POLICY EVALUATION USING MARKOV DECISION PROCESS FOR INVENTORY OPTIMIZATION

by

EAKEEN MUHAMMAD HAQUE, B.S.M.E

THESIS
Presented to the Graduate Faculty of
The University of Texas at San Antonio
in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

COMMITTEE MEMBERS:
Adel Alaeddini, Ph.D., Chair
F. Frank Chen, Ph.D.
Hung-da Wan, Ph.D.

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Engineering
Department of Mechanical Engineering
December, 2019

DEDICATION

This thesis is dedicated to my family. Thank you for sticking with me during my failures. Not that I am a huge success now. But you know… !!

ACKNOWLEDGEMENTS

The author wants to express his gratitude and gratefulness to Almighty Allah for His kindness, benevolence, and mercy to him, and for the successful completion of this thesis.

The author would like to express deep sense of gratitude, profound respect, and sincere appreciation to his supervisor Dr. Adel Alaeddini for his continuous and excellent supervision, and kind co-operation throughout the course of the whole project. His scholastic guidance in the field of Reinforcement Learning and Markov Decision Process has been invaluable for the author from day one.

The author is thankful to Dr. F. Frank Chen and Dr. Hung-da Wan for serving in the advisory committee. The author is also thankful to Syed Hasib Akhter Faruqui, Stanford Martinez, Mike Chi-Wen Chang, Rajitha Meka, and Sahidul Islam for their kind help and co-operation during the completion of the thesis.

December 2019

POLICY EVALUATION USING MARKOV DECISION PROCESS FOR INVENTORY OPTIMIZATION

Eakeen Muhammad Haque, M.S.M.E
The University of Texas at San Antonio, 2019

Supervising Professor: Adel Alaeddini, Ph.D.

Markov Decision Process (MDP) is a discrete-time state-transition system suited to progressive decision making for models with uncertain elements. The impact of it has been significant in the fields of operations management in recent times. This thesis aims to find an optimal policy for an adaptive inventory model through the help of MDP. We consider a situation where a seller has only one type of product, of unit quantity for each cycle, to order to satisfy demands. Whether or not there will be demand of that product is a case of stochasticity. The demand can also only be of unit quantity for each cycle. The seller can either stock or issue a backorder of the product. However, inventory size or the number of issued backorders is limited. For each state of the inventory, only one of two actions can be chosen. The seller can either decide to buy, or not to buy the product based on the stochastic demand pattern. We approach this problem by using the MDP to iterate through possible policies. Our goal is to find the optimal policy that will recommend which actions to take in each inventory state that minimizes the cost of stocking or backordering.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE: INTRODUCTION

Stochastic programming is a basis that provides outline for modeling optimization problems involving uncertainty [1]. Real world problems would always have some elements of uncertainty when mapped in a model. But the probability distribution outlining the data can be identifiable or estimated. Stochastic programming takes advantage of this fact to find an optimized solution for the model [17]. The goal in these cases would be to find some policy that satisfies all or at least, most of the conditions that define the problem, and then optimize the expectation or the outcome.

Markov Decision Process (MDP) is such a model that is suited to progressive decision making for problems with uncertain or stochastic elements [2]. It maps out a specific environment for an agent, where it can determine the ideal behavior that optimizes the outcome of the whole model [7]. This optimization is achieved with a reward feedback system, where different actions are explored and measured depending on the predicted situations the agent will find itself in for taking those actions.

They are used in the fields related to control and robotics, manufacturing, economics, and many more. MDPs are particularly convenient when implemented along with dynamic programming and reinforcement learning methods [2][3]. In the field of manufacturing, the application of this method has been quite impactful. MDP is being used in fields such as scheduling, resource allocation, maintenance and repairing, process control and planning, database management, and many other with efficient results [4][16]. One of such fields is the inventory optimization which is becoming increasingly important for industries and companies nowadays. As the market is full of all kinds of ambiguity, it is considered important and difficult

to keep the inventory management up to the mark [5]. The problem discussed in this project concerns how much to order the product in each time period in order to meet the demand.

1.1 Objective

The objective of this project is to evaluate policies and find a suitable one that will recommend which action to take in each state of the inventory that optimizes the whole model. The methods for evaluating and finding the optimal policies would be:

- Policy Iteration
- Value Iteration

CHAPTER TWO: LITERATURE REVIEW

2.1 Markov Decision Process

Markov decision process is a model suited to progressive decision making for models with uncertain elements. The model consists of an environment in which all states can be conditioned under the Markov property, that is that the future is independent of the past given the present. the model also consists of an agent that interacts with the environment by means of taking actions [6][13]. The environment reacts to these actions by presenting the agent with a reward and a new state.

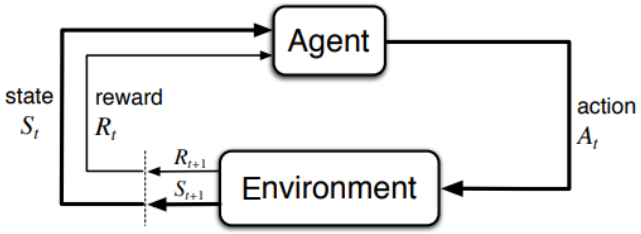

Figure 1: Agent environment interaction in a Markov decision process [6]

A Markov decision process is a tuple consisting of (S, A, P, R, γ).

- S signifies a finite set of states
- A signifies a finite set of actions
- P signifies a state transition probability matrix
- R signifies reward
- γ signifies a discount factor. $\gamma \in [0,1]$

State: The states in MDP are formulated based on Markov property. That is that the future is independent of the past given the present [6][7][8]. The state collects the necessary information from the history, and once the current state is known, the previous information of

that state can be discarded. Based on the Markov theory, the current state is an adequate statistic of the future. In mathematical terms, a state S_t is Markov if and only if

$$\mathbb{P}[S_{t+1} | S_t] = \mathbb{P}[S_{t+1} | S_1, \dots, S_t]$$

The equation signifies that, what happens next for a given state S_{t+1} depends only on the previous state S_t and not the states that came before that [2][7][16].

Action: For a given state, an agent is presented with a set of possible actions by the environment. The agent chooses the action based on the policy implemented in the model. By taking an action, the agent moves to the next state, while gaining a reward presented to it by the environment.

State Transition Probability: For a Markov state s and successor state s', the state transition probability can be defined by,

$$P_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s]$$

It is a probability distribution over next possible successor states for a given state [7]. For a given environment, it defines transition probabilities from all states *s* to all successor states *s'*.

$$\mathcal{P} = \begin{matrix} \\ from \end{matrix} \begin{matrix} \quad to \\ \end{matrix} \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix}$$

Figure 2: State transition probability in matrix form

Here, each row of the matrix will sum to 1 [7]. The probability that the agent moves into its new state is influenced by the chosen action.

