48E+02	0.00E+00
04/04/1997-18/04/1997	2.02E+01	0.00E+00
18/04/1997-02/05/1997	1.95E+02	0.00E+00
02/05/1997-16/05/1997	1.25E+01	0.00E+00
16/05/1997-30/05/1997	-5.88E-01	7.22E-01
30/05/1997-13/06/1997	3.61E+02	0.00E+00
13/06/1997-27/06/1997	6.60E+01	0.00E+00
27/06/1997-11/07/1997	6.66E+01	0.00E+00
11/07/1997-25/07/1997	6.23E+01	0.00E+00
25/07/1997-08/08/1997	1.18E+02	0.00E+00
08/08/1997-22/08/1997	5.44E+01	0.00E+00
22/08/1997-05/09/1997	3.56E+01	0.00E+00
05/09/1997-19/09/1997	3.64E+01	0.00E+00
19/09/1997-03/10/1997	3.33E+01	0.00E+00
03/10/1997-17/10/1997	1.00E+02	0.00E+00
17/10/1997-31/10/1997	5.32E+01	0.00E+00
31/10/1997-14/11/1997	7.29E+01	0.00E+00
14/11/1997-28/11/1997	1.30E+02	0.00E+00
28/11/1997-12/12/1997	5.32E+01	0.00E+00
12/12/1997-26/12/1997	9.81E+01	0.00E+00
26/12/1997-09/01/1998	5.81E+02	0.00E+00
09/01/1998-23/01/1998	1.05E+02	0.00E+00
23/01/1998-06/02/1998	2.34E+01	0.00E+00
06/02/1998-20/02/1998	4.14E+01	0.00E+00
20/02/1998-06/03/1998	2.90E+01	0.00E+00

Table B.15 continued from previous page

Dates	Test Statistic	P-value
06/03/1998-20/03/1998	1.32E+01	0.00E+00
20/03/1998-03/04/1998	8.06E+01	0.00E+00
03/04/1998-17/04/1998	4.92E+01	0.00E+00
17/04/1998-01/05/1998	7.68E+01	0.00E+00
01/05/1998-15/05/1998	2.47E+01	0.00E+00
15/05/1998-29/05/1998	4.13E+01	0.00E+00
29/05/1998-12/06/1998	1.37E+02	0.00E+00
12/06/1998-26/06/1998	8.85E+01	0.00E+00
26/06/1998-10/07/1998	7.90E+01	0.00E+00
10/07/1998-24/07/1998	8.08E+01	0.00E+00
24/07/1998-07/08/1998	1.60E+02	0.00E+00
07/08/1998-21/08/1998	7.59E+01	0.00E+00
21/08/1998-04/09/1998	5.83E+01	0.00E+00
04/09/1998-18/09/1998	2.27E+02	0.00E+00
18/09/1998-02/10/1998	1.05E+02	0.00E+00
02/10/1998-16/10/1998	5.94E+01	0.00E+00
16/10/1998-30/10/1998	3.45E+01	0.00E+00
30/10/1998-13/11/1998	5.44E+01	0.00E+00
13/11/1998-27/11/1998	2.06E+01	0.00E+00
27/11/1998-11/12/1998	1.55E+02	0.00E+00
11/12/1998-25/12/1998	7.10E+01	0.00E+00
25/12/1998-08/01/1999	8.99E+01	0.00E+00
08/01/1999-22/01/1999	5.37E+00	3.86E-08
22/01/1999-05/02/1999	4.19E+01	0.00E+00
05/02/1999-19/02/1999	1.61E+01	0.00E+00
19/02/1999-05/03/1999	4.06E+01	0.00E+00
05/03/1999-19/03/1999	2.45E+01	0.00E+00
19/03/1999-02/04/1999	7.79E+01	0.00E+00
02/04/1999-16/04/1999	1.11E+02	0.00E+00

Table B.15 continued from previous page

Dates	Test Statistic	P-value
16/04/1999-30/04/1999	2.18E+02	0.00E+00
30/04/1999-14/05/1999	9.13E+01	0.00E+00
14/05/1999-28/05/1999	1.24E+02	0.00E+00
28/05/1999-11/06/1999	6.80E+01	0.00E+00
11/06/1999-25/06/1999	1.24E+01	0.00E+00
25/06/1999-09/07/1999	1.61E+01	0.00E+00
09/07/1999-23/07/1999	5.88E-01	2.78E-01
23/07/1999-06/08/1999	5.52E+01	0.00E+00
06/08/1999-20/08/1999	2.62E+01	0.00E+00
20/08/1999-03/09/1999	1.48E+02	0.00E+00
03/09/1999-17/09/1999	9.11E+01	0.00E+00
17/09/1999-01/10/1999	6.82E+01	0.00E+00
01/10/1999-15/10/1999	8.27E+01	0.00E+00
15/10/1999-29/10/1999	2.24E+02	0.00E+00
29/10/1999-12/11/1999	5.11E+01	0.00E+00
12/11/1999-26/11/1999	2.94E+01	0.00E+00
26/11/1999-10/12/1999	8.17E+01	0.00E+00
10/12/1999-24/12/1999	5.95E+01	0.00E+00
24/12/1999-07/01/2000	6.18E+00	3.25E-10
07/01/2000-21/01/2000	7.02E+00	1.07E-12
21/01/2000-04/02/2000	2.83E+01	0.00E+00
04/02/2000-18/02/2000	2.21E+01	0.00E+00
18/02/2000-03/03/2000	3.64E+01	0.00E+00
03/03/2000-17/03/2000	1.17E+02	0.00E+00
17/03/2000-31/03/2000	4.65E+01	0.00E+00
31/03/2000-14/04/2000	3.38E+01	0.00E+00
14/04/2000-28/04/2000	1.01E+02	0.00E+00
28/04/2000-12/05/2000	9.33E+01	0.00E+00
12/05/2000-26/05/2000	4.51E+01	0.00E+00

Table B.15 continued from previous page

Dates	Test Statistic	P-value
26/05/2000-09/06/2000	2.75E+01	0.00E+00
09/06/2000-23/06/2000	3.34E+01	0.00E+00
23/06/2000-07/07/2000	2.51E+01	0.00E+00
07/07/2000-21/07/2000	3.93E+01	0.00E+00
21/07/2000-04/08/2000	2.55E+01	0.00E+00
04/08/2000-18/08/2000	2.40E+01	0.00E+00
18/08/2000-01/09/2000	3.95E+01	0.00E+00
01/09/2000-15/09/2000	4.08E+01	0.00E+00
15/09/2000-29/09/2000	4.12E+01	0.00E+00
29/09/2000-13/10/2000	4.33E+01	0.00E+00
13/10/2000-27/10/2000	4.51E+01	0.00E+00
27/10/2000-10/11/2000	3.97E+01	0.00E+00
10/11/2000-24/11/2000	5.71E+01	0.00E+00
24/11/2000-08/12/2000	4.65E+01	0.00E+00
08/12/2000-22/12/2000	3.24E+01	0.00E+00
22/12/2000-05/01/2001	5.45E+01	0.00E+00
05/01/2001-19/01/2001	1.47E-01	4.42E-01
19/01/2001-02/02/2001	9.92E+00	0.00E+00
02/02/2001-16/02/2001	1.46E+01	0.00E+00
16/02/2001-02/03/2001	1.53E+01	0.00E+00
02/03/2001-16/03/2001	2.19E+01	0.00E+00
16/03/2001-30/03/2001	1.67E+00	4.78E-02
30/03/2001-13/04/2001	7.38E-01	2.30E-01
13/04/2001-27/04/2001	1.64E+00	5.01E-02
27/04/2001-11/05/2001	5.75E+00	4.48E-09
11/05/2001-25/05/2001	4.51E+00	3.23E-06
25/05/2001-08/06/2001	6.98E+00	1.44E-12
08/06/2001-22/06/2001	8.15E+00	2.22E-16
22/06/2001-06/07/2001	9.41E-01	1.73E-01

Table B.15 continued from previous page

Dates	Test Statistic	P-value
06/07/2001-20/07/2001	-6.19E-02	5.25E-01
20/07/2001-03/08/2001	6.34E-01	2.63E-01
03/08/2001-17/08/2001	1.77E+00	3.88E-02
17/08/2001-31/08/2001	2.09E+00	1.82E-02
31/08/2001-14/09/2001	2.03E+00	2.11E-02
14/09/2001-28/09/2001	-6.54E-02	5.26E-01
28/09/2001-12/10/2001	3.53E+00	2.08E-04
12/10/2001-26/10/2001	3.14E-01	3.77E-01
26/10/2001-09/11/2001	5.10E+00	1.71E-07
09/11/2001-23/11/2001	-5.88E-01	7.22E-01
23/11/2001-07/12/2001	1.03E+00	1.52E-01
07/12/2001-21/12/2001	2.88E+00	2.01E-03
21/12/2001-04/01/2002	1.45E+01	0.00E+00
01/03/2002-15/03/2002	NaN	0.00E+00
15/03/2002-29/03/2002	NaN	0.00E+00
29/03/2002-12/04/2002	NaN	0.00E+00
12/04/2002-26/04/2002	NaN	0.00E+00
26/04/2002-10/05/2002	1.41E+00	7.90E-02
10/05/2002-24/05/2002	3.53E+00	2.08E-04
24/05/2002-07/06/2002	3.14E-01	3.77E-01
07/06/2002-21/06/2002	-1.96E-01	5.78E-01
21/06/2002-05/07/2002	-3.92E-01	6.53E-01
05/07/2002-19/07/2002	1.37E+00	8.49E-02
19/07/2002-02/08/2002	-5.66E-16	5.00E-01
02/08/2002-16/08/2002	2.03E+00	2.11E-02
16/08/2002-30/08/2002	1.03E+00	1.51E-01
30/08/2002-13/09/2002	8.50E-01	1.98E-01
13/09/2002-27/09/2002	4.51E+00	3.23E-06
27/09/2002-11/10/2002	-9.81E-01	8.37E-01

Table B.15 continued from previous page

Dates	Test Statistic	P-value
11/10/2002-25/10/2002	-5.88E-01	7.22E-01
25/10/2002-08/11/2002	6.86E-01	2.46E-01
08/11/2002-22/11/2002	-5.88E-01	7.22E-01
22/11/2002-06/12/2002	-3.92E-01	6.53E-01
06/12/2002-20/12/2002	7.84E-01	2.16E-01
20/12/2002-03/01/2003	1.53E+00	6.26E-02
03/01/2003-17/01/2003	9.81E-01	1.63E-01
17/01/2003-31/01/2003	2.16E+00	1.55E-02
14/02/2003-28/02/2003	1.53E+00	6.26E-02
28/02/2003-14/03/2003	2.46E+00	6.90E-03
14/03/2003-28/03/2003	-1.96E-01	5.78E-01
28/03/2003-11/04/2003	-3.92E-01	6.53E-01
11/04/2003-25/04/2003	NaN	0.00E+00
25/04/2003-09/05/2003	-1.96E-01	5.78E-01
09/05/2003-23/05/2003	3.14E-01	3.77E-01
23/05/2003-06/06/2003	-1.96E-01	5.78E-01
06/06/2003-20/06/2003	-3.92E-01	6.53E-01
20/06/2003-04/07/2003	5.16E+00	1.21E-07
04/07/2003-18/07/2003	5.88E-01	2.78E-01
18/07/2003-01/08/2003	1.41E+00	7.90E-02
01/08/2003-15/08/2003	-3.92E-01	6.53E-01
15/08/2003-29/08/2003	5.66E-16	5.00E-01
29/08/2003-12/09/2003	2.16E+00	1.55E-02
12/09/2003-26/09/2003	4.51E+00	3.23E-06
26/09/2003-10/10/2003	-1.96E-01	5.78E-01
10/10/2003-24/10/2003	-6.86E-01	7.54E-01
24/10/2003-07/11/2003	-6.86E-01	7.54E-01
07/11/2003-21/11/2003	3.76E+00	8.62E-05
21/11/2003-05/12/2003	2.94E-01	3.84E-01

Table B.15 continued from previous page

Dates	Test Statistic	P-value
05/12/2003-19/12/2003	6.13E+00	4.51E-10
19/12/2003-02/01/2004	-1.18E-01	5.47E-01
02/01/2004-16/01/2004	8.97E+00	0.00E+00
16/01/2004-30/01/2004	1.51E+01	0.00E+00
30/01/2004-13/02/2004	1.22E+01	0.00E+00
13/02/2004-27/02/2004	7.29E+00	1.55E-13
27/02/2004-12/03/2004	8.28E-01	2.04E-01
12/03/2004-26/03/2004	1.31E+01	0.00E+00
26/03/2004-09/04/2004	2.75E+00	3.02E-03
09/04/2004-23/04/2004	7.11E+00	5.65E-13
23/04/2004-07/05/2004	1.55E+00	6.09E-02
07/05/2004-21/05/2004	4.28E+00	9.43E-06
21/05/2004-04/06/2004	3.94E+00	4.10E-05
04/06/2004-18/06/2004	9.38E-01	1.74E-01
18/06/2004-02/07/2004	1.26E+01	0.00E+00
02/07/2004-16/07/2004	6.28E+00	1.74E-10
16/07/2004-30/07/2004	6.28E+00	1.66E-10
30/07/2004-13/08/2004	6.77E+00	6.51E-12
13/08/2004-27/08/2004	1.16E+01	0.00E+00
27/08/2004-10/09/2004	4.71E+00	1.26E-06
10/09/2004-24/09/2004	1.61E+00	5.34E-02
24/09/2004-08/10/2004	5.96E-01	2.76E-01
08/10/2004-22/10/2004	3.24E+00	5.99E-04
22/10/2004-05/11/2004	3.96E+00	3.83E-05
05/11/2004-19/11/2004	5.33E+00	4.97E-08
19/11/2004-03/12/2004	-1.83E-01	5.73E-01
03/12/2004-17/12/2004	3.33E+00	4.28E-04
17/12/2004-31/12/2004	-5.54E-01	7.10E-01
31/12/2004-14/01/2005	2.26E+01	0.00E+00

Table B.15 continued from previous page

Dates	Test Statistic	P-value
14/01/2005-28/01/2005	7.84E-01	2.16E-01
28/01/2005-11/02/2005	1.37E+00	8.49E-02
11/02/2005-25/02/2005	3.74E-01	3.54E-01
25/02/2005-11/03/2005	6.49E+00	4.25E-11
11/03/2005-25/03/2005	3.64E+04	0.00E+00
25/03/2005-08/04/2005	1.50E+01	0.00E+00
08/04/2005-22/04/2005	1.51E+01	0.00E+00
22/04/2005-06/05/2005	2.24E+01	0.00E+00
06/05/2005-20/05/2005	2.25E+01	0.00E+00
20/05/2005-03/06/2005	1.74E+01	0.00E+00
03/06/2005-17/06/2005	2.03E+01	0.00E+00
17/06/2005-01/07/2005	1.41E+01	0.00E+00
01/07/2005-15/07/2005	1.63E+01	0.00E+00
15/07/2005-29/07/2005	2.98E+01	0.00E+00
29/07/2005-12/08/2005	3.63E+01	0.00E+00
12/08/2005-26/08/2005	2.95E+01	0.00E+00
26/08/2005-09/09/2005	3.27E+01	0.00E+00
09/09/2005-23/09/2005	2.69E+01	0.00E+00
23/09/2005-07/10/2005	2.92E+01	0.00E+00
07/10/2005-21/10/2005	2.97E+01	0.00E+00
21/10/2005-04/11/2005	3.04E+01	0.00E+00
04/11/2005-18/11/2005	2.77E+01	0.00E+00
18/11/2005-02/12/2005	2.05E+01	0.00E+00
02/12/2005-16/12/2005	3.29E+01	0.00E+00
16/12/2005-30/12/2005	3.51E+01	0.00E+00
30/12/2005-13/01/2006	5.79E+01	0.00E+00
13/01/2006-27/01/2006	4.55E+01	0.00E+00
27/01/2006-10/02/2006	3.88E+01	0.00E+00
10/02/2006-24/02/2006	3.71E+01	0.00E+00

Table B.15 continued from previous page

Dates	Test Statistic	P-value
24/02/2006-10/03/2006	4.44E+01	0.00E+00
10/03/2006-24/03/2006	1.27E+02	0.00E+00
24/03/2006-07/04/2006	1.62E+01	0.00E+00
07/04/2006-21/04/2006	1.66E+01	0.00E+00
21/04/2006-05/05/2006	1.95E+01	0.00E+00
05/05/2006-19/05/2006	4.90E+01	0.00E+00
19/05/2006-02/06/2006	3.61E+01	0.00E+00
02/06/2006-16/06/2006	3.26E+01	0.00E+00
16/06/2006-30/06/2006	2.66E+01	0.00E+00
30/06/2006-14/07/2006	1.78E+01	0.00E+00
14/07/2006-28/07/2006	1.19E+01	0.00E+00
28/07/2006-11/08/2006	3.60E+01	0.00E+00
11/08/2006-25/08/2006	4.51E+00	3.19E-06
25/08/2006-08/09/2006	2.36E+01	0.00E+00
08/09/2006-22/09/2006	1.91E+01	0.00E+00
22/09/2006-06/10/2006	1.07E+01	0.00E+00
06/10/2006-20/10/2006	1.60E+01	0.00E+00
20/10/2006-03/11/2006	1.28E+01	0.00E+00
03/11/2006-17/11/2006	1.71E+01	0.00E+00
17/11/2006-01/12/2006	5.16E+00	1.22E-07
01/12/2006-15/12/2006	1.01E+01	0.00E+00
15/12/2006-29/12/2006	1.12E+01	0.00E+00
29/12/2006-12/01/2007	4.10E+01	0.00E+00
12/01/2007-26/01/2007	1.32E+01	0.00E+00
26/01/2007-09/02/2007	1.68E+01	0.00E+00
09/02/2007-23/02/2007	1.51E+01	0.00E+00
23/02/2007-09/03/2007	6.98E+01	0.00E+00
09/03/2007-23/03/2007	2.16E+02	0.00E+00
23/03/2007-06/04/2007	1.29E+01	0.00E+00

Table B.15 continued from previous page

Dates	Test Statistic	P-value
06/04/2007-20/04/2007	1.26E+01	0.00E+00
20/04/2007-04/05/2007	1.01E+01	0.00E+00
04/05/2007-18/05/2007	1.12E+01	0.00E+00
18/05/2007-01/06/2007	1.25E+01	0.00E+00
01/06/2007-15/06/2007	1.18E+01	0.00E+00
15/06/2007-29/06/2007	1.14E+01	0.00E+00
29/06/2007-13/07/2007	3.96E+01	0.00E+00
13/07/2007-27/07/2007	1.58E+01	0.00E+00
27/07/2007-10/08/2007	1.71E+01	0.00E+00
10/08/2007-24/08/2007	1.17E+01	0.00E+00
24/08/2007-07/09/2007	4.58E+01	0.00E+00
07/09/2007-21/09/2007	2.75E+01	0.00E+00
21/09/2007-05/10/2007	1.51E+01	0.00E+00
05/10/2007-19/10/2007	2.86E+01	0.00E+00
19/10/2007-02/11/2007	2.07E+01	0.00E+00
02/11/2007-16/11/2007	1.99E+01	0.00E+00
16/11/2007-30/11/2007	1.45E+01	0.00E+00
30/11/2007-14/12/2007	1.69E+01	0.00E+00
14/12/2007-28/12/2007	4.61E+01	0.00E+00
28/12/2007-11/01/2008	3.74E+01	0.00E+00
11/01/2008-25/01/2008	1.48E+01	0.00E+00
25/01/2008-08/02/2008	5.23E+01	0.00E+00
08/02/2008-22/02/2008	1.77E+01	0.00E+00
22/02/2008-07/03/2008	8.78E+01	0.00E+00
07/03/2008-21/03/2008	1.00E+02	0.00E+00
21/03/2008-04/04/2008	1.03E+01	0.00E+00
04/04/2008-18/04/2008	1.53E+01	0.00E+00
18/04/2008-02/05/2008	2.99E+01	0.00E+00
02/05/2008-16/05/2008	1.07E+01	0.00E+00