Reward: In a Markov decision process, a reward is a scaler feedback signal that indicates how well an agent is doing for taking actions in each time step. The agents' goal is to maximize

the cumulative reward signal over all the taken actions for a given policy. The reward function can be defined as,

$$R_s = \mathbb{E}\left[R_{t+1} \mid S_t = S\right]$$

Here, $\mathbb{E}$ is the expected reward for each action taken when the agent is in a state S [7][16].

Discount Factor: Discount factor is an element that is used in Markov models for mathematical convenience. It can range from 0 to 1. If the value of the discount factor is 1, that means the model is undiscounted. Discount factor signifies how much an agent should care about the immediate or future rewards in terms of taking actions. Using it has some advantages as it avoids infinite returns in cyclic Markov processes by converging the algorithm. Also, since the future is uncertain, it can also influence the agent to take the instant reward instead of waiting for a larger reward in the future [6][7].

2.2 Value Function

The value function represents how good a state is in the long run for an agent to be in [6][9]. They indicate the long-term potential of the states after considering the states that are likely to appear after taking actions and the rewards available for that. In short, it is equal to the expected total reward for an agent starting from that state [6][9]. It depends on the policy that the agent follows in terms of choosing which actions to take in each state. For all the states, there exist one value function that carry higher value than all other functions. That function is called the optimal value function. The concept of value function is key to the approach that is followed in this paper. It is very important for the efficient search in the space of policies [6][9].

State-Value Function: The state-value function of a Markov decision process is the expected return starting from state s, and then following policy π. If the state value can be denoted as v_π, then

$$v_\pi(s) = \mathbb{E}_{\boldsymbol{\pi}} [G_t \mid S_t = s]$$

Here, $\mathbb{E}_{\boldsymbol{\pi}}$ is the expected return from state s, and then following policy π [6][7].

Action-Value Function: The action-value function is the expected return starting from state s, taking an action a, and then following policy π. If the action value can be denoted as $q_\pi(s, a)$, then

$$q_\pi(s, a) = \mathbb{E}_{\boldsymbol{\pi}} [G_t \mid S_t = s, A_t = a]$$

Here, $\mathbb{E}_{\boldsymbol{\pi}}$ is the expected return from state s, taking an action a, and then following policy π [6][7].

2.3 Bellman Equation

Bellman equation is an optimality equation associated with dynamic programming. A dynamic problem refers to simplifying a complicated problem by breaking it down into simpler sub problems in a recursive manner [10]. To achieve the optimal value of the state-value and action-value functions, this approach is used. It uses the concept of dynamical system's state and of a value function, to define the functional equation.

The state-value function can be broken down into immediate reward and the discounted value of next state [7],

$$v_\pi(s) = \mathbb{E}_{\boldsymbol{\pi}} [R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

The action-value function can be similarly broken down [7],

$$q_\pi(s, a) = \mathbb{E}_{\boldsymbol{\pi}} [R_{t+1} + \gamma q_\pi((S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

2.4 Policy Evaluation

Policy in a Markov decision process maps a state to an action, or a distribution over actions [18]. Evaluation of policies is the exploration of possible policies applicable for the model through iteration. Policy evaluation for a model is needed to find the optimal policy which optimizes the value function for each and every possible state.

For a given policy $\pi\ (a \mid s)$, the probability of taking an action a in a state s is given by the policy π. To iterate through a set of possible policies, each successive approximation is obtained by using the Bellman equation. After a certain number of iterations, the sequence converges [6][18].

To produce each successive approximation, iterative policy evaluation applies the same operation to each state. It replaces the old value of s with a new value obtained from the old values of the successor states of s, and the expected immediate rewards, along all the one-step transitions possible under the policy being evaluated. Each iteration of iterative policy evaluation updates the value of every state once to produce the new approximative value function [6][18].

CHAPTER THREE: PROBLEM DESCRIPTION

3.1 Model Description

We consider an inventory problem where a seller has only one type of product, of unit quantity for each cycle, to order to satisfy demands. Whether or not there will be demand of that product is a case of stochasticity. The demand can also only be of unit quantity for each cycle. The seller can either stock or issue a backorder of the product. However, the seller can stock a maximum of N products and a backorder of maximum M products for the entire scenario. For each state of the inventory, only one of two actions can be chosen. The seller can either decide to buy, or not to buy the product based on the policy. We approach this problem by using the MDP to iterate through possible policies. Our objective is to find the optimal policy that will recommend which actions to take in each inventory state that minimizes the cost of stocking or backordering.

3.2 Assumptions

The model assumptions are mentioned below.

1. The demand is a single product type.
2. The demand is stochastic. There might or might not be a demand for the product each day. If there is a demand, it can only be one unit of the product per day.
3. The seller can only collect one unit of the product before the start of each cycle, which is, in this case, a day. He will follow a policy for this collection.
4. There can be 4 possible situations in this problem.
    a. The seller has collected the product, and there is a demand.
    b. The seller has collected the product, and there is no demand.
    c. The seller has not collected the product, and there is a demand.

d. The seller has not collected the product, and there is no demand.

5. For the situation of 4.a., when the seller has collected the product and there is a demand, he will end up being in the state from the previous cycle.
6. For the situation of 4.b., when the seller has collected the product and there is no demand, he can stock the product for the following day and increase inventory. However, whether he will collect the product for that day is part of the problem and not definite. The seller can stock a maximum of N products for the entire scenario. If the number of stocked products exceed N, the problem will meet terminal ends.
7. For the situation of 4.c., when the seller has not collected the product and there is a demand, the seller must backorder one unit of product. However, whether he will backorder the product for that day is part of the problem and not definite. The seller can do a backorder of maximum M products for the entire scenario. If the number of backorders exceed M, the problem will meet terminal ends.
8. For the situation of 4.d., when the seller has not collected the product and there is no demand, the seller will end up being in the state from the previous cycle.

3.3 Formulation

States: We consider an inventory problem with a single product type. The maximum number of the product the seller can stock is N, and the maximum number of the product the seller can backorder is M. The states for this model are formulated based on the states. If the maximum number of the product the seller can stock is N, and the maximum number of the product the seller can backorder is M, then based on the inventory levels as mentioned previously, the states are; $N, N-1, \dots, N-(N-1), 0, M-(M-1), \dots, M-1, M$.

Actions: The seller can choose from two options when he has to take an action before the start of each cycle (day). The actions are,

a. To buy

b. Not to buy.

The actions are controlled by a policy.

Rewards: The agents' goal is to maximize the cumulative reward signal over all the taken actions for a given policy. The reward has been set in such a way so that the agent always tries to stay in the states that optimize the model by minimizing the cost of stocking or backordering.