Table B.15 continued from previous page

Dates	Test Statistic	P-value
16/05/2008-30/05/2008	3.65E+01	0.00E+00
30/05/2008-13/06/2008	3.96E+01	0.00E+00
13/06/2008-27/06/2008	1.68E+01	0.00E+00
27/06/2008-11/07/2008	3.48E+01	0.00E+00
11/07/2008-25/07/2008	6.41E+00	7.15E-11
25/07/2008-08/08/2008	1.31E+01	0.00E+00
08/08/2008-22/08/2008	1.05E+01	0.00E+00
22/08/2008-05/09/2008	9.25E+00	0.00E+00
05/09/2008-19/09/2008	1.40E+01	0.00E+00
19/09/2008-03/10/2008	1.26E+01	0.00E+00
03/10/2008-17/10/2008	5.09E+01	0.00E+00
17/10/2008-31/10/2008	1.17E+01	0.00E+00
31/10/2008-14/11/2008	2.84E+01	0.00E+00
14/11/2008-28/11/2008	6.93E+00	2.13E-12
28/11/2008-12/12/2008	4.48E+01	0.00E+00
12/12/2008-26/12/2008	8.76E+00	0.00E+00
26/12/2008-09/01/2009	3.42E+01	0.00E+00
09/01/2009-23/01/2009	1.55E+02	0.00E+00
23/01/2009-06/02/2009	1.59E+01	0.00E+00
06/02/2009-20/02/2009	7.80E+01	0.00E+00
20/02/2009-06/03/2009	2.54E+01	0.00E+00
06/03/2009-20/03/2009	4.73E+01	0.00E+00
20/03/2009-03/04/2009	2.25E+01	0.00E+00
03/04/2009-17/04/2009	5.64E+00	8.40E-09
17/04/2009-01/05/2009	6.83E+00	4.18E-12
01/05/2009-15/05/2009	1.04E+01	0.00E+00
15/05/2009-29/05/2009	7.78E+01	0.00E+00
29/05/2009-12/06/2009	1.63E+02	0.00E+00
12/06/2009-26/06/2009	1.79E+01	0.00E+00

Table B.15 continued from previous page

Dates	Test Statistic	P-value
26/06/2009-10/07/2009	1.43E+01	0.00E+00
10/07/2009-24/07/2009	3.62E+01	0.00E+00
24/07/2009-07/08/2009	2.62E+01	0.00E+00
07/08/2009-21/08/2009	3.42E+01	0.00E+00
04/09/2009-18/09/2009	1.07E+01	0.00E+00
18/09/2009-02/10/2009	1.95E+01	0.00E+00
02/10/2009-16/10/2009	5.29E+00	6.28E-08
16/10/2009-30/10/2009	2.56E+02	0.00E+00
30/10/2009-13/11/2009	5.66E-16	5.00E-01
13/11/2009-27/11/2009	NaN	0.00E+00
27/11/2009-11/12/2009	-3.92E-01	6.53E-01
11/12/2009-25/12/2009	9.14E+02	0.00E+00
25/12/2009-08/01/2010	1.30E+01	0.00E+00
08/01/2010-22/01/2010	9.08E+00	0.00E+00
22/01/2010-05/02/2010	1.90E+01	0.00E+00
05/02/2010-19/02/2010	1.71E+01	0.00E+00
19/02/2010-05/03/2010	1.67E+01	0.00E+00
05/03/2010-19/03/2010	6.75E+00	7.58E-12
19/03/2010-02/04/2010	1.39E+02	0.00E+00
02/04/2010-16/04/2010	2.68E+01	0.00E+00
16/04/2010-30/04/2010	2.36E+01	0.00E+00
30/04/2010-14/05/2010	2.26E+01	0.00E+00
14/05/2010-28/05/2010	2.15E+02	0.00E+00
28/05/2010-11/06/2010	5.87E+01	0.00E+00
11/06/2010-25/06/2010	4.56E+01	0.00E+00
25/06/2010-09/07/2010	4.97E+01	0.00E+00
09/07/2010-23/07/2010	7.23E+01	0.00E+00
23/07/2010-06/08/2010	1.73E+01	0.00E+00
06/08/2010-20/08/2010	5.33E+01	0.00E+00

Table B.15 continued from previous page

Dates	Test Statistic	P-value
20/08/2010-03/09/2010	5.27E+01	0.00E+00
03/09/2010-17/09/2010	2.59E+02	0.00E+00
17/09/2010-01/10/2010	3.69E+01	0.00E+00
01/10/2010-15/10/2010	1.31E+01	0.00E+00
15/10/2010-29/10/2010	8.34E+01	0.00E+00
29/10/2010-12/11/2010	1.83E+01	0.00E+00
12/11/2010-26/11/2010	4.43E+00	4.66E-06
26/11/2010-10/12/2010	5.38E+01	0.00E+00
10/12/2010-24/12/2010	3.45E+01	0.00E+00
24/12/2010-07/01/2011	3.13E+01	0.00E+00
07/01/2011-21/01/2011	1.63E+01	0.00E+00
21/01/2011-04/02/2011	2.85E+02	0.00E+00
04/02/2011-18/02/2011	1.44E+02	0.00E+00
18/02/2011-04/03/2011	5.52E+01	0.00E+00
04/03/2011-18/03/2011	5.33E+01	0.00E+00
18/03/2011-01/04/2011	2.44E+02	0.00E+00
01/04/2011-15/04/2011	8.61E+01	0.00E+00
15/04/2011-29/04/2011	1.41E+01	0.00E+00
29/04/2011-13/05/2011	2.76E+01	0.00E+00
13/05/2011-27/05/2011	3.02E+01	0.00E+00
27/05/2011-10/06/2011	5.77E+01	0.00E+00
10/06/2011-24/06/2011	7.57E+01	0.00E+00
24/06/2011-08/07/2011	5.06E+01	0.00E+00
08/07/2011-22/07/2011	1.34E+02	0.00E+00
22/07/2011-05/08/2011	8.92E+01	0.00E+00
05/08/2011-19/08/2011	5.75E+01	0.00E+00
19/08/2011-02/09/2011	5.40E+01	0.00E+00
02/09/2011-16/09/2011	4.66E+01	0.00E+00
16/09/2011-30/09/2011	6.45E+01	0.00E+00

Table B.15 continued from previous page

Dates	Test Statistic	P-value
30/09/2011-14/10/2011	6.67E+01	0.00E+00
14/10/2011-28/10/2011	4.79E+01	0.00E+00
28/10/2011-11/11/2011	3.02E+01	0.00E+00
11/11/2011-25/11/2011	2.24E+02	0.00E+00
25/11/2011-09/12/2011	4.10E+01	0.00E+00
09/12/2011-23/12/2011	5.74E+01	0.00E+00
23/12/2011-06/01/2012	7.99E+01	0.00E+00
06/01/2012-20/01/2012	1.08E+02	0.00E+00
20/01/2012-03/02/2012	1.12E+02	0.00E+00
03/02/2012-17/02/2012	4.94E+01	0.00E+00
17/02/2012-02/03/2012	1.95E+02	0.00E+00
02/03/2012-16/03/2012	7.56E+01	0.00E+00
16/03/2012-30/03/2012	1.44E+02	0.00E+00
30/03/2012-13/04/2012	3.25E+01	0.00E+00
13/04/2012-27/04/2012	6.00E+01	0.00E+00
27/04/2012-11/05/2012	4.66E+01	0.00E+00
11/05/2012-25/05/2012	9.13E+01	0.00E+00
25/05/2012-08/06/2012	9.39E+01	0.00E+00
08/06/2012-22/06/2012	8.99E+01	0.00E+00
22/06/2012-06/07/2012	4.96E+01	0.00E+00
06/07/2012-20/07/2012	4.24E+01	0.00E+00
20/07/2012-03/08/2012	1.77E+02	0.00E+00
03/08/2012-17/08/2012	1.03E+02	0.00E+00
17/08/2012-31/08/2012	5.37E+01	0.00E+00
31/08/2012-14/09/2012	3.00E+01	0.00E+00
14/09/2012-28/09/2012	1.40E+02	0.00E+00
28/09/2012-12/10/2012	1.14E+02	0.00E+00
12/10/2012-26/10/2012	1.12E+02	0.00E+00
26/10/2012-09/11/2012	4.71E+01	0.00E+00

Table B.15 continued from previous page

Dates	Test Statistic	P-value
09/11/2012-23/11/2012	7.75E+01	0.00E+00
23/11/2012-07/12/2012	1.09E+02	0.00E+00
07/12/2012-21/12/2012	6.47E+01	0.00E+00
21/12/2012-04/01/2013	1.18E+02	0.00E+00
04/01/2013-18/01/2013	2.08E+02	0.00E+00
18/01/2013-01/02/2013	4.83E+02	0.00E+00
01/02/2013-15/02/2013	1.68E+02	0.00E+00
15/02/2013-01/03/2013	2.53E+02	0.00E+00
01/03/2013-15/03/2013	5.94E+01	0.00E+00
15/03/2013-29/03/2013	1.44E+02	0.00E+00
29/03/2013-12/04/2013	1.52E+01	0.00E+00
12/04/2013-26/04/2013	1.59E+02	0.00E+00
26/04/2013-10/05/2013	9.12E+01	0.00E+00
10/05/2013-24/05/2013	1.54E+02	0.00E+00
24/05/2013-07/06/2013	1.56E+02	0.00E+00
07/06/2013-21/06/2013	1.27E+02	0.00E+00
21/06/2013-05/07/2013	1.53E+02	0.00E+00
05/07/2013-19/07/2013	1.13E+02	0.00E+00
19/07/2013-02/08/2013	3.71E+02	0.00E+00
02/08/2013-16/08/2013	1.39E+02	0.00E+00
16/08/2013-30/08/2013	8.15E+01	0.00E+00
30/08/2013-13/09/2013	1.89E+02	0.00E+00
13/09/2013-27/09/2013	4.47E+01	0.00E+00
27/09/2013-11/10/2013	8.70E+01	0.00E+00
11/10/2013-25/10/2013	1.05E+02	0.00E+00
25/10/2013-08/11/2013	2.76E+02	0.00E+00
08/11/2013-22/11/2013	2.31E+02	0.00E+00
22/11/2013-06/12/2013	6.74E+01	0.00E+00
06/12/2013-20/12/2013	1.44E+02	0.00E+00

Table B.15 continued from previous page

Dates	Test Statistic	P-value
20/12/2013-03/01/2014	1.52E+02	0.00E+00
03/01/2014-17/01/2014	1.52E+02	0.00E+00
17/01/2014-31/01/2014	2.51E+02	0.00E+00
31/01/2014-14/02/2014	9.97E+01	0.00E+00
14/02/2014-28/02/2014	5.41E+01	0.00E+00
28/02/2014-14/03/2014	1.14E+02	0.00E+00
14/03/2014-28/03/2014	1.51E+02	0.00E+00
28/03/2014-11/04/2014	1.41E+01	0.00E+00
11/04/2014-25/04/2014	5.39E+01	0.00E+00
25/04/2014-09/05/2014	1.12E+02	0.00E+00
09/05/2014-23/05/2014	1.47E+02	0.00E+00
23/05/2014-06/06/2014	8.14E+02	0.00E+00
06/06/2014-20/06/2014	1.24E+02	0.00E+00
20/06/2014-04/07/2014	2.21E+02	0.00E+00
04/07/2014-18/07/2014	1.17E+02	0.00E+00
18/07/2014-01/08/2014	4.73E+02	0.00E+00
01/08/2014-15/08/2014	1.69E+02	0.00E+00
15/08/2014-29/08/2014	3.20E+02	0.00E+00
29/08/2014-12/09/2014	5.47E+01	0.00E+00
12/09/2014-26/09/2014	2.04E+02	0.00E+00
26/09/2014-10/10/2014	9.70E+01	0.00E+00
10/10/2014-24/10/2014	1.53E+02	0.00E+00
24/10/2014-07/11/2014	5.89E+01	0.00E+00
07/11/2014-21/11/2014	7.07E+01	0.00E+00
21/11/2014-05/12/2014	3.48E+02	0.00E+00
05/12/2014-19/12/2014	4.33E+01	0.00E+00
19/12/2014-02/01/2015	1.61E+02	0.00E+00
02/01/2015-16/01/2015	7.07E+01	0.00E+00
16/01/2015-30/01/2015	1.22E+02	0.00E+00

Table B.15 continued from previous page

Dates	Test Statistic	P-value
30/01/2015-13/02/2015	8.91E+01	0.00E+00
13/02/2015-27/02/2015	9.99E+01	0.00E+00
27/02/2015-13/03/2015	8.02E+01	0.00E+00
13/03/2015-27/03/2015	1.87E+02	0.00E+00
27/03/2015-10/04/2015	7.09E+01	0.00E+00
10/04/2015-24/04/2015	1.46E+01	0.00E+00
24/04/2015-08/05/2015	3.83E+01	0.00E+00
08/05/2015-22/05/2015	2.30E+02	0.00E+00
22/05/2015-05/06/2015	2.08E+02	0.00E+00
05/06/2015-19/06/2015	2.99E+02	0.00E+00
19/06/2015-03/07/2015	8.69E+01	0.00E+00
03/07/2015-17/07/2015	1.18E+02	0.00E+00
17/07/2015-31/07/2015	3.29E+02	0.00E+00
31/07/2015-14/08/2015	1.34E+02	0.00E+00
14/08/2015-28/08/2015	1.29E+02	0.00E+00
28/08/2015-11/09/2015	7.31E+01	0.00E+00
11/09/2015-25/09/2015	2.65E+02	0.00E+00
25/09/2015-09/10/2015	2.17E+02	0.00E+00
09/10/2015-23/10/2015	9.72E+01	0.00E+00
23/10/2015-06/11/2015	1.09E+02	0.00E+00
06/11/2015-20/11/2015	8.34E+01	0.00E+00
20/11/2015-04/12/2015	2.06E+02	0.00E+00
04/12/2015-18/12/2015	3.80E+01	0.00E+00
18/12/2015-01/01/2016	1.14E+02	0.00E+00
01/01/2016-15/01/2016	8.02E+01	0.00E+00
15/01/2016-29/01/2016	9.77E+01	0.00E+00
29/01/2016-12/02/2016	3.62E+01	0.00E+00
12/02/2016-26/02/2016	2.67E+02	0.00E+00
26/02/2016-11/03/2016	5.65E+01	0.00E+00

Table B.15 continued from previous page

Dates	Test Statistic	P-value
11/03/2016-25/03/2016	7.49E+02	0.00E+00
25/03/2016-08/04/2016	1.01E+02	0.00E+00
08/04/2016-22/04/2016	2.10E+02	0.00E+00
22/04/2016-06/05/2016	1.88E+01	0.00E+00
06/05/2016-20/05/2016	1.12E+02	0.00E+00
20/05/2016-03/06/2016	3.04E+02	0.00E+00
03/06/2016-17/06/2016	5.73E+01	0.00E+00
17/06/2016-01/07/2016	4.54E+02	0.00E+00
01/07/2016-15/07/2016	4.54E+01	0.00E+00
15/07/2016-29/07/2016	2.30E+02	0.00E+00
29/07/2016-12/08/2016	1.27E+02	0.00E+00
12/08/2016-26/08/2016	3.20E+02	0.00E+00
26/08/2016-09/09/2016	8.28E+01	0.00E+00
09/09/2016-23/09/2016	4.02E+02	0.00E+00
23/09/2016-07/10/2016	9.38E+01	0.00E+00
07/10/2016-21/10/2016	1.22E+02	0.00E+00
21/10/2016-04/11/2016	4.76E+02	0.00E+00
04/11/2016-18/11/2016	1.48E+02	0.00E+00
18/11/2016-02/12/2016	3.86E+02	0.00E+00
02/12/2016-16/12/2016	2.06E+02	0.00E+00
16/12/2016-30/12/2016	2.84E+02	0.00E+00
30/12/2016-13/01/2017	3.78E+02	0.00E+00
13/01/2017-27/01/2017	5.23E+02	0.00E+00
27/01/2017-10/02/2017	9.37E+01	0.00E+00
10/02/2017-24/02/2017	5.81E+02	0.00E+00
24/02/2017-10/03/2017	1.05E+02	0.00E+00
10/03/2017-24/03/2017	1.41E+02	0.00E+00
24/03/2017-07/04/2017	1.15E+03	0.00E+00
07/04/2017-21/04/2017	6.48E+01	0.00E+00

Table B.15 continued from previous page

Dates	Test Statistic	P-value
21/04/2017-05/05/2017	2.16E+01	0.00E+00
05/05/2017-19/05/2017	7.56E+01	0.00E+00
19/05/2017-02/06/2017	3.71E+02	0.00E+00
02/06/2017-16/06/2017	4.14E+01	0.00E+00
16/06/2017-30/06/2017	2.50E+02	0.00E+00
30/06/2017-14/07/2017	4.94E+01	0.00E+00
14/07/2017-28/07/2017	9.58E+01	0.00E+00
28/07/2017-11/08/2017	8.73E+01	0.00E+00
11/08/2017-25/08/2017	3.01E+02	0.00E+00
25/08/2017-08/09/2017	1.37E+02	0.00E+00
08/09/2017-22/09/2017	7.92E+01	0.00E+00
22/09/2017-06/10/2017	1.43E+02	0.00E+00
06/10/2017-20/10/2017	1.81E+02	0.00E+00
20/10/2017-03/11/2017	7.13E+02	0.00E+00
03/11/2017-17/11/2017	1.13E+02	0.00E+00
17/11/2017-01/12/2017	2.29E+02	0.00E+00
01/12/2017-15/12/2017	6.37E+01	0.00E+00
15/12/2017-29/12/2017	3.44E+02	0.00E+00
29/12/2017-12/01/2018	5.01E+01	0.00E+00
12/01/2018-26/01/2018	1.57E+02	0.00E+00
26/01/2018-09/02/2018	4.77E+01	0.00E+00
09/02/2018-23/02/2018	2.44E+02	0.00E+00
23/02/2018-09/03/2018	1.44E+02	0.00E+00
09/03/2018-23/03/2018	1.51E+02	0.00E+00
23/03/2018-06/04/2018	5.07E+02	0.00E+00
06/04/2018-20/04/2018	1.71E+01	0.00E+00
20/04/2018-04/05/2018	3.55E+01	0.00E+00
04/05/2018-18/05/2018	1.14E+02	0.00E+00
18/05/2018-01/06/2018	8.09E+01	0.00E+00

Table B.15 continued from previous page

Dates	Test Statistic	P-value
01/06/2018-15/06/2018	7.07E+01	0.00E+00
15/06/2018-29/06/2018	5.79E+01	0.00E+00
29/06/2018-13/07/2018	4.32E+01	0.00E+00
13/07/2018-27/07/2018	2.44E+02	0.00E+00
27/07/2018-10/08/2018	8.93E+01	0.00E+00
10/08/2018-24/08/2018	1.96E+02	0.00E+00
24/08/2018-07/09/2018	9.54E+01	0.00E+00
07/09/2018-21/09/2018	8.74E+01	0.00E+00
21/09/2018-05/10/2018	1.91E+02	0.00E+00
05/10/2018-19/10/2018	3.83E+01	0.00E+00
19/10/2018-02/11/2018	9.33E+01	0.00E+00
02/11/2018-16/11/2018	5.78E+01	0.00E+00
16/11/2018-30/11/2018	5.65E+01	0.00E+00
30/11/2018-14/12/2018	6.19E+00	2.93E-10
14/12/2018-28/12/2018	9.01E+01	0.00E+00
28/12/2018-11/01/2019	9.27E+01	0.00E+00
11/01/2019-25/01/2019	2.21E+01	0.00E+00
25/01/2019-08/02/2019	2.67E+01	0.00E+00
08/02/2019-22/02/2019	3.77E+01	0.00E+00
22/02/2019-08/03/2019	4.26E+01	0.00E+00
08/03/2019-22/03/2019	4.27E+01	0.00E+00
22/03/2019-05/04/2019	1.70E+02	0.00E+00
05/04/2019-19/04/2019	9.57E+00	0.00E+00
19/04/2019-03/05/2019	2.10E+01	0.00E+00
03/05/2019-17/05/2019	1.46E+01	0.00E+00
17/05/2019-31/05/2019	1.31E+02	0.00E+00
31/05/2019-14/06/2019	1.49E+01	0.00E+00
14/06/2019-28/06/2019	5.99E+01	0.00E+00
28/06/2019-12/07/2019	3.49E+01	0.00E+00

Table B.15 continued from previous page

Dates	Test Statistic	P-value
12/07/2019-26/07/2019	1.05E+02	0.00E+00
26/07/2019-09/08/2019	3.37E+01	0.00E+00
09/08/2019-23/08/2019	8.84E+00	0.00E+00
23/08/2019-06/09/2019	1.53E+02	0.00E+00
06/09/2019-20/09/2019	4.63E+01	0.00E+00
20/09/2019-04/10/2019	7.40E+01	0.00E+00
04/10/2019-18/10/2019	7.38E-01	2.30E-01
18/10/2019-01/11/2019	3.33E+01	0.00E+00
01/11/2019-15/11/2019	3.81E+01	0.00E+00
15/11/2019-29/11/2019	9.40E+01	0.00E+00
29/11/2019-13/12/2019	6.58E+01	0.00E+00
13/12/2019-27/12/2019	6.21E+01	0.00E+00
27/12/2019-10/01/2020	3.04E+01	0.00E+00
10/01/2020-24/01/2020	1.68E+02	0.00E+00
24/01/2020-07/02/2020	5.28E+01	0.00E+00
07/02/2020-21/02/2020	3.85E+01	0.00E+00
21/02/2020-06/03/2020	4.63E+01	0.00E+00
06/03/2020-20/03/2020	3.60E+02	0.00E+00
20/03/2020-03/04/2020	6.08E+01	0.00E+00
03/04/2020-17/04/2020	4.71E+00	1.26E-06
17/04/2020-01/05/2020	9.66E+01	0.00E+00
01/05/2020-15/05/2020	3.34E+01	0.00E+00
15/05/2020-29/05/2020	1.12E+02	0.00E+00
29/05/2020-12/06/2020	7.56E+01	0.00E+00
12/06/2020-26/06/2020	1.67E+01	0.00E+00
26/06/2020-10/07/2020	1.44E+01	0.00E+00
10/07/2020-24/07/2020	1.74E+01	0.00E+00
24/07/2020-07/08/2020	3.05E+02	0.00E+00
07/08/2020-21/08/2020	1.13E+01	0.00E+00

Table B.15 continued from previous page

Dates	Test Statistic	P-value
21/08/2020-04/09/2020	8.97E+00	0.00E+00
04/09/2020-18/09/2020	3.09E+01	0.00E+00
18/09/2020-02/10/2020	1.75E+02	0.00E+00
02/10/2020-16/10/2020	5.04E+01	0.00E+00
16/10/2020-30/10/2020	3.01E+01	0.00E+00
30/10/2020-13/11/2020	4.51E+00	3.23E-06
13/11/2020-27/11/2020	7.84E-01	2.16E-01
27/11/2020-11/12/2020	5.60E+01	0.00E+00
11/12/2020-25/12/2020	1.44E+02	0.00E+00

The code block developed for Böhning's Test using months as consecutive time intervals is given below, and the test results are given in Table B.16.

```
test_stat=[]
p_val=[]

months=["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"]
years=[]
months=[]
mean_sales=[]

for year in range(1996,2021):
    for month in range(1,13):
        if (year==1996) & ((month!=10) & (month!=11) & (month!=12)):
            continue

        df=sample.loc[(sample.index.year==year) & (sample.index.month==month)]

        #some samples' sales records are consist of only zeros
        #those days are given a p-value of 0
        #to prevent error arising due to division by zero
        if all([ v == 0 for v in df.chassis ]):
            test_stat.append(None)
            p_val.append(0)
            years.append(year)
            months.append(month)
            mean_sales.append(df.chassis.mean())
            continue

        mean_sales.append(df.sales_quant.mean())

        s=bohning_test(df.sales_quant)
```

```
        test_stat.append(s[0])
        p_val.append(s[1])
        years.append(year)
        months.append(month)

result=pd.DataFrame(list(zip(years, months, test_stat, p_val,
    mean_sales)), columns=['years', 'months', 'test_stat', 'p_val',
    'mean_sales'])
pd.set_option('display.max_rows', result.shape[0]+1)
result
```

Table B.16 Results of the Böhning's Test when using months as successive time intervals.

Years	Months	Test Statistic	P-value	Mean Sales
1996	10	1.35E+02	0.00E+00	2.06E+01
1996	11	2.67E+02	0.00E+00	2.18E+01
1996	12	9.05E+01	0.00E+00	2.09E+01
1997	1	1.68E+02	0.00E+00	1.01E+01
1997	2	4.99E+01	0.00E+00	9.71E+00
1997	3	1.26E+02	0.00E+00	1.08E+01
1997	4	1.70E+02	0.00E+00	1.11E+01
1997	5	6.48E+02	0.00E+00	9.94E+00
1997	6	9.14E+01	0.00E+00	1.57E+01
1997	7	9.28E+01	0.00E+00	1.51E+01
1997	8	1.15E+02	0.00E+00	1.50E+01
1997	9	5.72E+01	0.00E+00	1.24E+01
1997	10	1.07E+02	0.00E+00	1.65E+01
1997	11	1.89E+02	0.00E+00	2.45E+01
1997	12	4.69E+02	0.00E+00	1.97E+01
1998	1	1.73E+02	0.00E+00	8.81E+00
1998	2	4.19E+01	0.00E+00	1.25E+01
1998	3	1.36E+02	0.00E+00	1.57E+01
1998	4	1.44E+02	0.00E+00	1.55E+01
1998	5	6.04E+01	0.00E+00	1.25E+01
1998	6	1.60E+02	0.00E+00	2.78E+01

Table B.16 continued from previous page

Years	Months	Test Statistic	P-value	Mean Sales
1998	7	1.46E+02	0.00E+00	3.03E+01
1998	8	1.04E+02	0.00E+00	1.51E+01
1998	9	2.37E+02	0.00E+00	3.04E+01
1998	10	8.12E+01	0.00E+00	1.31E+01
1998	11	1.68E+02	0.00E+00	1.45E+01
1998	12	9.06E+01	0.00E+00	2.06E+01
1999	1	6.62E+01	0.00E+00	1.77E+00
1999	2	4.72E+01	0.00E+00	8.04E+00
1999	3	6.49E+01	0.00E+00	1.24E+01
1999	4	2.44E+02	0.00E+00	2.83E+01
1999	5	1.55E+02	0.00E+00	2.32E+01
1999	6	1.54E+01	0.00E+00	5.87E+00
1999	7	9.12E+01	0.00E+00	3.42E+00
1999	8	1.37E+02	0.00E+00	1.00E+01
1999	9	1.13E+02	0.00E+00	1.92E+01
1999	10	2.65E+02	0.00E+00	1.37E+01
1999	11	8.01E+01	0.00E+00	1.45E+01
1999	12	1.36E+02	0.00E+00	1.41E+01
2000	1	8.50E+01	0.00E+00	5.65E+00
2000	2	3.83E+01	0.00E+00	1.62E+01
2000	3	1.05E+02	0.00E+00	2.67E+01
2000	4	1.26E+02	0.00E+00	2.16E+01
2000	5	1.04E+02	0.00E+00	2.42E+01
2000	6	3.92E+01	0.00E+00	1.50E+01
2000	7	5.79E+01	0.00E+00	1.50E+01
2000	8	4.64E+01	0.00E+00	1.53E+01
2000	9	6.39E+01	0.00E+00	3.15E+01
2000	10	5.78E+01	0.00E+00	2.53E+01
2000	11	7.24E+01	0.00E+00	3.80E+01

Table B.16 continued from previous page

Years	Months	Test Statistic	P-value	Mean Sales
2000	12	7.67E+01	0.00E+00	2.06E+01
2001	1	8.54E+00	0.00E+00	1.71E+00
2001	2	1.87E+01	0.00E+00	9.93E+00
2001	3	3.95E+01	0.00E+00	2.94E+00
2001	4	5.86E+00	2.37E-09	1.67E+00
2001	5	7.49E+00	3.40E-14	4.13E+00
2001	6	1.55E+01	0.00E+00	4.97E+00
2001	7	5.36E-01	2.96E-01	1.71E+00
2001	8	2.42E+00	7.85E-03	1.90E+00
2001	9	2.36E+00	9.05E-03	5.00E-01
2001	10	2.52E+00	5.79E-03	2.90E-01
2001	11	3.36E+00	3.93E-04	4.67E-01
2001	12	1.53E+01	0.00E+00	2.39E+00
2002	1	NaN	0.00E+00	0.00E+00
2002	2	NaN	0.00E+00	0.00E+00
2002	3	3.07E+00	1.05E-03	3.55E-01
2002	4	-5.25E-01	7.00E-01	1.67E-01
2002	5	1.07E+00	1.43E-01	5.16E-01
2002	6	9.54E-01	1.70E-01	6.33E-01
2002	7	4.03E+00	2.78E-05	5.81E-01
2002	8	-1.16E+00	8.77E-01	3.23E-01
2002	9	5.25E-01	3.00E-01	4.00E-01
2002	10	-3.61E-01	6.41E-01	3.23E-01
2002	11	3.02E+00	1.26E-03	6.00E-01
2002	12	2.66E+00	3.90E-03	1.06E+00
2003	1	-5.16E-01	6.97E-01	1.61E-01
2003	2	-1.36E-01	5.54E-01	7.14E-02
2003	3	9.68E-02	4.61E-01	2.58E-01
2003	4	5.10E+00	1.70E-07	6.33E-01

Table B.16 continued from previous page

Years	Months	Test Statistic	P-value	Mean Sales
2003	5	2.15E+00	1.60E-02	4.19E-01
2003	6	-2.63E-01	6.04E-01	1.00E-01
2003	7	3.92E+00	4.50E-05	3.87E-01
2003	8	-9.36E-01	8.25E-01	5.16E-01
2003	9	3.01E+00	1.30E-03	1.70E+00
2003	10	9.73E+00	0.00E+00	2.48E+00
2003	11	2.58E+01	0.00E+00	2.07E+00
2003	12	5.32E+00	5.30E-08	4.16E+00
2004	1	1.29E+01	0.00E+00	2.55E+00
2004	2	6.61E+00	1.95E-11	1.38E+00
2004	3	5.07E+00	2.01E-07	3.23E+00
2004	4	1.49E+01	0.00E+00	2.87E+00
2004	5	1.07E+01	0.00E+00	3.03E+00
2004	6	1.01E+01	0.00E+00	2.30E+00
2004	7	4.32E+00	7.81E-06	2.03E+00
2004	8	4.79E+00	8.52E-07	1.87E+00
2004	9	5.35E+00	4.45E-08	1.10E+00
2004	10	1.08E+00	1.39E-01	1.32E+00
2004	11	3.92E+01	0.00E+00	1.57E+00
2004	12	1.29E+01	0.00E+00	2.71E+00
2005	1	5.51E+04	0.00E+00	4.67E+02
2005	2	3.20E+01	0.00E+00	1.63E+01
2005	3	2.38E+01	0.00E+00	1.89E+01
2005	4	2.43E+01	0.00E+00	1.17E+01
2005	5	5.38E+01	0.00E+00	2.51E+01
2005	6	4.45E+01	0.00E+00	2.78E+01
2005	7	4.45E+01	0.00E+00	2.39E+01
2005	8	4.14E+01	0.00E+00	3.02E+01
2005	9	3.94E+01	0.00E+00	2.26E+01

Table B.16 continued from previous page

Years	Months	Test Statistic	P-value	Mean Sales
2005	10	6.43E+01	0.00E+00	2.97E+01
2005	11	6.65E+01	0.00E+00	2.88E+01
2005	12	7.04E+01	0.00E+00	4.24E+01
2006	1	4.22E+01	0.00E+00	1.08E+01
2006	2	4.17E+01	0.00E+00	1.85E+01
2006	3	5.76E+01	0.00E+00	2.70E+01
2006	4	3.71E+01	0.00E+00	2.32E+01
2006	5	4.00E+01	0.00E+00	1.63E+01
2006	6	4.84E+01	0.00E+00	1.38E+01
2006	7	2.37E+01	0.00E+00	1.11E+01
2006	8	2.21E+01	0.00E+00	1.22E+01
2006	9	1.31E+01	0.00E+00	7.43E+00
2006	10	3.33E+01	0.00E+00	9.10E+00
2006	11	2.17E+01	0.00E+00	1.24E+01
2006	12	1.39E+02	0.00E+00	2.84E+01
2007	1	2.40E+01	0.00E+00	9.45E+00
2007	2	1.55E+01	0.00E+00	1.08E+01
2007	3	1.76E+01	0.00E+00	1.05E+01
2007	4	2.43E+01	0.00E+00	9.97E+00
2007	5	3.59E+01	0.00E+00	1.23E+01
2007	6	5.72E+01	0.00E+00	1.30E+01
2007	7	2.56E+01	0.00E+00	1.14E+01
2007	8	3.34E+01	0.00E+00	1.80E+01
2007	9	2.64E+01	0.00E+00	1.41E+01
2007	10	5.53E+01	0.00E+00	1.61E+01
2007	11	5.76E+01	0.00E+00	1.96E+01
2007	12	1.09E+02	0.00E+00	2.61E+01
2008	1	3.67E+01	0.00E+00	8.52E+00
2008	2	3.44E+01	0.00E+00	1.09E+01

Table B.16 continued from previous page

Years	Months	Test Statistic	P-value	Mean Sales
2008	3	5.37E+01	0.00E+00	1.72E+01
2008	4	4.11E+01	0.00E+00	1.02E+01
2008	5	1.76E+01	0.00E+00	8.32E+00
2008	6	1.66E+01	0.00E+00	8.60E+00
2008	7	5.01E+01	0.00E+00	1.19E+01
2008	8	2.87E+01	0.00E+00	1.11E+01
2008	9	3.53E+01	0.00E+00	8.47E+00
2008	10	4.14E+01	0.00E+00	8.32E+00
2008	11	1.33E+02	0.00E+00	1.44E+01
2008	12	6.98E+01	0.00E+00	1.57E+01
2009	1	2.10E+01	0.00E+00	5.26E+00
2009	2	1.32E+01	0.00E+00	6.86E+00
2009	3	3.12E+02	0.00E+00	3.45E+01
2009	4	4.24E+01	0.00E+00	1.46E+01
2009	5	5.05E+01	0.00E+00	8.26E+00
2009	6	5.47E+01	0.00E+00	6.97E+00
2009	7	2.28E+02	0.00E+00	8.97E+00
2009	8	-2.58E-01	6.02E-01	9.68E-02
2009	9	1.52E+03	0.00E+00	3.29E+01
2009	10	2.90E+01	0.00E+00	7.74E+00
2009	11	3.18E+01	0.00E+00	6.53E+00
2009	12	1.72E+02	0.00E+00	1.32E+01
2010	1	3.25E+01	0.00E+00	1.04E+01
2010	2	2.52E+02	0.00E+00	2.46E+01
2010	3	6.22E+01	0.00E+00	1.98E+01
2010	4	8.92E+01	0.00E+00	2.43E+01
2010	5	7.13E+01	0.00E+00	1.87E+01
2010	6	2.47E+02	0.00E+00	2.68E+01
2010	7	7.89E+01	0.00E+00	1.90E+01

Table B.16 continued from previous page

Years	Months	Test Statistic	P-value	Mean Sales
2010	8	2.23E+01	0.00E+00	4.26E+00
2010	9	7.03E+01	0.00E+00	1.39E+01
2010	10	4.06E+01	0.00E+00	1.64E+01
2010	11	3.18E+02	0.00E+00	2.70E+01
2010	12	2.31E+02	0.00E+00	5.23E+01
2011	1	1.12E+02	0.00E+00	1.40E+01
2011	2	4.56E+01	0.00E+00	2.08E+01
2011	3	1.00E+02	0.00E+00	4.11E+01
2011	4	1.71E+02	0.00E+00	5.67E+01
2011	5	1.52E+02	0.00E+00	5.45E+01
2011	6	7.17E+01	0.00E+00	3.71E+01
2011	7	9.42E+01	0.00E+00	3.41E+01
2011	8	2.14E+02	0.00E+00	3.06E+01
2011	9	1.03E+02	0.00E+00	3.99E+01
2011	10	1.45E+02	0.00E+00	4.94E+01
2011	11	2.65E+02	0.00E+00	4.46E+01
2011	12	1.31E+02	0.00E+00	6.80E+01
2012	1	7.87E+01	0.00E+00	2.60E+01
2012	2	1.09E+02	0.00E+00	3.97E+01
2012	3	1.31E+02	0.00E+00	5.00E+01
2012	4	7.33E+01	0.00E+00	2.41E+01
2012	5	2.00E+02	0.00E+00	6.57E+01
2012	6	1.83E+02	0.00E+00	3.93E+01
2012	7	1.74E+02	0.00E+00	5.18E+01
2012	8	9.94E+01	0.00E+00	2.75E+01
2012	9	1.40E+02	0.00E+00	3.55E+01
2012	10	5.26E+02	0.00E+00	6.81E+01
2012	11	3.09E+02	0.00E+00	9.12E+01
2012	12	1.40E+02	0.00E+00	5.50E+01