The rewards for 3 separate case studies are mentioned here,

Case Study 1 – 3 States: The rewards for the case study with 3 possible states are mentioned below,

a. State Zero, Action Buy = 12.5

b. State Zero, Action No Buy = 12.5

c. State Positive One, Action Buy = -10

d. State Positive One, Action No Buy = 10

e. State Negative One, Action Buy = 10

f. State Negative One, Action No Buy = -10

Case Study 2 – 5 States: The rewards for the case study with 5 possible states are mentioned below,

a. State Zero, Action Buy = 12.5

b. State Zero, Action No Buy = 12.5

c. State Positive One, Action Buy = -10

d. State Positive One, Action No Buy = 10

e. State Positive Two, Action Buy = -7.5

f. State Positive Two, Action No Buy = 7.5

g. State Negative One, Action Buy = 10

h. State Negative One, Action No Buy = -10

i. State Negative Two, Action Buy = 7.5

j. State Negative Two, Action No Buy = -7.5

Case Study 3 – 9 States: The rewards for the case study with 5 possible states are mentioned below,

a. State Zero, Action Buy = 12.5

b. State Zero, Action No Buy = 12.5

c. State Positive One, Action Buy = -10

d. State Positive One, Action No Buy = 10

e. State Positive Two, Action Buy = -7.5

f. State Positive Two, Action No Buy = 7.5

g. State Positive Three, Action Buy = -5

h. State Positive Three, Action No Buy = 5

i. State Positive Four, Action Buy = -2.5

j. State Positive Four, Action No Buy = 2.5

k. State Negative One, Action Buy = 10

l. State Negative One, Action No Buy = -10

m. State Negative Two, Action Buy = 7.5

n. State Negative Two, Action No Buy = -7.5

o. State Negative Three, Action Buy = 5

p. State Negative Three, Action No Buy = -5

q. State Negative Four, Action Buy = 2.5

r. State Negative Four, Action No Buy = -2.5

Policy: The policy has been set in such a way that the agent will always choose the greedy option when he switches to the next state by choosing an action. The agent will choose the greedy option for 90% of the time in these cases.

Transition Probability Matrix: The transition probability matrix has been set in such a way that epitomizes the environment.

Discount Factor: The discount factor for this model has been set as 0.75 for all the scenarios.

3.4 Approach

The model is approached through both policy iteration and value iteration. 3 case studies have been shown with 3 different number of states, with demand probability varying from 0 to 1.

For the policy iteration, initially, an arbitrary policy has been applied to the model. This resulted in the convergence of the action value functions. These action value functions then suggested or recommended the next possible policy. Applying the new improved policy, a new set of action value functions were obtained, which would then go on to suggest the next possible policy. This evaluation and iteration would go on until the action value functions suggest the same policy from where they were extracted. This resulted in the convergence of the optimal policy that would suggest which actions to take in each state of the whole model.

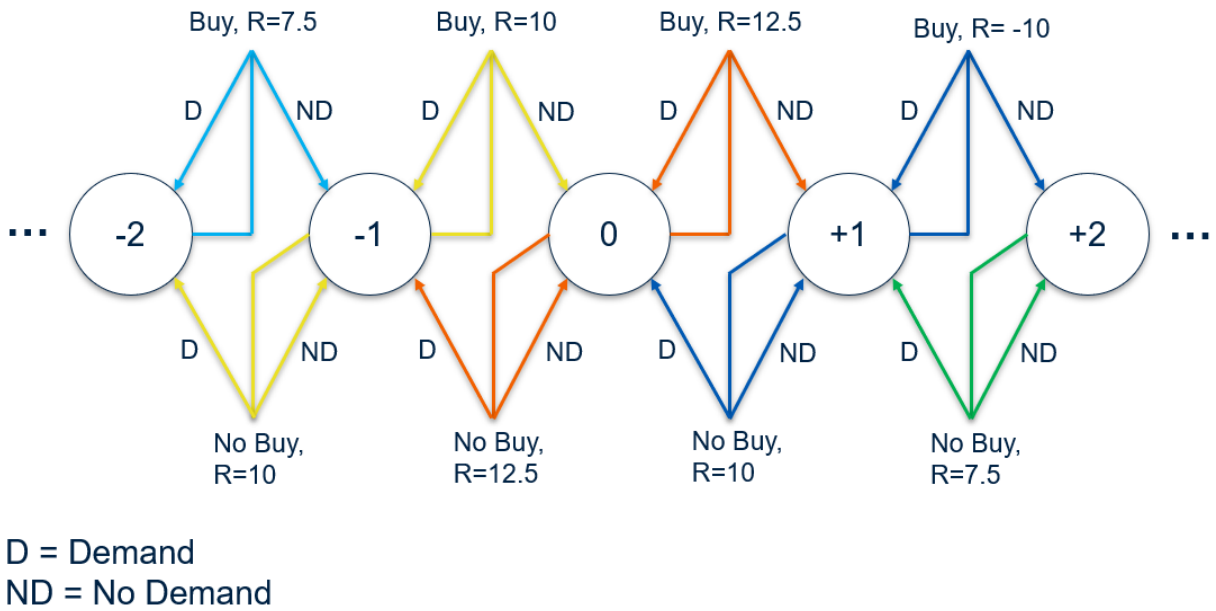

Figure 3: Visual representation of the MDP model

For the value iteration, the same theory has been applied. But instead of waiting for the action values to converge, this methodology would extract the action value functions after one iteration. This new set of action value functions would recommend the next policy.

CHAPTER FOUR: RESULT AND DISCUSSION

4.1 Case Study 1 – 3 States

In case study 1, the scenario has 3 inventory states: Negative one, Zero, and Positive one. Figure 4 illustrates the action value functions for each state, which represents the X axis, based on the demand, which represents Y axis. Each graph signifies a single state. From figure 4, it can be seen that when the inventory is in negative one state, the more profitable action would always be to 'Buy', regardless of the demand. In terms of positive one inventory state, it is the opposite, where 'No Buy' is the more profitable option.

Figure 4: Action value functions for 3 states inventory

When it comes to zero inventory state, it can be seen that the recommended actions change based on the demand pattern. When the demand probability is more than 0.5, the more

profitable action would be to choose ‘Buy’, and when the demand pattern is less than 0.5, the more profitable action would be to choose ‘No Buy’. The recommended actions based on the demand pattern is showed in table 1. This table is the optimal policy for an inventory system with 3 states under conditions mentioned in Chapter 3.1 and 3.2.