Table B.16 continued from previous page

Years	Months	Test Statistic	P-value	Mean Sales
2013	1	1.98E+02	0.00E+00	2.21E+01
2013	2	1.90E+02	0.00E+00	5.67E+01
2013	3	2.21E+02	0.00E+00	7.20E+01
2013	4	3.75E+02	0.00E+00	8.04E+01
2013	5	1.69E+02	0.00E+00	5.40E+01
2013	6	2.48E+02	0.00E+00	5.00E+01
2013	7	1.31E+02	0.00E+00	6.27E+01
2013	8	3.58E+02	0.00E+00	7.23E+01
2013	9	2.72E+02	0.00E+00	6.87E+01
2013	10	3.35E+02	0.00E+00	6.61E+01
2013	11	1.09E+02	0.00E+00	5.60E+01
2013	12	1.87E+02	0.00E+00	7.68E+01
2014	1	1.65E+02	0.00E+00	2.09E+01
2014	2	7.66E+02	0.00E+00	6.07E+01
2014	3	2.56E+02	0.00E+00	4.79E+01
2014	4	5.14E+02	0.00E+00	6.98E+01
2014	5	4.08E+02	0.00E+00	5.72E+01
2014	6	1.98E+02	0.00E+00	4.10E+01
2014	7	1.71E+02	0.00E+00	5.73E+01
2014	8	2.94E+02	0.00E+00	5.67E+01
2014	9	9.41E+01	0.00E+00	4.24E+01
2014	10	1.73E+02	0.00E+00	5.30E+01
2014	11	1.52E+02	0.00E+00	5.84E+01
2014	12	1.92E+02	0.00E+00	8.44E+01
2015	1	6.49E+01	0.00E+00	1.34E+01
2015	2	3.72E+02	0.00E+00	6.96E+01
2015	3	3.09E+02	0.00E+00	8.76E+01
2015	4	3.30E+02	0.00E+00	9.15E+01
2015	5	2.13E+02	0.00E+00	6.83E+01

Table B.16 continued from previous page

Years	Months	Test Statistic	P-value	Mean Sales
2015	6	2.22E+02	0.00E+00	6.25E+01
2015	7	2.14E+02	0.00E+00	5.81E+01
2015	8	1.85E+02	0.00E+00	4.82E+01
2015	9	1.50E+02	0.00E+00	1.82E+01
2015	10	1.28E+02	0.00E+00	4.12E+01
2015	11	2.67E+02	0.00E+00	5.16E+01
2015	12	8.87E+02	0.00E+00	1.04E+02
2016	1	2.56E+02	0.00E+00	2.61E+01
2016	2	3.56E+02	0.00E+00	5.83E+01
2016	3	4.65E+02	0.00E+00	6.95E+01
2016	4	2.62E+02	0.00E+00	5.76E+01
2016	5	4.66E+02	0.00E+00	7.89E+01
2016	6	4.70E+02	0.00E+00	1.07E+02
2016	7	5.77E+02	0.00E+00	6.82E+01
2016	8	3.41E+02	0.00E+00	6.97E+01
2016	9	5.15E+02	0.00E+00	7.14E+01
2016	10	7.03E+02	0.00E+00	1.14E+02
2016	11	1.52E+02	0.00E+00	6.97E+01
2016	12	8.25E+02	0.00E+00	1.13E+02
2017	1	8.67E+01	0.00E+00	1.66E+01
2017	2	3.86E+02	0.00E+00	3.13E+01
2017	3	3.19E+02	0.00E+00	4.73E+01
2017	4	1.52E+02	0.00E+00	5.51E+01
2017	5	3.44E+02	0.00E+00	6.78E+01
2017	6	2.46E+02	0.00E+00	4.79E+01
2017	7	6.39E+02	0.00E+00	7.80E+01
2017	8	1.96E+02	0.00E+00	6.11E+01
2017	9	4.55E+02	0.00E+00	7.88E+01
2017	10	1.91E+02	0.00E+00	4.79E+01

Table B.16 continued from previous page

Years	Months	Test Statistic	P-value	Mean Sales
2017	11	2.68E+02	0.00E+00	6.23E+01
2017	12	3.13E+02	0.00E+00	7.59E+01
2018	1	3.76E+01	0.00E+00	1.28E+01
2018	2	1.49E+02	0.00E+00	2.53E+01
2018	3	8.89E+01	0.00E+00	3.56E+01
2018	4	2.00E+02	0.00E+00	3.78E+01
2018	5	2.36E+02	0.00E+00	5.59E+01
2018	6	2.50E+02	0.00E+00	3.69E+01
2018	7	9.50E+01	0.00E+00	3.18E+01
2018	8	7.52E+01	0.00E+00	1.82E+01
2018	9	8.67E+01	0.00E+00	1.02E+01
2018	10	9.50E+01	0.00E+00	9.48E+00
2018	11	5.59E+01	0.00E+00	2.47E+01
2018	12	1.24E+02	0.00E+00	2.74E+01
2019	1	3.08E+01	0.00E+00	5.00E+00
2019	2	1.33E+02	0.00E+00	1.55E+01
2019	3	8.56E+01	0.00E+00	1.84E+01
2019	4	1.12E+02	0.00E+00	2.07E+01
2019	5	4.25E+01	0.00E+00	1.15E+01
2019	6	1.39E+02	0.00E+00	1.47E+01
2019	7	7.86E+00	1.89E-15	1.23E+00
2019	8	1.42E+02	0.00E+00	1.87E+01
2019	9	9.01E+01	0.00E+00	2.22E+01
2019	10	2.23E+02	0.00E+00	3.07E+01
2019	11	6.62E+01	0.00E+00	1.38E+01
2019	12	4.59E+02	0.00E+00	2.28E+01
2020	1	1.35E+02	0.00E+00	5.87E+00
2020	2	1.97E+02	0.00E+00	1.64E+01
2020	3	9.15E+01	0.00E+00	1.77E+01

Table B.16 continued from previous page

Years	Months	Test Statistic	P-value	Mean Sales
2020	4	2.37E+01	0.00E+00	5.30E+00
2020	5	4.10E+02	0.00E+00	9.45E+00
2020	6	1.71E+02	0.00E+00	1.24E+01
2020	7	4.48E+01	0.00E+00	5.42E+00
2020	8	6.75E+00	7.48E-12	7.10E-01
2020	9	2.44E+02	0.00E+00	3.45E+01
2020	10	2.66E+02	0.00E+00	2.84E+01
2020	11	2.04E+03	0.00E+00	6.13E+01
2020	12	3.03E+02	0.00E+00	5.51E+01

The code block developed for Böhning's Test using quarters as consecutive time intervals is given below, and the test results are given in Table B.17.

```
test_stat=[]
p_val=[]

quarters=[4]
for year in range(1997,2021):
        quarters.append(1)
        quarters.append(2)
        quarters.append(3)
        quarters.append(4)

years=[1996]
for year in range(1997,2021):
        for t in range(0,4):
            years.append(year)

for year in range(1996,2021):
    for quarter in range(1,5):
        if (year==1996) & ((quarter==1) | (quarter==2) | (quarter
    ==3)):
            continue

        df=sample.loc[(sample.index.year==year) & (sample.index.
    quarter==quarter)]

        s=bohning_test(df.chassis)
        test_stat.append(s[0])
        p_val.append(s[1])

result=pd.DataFrame(list(zip(years, quarters, test_stat, p_val)),
    columns=['years', 'quarters', 'test_stat', 'p_val'])
pd.set_option('display.max_rows', result.shape[0]+1)
result
```

Table B.17 Results of the Böhning's Test when using quarters as successive time intervals.

Years	Quarters	Test Statistic	P-value
1996	4	2.84E+02	0.00E+00
1997	1	1.99E+02	0.00E+00
1997	2	4.62E+02	0.00E+00
1997	3	1.55E+02	0.00E+00
1997	4	4.47E+02	0.00E+00
1998	1	2.00E+02	0.00E+00
1998	2	2.44E+02	0.00E+00
1998	3	3.12E+02	0.00E+00
1998	4	1.94E+02	0.00E+00
1999	1	1.18E+02	0.00E+00
1999	2	3.47E+02	0.00E+00
1999	3	2.29E+02	0.00E+00
1999	4	2.72E+02	0.00E+00
2000	1	1.71E+02	0.00E+00
2000	2	1.69E+02	0.00E+00
2000	3	1.19E+02	0.00E+00
2000	4	1.31E+02	0.00E+00
2001	1	5.60E+01	0.00E+00
2001	2	2.22E+01	0.00E+00
2001	3	4.51E+00	3.18E-06
2001	4	2.69E+01	0.00E+00
2002	1	6.61E+00	1.93E-11
2002	2	1.84E+00	3.28E-02
2002	3	2.91E+00	1.83E-03
2002	4	4.72E+00	1.18E-06
2003	1	-1.50E-01	5.60E-01

Table B.17 continued from previous page

Years	Quarters	Test Statistic	P-value
2003	2	6.77E+00	6.54E-12
2003	3	6.65E+00	1.48E-11
2003	4	2.12E+01	0.00E+00
2004	1	1.54E+01	0.00E+00
2004	2	2.06E+01	0.00E+00
2004	3	8.59E+00	0.00E+00
2004	4	3.09E+01	0.00E+00
2005	1	8.84E+04	0.00E+00
2005	2	9.15E+01	0.00E+00
2005	3	7.44E+01	0.00E+00
2005	4	1.23E+02	0.00E+00
2006	1	1.01E+02	0.00E+00
2006	2	7.59E+01	0.00E+00
2006	3	3.77E+01	0.00E+00
2006	4	1.85E+02	0.00E+00
2007	1	3.24E+01	0.00E+00
2007	2	7.01E+01	0.00E+00
2007	3	5.32E+01	0.00E+00
2007	4	1.40E+02	0.00E+00
2008	1	8.28E+01	0.00E+00
2008	2	4.51E+01	0.00E+00
2008	3	6.73E+01	0.00E+00
2008	4	1.54E+02	0.00E+00
2009	1	4.80E+02	0.00E+00
2009	2	8.89E+01	0.00E+00
2009	3	2.23E+03	0.00E+00
2009	4	1.75E+02	0.00E+00
2010	1	2.53E+02	0.00E+00
2010	2	2.51E+02	0.00E+00

Table B.17 continued from previous page

Years	Quarters	Test Statistic	P-value
2010	3	1.39E+02	0.00E+00
2010	4	4.29E+02	0.00E+00
2011	1	1.86E+02	0.00E+00
2011	2	2.48E+02	0.00E+00
2011	3	2.30E+02	0.00E+00
2011	4	3.07E+02	0.00E+00
2012	1	2.09E+02	0.00E+00
2012	2	3.39E+02	0.00E+00
2012	3	2.68E+02	0.00E+00
2012	4	5.95E+02	0.00E+00
2013	1	4.14E+02	0.00E+00
2013	2	4.99E+02	0.00E+00
2013	3	4.46E+02	0.00E+00
2013	4	3.75E+02	0.00E+00
2014	1	8.59E+02	0.00E+00
2014	2	7.03E+02	0.00E+00
2014	3	3.40E+02	0.00E+00
2014	4	3.20E+02	0.00E+00
2015	1	6.63E+02	0.00E+00
2015	2	4.65E+02	0.00E+00
2015	3	3.77E+02	0.00E+00
2015	4	1.05E+03	0.00E+00
2016	1	7.12E+02	0.00E+00
2016	2	7.53E+02	0.00E+00
2016	3	8.17E+02	0.00E+00
2016	4	1.09E+03	0.00E+00
2017	1	5.46E+02	0.00E+00
2017	2	4.44E+02	0.00E+00
2017	3	7.75E+02	0.00E+00

Table B.17 continued from previous page

Years	Quarters	Test Statistic	P-value
2017	4	4.71E+02	0.00E+00
2018	1	1.95E+02	0.00E+00
2018	2	4.05E+02	0.00E+00
2018	3	1.77E+02	0.00E+00
2018	4	1.79E+02	0.00E+00
2019	1	1.84E+02	0.00E+00
2019	2	1.83E+02	0.00E+00
2019	3	2.32E+02	0.00E+00
2019	4	4.78E+02	0.00E+00
2020	1	2.56E+02	0.00E+00
2020	2	3.94E+02	0.00E+00
2020	3	4.81E+02	0.00E+00
2020	4	1.79E+03	0.00E+00

The code block developed for Böhning's Test using seasons as consecutive time intervals is given below, and the test results are given in Table B.18.

```
test_stat=[]
p_val=[]

years=[1996]
for year in range(1997,2021):
    for t in range(0,4):
        years.append(year)

seasons=["autumn"]
for i in range(0,24):
    for season in ["winter","spring","summer","autumn"]:
        seasons.append(season)

start_date=pd.Timestamp(1996,9,1)

for iteration in range(0,98):
    fin_date = start_date + relativedelta(months=+3)

    df=sample[(sample.index>=start_date) & (sample.index<fin_date)]

    #some samples' sales records are consist of only zeros
    #those days are given a p-value of 0
    #to prevent error arising due to division by zero
    if all([ v == 0 for v in df.sales_quant]):
```

```
        test_stat.append(None)
        p_val.append(0)
        continue

    start_date = fin_date

    s=bohning_test(df.chassis)
    test_stat.append(s[0])
    p_val.append(s[1])

result=pd.DataFrame(list(zip(years, seasons, test_stat, p_val)),
   columns=['years', 'seasons','test_stat', 'p_val'])
pd.set_option('display.max_rows', result.shape[0]+1)
result
```

Table B.18 Results of the Böhning's Test when using seasons as successive time intervals.

Years	Seasons	Test Statistic	P-value
1996	autumn	2.88E+02	0.00E+00
1997	winter	1.85E+02	0.00E+00
1997	spring	5.23E+02	0.00E+00
1997	summer	1.71E+02	0.00E+00
1997	autumn	2.37E+02	0.00E+00
1998	winter	4.87E+02	0.00E+00
1998	spring	2.01E+02	0.00E+00
1998	summer	2.57E+02	0.00E+00
1998	autumn	3.37E+02	0.00E+00
1999	winter	1.75E+02	0.00E+00
1999	spring	3.16E+02	0.00E+00
1999	summer	1.66E+02	0.00E+00
1999	autumn	2.55E+02	0.00E+00
2000	winter	1.57E+02	0.00E+00
2000	spring	1.91E+02	0.00E+00
2000	summer	8.19E+01	0.00E+00
2000	autumn	1.18E+02	0.00E+00
2001	winter	1.34E+02	0.00E+00
2001	spring	3.29E+01	0.00E+00

Table B.18 continued from previous page

Years	Seasons	Test Statistic	P-value
2001	summer	2.15E+01	0.00E+00
2001	autumn	4.74E+00	1.07E-06
2002	winter	3.63E+01	0.00E+00
2002	spring	2.82E+00	2.43E-03
2002	summer	2.97E+00	1.47E-03
2002	autumn	2.52E+00	5.88E-03
2003	winter	6.52E+00	3.49E-11
2003	spring	5.63E+00	8.83E-09
2003	summer	2.18E+00	1.48E-02
2003	autumn	2.28E+01	0.00E+00
2004	winter	1.66E+01	0.00E+00
2004	spring	1.70E+01	0.00E+00
2004	summer	1.13E+01	0.00E+00
2004	autumn	2.94E+01	0.00E+00
2005	winter	9.15E+04	0.00E+00
2005	spring	7.49E+01	0.00E+00
2005	summer	7.58E+01	0.00E+00
2005	autumn	1.02E+02	0.00E+00
2006	winter	1.53E+02	0.00E+00
2006	spring	8.51E+01	0.00E+00
2006	summer	5.59E+01	0.00E+00
2006	autumn	4.25E+01	0.00E+00
2007	winter	1.85E+02	0.00E+00
2007	spring	4.58E+01	0.00E+00
2007	summer	6.96E+01	0.00E+00
2007	autumn	8.42E+01	0.00E+00
2008	winter	1.60E+02	0.00E+00
2008	spring	7.96E+01	0.00E+00
2008	summer	5.86E+01	0.00E+00

Table B.18 continued from previous page

Years	Seasons	Test Statistic	P-value
2008	autumn	1.45E+02	0.00E+00
2009	winter	9.56E+01	0.00E+00
2009	spring	3.98E+02	0.00E+00
2009	summer	2.80E+02	0.00E+00
2009	autumn	1.89E+03	0.00E+00
2010	winter	3.28E+02	0.00E+00
2010	spring	1.31E+02	0.00E+00
2010	summer	3.16E+02	0.00E+00
2010	autumn	3.16E+02	0.00E+00
2011	winter	3.54E+02	0.00E+00
2011	spring	2.54E+02	0.00E+00
2011	summer	2.11E+02	0.00E+00
2011	autumn	2.97E+02	0.00E+00
2012	winter	2.43E+02	0.00E+00
2012	spring	3.05E+02	0.00E+00
2012	summer	2.91E+02	0.00E+00
2012	autumn	6.61E+02	0.00E+00
2013	winter	3.32E+02	0.00E+00
2013	spring	4.69E+02	0.00E+00
2013	summer	4.39E+02	0.00E+00
2013	autumn	4.25E+02	0.00E+00
2014	winter	7.58E+02	0.00E+00
2014	spring	7.09E+02	0.00E+00
2014	summer	3.90E+02	0.00E+00
2014	autumn	2.51E+02	0.00E+00
2015	winter	5.54E+02	0.00E+00
2015	spring	5.05E+02	0.00E+00
2015	summer	3.62E+02	0.00E+00
2015	autumn	3.71E+02	0.00E+00

Table B.18 continued from previous page

Years	Seasons	Test Statistic	P-value
2016	winter	1.21E+03	0.00E+00
2016	spring	7.08E+02	0.00E+00
2016	summer	8.21E+02	0.00E+00
2016	autumn	8.90E+02	0.00E+00
2017	winter	1.36E+03	0.00E+00
2017	spring	4.80E+02	0.00E+00
2017	summer	6.91E+02	0.00E+00
2017	autumn	5.76E+02	0.00E+00
2018	winter	5.54E+02	0.00E+00
2018	spring	3.29E+02	0.00E+00
2018	summer	2.83E+02	0.00E+00
2018	autumn	1.45E+02	0.00E+00
2019	winter	2.37E+02	0.00E+00
2019	spring	1.54E+02	0.00E+00
2019	summer	2.66E+02	0.00E+00
2019	autumn	2.65E+02	0.00E+00
2020	winter	5.76E+02	0.00E+00
2020	spring	3.13E+02	0.00E+00
2020	summer	2.44E+02	0.00E+00
2020	autumn	1.98E+03	0.00E+00

B.3 Brown's Test

In this section, codes blocks and results of the Brown's Tests for successive time intervals are presented. Following libraries are imported for data manipulation and constructing the test procedure.

```
import pandas as pd
import numpy as np
import scipy.stats as stats
from dateutil.relativedelta import *
from datetime import datetime
```

The code block given below is the function that implements Brown's test procedure.

```
 def browns_test(x,block_length):
   #x: arrival times
   #block_length: length of arrival period, e.g. month, quarter, week etc.
   n = x.size
   x_prime=np.delete(x,n-1)
   x_prime=np.insert(x_prime, 0, 0)
   jvect = np.arange(1,n+1)
   std_exp_var = -(n+1-jvect)*np.log((block_length-x)/(block_length-x_prime))
   test_res = stats.kstest(std_exp_var,"expon")
   return test_res[0], test_res[1]
```

The code block developed for Brown's Test using fortnights as consecutive time intervals is given below, and the test results are given in Table B.19.

```
test_stat=[]
p_val=[]
dates=[]
start_date=pd.Timestamp(2005,1,6)

while start_date<=pd.Timestamp(2020,12,17):
    fin_date = start_date + relativedelta(weeks=+2)

    intersales_time=[]
    df=sales_after2005.loc[(sales_after2005.index>=start_date) & (sales_after2005.index<fin_date)]
    df.reset_index(drop=True, inplace=True)
    #print(start_date)
    date='%s & %s' % (start_date, fin_date)

    #some samples have a sole day with sales record
    #those days are given a p-value of 0
    #to prevent errors arising since std. dev and variance are cannot be calculated
    if df.shape[0]<2:
        test_stat.append(None)
        p_val.append(0)
        start_date = fin_date
        continue

    for i in range(df.index.max()):
        intersales_time.append(df.invoice_date.loc[i+1]-df.invoice_date.loc[i])

    intersales_time=pd.Series(intersales_time)

    intersales_time_h=convert_to_hours(intersales_time)
    intersales_time_h=np.array(intersales_time_h)
    sales_time_h=np.cumsum(intersales_time_h)

    block_length_hour=((fin_date-start_date).days)*24

    res_hour=browns_test(sales_time_h,block_length_hour)

    test_stat.append(res_hour[0])
    p_val.append(res_hour[1])
```

```
    dates.append(date)
    start_date = fin_date

result=pd.DataFrame(list(zip(dates, test_stat, p_val)), columns=['
    dates','test_stat', 'p_val'])
pd.set_option('display.max_rows', result.shape[0]+1)
result
```

Table B.19 Results of the Brown's Test when using fortnights as successive time intervals.

Dates	Test Statistic	P-value
06/01/2005-20/01/2005	4.32E-01	1.15E-09
20/01/2005-03/02/2005	4.89E-01	6.40E-18
03/02/2005-17/02/2005	3.28E-01	1.70E-17
17/02/2005-03/03/2005	4.33E-01	5.42E-45
03/03/2005-17/03/2005	4.52E-01	1.35E-43
17/03/2005-31/03/2005	4.21E-01	1.19E-42
31/03/2005-14/04/2005	4.62E-01	3.22E-43
14/04/2005-28/04/2005	5.02E-01	4.19E-34
28/04/2005-12/05/2005	3.92E-01	3.44E-31
12/05/2005-26/05/2005	3.79E-01	1.20E-46
26/05/2005-09/06/2005	4.89E-01	4.91E-98
09/06/2005-23/06/2005	3.81E-01	4.93E-46
23/06/2005-07/07/2005	4.10E-01	4.93E-58
07/07/2005-21/07/2005	4.09E-01	3.77E-57
21/07/2005-04/08/2005	4.16E-01	9.54E-53
04/08/2005-18/08/2005	4.36E-01	4.86E-69
18/08/2005-01/09/2005	4.70E-01	4.99E-81
01/09/2005-15/09/2005	4.29E-01	3.67E-44
15/09/2005-29/09/2005	4.48E-01	3.79E-59
29/09/2005-13/10/2005	4.36E-01	1.44E-59
13/10/2005-27/10/2005	4.34E-01	1.55E-78
27/10/2005-10/11/2005	5.61E-01	1.19E-100

Table B.19 continued from previous page

Dates	Test Statistic	P-value
10/11/2005-24/11/2005	4.52E-01	3.37E-88
24/11/2005-08/12/2005	4.34E-01	2.70E-86
08/12/2005-22/12/2005	4.15E-01	5.16E-74
22/12/2005-05/01/2006	4.65E-01	5.84E-122
05/01/2006-19/01/2006	6.03E-01	2.53E-35
19/01/2006-02/02/2006	3.84E-01	6.93E-24
02/02/2006-16/02/2006	4.13E-01	1.73E-30
16/02/2006-02/03/2006	4.78E-01	3.47E-65
02/03/2006-16/03/2006	4.16E-01	3.08E-43
16/03/2006-30/03/2006	4.20E-01	2.52E-71
30/03/2006-13/04/2006	4.40E-01	1.55E-65
13/04/2006-27/04/2006	4.71E-01	9.65E-68
27/04/2006-11/05/2006	4.72E-01	4.73E-48
11/05/2006-25/05/2006	3.60E-01	1.73E-22
25/05/2006-08/06/2006	5.53E-01	2.32E-55
08/06/2006-22/06/2006	3.66E-01	4.74E-19
22/06/2006-06/07/2006	4.87E-01	3.77E-64
06/07/2006-20/07/2006	5.06E-01	2.03E-31
20/07/2006-03/08/2006	3.97E-01	1.94E-27
03/08/2006-17/08/2006	4.54E-01	3.08E-31
17/08/2006-31/08/2006	5.47E-01	6.32E-39
31/08/2006-14/09/2006	4.91E-01	5.64E-25
14/09/2006-28/09/2006	4.14E-01	6.01E-17
28/09/2006-12/10/2006	4.32E-01	6.44E-21
12/10/2006-26/10/2006	6.12E-01	7.56E-36
26/10/2006-09/11/2006	5.03E-01	5.00E-34
09/11/2006-23/11/2006	4.85E-01	7.22E-37
23/11/2006-07/12/2006	4.38E-01	3.11E-39
07/12/2006-21/12/2006	4.64E-01	3.78E-53

Table B.19 continued from previous page

Dates	Test Statistic	P-value
21/12/2006-04/01/2007	5.74E-01	1.50E-163
04/01/2007-18/01/2007	4.75E-01	5.34E-27
18/01/2007-01/02/2007	4.77E-01	1.36E-34
01/02/2007-15/02/2007	4.27E-01	2.15E-24
15/02/2007-01/03/2007	4.26E-01	4.37E-26
01/03/2007-15/03/2007	4.69E-01	1.59E-27
15/03/2007-29/03/2007	4.80E-01	1.28E-31
29/03/2007-12/04/2007	4.76E-01	4.35E-31
12/04/2007-26/04/2007	4.38E-01	4.05E-22
26/04/2007-10/05/2007	5.02E-01	7.96E-45
10/05/2007-24/05/2007	5.15E-01	3.17E-37
24/05/2007-07/06/2007	5.06E-01	1.07E-34
07/06/2007-21/06/2007	5.10E-01	1.24E-46
21/06/2007-05/07/2007	4.34E-01	4.77E-35
05/07/2007-19/07/2007	4.41E-01	6.67E-26
19/07/2007-02/08/2007	3.23E-01	4.85E-19
02/08/2007-16/08/2007	4.42E-01	3.09E-46
16/08/2007-30/08/2007	4.19E-01	3.30E-38
30/08/2007-13/09/2007	4.43E-01	1.76E-32
13/09/2007-27/09/2007	4.07E-01	5.28E-34
27/09/2007-11/10/2007	4.38E-01	9.55E-45
11/10/2007-25/10/2007	4.25E-01	5.44E-37
25/10/2007-08/11/2007	4.12E-01	3.45E-26
08/11/2007-22/11/2007	4.70E-01	1.28E-55
22/11/2007-06/12/2007	4.79E-01	9.96E-59
06/12/2007-20/12/2007	4.17E-01	5.03E-43
20/12/2007-03/01/2008	6.25E-01	8.17E-197
03/01/2008-17/01/2008	4.00E-01	2.66E-09
17/01/2008-31/01/2008	4.16E-01	2.29E-21

Table B.19 continued from previous page

Dates	Test Statistic	P-value
31/01/2008-14/02/2008	4.95E-01	7.87E-19
14/02/2008-28/02/2008	4.91E-01	7.86E-47
28/02/2008-13/03/2008	4.99E-01	5.02E-58
13/03/2008-27/03/2008	4.63E-01	7.46E-45
27/03/2008-10/04/2008	5.48E-01	1.23E-54
10/04/2008-24/04/2008	4.55E-01	5.77E-21
24/04/2008-08/05/2008	5.17E-01	2.83E-40
08/05/2008-22/05/2008	3.85E-01	2.26E-12
22/05/2008-05/06/2008	4.79E-01	8.22E-32
05/06/2008-19/06/2008	4.92E-01	1.77E-23
19/06/2008-03/07/2008	4.12E-01	3.53E-25
03/07/2008-17/07/2008	4.30E-01	5.84E-19
17/07/2008-31/07/2008	4.71E-01	3.39E-30
31/07/2008-14/08/2008	5.07E-01	1.80E-51
14/08/2008-28/08/2008	4.48E-01	5.16E-26
28/08/2008-11/09/2008	5.08E-01	1.27E-32
11/09/2008-25/09/2008	3.61E-01	4.56E-12
25/09/2008-09/10/2008	5.85E-01	1.01E-33
09/10/2008-23/10/2008	4.47E-01	4.61E-31
23/10/2008-06/11/2008	4.78E-01	7.66E-20
06/11/2008-20/11/2008	5.95E-01	7.70E-87
20/11/2008-04/12/2008	4.40E-01	2.49E-46
04/12/2008-18/12/2008	5.19E-01	1.33E-20
18/12/2008-01/01/2009	3.83E-01	1.42E-38
01/01/2009-15/01/2009	5.10E-01	4.03E-12
15/01/2009-29/01/2009	4.77E-01	3.08E-21
29/01/2009-12/02/2009	4.64E-01	1.10E-15
12/02/2009-26/02/2009	3.74E-01	6.57E-13
26/02/2009-12/03/2009	4.01E-01	7.63E-20

Table B.19 continued from previous page

Dates	Test Statistic	P-value
12/03/2009-26/03/2009	4.12E-01	3.66E-106
26/03/2009-09/04/2009	5.14E-01	1.96E-115
09/04/2009-23/04/2009	4.28E-01	3.09E-30
23/04/2009-07/05/2009	5.21E-01	1.48E-23
07/05/2009-21/05/2009	4.98E-01	5.17E-39
21/05/2009-04/06/2009	5.07E-01	6.03E-27
04/06/2009-18/06/2009	6.08E-01	7.32E-45
18/06/2009-02/07/2009	4.34E-01	1.07E-06
02/07/2009-16/07/2009	4.75E-01	4.28E-17
16/07/2009-30/07/2009	5.93E-01	4.21E-54
30/07/2009-13/08/2009	8.07E-01	2.17E-22
10/09/2009-24/09/2009	NaN	0.00E+00
24/09/2009-08/10/2009	NaN	0.00E+00
08/10/2009-22/10/2009	7.11E-01	3.03E-12
22/10/2009-05/11/2009	6.75E-01	0.00E+00
05/11/2009-19/11/2009	4.22E-01	1.59E-20
19/11/2009-03/12/2009	3.88E-01	5.61E-18
03/12/2009-17/12/2009	4.91E-01	2.06E-15
17/12/2009-31/12/2009	5.92E-01	1.25E-24
31/12/2009-14/01/2010	4.62E-01	7.15E-20
14/01/2010-28/01/2010	3.20E-01	6.37E-16
28/01/2010-11/02/2010	6.17E-01	7.44E-86
11/02/2010-25/02/2010	4.94E-01	7.41E-43
25/02/2010-11/03/2010	4.17E-01	1.06E-36
11/03/2010-25/03/2010	5.29E-01	1.15E-113
25/03/2010-08/04/2010	4.75E-01	1.37E-58
08/04/2010-22/04/2010	4.69E-01	1.53E-59
22/04/2010-06/05/2010	4.10E-01	2.06E-52
06/05/2010-20/05/2010	4.16E-01	1.25E-37

Table B.19 continued from previous page

Dates	Test Statistic	P-value
20/05/2010-03/06/2010	5.34E-01	7.84E-77
03/06/2010-17/06/2010	4.81E-01	1.01E-54
17/06/2010-01/07/2010	5.24E-01	2.99E-97
01/07/2010-15/07/2010	5.12E-01	1.47E-108
15/07/2010-29/07/2010	4.55E-01	9.91E-48
29/07/2010-12/08/2010	5.40E-01	6.60E-75
12/08/2010-26/08/2010	4.25E-01	3.51E-29
26/08/2010-09/09/2010	5.92E-01	4.21E-69
09/09/2010-23/09/2010	3.94E-01	4.40E-07
23/09/2010-07/10/2010	5.06E-01	3.60E-07
07/10/2010-21/10/2010	5.90E-01	3.26E-63
21/10/2010-04/11/2010	4.58E-01	6.40E-71
04/11/2010-18/11/2010	4.55E-01	1.71E-41
18/11/2010-02/12/2010	4.08E-01	3.48E-32
02/12/2010-16/12/2010	8.05E-01	1.60E-261
16/12/2010-30/12/2010	5.93E-01	8.77E-143
30/12/2010-13/01/2011	4.21E-01	9.86E-69
13/01/2011-27/01/2011	4.00E-01	7.45E-110
27/01/2011-10/02/2011	5.71E-01	2.86E-208
10/02/2011-24/02/2011	4.97E-01	2.54E-36
24/02/2011-10/03/2011	4.11E-01	2.07E-33
10/03/2011-24/03/2011	4.21E-01	2.72E-53
24/03/2011-07/04/2011	4.52E-01	4.28E-88
07/04/2011-21/04/2011	4.14E-01	4.08E-88
21/04/2011-05/05/2011	4.59E-01	1.16E-114
05/05/2011-19/05/2011	3.72E-01	2.10E-96
19/05/2011-02/06/2011	4.94E-01	1.56E-249
02/06/2011-16/06/2011	5.22E-01	1.79E-215
16/06/2011-30/06/2011	4.62E-01	1.09E-104

Table B.19 continued from previous page

Dates	Test Statistic	P-value
30/06/2011-14/07/2011	4.92E-01	2.54E-120
14/07/2011-28/07/2011	5.14E-01	1.28E-110
28/07/2011-11/08/2011	5.07E-01	6.63E-126
11/08/2011-25/08/2011	4.44E-01	3.88E-86
25/08/2011-08/09/2011	5.03E-01	1.37E-91
08/09/2011-22/09/2011	4.43E-01	5.76E-55
22/09/2011-06/10/2011	6.01E-01	2.85E-215
06/10/2011-20/10/2011	4.37E-01	1.52E-79
20/10/2011-03/11/2011	4.72E-01	4.71E-137
03/11/2011-17/11/2011	4.63E-01	4.74E-111
17/11/2011-01/12/2011	5.37E-01	1.06E-255
01/12/2011-15/12/2011	4.45E-01	2.98E-61
15/12/2011-29/12/2011	3.77E-01	9.06E-112
29/12/2011-12/01/2012	4.95E-01	3.79E-154
12/01/2012-26/01/2012	4.56E-01	7.30E-203
26/01/2012-09/02/2012	5.97E-01	1.66E-189
09/02/2012-23/02/2012	4.52E-01	4.83E-70
23/02/2012-08/03/2012	4.61E-01	5.83E-98
08/03/2012-22/03/2012	5.19E-01	1.34E-105
22/03/2012-05/04/2012	5.09E-01	1.55E-185
05/04/2012-19/04/2012	5.05E-01	1.20E-201
19/04/2012-03/05/2012	5.33E-01	1.15E-126
03/05/2012-17/05/2012	5.21E-01	6.16E-119
17/05/2012-31/05/2012	4.29E-01	2.21E-61
31/05/2012-14/06/2012	5.79E-01	0.00E+00
14/06/2012-28/06/2012	5.36E-01	6.09E-188
28/06/2012-12/07/2012	4.75E-01	3.76E-80
12/07/2012-26/07/2012	4.72E-01	5.12E-97
26/07/2012-09/08/2012	4.97E-01	2.87E-199

Table B.19 continued from previous page

Dates	Test Statistic	P-value
09/08/2012-23/08/2012	5.16E-01	7.94E-195
23/08/2012-06/09/2012	5.84E-01	1.41E-160
06/09/2012-20/09/2012	6.21E-01	3.77E-127
20/09/2012-04/10/2012	5.90E-01	1.09E-120
04/10/2012-18/10/2012	5.81E-01	4.50E-136
18/10/2012-01/11/2012	6.06E-01	1.88E-227
01/11/2012-15/11/2012	4.35E-01	3.40E-144
15/11/2012-29/11/2012	4.96E-01	1.98E-283
29/11/2012-13/12/2012	4.45E-01	5.50E-146
13/12/2012-27/12/2012	5.28E-01	0.00E+00
27/12/2012-10/01/2013	5.85E-01	0.00E+00
10/01/2013-24/01/2013	4.24E-01	5.01E-110
24/01/2013-07/02/2013	6.41E-01	2.84E-223
07/02/2013-21/02/2013	4.81E-01	5.12E-32
21/02/2013-07/03/2013	6.03E-01	9.71E-192
07/03/2013-21/03/2013	5.13E-01	9.74E-180
21/03/2013-04/04/2013	6.34E-01	0.00E+00
04/04/2013-18/04/2013	5.77E-01	0.00E+00
18/04/2013-02/05/2013	5.36E-01	3.37E-236
02/05/2013-16/05/2013	4.94E-01	3.14E-281
16/05/2013-30/05/2013	5.23E-01	6.90E-268
30/05/2013-13/06/2013	4.63E-01	2.57E-189
13/06/2013-27/06/2013	5.12E-01	1.12E-131
27/06/2013-11/07/2013	5.08E-01	9.95E-156
11/07/2013-25/07/2013	5.67E-01	1.53E-269
25/07/2013-08/08/2013	4.24E-01	2.81E-120
08/08/2013-22/08/2013	4.35E-01	6.80E-152
22/08/2013-05/09/2013	4.81E-01	6.81E-203
05/09/2013-19/09/2013	6.12E-01	0.00E+00

Table B.19 continued from previous page

Dates	Test Statistic	P-value
19/09/2013-03/10/2013	5.54E-01	7.35E-196
03/10/2013-17/10/2013	4.30E-01	3.94E-120
17/10/2013-31/10/2013	5.17E-01	0.00E+00
31/10/2013-14/11/2013	6.19E-01	2.55E-273
14/11/2013-28/11/2013	6.08E-01	0.00E+00
28/11/2013-12/12/2013	4.99E-01	2.26E-183
12/12/2013-26/12/2013	4.78E-01	8.14E-179
26/12/2013-09/01/2014	4.20E-01	1.69E-127
09/01/2014-23/01/2014	4.90E-01	1.00E-263
23/01/2014-06/02/2014	6.39E-01	1.27E-298
06/02/2014-20/02/2014	4.53E-01	1.62E-35
20/02/2014-06/03/2014	5.26E-01	1.09E-146
06/03/2014-20/03/2014	4.29E-01	9.00E-63
20/03/2014-03/04/2014	6.03E-01	0.00E+00
03/04/2014-17/04/2014	4.91E-01	8.17E-114
17/04/2014-01/05/2014	5.20E-01	2.53E-240
01/05/2014-15/05/2014	4.76E-01	1.38E-113
15/05/2014-29/05/2014	3.84E-01	3.06E-199
29/05/2014-12/06/2014	5.36E-01	1.29E-138
12/06/2014-26/06/2014	5.08E-01	5.96E-167
26/06/2014-10/07/2014	6.12E-01	0.00E+00
10/07/2014-24/07/2014	5.04E-01	4.21E-101
24/07/2014-07/08/2014	5.43E-01	9.56E-213
07/08/2014-21/08/2014	4.45E-01	1.00E-190
21/08/2014-04/09/2014	5.66E-01	2.91E-147
04/09/2014-18/09/2014	4.17E-01	1.61E-115
18/09/2014-02/10/2014	4.71E-01	3.72E-199
02/10/2014-16/10/2014	4.44E-01	1.06E-91
16/10/2014-30/10/2014	4.01E-01	5.30E-111