Table 1: Optimal policy for 3 states inventory

Demand	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
N1 Inventory State	B	B	B	B	B	B	B	B	B	B	B
Zero Inventory State	NB	NB	NB	NB	NB	B/NB	B	B	B	B	B
P1 Inventory State	NB	NB	NB	NB	NB	NB	NB	NB	NB	NB	NB

4.2 Case Study 2 – 5 States

In case study 2, the scenario has 5 inventory states: Negative two, Negative one, Zero, Positive one, and Positive two. Figure 5 illustrates the action value functions for each state, which represents the X axis, based on the demand, which represents Y axis. Each graph signifies a single state. From figure 5, it can be seen that when the inventory is in negative two and negative one states, the more profitable action would always be to ‘Buy’, regardless of the demand. It can also be seen that the difference between the action value functions of ‘Buy’ and ‘No Buy’ reduces when the inventory states get closer to zero inventory state. In terms of positive two and positive one inventory states, it is the opposite, where ‘No Buy’ is the more profitable option. But the difference between the action value functions of ‘Buy’ and ‘No Buy’ reduces here as well, albeit, the ‘No Buy’ actions values boasting more profit than ‘Buy’ actions, which is the inverse in terms of the negative two and negative one states.

When it comes to zero inventory state, the results are similar to case study 1. It can be seen from the figure 5 that the recommended actions change based on the demand pattern. When the demand probability is more than 0.5, the more profitable action would be to choose ‘Buy’,

and when the demand pattern is less than 0.5, the more profitable action would be to choose 'No Buy'.

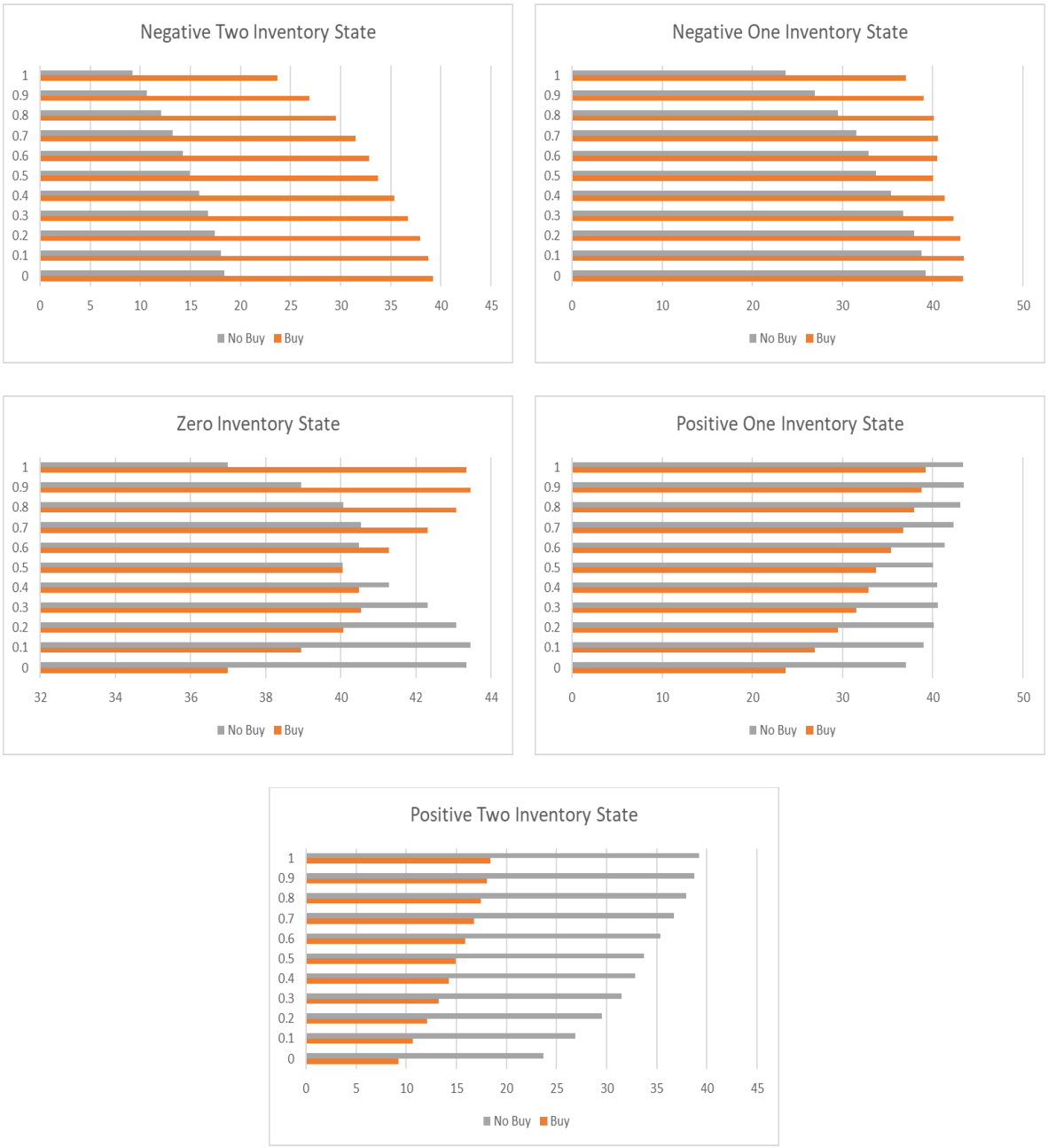

Figure 5: Action value functions for 5 states inventory

The recommended actions based on the demand pattern is showed in table 1. This table is the optimal policy for an inventory system with 5 states under conditions mentioned in Chapter 3.1 and 3.2.

Table 2: Optimal policy for 5 states inventory

Demand	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
N2 Inventory State	B	B	B	B	B	B	B	B	B	B	B
N1 Inventory State	B	B	B	B	B	B	B	B	B	B	B
Zero Inventory State	NB	NB	NB	NB	NB	B/NB	B	B	B	B	B
P1 Inventory State	NB	NB	NB	NB	NB	NB	NB	NB	NB	NB	NB
P2 Inventory State	NB	NB	NB	NB	NB	NB	NB	NB	NB	NB	NB

4.3 Case Study 3: 9 States

In case study 3, the scenario has 9 inventory states: Negative Four, Negative three, Negative two, Negative one, Zero, Positive one, Positive two, Positive three, and Positive Four. Figure 6 illustrates the action value functions for each state, which represents the X axis, based on the demand, which represents Y axis. Each graph signifies a single state. From figure 6, it can be seen that when the inventory is in negative four, negative three, negative two and negative one states, the more profitable action would always be to ‘Buy’, regardless of the demand. It can also be seen, as it can be seen from case study 2, that the difference between the action value functions of ‘Buy’ and ‘No Buy’ reduces as the inventory states get closer to zero inventory state. In terms of positive four, positive three, positive two and positive one inventory states, it is the opposite, where ‘No Buy’ is the more profitable option. But the difference between the action value functions of ‘Buy’ and ‘No Buy’ reduces here as well, albeit, the ‘No Buy’ actions values boasting more profit than ‘Buy’ actions, which is the inverse in terms of the negative states.

When it comes to zero inventory state, the results are similar to case study 1 and case study 2. It can be seen from the figure 6 that the recommended actions change based on the

demand pattern. When the demand probability is more than 0.5, the more profitable action would be to choose ‘Buy’, and when the demand pattern is less than 0.5, the more profitable action would be to choose ‘No Buy’.