Table B.19 continued from previous page

Dates	Test Statistic	P-value
30/10/2014-13/11/2014	4.50E-01	1.82E-102
13/11/2014-27/11/2014	5.16E-01	4.59E-191
27/11/2014-11/12/2014	4.70E-01	9.92E-142
11/12/2014-25/12/2014	4.06E-01	2.06E-136
25/12/2014-08/01/2015	4.80E-01	2.89E-193
08/01/2015-22/01/2015	4.09E-01	2.21E-180
22/01/2015-05/02/2015	6.07E-01	0.00E+00
05/02/2015-19/02/2015	4.36E-01	5.81E-21
19/02/2015-05/03/2015	3.61E-01	2.81E-44
05/03/2015-19/03/2015	4.36E-01	3.09E-109
19/03/2015-02/04/2015	5.07E-01	0.00E+00
02/04/2015-16/04/2015	4.07E-01	2.59E-231
16/04/2015-30/04/2015	4.33E-01	1.48E-153
30/04/2015-14/05/2015	4.32E-01	2.45E-175
14/05/2015-28/05/2015	3.77E-01	7.02E-156
28/05/2015-11/06/2015	5.25E-01	0.00E+00
11/06/2015-25/06/2015	4.51E-01	5.49E-168
25/06/2015-09/07/2015	5.18E-01	1.09E-250
09/07/2015-23/07/2015	5.34E-01	1.31E-179
23/07/2015-06/08/2015	5.20E-01	1.26E-306
06/08/2015-20/08/2015	4.50E-01	5.89E-95
20/08/2015-03/09/2015	5.33E-01	2.22E-191
03/09/2015-17/09/2015	4.93E-01	5.51E-142
17/09/2015-01/10/2015	4.82E-01	1.88E-192
01/10/2015-15/10/2015	4.14E-01	4.15E-24
15/10/2015-29/10/2015	4.36E-01	1.10E-52
29/10/2015-12/11/2015	4.30E-01	5.25E-78
12/11/2015-26/11/2015	4.69E-01	1.44E-134
26/11/2015-10/12/2015	5.28E-01	6.16E-94

Table B.19 continued from previous page

Dates	Test Statistic	P-value
10/12/2015-24/12/2015	4.07E-01	2.56E-101
24/12/2015-07/01/2016	5.57E-01	0.00E+00
07/01/2016-21/01/2016	4.75E-01	1.94E-132
21/01/2016-04/02/2016	5.99E-01	0.00E+00
04/02/2016-18/02/2016	4.95E-01	4.61E-130
18/02/2016-03/03/2016	4.31E-01	2.91E-42
03/03/2016-17/03/2016	3.54E-01	2.57E-59
17/03/2016-31/03/2016	4.43E-01	3.54E-238
31/03/2016-14/04/2016	3.91E-01	5.83E-69
14/04/2016-28/04/2016	4.05E-01	4.61E-152
28/04/2016-12/05/2016	6.05E-01	1.471734e-314
12/05/2016-26/05/2016	4.86E-01	1.28E-172
26/05/2016-09/06/2016	4.83E-01	4.00E-224
09/06/2016-23/06/2016	4.54E-01	5.07E-170
23/06/2016-07/07/2016	5.39E-01	0.00E+00
07/07/2016-21/07/2016	4.46E-01	5.77E-206
21/07/2016-04/08/2016	6.06E-01	0.00E+00
04/08/2016-18/08/2016	4.60E-01	1.87E-128
18/08/2016-01/09/2016	4.92E-01	0.00E+00
01/09/2016-15/09/2016	4.57E-01	1.42E-128
15/09/2016-29/09/2016	3.92E-01	1.82E-182
29/09/2016-13/10/2016	6.11E-01	6.27E-123
13/10/2016-27/10/2016	4.98E-01	4.37E-254
27/10/2016-10/11/2016	5.72E-01	0.00E+00
10/11/2016-24/11/2016	4.82E-01	0.00E+00
24/11/2016-08/12/2016	5.95E-01	0.00E+00
08/12/2016-22/12/2016	4.13E-01	3.19E-141
22/12/2016-05/01/2017	5.53E-01	0.00E+00
05/01/2017-19/01/2017	4.80E-01	8.00E-180

Table B.19 continued from previous page

Dates	Test Statistic	P-value
19/01/2017-02/02/2017	5.85E-01	0.00E+00
02/02/2017-16/02/2017	4.37E-01	4.86E-52
16/02/2017-02/03/2017	4.35E-01	1.07E-36
02/03/2017-16/03/2017	4.53E-01	1.46E-53
16/03/2017-30/03/2017	4.51E-01	5.66E-111
30/03/2017-13/04/2017	3.58E-01	1.03E-41
13/04/2017-27/04/2017	3.59E-01	6.24E-79
27/04/2017-11/05/2017	5.34E-01	2.09E-256
11/05/2017-25/05/2017	3.60E-01	1.55E-90
25/05/2017-08/06/2017	4.88E-01	6.63E-177
08/06/2017-22/06/2017	4.53E-01	4.24E-118
22/06/2017-06/07/2017	5.48E-01	0.00E+00
06/07/2017-20/07/2017	5.24E-01	3.51E-192
20/07/2017-03/08/2017	5.01E-01	1.34E-185
03/08/2017-17/08/2017	4.69E-01	1.38E-174
17/08/2017-31/08/2017	5.14E-01	0.00E+00
31/08/2017-14/09/2017	3.97E-01	4.62E-114
14/09/2017-28/09/2017	4.64E-01	1.05E-191
28/09/2017-12/10/2017	4.25E-01	5.19E-77
12/10/2017-26/10/2017	4.04E-01	1.44E-161
26/10/2017-09/11/2017	6.12E-01	0.00E+00
09/11/2017-23/11/2017	4.51E-01	2.43E-108
23/11/2017-07/12/2017	4.95E-01	4.59E-206
07/12/2017-21/12/2017	4.17E-01	4.70E-102
21/12/2017-04/01/2018	5.12E-01	1.51E-302
04/01/2018-18/01/2018	3.97E-01	4.79E-101
18/01/2018-01/02/2018	4.80E-01	6.83E-277
01/02/2018-15/02/2018	4.26E-01	3.45E-24
15/02/2018-01/03/2018	3.91E-01	2.80E-35

Table B.19 continued from previous page

Dates	Test Statistic	P-value
01/03/2018-15/03/2018	5.02E-01	2.53E-71
15/03/2018-29/03/2018	3.39E-01	1.18E-41
29/03/2018-12/04/2018	4.13E-01	5.91E-55
12/04/2018-26/04/2018	4.17E-01	9.44E-89
26/04/2018-10/05/2018	4.36E-01	1.62E-89
10/05/2018-24/05/2018	4.81E-01	8.09E-95
24/05/2018-07/06/2018	5.13E-01	3.89E-149
07/06/2018-21/06/2018	5.06E-01	1.47E-169
21/06/2018-05/07/2018	5.83E-01	1.56E-275
05/07/2018-19/07/2018	5.85E-01	8.59E-117
19/07/2018-02/08/2018	4.33E-01	1.54E-125
02/08/2018-16/08/2018	4.24E-01	2.64E-72
16/08/2018-30/08/2018	3.82E-01	4.40E-70
30/08/2018-13/09/2018	3.71E-01	7.67E-41
13/09/2018-27/09/2018	5.58E-01	4.36E-40
27/09/2018-11/10/2018	5.47E-01	1.61E-41
11/10/2018-25/10/2018	4.64E-01	2.82E-20
25/10/2018-08/11/2018	5.92E-01	6.53E-60
08/11/2018-22/11/2018	5.79E-01	7.39E-57
22/11/2018-06/12/2018	3.90E-01	9.26E-18
06/12/2018-20/12/2018	4.07E-01	8.09E-58
20/12/2018-03/01/2019	5.27E-01	2.64E-80
03/01/2019-17/01/2019	3.96E-01	1.89E-37
17/01/2019-31/01/2019	5.06E-01	1.72E-133
31/01/2019-14/02/2019	4.98E-01	1.28E-07
14/02/2019-28/02/2019	4.60E-01	3.64E-18
28/02/2019-14/03/2019	5.06E-01	5.61E-33
14/03/2019-28/03/2019	4.39E-01	3.77E-40
28/03/2019-11/04/2019	5.66E-01	3.83E-67

Table B.19 continued from previous page

Dates	Test Statistic	P-value
11/04/2019-25/04/2019	3.65E-01	5.13E-34
25/04/2019-09/05/2019	4.85E-01	1.66E-72
09/05/2019-23/05/2019	4.97E-01	2.30E-52
23/05/2019-06/06/2019	5.63E-01	6.05E-93
06/06/2019-20/06/2019	4.93E-01	7.58E-40
20/06/2019-04/07/2019	5.29E-01	6.22E-27
04/07/2019-18/07/2019	7.24E-01	9.15E-126
18/07/2019-01/08/2019	5.40E-01	1.48E-56
01/08/2019-15/08/2019	1.48E-01	9.84E-01
15/08/2019-29/08/2019	3.08E-01	6.15E-03
29/08/2019-12/09/2019	6.46E-01	1.43E-39
12/09/2019-26/09/2019	3.51E-01	2.18E-40
26/09/2019-10/10/2019	4.84E-01	7.39E-64
10/10/2019-24/10/2019	4.31E-01	2.09E-52
24/10/2019-07/11/2019	5.49E-01	7.61E-82
07/11/2019-21/11/2019	3.80E-01	2.67E-41
21/11/2019-05/12/2019	5.88E-01	2.85E-187
05/12/2019-19/12/2019	5.62E-01	5.17E-59
19/12/2019-02/01/2020	5.78E-01	1.50E-71
02/01/2020-16/01/2020	5.96E-01	1.16E-30
16/01/2020-30/01/2020	8.08E-01	0.00E+00
30/01/2020-13/02/2020	8.01E-01	7.17E-11
13/02/2020-27/02/2020	3.95E-01	2.29E-11
27/02/2020-12/03/2020	5.69E-01	5.31E-43
12/03/2020-26/03/2020	4.40E-01	7.69E-33
26/03/2020-09/04/2020	5.28E-01	7.23E-122
09/04/2020-23/04/2020	5.21E-01	1.73E-65
23/04/2020-07/05/2020	5.73E-01	8.13E-29
07/05/2020-21/05/2020	5.17E-01	3.48E-17

Table B.19 continued from previous page

Dates	Test Statistic	P-value
21/05/2020-04/06/2020	5.73E-01	2.54E-36
04/06/2020-18/06/2020	7.74E-01	7.13E-132
18/06/2020-02/07/2020	5.06E-01	2.70E-09
02/07/2020-16/07/2020	5.00E-01	2.12E-20
16/07/2020-30/07/2020	4.76E-01	1.77E-59
30/07/2020-13/08/2020	5.29E-01	1.19E-27
13/08/2020-27/08/2020	3.81E-01	1.18E-06
27/08/2020-10/09/2020	5.74E-01	2.90E-09
10/09/2020-24/09/2020	9.50E-01	4.94E-03
24/09/2020-08/10/2020	6.23E-01	1.32E-34
08/10/2020-22/10/2020	5.16E-01	7.10E-75
22/10/2020-05/11/2020	6.00E-01	2.14E-238
05/11/2020-19/11/2020	4.29E-01	2.76E-25
19/11/2020-03/12/2020	6.41E-01	6.11E-276
03/12/2020-17/12/2020	6.30E-01	1.05E-159
17/12/2020-31/12/2020	5.44E-01	0.00E+00

The code block developed for Brown's Test using months as consecutive time intervals is given below, and the test results are given in Table B.20.

```
test_stat=[]
p_val=[]
years=[]
months=[]
for year in range(2005,2021):
    for month in range(1,13):
        intersales_time=[]
        a=sales_after2005.loc[(sales_after2005.index.year==year) & (sales_after2005.index.month==month)]
        a.reset_index(drop=True, inplace=True)

        for i in range(a.index.max()):
            intersales_time.append(a.invoice_date.loc[i+1]-a.invoice_date.loc[i])

        intersales_time=pd.Series(intersales_time)

        intersales_time_h=convert_to_hours(intersales_time)
```

```python
        intersales_time_h=np.array(intersales_time_h)
        sales_time_h=np.cumsum(intersales_time_h)

        start_date=pd.Timestamp(year,month,1)
        fin_date = start_date+relativedelta(day=31)
        block_length_hour=((fin_date-start_date).days+1)*24

        res_hour=browns_test(sales_time_h,block_length_hour)

        test_stat.append(res_hour[0])
        p_val.append(res_hour[1])

        years.append(year)
        months.append(month)

result=pd.DataFrame(list(zip(years, months, test_stat, p_val)),
    columns=['years', 'months','test_stat', 'p_val'])
pd.set_option('display.max_rows', result.shape[0]+1)
result
```

Table B.20 Results of the Brown's Test when using months as successive time intervals.

Years	Months	Test Statistic	P-value
2005	1	5.28E-01	9.84E-26
2005	2	3.64E-01	1.49E-51
2005	3	3.98E-01	4.96E-83
2005	4	4.98E-01	1.01E-79
2005	5	4.36E-01	5.92E-132
2005	6	4.25E-01	6.24E-135
2005	7	4.74E-01	5.12E-152
2005	8	4.38E-01	6.64E-162
2005	9	4.09E-01	1.70E-101
2005	10	4.18E-01	6.03E-143
2005	11	4.23E-01	2.63E-138
2005	12	3.60E-01	3.13E-150
2006	1	4.84E-01	3.89E-71
2006	2	3.85E-01	1.08E-68
2006	3	3.93E-01	1.92E-115
2006	4	4.63E-01	1.33E-134

Table B.20 continued from previous page

Years	Months	Test Statistic	P-value
2006	5	3.97E-01	9.63E-72
2006	6	3.76E-01	1.70E-52
2006	7	5.09E-01	8.05E-82
2006	8	4.95E-01	8.48E-85
2006	9	4.39E-01	1.39E-38
2006	10	5.14E-01	1.05E-68
2006	11	3.88E-01	1.15E-50
2006	12	4.21E-01	2.00E-141
2007	1	5.34E-01	1.93E-78
2007	2	4.21E-01	1.22E-48
2007	3	4.40E-01	7.41E-58
2007	4	4.64E-01	3.05E-59
2007	5	4.41E-01	1.71E-67
2007	6	4.06E-01	1.13E-58
2007	7	3.85E-01	2.22E-47
2007	8	4.34E-01	4.36E-96
2007	9	4.82E-01	9.17E-91
2007	10	5.07E-01	1.35E-119
2007	11	4.28E-01	2.69E-98
2007	12	4.53E-01	2.90E-152
2008	1	4.32E-01	5.00E-45
2008	2	4.03E-01	1.26E-46
2008	3	4.56E-01	3.13E-101
2008	4	4.04E-01	1.13E-45
2008	5	4.26E-01	6.76E-43
2008	6	4.98E-01	2.07E-59
2008	7	3.72E-01	3.09E-46
2008	8	4.38E-01	1.23E-60
2008	9	4.37E-01	1.53E-44

Table B.20 continued from previous page

Years	Months	Test Statistic	P-value
2008	10	5.23E-01	4.35E-66
2008	11	5.35E-01	2.01E-115
2008	12	4.17E-01	5.94E-77
2009	1	4.50E-01	1.89E-30
2009	2	4.26E-01	5.38E-32
2009	3	4.79E-01	2.44E-225
2009	4	5.14E-01	3.49E-108
2009	5	6.10E-01	1.09E-91
2009	6	6.14E-01	7.69E-76
2009	7	5.30E-01	5.37E-73
2009	8	7.41E-01	1.34E-01
2009	9	7.45E-01	0.00E+00
2009	10	4.63E-01	1.79E-47
2009	11	5.96E-01	1.38E-66
2009	12	2.97E-01	1.45E-32
2010	1	5.00E-01	1.47E-74
2010	2	5.29E-01	1.68E-179
2010	3	4.97E-01	2.53E-140
2010	4	4.49E-01	2.12E-134
2010	5	5.37E-01	1.51E-156
2010	6	5.65E-01	1.75E-242
2010	7	4.50E-01	3.66E-109
2010	8	5.01E-01	6.57E-31
2010	9	4.43E-01	8.98E-75
2010	10	5.06E-01	5.09E-123
2010	11	5.33E-01	8.84E-219
2010	12	3.31E-01	1.12E-160
2011	1	5.50E-01	2.15E-136
2011	2	4.32E-01	1.41E-103

Table B.20 continued from previous page

Years	Months	Test Statistic	P-value
2011	3	4.26E-01	7.06E-216
2011	4	4.35E-01	8.52E-297
2011	5	5.45E-01	0.00E+00
2011	6	4.62E-01	3.43E-217
2011	7	4.74E-01	1.79E-218
2011	8	5.01E-01	1.49E-220
2011	9	4.41E-01	1.34E-212
2011	10	4.74E-01	7.188755e-317
2011	11	3.80E-01	2.22E-174
2011	12	4.25E-01	0.00E+00
2012	1	4.50E-01	3.52E-150
2012	2	4.25E-01	1.63E-189
2012	3	5.15E-01	0.00E+00
2012	4	5.38E-01	2.30E-196
2012	5	5.69E-01	0.00E+00
2012	6	4.28E-01	1.39E-196
2012	7	5.41E-01	0.00E+00
2012	8	4.81E-01	1.85E-181
2012	9	5.82E-01	0.00E+00
2012	10	4.87E-01	0.00E+00
2012	11	4.30E-01	0.00E+00
2012	12	3.77E-01	8.59E-219
2013	1	3.96E-01	1.48E-97
2013	2	4.70E-01	0.00E+00
2013	3	5.70E-01	0.00E+00
2013	4	4.62E-01	0.00E+00
2013	5	5.08E-01	0.00E+00
2013	6	5.56E-01	0.00E+00
2013	7	4.56E-01	0.00E+00

Table B.20 continued from previous page

Years	Months	Test Statistic	P-value
2013	8	5.45E-01	0.00E+00
2013	9	4.82E-01	0.00E+00
2013	10	4.47E-01	0.00E+00
2013	11	4.17E-01	2.58E-265
2013	12	4.55E-01	0.00E+00
2014	1	3.93E-01	7.50E-91
2014	2	3.36E-01	5.07E-172
2014	3	4.08E-01	9.24E-225
2014	4	3.48E-01	6.07E-227
2014	5	4.60E-01	0.00E+00
2014	6	4.72E-01	5.56E-252
2014	7	5.12E-01	0.00E+00
2014	8	3.92E-01	8.32E-245
2014	9	3.95E-01	4.23E-180
2014	10	4.40E-01	1.37E-289
2014	11	4.16E-01	9.08E-275
2014	12	3.71E-01	0.00E+00
2015	1	4.49E-01	1.08E-76
2015	2	4.24E-01	1.061757e-318
2015	3	5.18E-01	0.00E+00
2015	4	3.28E-01	4.99E-264
2015	5	4.65E-01	0.00E+00
2015	6	4.28E-01	4.468000e-313
2015	7	4.43E-01	0.00E+00
2015	8	4.00E-01	2.10E-216
2015	9	3.84E-01	6.70E-73
2015	10	4.54E-01	5.16E-241
2015	11	4.17E-01	9.82E-245
2015	12	2.75E-01	8.54E-217