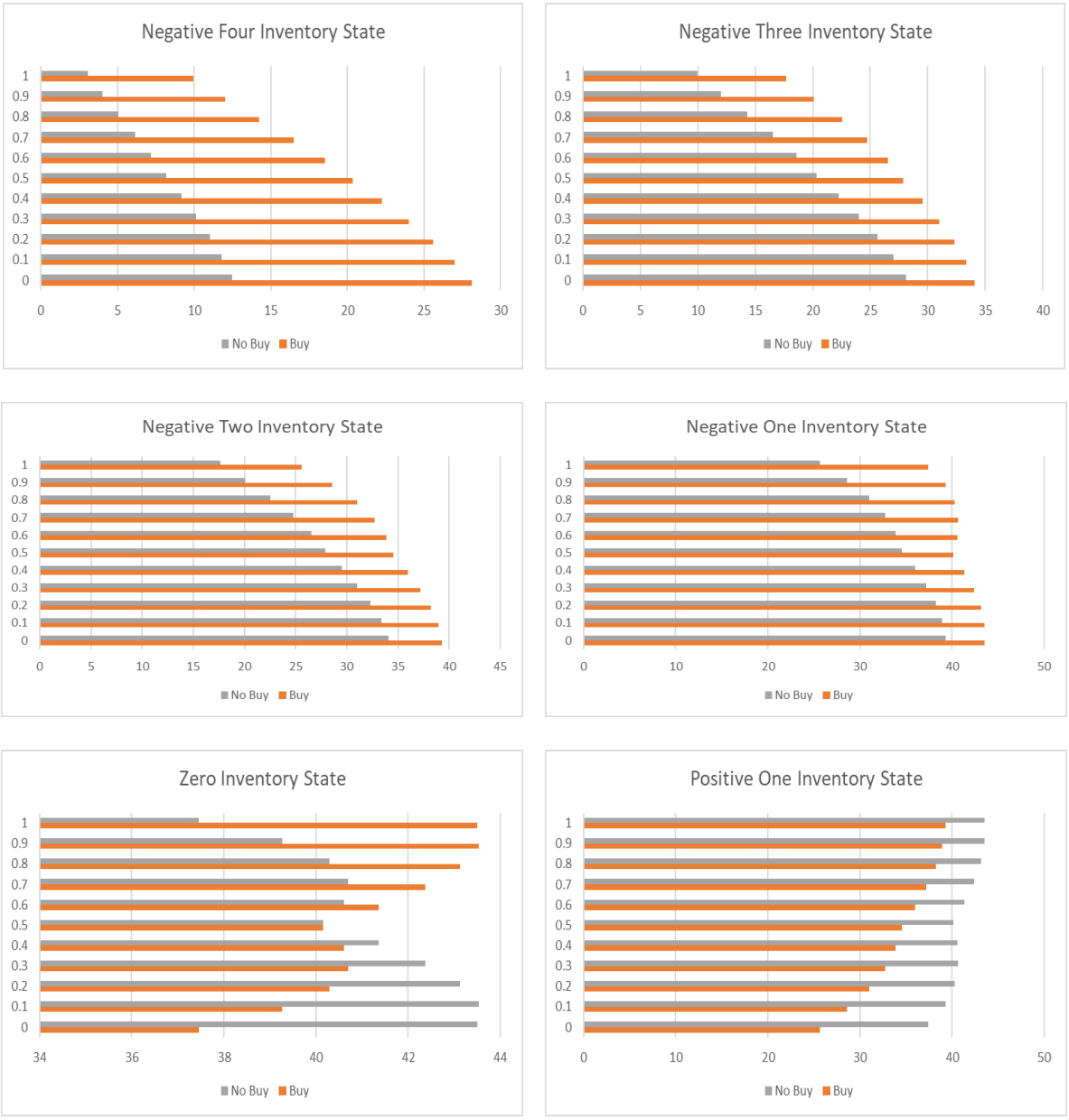

Figure 6: Action value functions for 9 states inventory (a)

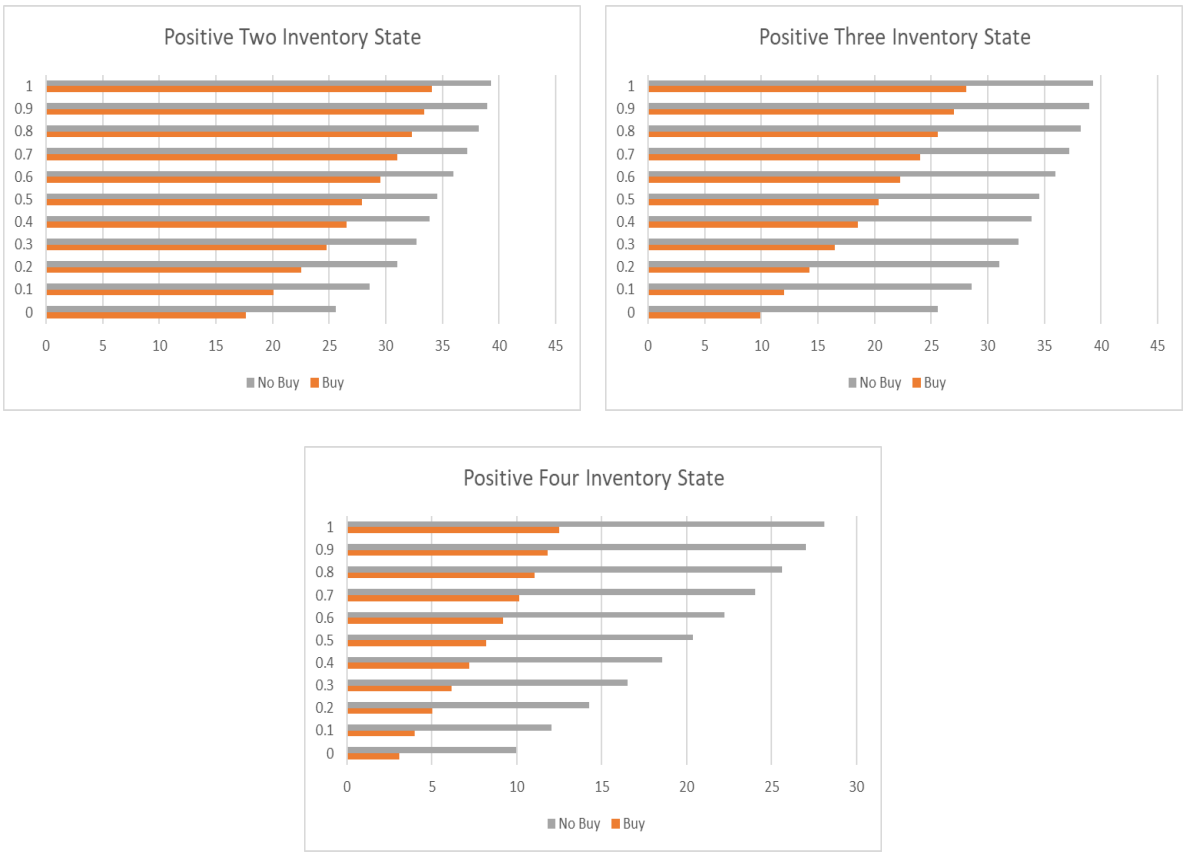

Figure 7: Action value functions for 9 states inventory (b)

The recommended actions based on the demand pattern for case study 3 is showed in table 1. This table is the optimal policy for an inventory system with 5 states under conditions mentioned in Chapter 3.1 and 3.2.