Table B.20 continued from previous page

Years	Months	Test Statistic	P-value
2016	1	5.34E-01	8.17E-216
2016	2	3.64E-01	1.97E-202
2016	3	3.31E-01	5.51E-211
2016	4	4.33E-01	8.94E-295
2016	5	4.17E-01	0.00E+00
2016	6	2.81E-01	5.20E-225
2016	7	3.80E-01	1.30E-274
2016	8	3.66E-01	1.03E-260
2016	9	3.56E-01	1.11E-242
2016	10	3.91E-01	0.00E+00
2016	11	3.56E-01	1.54E-238
2016	12	4.55E-01	0.00E+00
2017	1	4.92E-01	2.57E-115
2017	2	2.86E-01	5.36E-64
2017	3	2.99E-01	2.47E-116
2017	4	4.59E-01	2.782775e-319
2017	5	4.00E-01	3.96E-304
2017	6	4.30E-01	9.17E-242
2017	7	3.64E-01	5.24E-288
2017	8	4.46E-01	0.00E+00
2017	9	4.90E-01	0.00E+00
2017	10	3.86E-01	3.34E-199
2017	11	2.60E-01	2.77E-112
2017	12	3.37E-01	3.09E-239
2018	1	4.09E-01	1.01E-60
2018	2	3.90E-01	2.76E-97
2018	3	3.84E-01	9.61E-147
2018	4	4.21E-01	4.65E-183
2018	5	4.22E-01	8.16E-280

Table B.20 continued from previous page

Years	Months	Test Statistic	P-value
2018	6	2.83E-01	9.34E-79
2018	7	4.14E-01	7.31E-154
2018	8	4.76E-01	3.12E-118
2018	9	4.31E-01	1.00E-51
2018	10	4.79E-01	3.18E-62
2018	11	3.79E-01	8.57E-96
2018	12	3.78E-01	1.80E-109
2019	1	3.41E-01	1.15E-16
2019	2	3.86E-01	2.20E-58
2019	3	3.66E-01	9.05E-69
2019	4	3.99E-01	9.71E-90
2019	5	4.25E-01	4.65E-59
2019	6	5.42E-01	5.22E-122
2019	7	2.31E-01	2.64E-02
2019	8	4.34E-01	7.70E-100
2019	9	4.00E-01	1.07E-96
2019	10	2.78E-01	1.50E-65
2019	11	4.63E-01	9.29E-82
2019	12	7.62E-01	0.00E+00
2020	1	5.50E-01	7.01E-52
2020	2	3.62E-01	2.13E-56
2020	3	5.25E-01	5.03E-141
2020	4	4.53E-01	4.09E-30
2020	5	7.44E-01	3.82E-167
2020	6	4.29E-01	1.01E-62
2020	7	5.05E-01	3.11E-40
2020	8	5.49E-01	9.46E-07
2020	9	4.47E-01	3.90E-189
2020	10	4.68E-01	1.30E-176

Table B.20 continued from previous page

Years	Months	Test Statistic	P-value
2020	11	4.48E-01	0.00E+00
2020	12	3.46E-01	2.41E-183

The code block developed for Brown's Test using quarters as consecutive time intervals is given below, and the test results are given in Table B.21.

```
test_stat=[]
p_val=[]
quarters=[1,2,3,4]*20
years=[2005, 2005, 2005, 2005]
for year in range(2006,2021):
        for t in range(0,4):
            years.append(year)

for year in range(2005,2021):
    for quarter in range(1,5):
        intersales_time=[]
        a=sales_after2005.loc[(sales_after2005.index.year==year) &
    (sales_after2005.index.quarter==quarter)]
        a.reset_index(drop=True, inplace=True)

        for i in range(a.index.max()):
            intersales_time.append(a.invoice_date.loc[i+1]-a.
    invoice_date.loc[i])

        intersales_time=pd.Series(intersales_time)

        intersales_time_h=convert_to_hours(intersales_time)
        intersales_time_h=np.array(intersales_time_h)
        sales_time_h=np.cumsum(intersales_time_h)

        start_date=pd.Timestamp(year,1+(quarter-1)*3,1)
        fin_date =pd.Timestamp(year,3+(quarter-1)*3,1)
        fin_date = fin_date+relativedelta(day=31)

        block_length_hour=((fin_date-start_date).days+1)*24

        res_hour=browns_test(sales_time_h,block_length_hour)

        test_stat.append(res_hour[0])
        p_val.append(res_hour[1])

result=pd.DataFrame(list(zip(years, quarters, test_stat, p_val)),
    columns=['years', 'quarters','test_stat', 'p_val'])
pd.set_option('display.max_rows', result.shape[0]+1)
result
```

Table B.21 Results of the Brown's Test when using quarters as successive time intervals.

Years	Quarters	Test Statistic	P-value
2005	1	4.25E-01	2.95E-181
2005	2	3.95E-01	2.45E-271
2005	3	4.30E-01	0.00E+00
2005	4	3.85E-01	0.00E+00
2006	1	3.88E-01	1.25E-225
2006	2	4.47E-01	4.91E-291
2006	3	5.08E-01	1.07E-222
2006	4	3.85E-01	3.98E-204
2007	1	4.78E-01	4.41E-194
2007	2	4.22E-01	9.86E-173
2007	3	4.22E-01	3.01E-216
2007	4	4.06E-01	2.08E-283
2008	1	4.27E-01	8.58E-184
2008	2	4.63E-01	1.29E-161
2008	3	4.59E-01	4.50E-187
2008	4	4.64E-01	1.87E-232
2009	1	3.98E-01	1.71E-203
2009	2	5.61E-01	1.16E-268
2009	3	3.14E-01	1.43E-111
2009	4	3.45E-01	3.87E-90
2010	1	4.79E-01	0.00E+00
2010	2	4.96E-01	0.00E+00
2010	3	4.94E-01	6.66E-256
2010	4	3.61E-01	0.00E+00
2011	1	4.14E-01	0.00E+00
2011	2	5.06E-01	0.00E+00

Table B.21 continued from previous page

Years	Quarters	Test Statistic	P-value
2011	3	4.49E-01	0.00E+00
2011	4	4.40E-01	0.00E+00
2012	1	4.69E-01	0.00E+00
2012	2	4.97E-01	0.00E+00
2012	3	5.57E-01	0.00E+00
2012	4	5.16E-01	0.00E+00
2013	1	5.12E-01	0.00E+00
2013	2	5.51E-01	0.00E+00
2013	3	4.78E-01	0.00E+00
2013	4	4.47E-01	0.00E+00
2014	1	4.56E-01	0.00E+00
2014	2	5.15E-01	0.00E+00
2014	3	4.76E-01	0.00E+00
2014	4	3.82E-01	0.00E+00
2015	1	4.68E-01	0.00E+00
2015	2	4.76E-01	0.00E+00
2015	3	5.39E-01	0.00E+00
2015	4	3.42E-01	0.00E+00
2016	1	4.09E-01	0.00E+00
2016	2	3.88E-01	0.00E+00
2016	3	4.19E-01	0.00E+00
2016	4	4.55E-01	0.00E+00
2017	1	3.69E-01	0.00E+00
2017	2	4.77E-01	0.00E+00
2017	3	4.46E-01	0.00E+00
2017	4	3.85E-01	0.00E+00
2018	1	3.87E-01	4.31E-299
2018	2	4.49E-01	0.00E+00
2018	3	5.45E-01	0.00E+00

Table B.21 continued from previous page

Years	Quarters	Test Statistic	P-value
2018	4	3.76E-01	4.14E-240
2019	1	3.87E-01	1.80E-156
2019	2	4.84E-01	1.65E-307
2019	3	4.07E-01	2.14E-193
2019	4	5.56E-01	0.00E+00
2020	1	5.53E-01	0.00E+00
2020	2	4.93E-01	4.16E-185
2020	3	3.80E-01	7.12E-160
2020	4	4.63E-01	0.00E+00

The code block developed for Brown's Test using seasons as consecutive time intervals is given below, and the test results are given in Table B.22.

```
test_stat=[]
p_val=[]

years=[2005, 2005, 2005]
for year in range(2006,2021):
    for t in range(0,4):
        years.append(year)

seasons=["spring","summer","autumn"]
for i in range(0,15):
    for season in ["winter","spring","summer","autumn"]:
        seasons.append(season)

start_date=pd.Timestamp(2005,3,1)

for iteration in range(0,64):
    fin_date = start_date + relativedelta(months=+3)
    intersales_time=[]
    a=sales_after2005.loc[(sales_after2005.invoice_date>=start_date) & (sales_after2005.invoice_date<fin_date)]
    a.reset_index(drop=True, inplace=True)

    for i in range(a.index.max()):
        intersales_time.append(a.invoice_date.loc[i+1]-a.invoice_date.loc[i])

    intersales_time=pd.Series(intersales_time)

    intersales_time_h=convert_to_hours(intersales_time)
    intersales_time_h=np.array(intersales_time_h)
    sales_time_h=np.cumsum(intersales_time_h)

```

```
    block_length_hour=((fin_date-start_date).days)*24

    start_date = fin_date

    res_hour=browns_test(sales_time_h,block_length_hour)

    test_stat.append(res_hour[0])
    p_val.append(res_hour[1])

result=pd.DataFrame(list(zip(years, seasons, test_stat, p_val)),
    columns=['Years', 'Seasons','test_stat', 'p_val'])
pd.set_option('display.max_rows', result.shape[0]+1)
result
```

Table B.22 Results of the Brown's Test when using seasons as successive time intervals.

Years	Seasons	Test Statistic	P-value
2005	spring	3.89E-01	1.70E-229
2005	summer	4.26E-01	0.00E+00
2005	autumn	4.26E-01	0.00E+00
2006	winter	5.03E-01	0.00E+00
2006	spring	4.56E-01	0.00E+00
2006	summer	4.71E-01	4.83E-229
2006	autumn	3.90E-01	2.34E-119
2007	winter	5.33E-01	0.00E+00
2007	spring	4.22E-01	4.85E-163
2007	summer	4.00E-01	2.30E-188
2007	autumn	4.51E-01	1.00E-280
2008	winter	5.19E-01	0.00E+00
2008	spring	5.13E-01	1.20E-268
2008	summer	4.42E-01	1.17E-173
2008	autumn	4.59E-01	1.91E-182
2009	winter	5.07E-01	9.01E-201
2009	spring	6.05E-01	0.00E+00
2009	summer	6.39E-01	2.38E-194
2009	autumn	7.39E-01	0.00E+00

Table B.22 continued from previous page

Years	Seasons	Test Statistic	P-value
2010	winter	4.43E-01	8.24E-254
2010	spring	4.82E-01	0.00E+00
2010	summer	6.05E-01	0.00E+00
2010	autumn	4.60E-01	0.00E+00
2011	winter	5.19E-01	0.00E+00
2011	spring	4.71E-01	0.00E+00
2011	summer	4.89E-01	0.00E+00
2011	autumn	4.59E-01	0.00E+00
2012	winter	4.98E-01	0.00E+00
2012	spring	5.08E-01	0.00E+00
2012	summer	5.39E-01	0.00E+00
2012	autumn	5.03E-01	0.00E+00
2013	winter	4.75E-01	0.00E+00
2013	spring	5.43E-01	0.00E+00
2013	summer	4.94E-01	0.00E+00
2013	autumn	5.14E-01	0.00E+00
2014	winter	4.82E-01	0.00E+00
2014	spring	4.77E-01	0.00E+00
2014	summer	4.56E-01	0.00E+00
2014	autumn	4.20E-01	0.00E+00
2015	winter	4.59E-01	0.00E+00
2015	spring	4.88E-01	0.00E+00
2015	summer	4.72E-01	0.00E+00
2015	autumn	3.58E-01	0.00E+00
2016	winter	4.71E-01	0.00E+00
2016	spring	4.19E-01	0.00E+00
2016	summer	4.60E-01	0.00E+00
2016	autumn	4.80E-01	0.00E+00
2017	winter	5.74E-01	0.00E+00

Table B.22 continued from previous page

Years	Seasons	Test Statistic	P-value
2017	spring	3.87E-01	0.00E+00
2017	summer	4.80E-01	0.00E+00
2017	autumn	4.65E-01	0.00E+00
2018	winter	5.26E-01	0.00E+00
2018	spring	3.84E-01	0.00E+00
2018	summer	5.11E-01	0.00E+00
2018	autumn	3.65E-01	1.74E-160
2019	winter	4.60E-01	5.84E-279
2019	spring	4.65E-01	2.18E-307
2019	summer	4.62E-01	2.80E-208
2019	autumn	4.76E-01	0.00E+00
2020	winter	5.95E-01	0.00E+00
2020	spring	5.83E-01	0.00E+00
2020	summer	6.48E-01	1.02E-231
2020	autumn	4.56E-01	0.00E+00

B.4 Chi-square Goodness of Fit Test

In this section, the code blocks developed to implement chi-square goodness of fit test is presented. The libraries given below are required for data manipulation and constructing the test procedure.

```
import pandas as pd
import numpy as np
import math
import scipy.stats as stats
```

The following code block is developed to calculate statistics of probability distributions to be subjected to the chi-square test. The function in the code block first calculates the moments based on parameters such as installed base type (growing or declining), distribution selection algorithm (Ord or Adan), α, λ, et cetera. Note that not all probability distributions given in Ord's and Adan's algorithms are included in this function since some of them are not selected due to the low α rates.

```python
def DynamicDistStats(lambd,alph,plan_hor,stepsize,N0,IB,dist_sel_alg):
    ########  PARAMETERS  ########
    lambd=lambd#parameter of poisson process that represents capital product sales
    alpha=alph#parameter of poisson process that represents spare part breakdowns
    tmax=plan_hor #total time period
    t=plan_hor
    step_size=stepsize
    tvect=np.arange(1,tmax*step_size+1)/step_size
    alphat=alpha*tvect
    lambdat=lambd*tvect

    h=1/step_size
    alphah=alpha*h
    lambdah=lambd*h

    if IB=="Growing":
        ########  h - CUMULATIVE MOMENTS  ########
        #FIRST MOMENT
        moment1_h = alphah*(0.5*lambdah+1)

        #SECOND MOMENT
        moment2_h = alphah**2 * (0.25*lambdah**2 + 4/3*lambdah+1) + moment1_h

        #THIRD MOMENT
        moment3_h = alphah**3*(1/8*lambdah**3 + 5/4*lambdah**2+11/4*lambdah+1) + 3*moment2_h -2*moment1_h

        ########  NON-CENTRAL CUMULATIVE MOMENTS  ########
        #FIRST MOMENT
        Dmoment1_noncentral = alphah*(lambdat+0.5*lambdah+1)

        #SECOND MOMENT
        Dmoment2_noncentral = alphah**2*(lambdat**2+lambdat)+alphah*lambdat+alphah**2 * (0.25*lambdah**2 + 4/3*lambdah+1) + alphah*(0.5*lambdah+1)+2*lambdat*alphah*alphah*(0.5*lambdah+1)

        #THIRD MOMENT
        Dmoment3_noncentral = alphah**3*(lambdat**3+3*lambdat**2+lambdat)+3*alphah**2*(lambdat**2+lambdat)+alphah*lambdat+3*moment1_h*(alphah**2*(lambdat**2+lambdat)+alphah*lambdat)+3*moment2_h*alphah*lambdat+moment3_h

        ########  CENTRAL CUMULATIVE MOMENTS  ########
        #FIRST MOMENT
        Dmom1=Dmoment1_noncentral

        #SECOND MOMENT
        Dmom2=Dmoment2_noncentral-Dmoment1_noncentral**2

        #THIRD MOMENT
        Dmom3=Dmoment3_noncentral - 3*Dmoment1_noncentral*Dmoment2_noncentral + 2*Dmoment1_noncentral**3

```

```
     #CV MARGINAL
     DmomCV=Dmom2/Dmom1

     #ZERO PROBABILITY
     P0 = np.exp(lambdat*(np.exp(-alphah)-1))* np.exp(-h*(lambd+
alpha))
     D_Zero_Prob = P0*np.exp(-lambd/alpha*(np.exp(-alphah)-1))

 elif IB=="Declining":
     ########  FUNCTIONS  ########
     def qt2(x,lambd,t):
         lambdat=t*lambd
         return(stats.poisson.cdf(x,lambdat))

     def beta2(k,N0):
         res=0
         kvect=np.arange(0,k+1)
         return(1/(k+1)*math.prod(N0+kvect))

     alphat=alpha*tvect
     lambdat=lambd*tvect

     #FIRST MOMENT
     def NonCentralMom1(a,L,tvector,N):
         at=a*tvector
         Lt=L*tvector
         return(at*(N*qt2(N,L,tvector)-(1/2)*Lt*qt2(N-1,L,
tvector))+(a/L)*beta2(1,N)*(1-qt2(N+1,L,tvector)))

     #SECOND MOMENT
     def NonCentralMom2(a,L,tvector,N):
         at=a*tvector
         Lt=L*tvector
         return(Lt*qt2(N-1,L,tvector)*(at**2*((1/12)-N)-0.5*at)
+ (Lt**2* qt2(N-2,L,tvector)+Lt*qt2(N-1,L,tvector))*0.25*at**2 +
 qt2(N,L,tvector)*(N*at+N**2*at**2) + (a/L)*beta2(1,N)*(1-qt2(N
+1,L,tvector)) + 0.25* (a/L)**2*beta2(2,N)*(3*N+1)*(1-qt2(N+2,L,
tvector)))

     #THIRD MOMENT
     def NonCentralMom3(a,L,tvector,N):
         at=a*tvector
         Lt=L*tvector
         term1=-1/8*at**3*(Lt**3 * qt2(N-3,L,tvector) +3*Lt**2 *
 qt2(N-2,L,tvector) +Lt * qt2(N-1,L,tvector) )
         term2=(1/8)*at**2*( at*(6*N-1)+6)*(Lt**2 * qt2(N-2,L,
tvector) +Lt * qt2(N-1,L,tvector)  )
         term3=-(1/4)*at*(at**2*N*(6*N-1)-at*(1-12*N)+2)*Lt *
qt2(N-1,L,tvector)
         term4=(at**3*(N**3)+at**2*(3*N**2)+N*at)*qt2(N,L,
tvector)
         term5=(at/Lt)*beta2(1,N)*(1-qt2(N+1,L,tvector)) + (3/4)
*(at/Lt)**2*beta2(2,N)*(3*N+1)*(1-qt2(N+2,L,tvector)) + (at/Lt)
**3*beta2(1,N)*beta2(3,N)*(1-qt2(N+3,L,tvector))
         return(term1+term2+term3+term4+term5)

     ########  CENTRAL CUMULATIVE MOMENTS  ########
     centralmoment1=NonCentralMom1(alpha,lambd,tvect,N0)
     centralmoment2=NonCentralMom2(alpha,lambd,tvect,N0) - (
NonCentralMom1(alpha,lambd,tvect,N0))**2
```

```
        centralmoment3=(NonCentralMom3(alpha,lambd,tvect,N0) -3*centralmoment1*centralmoment2 - centralmoment1**3)

        ########  CENTRAL MARGINAL MOMENTS  ########
        ########  PARAMETERS  ########
        h=1/step_size
        alphah=alpha*h
        lambdah=lambd*h
        epsilon=0.1
        epsilon2=0.05

        NonCM1=0 #FIRST NON-CENTRAL MOMENT OF MARGINAL
        NonCM2=0 #SECOND NON-CENTRAL MOMENT OF MARGINAL
        NonCM3=0 #THIRD NON-CENTRAL MOMENT OF MARGINAL

        for n in range(0,N0+1):
            NonCM1=NonCM1+NonCentralMom1(alpha,lambd,h,N0-n)*stats.poisson.pmf(n,lambd*tvect)
            NonCM2=NonCM2+NonCentralMom2(alpha,lambd,h,N0-n)*stats.poisson.pmf(n,lambd*tvect)
            NonCM3=NonCM3+NonCentralMom3(alpha,lambd,h,N0-n)*stats.poisson.pmf(n,lambd*tvect)

        #FIRST CENTRAL MOMENT OF MARGINAL
        Dmom1=NonCM1

        #SECOND CENTRAL MOMENT OF MARGINAL
        Dmom2=NonCM2-NonCM1**2

        #THIRD CENTRAL MOMENT OF MARGINAL
        Dmom3=(NonCM3 -3*NonCM1*Dmom2 - NonCM1**3)

        #CV MARGINAL
        DmomCV=Dmom2/Dmom1

        part1_1=lambd*np.exp(alphah)*tvect+lambd/alpha*(np.exp(alphah)-1)
        part1_2=(lambdat*np.exp(alphah))**N0/math.factorial(N0)

    param_Pois = np.zeros(len(Dmom3))
    Param_NegBinom = np.zeros((len(Dmom3),2))
    param_binomMix = np.zeros((len(Dmom3),3))
    Param_NegBinomMix = np.zeros((len(Dmom3),3))
    param_geomMix = np.zeros((len(Dmom3),4))

    dist_vect=[]

    if dist_sel_alg=="Ord":
        print("Ord")
        epsilon=0.1
        epsilon2=0.05
        S_Index_vect=np.zeros(len(Dmom3))
        I_Index_vect=np.zeros(len(Dmom3))

        for v in range(0,len(Dmom3)):

            S_Index = Dmom3[v] / Dmom2[v]
            I_Index = Dmom2[v] / Dmom1[v]

            S_Index_vect[v] = S_Index
```

```
            I_Index_vect[v] = I_Index

            #Poisson Dist.
            cond1 = ((abs(S_Index-1) + abs(I_Index-1))<epsilon2)
            if cond1:
                dist_vect.append("Pois")
                param_Pois[v] = Dmom1[v]

            #Negative Binomial Dist.
            cond2 = (abs(S_Index - 2*I_Index+1)<epsilon)
            if((not cond1) & cond2):
                dist_vect.append("NBinom")
                var_v = Dmom2[v]
                Param_NegBinom[v,0] = Dmom1[v] / var_v  #prob(p)
                Param_NegBinom[v,1] = Dmom1[v] * Param_NegBinom[v
,0] / (1-Param_NegBinom[v,0])#N

            #Beta Binomial Distribution
            cond3 = (S_Index - 2*I_Index+1 < (-epsilon))

            #Beta Pascal Distribution
            cond4 = (S_Index - 2*I_Index+1 > epsilon)

            if ((not cond1) & (not cond2) & cond3):
                print("Beta Binom Dist. Selected!")
                break

            if((not cond1) & (not cond2) & cond4):
                print("Beta Pascal Dist. Selected!")
                break

            if(cond1 + ((not cond1) & cond2) + ((not cond1)&(not
cond2) & cond3) + ((not cond1)&(not cond2) & cond4)) > 1:
                print("Multiple True Conditions!")
                break

    elif dist_sel_alg=="Adan":
        print("Adan")
        epsilon=0.001
        for v in range(0,len(Dmom3)):
            cx_vect=np.zeros(len(Dmom3))
            a_vect=np.zeros(len(Dmom3))

            cx=math.sqrt(Dmom2[v])/Dmom1[v]
            a=cx**2-1/Dmom1[v]

            cx_vect[v] = cx
            a_vect[v] = a

            #Binomial Dist.
            if((a<-epsilon) & (a>=-1)):
                dist_vect.append("BinomMix")

                k=0
                while(-1/(k+1)<a):
                    k=k+1

                k_star=k

                q=(1+a*(1+k_star)+math.sqrt(-a*k_star*(1+k_star)-
```

```
k_star))/(1+a)

             p=Dmom1[v]/(k_star+1-q)

             param_binomMix[v,0]=p
             param_binomMix[v,1]=k_star
             param_binomMix[v,2]=q

             probabilities[v,:dmax]=stats.binom.pmf(np.arange(
dmax),k.star,p)*q + (1-q)*stats.binom.pmf(np.arange(dmax),k.star
+1,p)

         #Poisson Dist.
         if((a<=epsilon) & (a>=-epsilon)):
             dist_vect.append("Pois")
             param_Pois[v] = Dmom1[v]

         #Negative Binomial Dist.
         if((a<1) & (a>epsilon)):
             dist_vect.append("NBinomMix")

             k= math.ceil(1/a)*10
             while((1/k)<a):
                 k=k-1

             k_star = k

             q = ((1+k_star)*a - math.sqrt((1+k_star)*(1-a*
k_star)))/(1+a)
             p = Dmom1[v]/(k_star+1-q+Dmom1[v])

             Param_NegBinom[v,0] = 1-p #prob(p)
             Param_NegBinom[v,1] = k_star
             Param_NegBinomMix[v,2] = q

         #Geometric Dist.
         if(a>=1):
             dist_vect.append("GeomMix")

             r1= (1+a + math.sqrt(a**2-1))
             r2= (1+a - math.sqrt(a**2-1))
             p1= Dmom1[v] * r1 / (2+Dmom1[v]*r1)
             p2= Dmom1[v] * r2 / (2+Dmom1[v]*r2)
             q1= 1 / r1
             q2= 1 / r2

             param_geomMix[v,0] = 1-p1
             param_geomMix[v,1] = 1-p2
             param_geomMix[v,2] = q1
             param_geomMix[v,3] = q2

 return dist_vect, param_Pois, Param_NegBinom, param_binomMix,
Param_NegBinomMix, param_geomMix
```

The function below is used to calculate CDFs and PMFs of respective probability distributions, especially for mixture distributions provided in Adan's algorithm. There are no generic functions provided in software languages' libraries to calculate those mixture distributions' CDFs and PMFs.

```python
def pmf(x,dist,parameters):
    if dist=="Pois":
        return stats.poisson.pmf(x,parameters)
    elif dist=="NBinom":
        return stats.nbinom.pmf(x, parameters[0], parameters[1])
    elif dist=="BinomMix":
        return stats.binom.pmf(x,parameters[1],parameters[0])*parameters[2] + (1-parameters[2])*stats.binom.pmf(x,parameters[1]+1,parameters[0])
    elif dist=="NBinomMix":
        return stats.nbinom.pmf(x,parameters[1],parameters[0])*parameters[2] + (1-parameters[2])*stats.nbinom.pmf(x,parameters[1]+1,parameters[0])
    elif dist=="GeomMix":
        return stats.geom.pmf(x,parameters[0])*parameters[2]+stats.geom.pmf(x,parameters[1])*parameters[3]

def cdf(x,dist,parameters):
    if dist=="Pois":
        return stats.poisson.cdf(x,parameters)
    elif dist=="NBinom":
        return stats.nbinom.cdf(x, parameters[0], parameters[1])
    elif dist=="BinomMix":
        return stats.binom.cdf(x,parameters[1],parameters[0])*parameters[2] + (1-parameters[2])*stats.binom.cdf(x,parameters[1]+1,parameters[0])
    elif dist=="NBinomMix":
        return stats.nbinom.cdf(x,parameters[1],parameters[0])*parameters[2] + (1-parameters[2])*stats.nbinom.cdf(x,parameters[1]+1,parameters[0])
    elif dist=="GeomMix":
        return stats.geom.cdf(x,parameters[0])*parameters[2] + stats.geom.cdf(x,parameters[1])*parameters[3]
```

The code block below is the function that calculates empirical α rates of respective spare parts.

```python
def CalcAlpha(part_num,Model1_parts_20181920,Model1_sales_20181920):
    def convert_to_days(td):
        return (td.dt.days)+(td.dt.seconds/(60*60*24))

    part=Model1_parts_20181920[Model1_parts_20181920.recent_part_num==part_num]
    part_changes=part.groupby('VehicleID')['total quantity'].sum().to_frame().reset_index()
    part_VehicleSales=Model1_sales_20181920[Model1_sales_20181920.VehicleID.isin(part_changes.VehicleID)]
    part_VehicleSales=part_VehicleSales[['VehicleID', 'selling_date']]
    part_VehicleSales['time']=convert_to_days(pd.to_datetime('2020-12-31', format='%Y-%m-%d')-part_VehicleSales.selling_date)
    part_changes=part_changes.merge(part_VehicleSales, on='VehicleID')
    part_changes['alpha']=part_changes['total quantity']/part_changes.time
```

```
    AlphaRate=part_changes.alpha.sum()/Model1_sales_20181920.VehicleID.nunique()
    return AlphaRate
```

The code block given below is the test procedure developed for chi-square test.

```
alph=0.05 #significance level (not the parameter of poisson process)
acceptance_rates=[]
alphas=[]
for part_num in selectedparts.part_number:
    #filter respective part_num
    z=Model1_parts_20181920[Model1_parts_20181920.recent_part_num==part_num]

    #transform work_order_dates to yy/mm/dd format
    z['work_order_date_days']=pd.to_datetime(z['work_order_date']).dt.date

    alpha=CalcAlpha(part_num,Model1_parts_20181920,Model1_sales_20181920)
    alphas.append(alpha)

    #lambda for Model 1 is 21.53888381
    DistStats=DynamicDistStats(21.53888381,alpha,365*3,1,0,"Growing","Adan")
    dist_vect=DistStats[0]
    param_Pois=DistStats[1]
    Param_NegBinom=DistStats[2]
    param_binomMix=DistStats[3]
    Param_NegBinomMix=DistStats[4]
    param_geomMix=DistStats[5]

    dists=np.array(dist_vect)[14:365*3:30]

    #create empty lists
    chi_stat=[]
    crit=[]
    p_value=[]
    results=[]

    #there are 36 time intervals
    for i in range(36):
        #filter respective time interval
        x=z[(z['work_order_date_days']>=periods.start_date.loc[i]) & (z['work_order_date_days']<=periods.end_date.loc[i])].groupby('work_order_date_days')['total quantity'].sum().to_frame().sort_values('total quantity').reset_index()

        #count order quantities
        y=x.groupby('total quantity').count().reset_index()

        #add days without orders as 0
        y=y.append({'total quantity':0,'work_order_date_days':30-len(x)}, ignore_index=True)
        y.sort_values('total quantity', inplace=True)
        y.reset_index(drop=True, inplace=True)

        #aralardaki eksik degerleri ekle
```

```
    for kk in range(0,int(max(y['total quantity']))):
        if kk in list(y['total quantity']):
            continue
        else:
            y=y.append({'total quantity':kk, '
work_order_date_days':0},ignore_index=True).sort_values('total
quantity').reset_index(drop=True)

    #choose distribution and its parameters for time interval i
    distribution=dists[i]
    if distribution=="Pois":
        parameters=param_Pois[14+30*i]
    elif distribution=="NBinom":
        parameters=Param_NegBinom[14+30*i,:]
    elif distribution=="BinomMix":
        parameters=param_binomMix[14+30*i,:]
    elif distribution=="NBinomMix":
        parameters=Param_NegBinomMix[14+30*i,:]
    elif distribution=="GeomMix":
        parameters=param_geomMix[14+30*i,:]

    #calculate expected frequencies (based on chosen dist.)
    y['expected_freq']=pmf(y['total quantity'],distribution,
parameters)*30
    y.columns=['total_quantity','observed_freq', 'expected_freq
']

    #add the missing values in observed frequencies
    max_obs=y.total_quantity.max()
    k=1
    if pmf(max_obs+k,distribution,parameters)*30>=2:
        while pmf(max_obs+k,distribution,parameters)*30>=2:
            y=y.append({'total_quantity':max_obs+k, '
observed_freq':0, 'expected_freq':pmf(max_obs+k,distribution,
parameters)*30},ignore_index=True)
            k+=1
        if (1-cdf(max_obs+k-1,distribution,parameters))*30>=2:
            y=y.append({'total_quantity':max_obs+k, '
observed_freq':0, 'expected_freq':(1-cdf(max_obs+k-1,
distribution,parameters))*30},ignore_index=True)
    elif (1-cdf(max_obs+k-1,distribution,parameters))*30>=2:
        y=y.append({'total_quantity':max_obs+k, 'observed_freq'
:0, 'expected_freq':(1-cdf(max_obs+k-1,distribution,parameters))
*30},ignore_index=True)

    if y.expected_freq.size>1:
        y.expected_freq[y.index.max()]=(1-cdf(y.total_quantity[
y.index.max()-1],distribution,parameters))*30

        ind=0
        while ind<=max(y.index):
            #merge the rows whose expected frequency less is
than two with the row below until resulting expected frequency
becomes at least two
            if y.expected_freq[ind]<2:
                while y.expected_freq[ind]<2:
                    ###print(y)
                    if ind+1<=max(y.index):
                        a=[ind,ind+1]
                        y=y.append({'total_quantity':y.iloc[ind
```

```
,0], 'observed_freq':y.iloc[a,1].sum(), 'expected_freq':y.iloc[a,2].sum()}, ignore_index=True).drop(a).sort_values('total_quantity').reset_index(drop=True)
                    else:
                        a=[ind,ind-1]
                        y=y.append({'total_quantity':y.iloc[ind-1,0], 'observed_freq':y.iloc[a,1].sum(), 'expected_freq':y.iloc[a,2].sum()}, ignore_index=True).drop(a).sort_values('total_quantity').reset_index(drop=True)

                    #do not increase the index if it's the last row
                    if ind>max(y.index):
                        ind=max(y.index)
                else:
                    ind+=1
        else:
            y.expected_freq=30

        #change the exp.freq of last tot.quant. using CDF(probability of being greater than or equal to the last tot.quant.)
        chi_stat.append(((y.observed_freq-y.expected_freq)**2/y.expected_freq).sum())
        crit.append(stats.chi2.ppf(q = 1-alpha, df = len(y.observed_freq)-1))
        p_value.append(1-stats.chi2.cdf(x=chi_stat[i], df=len(y.observed_freq)-1))
        if p_value[i]>=alph:
            results.append("accept")
        else:
            results.append("reject")

    res=pd.DataFrame(list(zip(dists, chi_stat, crit, p_value, results)), columns=['Dist' ,'Chi_stat', 'Crit_val', 'P_value', 'Result'])
    acceptance_rates.append((res.Result=="accept").mean())
```

REFERENCES

Adan, I., van Eenige, M. and Resing, J., 1995. Fitting Discrete Distributions on the First Two Moments. Probability in the Engineering and Informational Sciences, 9(4), pp.623-632.

Allon, G. and Van Mieghem, J., 2010. Global Dual Sourcing: Tailored Base-Surge Allocation to Near- and Offshore Production. Management Science, 56(1), pp.110-124.

Böhning, D., 1994. A Note on a Test for Poisson Overdispersion. Biometrika, 81(2), p.418.

Brown, L., Gans, N., Mandelbaum, A., Sakov, A., Shen, H., Zeltyn, S. and Zhao, L., 2005. Statistical Analysis of a Telephone Call Center. Journal of the American Statistical Association, 100(469), pp.36-50.

Dekker, R., Pinçe, Ç., Zuidwijk, R. and Jalil, M., 2013. On the use of installed base information for spare parts logistics: A review of ideas and industry practice. International Journal of Production Economics, 143(2), pp.536-545.

Federgruen, A. and Zipkin, P., 1986. An Inventory Model with Limited Production Capacity and Uncertain Demands I. The Average-Cost Criterion. Mathematics of Operations Research, 11(2), pp.193-207.

Fukuda, Y., 1964. Optimal Policies for the Inventory Problem with Negotiable Leadtime. Management Science, 10(4), pp.690-708.

Hekimoğlu, M., 2015. Spare Parts Management of Aging Capital Products. Ph.D thesis. Erasmus University Rotterdam.

Hekimoğlu, M. and Karlı, D., 2021. Modeling Repair Demand In Existence of a Nonstationary Installed Base. European Journal of Operational Research, submitted.

Hu, M., Pavlin, J. and Shi, M., 2013. When Gray Markets Have Silver Linings: All-Unit Discounts, Gray Markets, and Channel Management. Manufacturing & Service Operations Management, 15(2), pp.250-262.

Jin, T. and Liao, H., 2009. Spare parts inventory control considering stochastic growth of an installed base. Computers & Industrial Engineering, 56(1), pp.452-460.

Karlis, D. and Xekalaki, E., 2000. A Simulation Comparison of Several Procedures for Testing the Poisson Assumption. Journal of the Royal Statistical Society: Series D (The Statistician), 49(3), pp.355-382.

Longman, 2017. Business English Dictionary. London: Pearson Publishing.

Ord, J., 1967. On a System of Discrete Distributions. Biometrika, 54(3/4), p.649.

Pinçe, Ç. and Dekker, R., 2011. An inventory model for slow moving items subject to obsolescence. European Journal of Operational Research, 213(1), pp.83-95.

Porteus, E., 2009. Foundations of stochastic inventory theory. Stanford, Calif.: Stanford Univ. Press.

Rayner, J., Thas, O. and Best, D., 2009. Smooth tests of goodness of fit. 2nd ed. Singapore [etc.]: John Wiley & Sons (Asia).

Scarf, H., 1959. The optimality of (S, s) policies in the dynamic inventory problem. Stanford: Stanford University Press.

Sun, J. and Van Mieghem, J., 2019. Robust Dual Sourcing Inventory Management: Optimality of Capped Dual Index Policies and Smoothing. Manufacturing & Service Operations Management, 21(4), pp.912-931.

Tan, B., Feng, Q. and Chen, W., 2016. Dual Sourcing Under Random Supply Capacities: The Role of the Slow Supplier. Production and Operations Management, 25(7), pp.1232-1244.

Van der Auweraer, S., Zhu, S. and Boute, R., 2021. The value of installed base information for spare part inventory control. International Journal of Production Economics, 239, p.108186.

Veeraraghavan, S. and Scheller-Wolf, A., 2008. Now or Later: A Simple Policy for Effective Dual Sourcing in Capacitated Systems. Operations Research, 56(4), pp.850-864.

Whittemore, A. and Saunders, S., 1977. Optimal Inventory Under Stochastic Demand with Two Supply Options. SIAM Journal on Applied Mathematics, 32(2), pp.293-305.

Yang, J., Qi, X. and Xia, Y., 2005. A Production-Inventory System with Markovian Capacity and Outsourcing Option. Operations Research, 53(2), pp.328-349.

Yazlalı, Ö. and Erhun, F., 2009. Dual-supply inventory problem with capacity limits on order sizes and unrestricted ordering costs. IIE Transactions, 41(8), pp.716-729.

CURRICULUM VITAE

Personal Information
Name and Surname: Ali Kök

Academic Background
Bachelor's Degree Education: Department of Industrial Engineering, Kadir Has University
Foreign Languages: English

Work Experience
Institutions Served and Their Dates:
Doğuş Teknoloji Jan 2021 – Present
USO Exhaust Systems Aug 2018 – Sep 2018
JLL Turkey Jun 2018 – Aug 2018