Table 3: Optimal policy for 9 states inventory

Demand	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
N4 Inventory State	B	B	B	B	B	B	B	B	B	B	B
N3 Inventory State	B	B	B	B	B	B	B	B	B	B	B
N2 Inventory State	B	B	B	B	B	B	B	B	B	B	B
N1 Inventory State	B	B	B	B	B	B	B	B	B	B	B
Zero Inventory State	NB	NB	NB	NB	NB	B/NB	B	B	B	B	B
P1 Inventory State	NB	NB	NB	NB	NB	NB	NB	NB	NB	NB	NB
P2 Inventory State	NB	NB	NB	NB	NB	NB	NB	NB	NB	NB	NB
P3 Inventory State	NB	NB	NB	NB	NB	NB	NB	NB	NB	NB	NB
P4 Inventory State	NB	NB	NB	NB	NB	NB	NB	NB	NB	NB	NB

CONCLUSION

Inventory optimization is a critical part of a manufacturing operation. The management of it has a large impact on the profit of the whole system. This project aimed to find a suitable solution for a simple inventory problem consisting of unit demand quantity for each cycle (day). The goal was to find an optimal policy for each inventory state.

From the results, it can be concluded that, in no situation is buying a good action when the inventory is in a positive state. Neither is not buying when the inventory is in a negative state. Even though the solution can be seen as straightforward, this model offered an insight on how the possible actions compare in terms of profit. The symmetry in the results was purely because of the reward functions. If the reward function had followed variation instead of the symmetry, the action value functions would not have followed a pattern like it did in terms of the solution.

This work can be further expanded to inventory problems with more than one demand quantity. By setting the pattern of the demand as a probability distribution, and by increasing the number of inventory states, this model, via a few modifications, can be used to optimize the inventory states with larger demand quantity. This model can also be used in single state inventory problems and perishable inventory problems with necessary modifications.

APPENDICES

APPENDIX A: CODE

A1: CASE STUDY 1

```
import numpy as np


iteration = 500

df        = 0.75


zero0 = 0

p10   = 0

p20   = 0

n10   = 0

n20   = 0


rz_buy    = 12.5

rz_nobuy  = 12.5

rp1_buy   = -10

rp1_nobuy = 10

rn1_buy   = 10

rn1_nobuy = -10


zero = np.zeros([1,iteration])

p1   = np.zeros([1,iteration])
```

```
p2   = 0
n1   = np.zeros([1,iteration])
n2   = 0

policy = [.1, .9]
pc_buy_z  = policy[np.random.randint(0,2)]
pc_buy_p1 = policy[np.random.randint(0,2)]
pc_buy_n1 = policy[np.random.randint(0,2)]

pc_buy_z_tmp = 0
pc_buy_p1_tmp = 0
pc_buy_n1_tmp = 0

count = 0

Final_Data = []

for i, prod in enumerate(np.arange(0,1.1,0.1)):
    while (True):
        count += 1

        zero[0,0] = pc_buy_z *(rz_buy  + df*(prod*zero0 + (1 - prod)*p10)) + (1-pc_buy_z)
*(rz_nobuy  + df*((1 - prod)*zero0 + prod*n10))
```

```
    p1[0,0]   = pc_buy_p1*(rp1_buy + df*(prod*p10   + (1 - prod)*p20)) + (1-
pc_buy_p1)*(rp1_nobuy + df*((1 - prod)*p10   + prod*zero0))
    n1[0,0]   = pc_buy_n1*(rn1_buy + df*((1 - prod)*zero0 + prod*n10)) + (1-
pc_buy_n1)*(rn1_nobuy + df*((1 - prod)*n10   + prod*n20))

    for i in range(iteration-1):
      zero[0,i+1] = pc_buy_z *(rz_buy  + df*(prod*zero[0,i] + (1 - prod)*p1[0,i])) + (1-
pc_buy_z) *(rz_nobuy  + df*((1 - prod)*zero[0,i] + prod*n1[0,i]))
      p1[0,i+1]   = pc_buy_p1*(rp1_buy + df*(prod*p1[0,i]   + (1 - prod)*p2))     + (1-
pc_buy_p1)*(rp1_nobuy + df*((1 - prod)*p1[0,i]   + prod*zero[0,i]))
      n1[0,i+1]   = pc_buy_n1*(rn1_buy + df*((1 - prod)*zero[0,i] + prod*n1[0,i])) + (1-
pc_buy_n1)*(rn1_nobuy + df*((1 - prod)*n1[0,i]   + prod*n2))

    zero_buy   = (zero[0,iteration-1] + p1[0,iteration-1])/2
    zero_nobuy = (zero[0,iteration-1] + n1[0,iteration-1])/2

    p1_buy     = (p1[0,iteration-1] + p2)/2
    p1_nobuy   = (p1[0,iteration-1] + zero[0,iteration-1])/2

    n1_buy     = (zero[0,iteration-1] + n1[0,iteration-1])/2
    n1_nobuy   = (n1[0,iteration-1] + n2)/2

    if zero_buy > zero_nobuy:
```

```
            pc_buy_z_tmp = 0.9
        else:
            pc_buy_z_tmp = 0.1

        if p1_buy > p1_nobuy:
            pc_buy_p1_tmp = 0.9
        else:
            pc_buy_p1_tmp = 0.1

        if n1_buy > n1_nobuy:
            pc_buy_n1_tmp = 0.9
        else:
            pc_buy_n1_tmp = 0.1

        if pc_buy_z == pc_buy_z_tmp and pc_buy_p1 == pc_buy_p1_tmp and pc_buy_n1 ==
pc_buy_n1_tmp:
            break

        else:
            pc_buy_z = pc_buy_z_tmp
            pc_buy_p1 = pc_buy_p1_tmp
            pc_buy_n1 = pc_buy_n1_tmp
```

```
    avf_final = np.array([[prod, n1_buy, n1_nobuy, zero_buy, zero_nobuy, p1_buy, p1_nobuy]])

    Final_Data.append(avf_final)

Final_Data = np.array(Final_Data).reshape([11,7])
```

A2: Case Study 2

```
import numpy as np

iteration = 5000
df        = 0.75

zero0 = 0
p10   = 0
p20   = 0
p30   = 0
n10   = 0
n20   = 0
n30   = 0

rz_buy    = 12.5
rz_nobuy  = 12.5
rp1_buy   = -10
```

```
rp1_nobuy = 10

rp2_buy   = -7.5

rp2_nobuy = 7.5

rn1_buy   = 10

rn1_nobuy = -10

rn2_buy   = 7.5

rn2_nobuy = -7.5


zero = np.zeros([1,iteration])

p1   = np.zeros([1,iteration])

p2   = np.zeros([1,iteration])

p3   = 0

n1   = np.zeros([1,iteration])

n2   = np.zeros([1,iteration])

n3   = 0


policy = [.1, .9]

pc_buy_z  = policy[np.random.randint(0,2)]

pc_buy_p1 = policy[np.random.randint(0,2)]

pc_buy_p2 = policy[np.random.randint(0,2)]

pc_buy_n1 = policy[np.random.randint(0,2)]

pc_buy_n2 = policy[np.random.randint(0,2)]
```

```
pc_buy_z_tmp  = 0

pc_buy_p1_tmp = 0

pc_buy_p2_tmp = 0

pc_buy_n1_tmp = 0

pc_buy_n2_tmp = 0


count = 0


Final_Data = []


for i, prod in enumerate(np.arange(0,1.1,0.1)):

    while (True):

        count += 1


        zero[0,0] = pc_buy_z *(rz_buy  + df*(prod*zero0 + (1 - prod)*p10)) + (1-pc_buy_z)
*(rz_nobuy  + df*((1 - prod)*zero0 + prod*n10))

        p1[0,0]   = pc_buy_p1*(rp1_buy + df*(prod*p10   + (1 - prod)*p20)) + (1-
pc_buy_p1)*(rp1_nobuy + df*((1 - prod)*p10   + prod*zero0))

        p2[0,0]   = pc_buy_p2*(rp2_buy + df*(prod*p20   + (1 - prod)*p30)) + (1-
pc_buy_p2)*(rp2_nobuy + df*((1 - prod)*p20   + prod*p10))

        n1[0,0]   = pc_buy_n1*(rn1_buy + df*((1 - prod)*zero0 + prod*n10)) + (1-
pc_buy_n1)*(rn1_nobuy + df*((1 - prod)*n10   + prod*n20))
```

```
    n2[0,0]   = pc_buy_n2*(rn2_buy + df*((1 - prod)*n10   + prod*n20)) + (1-
pc_buy_n2)*(rn2_nobuy + df*((1 - prod)*n20   + prod*n30))


    for i in range(iteration-1):
        zero[0,i+1] = pc_buy_z *(rz_buy  + df*(prod*zero[0,i] + (1 - prod)*p1[0,i])) + (1-
pc_buy_z) *(rz_nobuy  + df*((1 - prod)*zero[0,i] + prod*n1[0,i]))
        p1[0,i+1]   = pc_buy_p1*(rp1_buy + df*(prod*p1[0,i]   + (1 - prod)*p2[0,i])) + (1-
pc_buy_p1)*(rp1_nobuy + df*((1 - prod)*p1[0,i]   + prod*zero[0,i]))
        p2[0,i+1]   = pc_buy_p2*(rp2_buy + df*(prod*p2[0,i]   + (1 - prod)*p3))      + (1-
pc_buy_p2)*(rp2_nobuy + df*((1 - prod)*p2[0,i]   + prod*p1[0,i]))
        n1[0,i+1]   = pc_buy_n1*(rn1_buy + df*((1 - prod)*zero[0,i] + prod*n1[0,i])) + (1-
pc_buy_n1)*(rn1_nobuy + df*((1 - prod)*n1[0,i]   + prod*n2[0,i]))
        n2[0,i+1]   = pc_buy_n2*(rn2_buy + df*((1 - prod)*n1[0,i]   + prod*n2[0,i])) + (1-
pc_buy_n2)*(rn2_nobuy + df*((1 - prod)*n2[0,i]   + prod*n3))


    zero_buy = (zero[0,iteration-1] + p1[0,iteration-1])/2
    zero_nobuy = (zero[0,iteration-1] + n1[0,iteration-1])/2


    p1_buy = (p1[0,iteration-1] + p2[0,iteration-1])/2
    p1_nobuy = (p1[0,iteration-1] + zero[0,iteration-1])/2


    p2_buy = (p2[0,iteration-1] + p3)/2
```

```
p2_nobuy = (p2[0,iteration-1] + p1[0,iteration-1])/2

n1_buy = (zero[0,iteration-1] + n1[0,iteration-1])/2
n1_nobuy = (n1[0,iteration-1] + n2[0,iteration-1])/2

n2_buy = (n2[0,iteration-1] + n1[0,iteration-1])/2
n2_nobuy = (n2[0,iteration-1] + n3)/2

if zero_buy > zero_nobuy:
    pc_buy_z_tmp = 0.9
else:
    pc_buy_z_tmp = 0.1

if p1_buy > p1_nobuy:
    pc_buy_p1_tmp = 0.9
else:
    pc_buy_p1_tmp = 0.1

if p2_buy > p2_nobuy:
    pc_buy_p2_tmp = 0.9
else:
    pc_buy_p2_tmp = 0.1
```

```
        if n1_buy > n1_nobuy:
            pc_buy_n1_tmp = 0.9
        else:
            pc_buy_n1_tmp = 0.1

        if n2_buy > n2_nobuy:
            pc_buy_n2_tmp = 0.9
        else:
            pc_buy_n2_tmp = 0.1

        if pc_buy_z == pc_buy_z_tmp and pc_buy_p1 == pc_buy_p1_tmp and pc_buy_p2 ==
pc_buy_p2_tmp and pc_buy_n1 == pc_buy_n1_tmp and pc_buy_n2 == pc_buy_n2_tmp:
            break

        else:
            pc_buy_z  = pc_buy_z_tmp
            pc_buy_p1 = pc_buy_p1_tmp
            pc_buy_p2 = pc_buy_p2_tmp
            pc_buy_n1 = pc_buy_n1_tmp
            pc_buy_n2 = pc_buy_n2_tmp

    avf_final = np.array([[prod,n2_buy, n2_nobuy, n1_buy, n1_nobuy, zero_buy, zero_nobuy,
p1_buy, p1_nobuy, p2_buy, p2_nobuy]])
```

```
    Final_Data.append(avf_final)


Final_Data = np.array(Final_Data).reshape([11,11])
```

A3: CASE STUDY 3

```
import numpy as np


iteration = 500

df        = 0.75


zero0 = 0

p10   = 0

p20   = 0

p30   = 0

p40   = 0

p50   = 0

n10   = 0

n20   = 0

n30   = 0

n40   = 0

n50   = 0


rz_buy    = 12.5
```

```
rz_nobuy  = 12.5

rp1_buy   = -10

rp1_nobuy = 10

rp2_buy   = -7.5

rp2_nobuy = 7.5

rp3_buy   = -5

rp3_nobuy = 5

rp4_buy   = -2.5

rp4_nobuy = 2.5

rn1_buy   = 10

rn1_nobuy = -10

rn2_buy   = 7.5

rn2_nobuy = -7.5

rn3_buy   = 5

rn3_nobuy = -5

rn4_buy   = 2.5

rn4_nobuy = -2.5


zero = np.zeros([1,iteration])

p1   = np.zeros([1,iteration])

p2   = np.zeros([1,iteration])

p3   = np.zeros([1,iteration])

p4   = np.zeros([1,iteration])
```

```
p5   = 0
n1   = np.zeros([1,iteration])
n2   = np.zeros([1,iteration])
n3   = np.zeros([1,iteration])
n4   = np.zeros([1,iteration])
n5   = 0

policy = [.1, .9]
pc_buy_z  = policy[np.random.randint(0,2)]
pc_buy_p1 = policy[np.random.randint(0,2)]
pc_buy_p2 = policy[np.random.randint(0,2)]
pc_buy_p3 = policy[np.random.randint(0,2)]
pc_buy_p4 = policy[np.random.randint(0,2)]
pc_buy_n1 = policy[np.random.randint(0,2)]
pc_buy_n2 = policy[np.random.randint(0,2)]
pc_buy_n3 = policy[np.random.randint(0,2)]
pc_buy_n4 = policy[np.random.randint(0,2)]

pc_buy_z_tmp  = 0
pc_buy_p1_tmp = 0
pc_buy_p2_tmp = 0
pc_buy_p3_tmp = 0
pc_buy_p4_tmp = 0
```

```
pc_buy_n1_tmp = 0

pc_buy_n2_tmp = 0

pc_buy_n3_tmp = 0

pc_buy_n4_tmp = 0


count = 0


Final_Data = []


for i, prod in enumerate(np.arange(0,1.1,0.1)):
    while (True):
        count += 1


        zero[0,0] = pc_buy_z *(rz_buy  + df*(prod*zero0 + (1 - prod)*p10)) + (1-pc_buy_z)
*(rz_nobuy  + df*((1 - prod)*zero0 + prod*n10))
        p1[0,0]   = pc_buy_p1*(rp1_buy + df*(prod*p10   + (1 - prod)*p20)) + (1-
pc_buy_p1)*(rp1_nobuy + df*((1 - prod)*p10   + prod*zero0))
        p2[0,0]   = pc_buy_p2*(rp2_buy + df*(prod*p20   + (1 - prod)*p30)) + (1-
pc_buy_p2)*(rp2_nobuy + df*((1 - prod)*p20   + prod*p10))
        p3[0,0]   = pc_buy_p3*(rp3_buy + df*(prod*p30   + (1 - prod)*p40)) + (1-
pc_buy_p3)*(rp3_nobuy + df*((1 - prod)*p30   + prod*p20))
        p4[0,0]   = pc_buy_p4*(rp4_buy + df*(prod*p40   + (1 - prod)*p50)) + (1-
pc_buy_p4)*(rp4_nobuy + df*((1 - prod)*p40   + prod*p30))
```

```
        n1[0,0]   = pc_buy_n1*(rn1_buy + df*((1 - prod)*zero0 + prod*n10)) + (1-
pc_buy_n1)*(rn1_nobuy + df*((1 - prod)*n10   + prod*n20))
        n2[0,0]   = pc_buy_n2*(rn2_buy + df*((1 - prod)*n10   + prod*n20)) + (1-
pc_buy_n2)*(rn2_nobuy + df*((1 - prod)*n20   + prod*n30))
        n3[0,0]   = pc_buy_n3*(rn3_buy + df*((1 - prod)*n20   + prod*n30)) + (1-
pc_buy_n3)*(rn3_nobuy + df*((1 - prod)*n30   + prod*n40))
        n4[0,0]   = pc_buy_n4*(rn4_buy + df*((1 - prod)*n30   + prod*n40)) + (1-
pc_buy_n4)*(rn4_nobuy + df*((1 - prod)*n40   + prod*n50))


        for i in range(iteration-1):
            zero[0,i+1] = pc_buy_z *(rz_buy  + df*(prod*zero[0,i] + (1 - prod)*p1[0,i])) + (1-
pc_buy_z) *(rz_nobuy  + df*((1 - prod)*zero[0,i] + prod*n1[0,i]))
            p1[0,i+1]   = pc_buy_p1*(rp1_buy + df*(prod*p1[0,i]   + (1 - prod)*p2[0,i])) + (1-
pc_buy_p1)*(rp1_nobuy + df*((1 - prod)*p1[0,i]   + prod*zero[0,i]))
            p2[0,i+1]   = pc_buy_p2*(rp2_buy + df*(prod*p2[0,i]   + (1 - prod)*p3[0,i])) + (1-
pc_buy_p2)*(rp2_nobuy + df*((1 - prod)*p2[0,i]   + prod*p1[0,i]))
            p3[0,i+1]   = pc_buy_p3*(rp3_buy + df*(prod*p3[0,i]   + (1 - prod)*p4[0,i])) + (1-
pc_buy_p3)*(rp3_nobuy + df*((1 - prod)*p3[0,i]   + prod*p2[0,i]))
            p4[0,i+1]   = pc_buy_p4*(rp4_buy + df*(prod*p4[0,i]   + (1 - prod)*p50))     + (1-
pc_buy_p4)*(rp4_nobuy + df*((1 - prod)*p4[0,i]   + prod*p3[0,i]))
            n1[0,i+1]   = pc_buy_n1*(rn1_buy + df*((1 - prod)*zero[0,i] + prod*n1[0,i])) + (1-
pc_buy_n1)*(rn1_nobuy + df*((1 - prod)*n1[0,i]   + prod*n2[0,i]))
```

n2[0,i+1] = pc_buy_n2*(rn2_buy + df*((1 - prod)*n1[0,i] + prod*n2[0,i])) + (1-pc_buy_n2)*(rn2_nobuy + df*((1 - prod)*n2[0,i] + prod*n3[0,i]))

n3[0,i+1] = pc_buy_n3*(rn3_buy + df*((1 - prod)*n2[0,i] + prod*n3[0,i])) + (1-pc_buy_n3)*(rn3_nobuy + df*((1 - prod)*n3[0,i] + prod*n4[0,i]))

n4[0,i+1] = pc_buy_n4*(rn4_buy + df*((1 - prod)*n3[0,i] + prod*n4[0,i])) + (1-pc_buy_n4)*(rn4_nobuy + df*((1 - prod)*n4[0,i] + prod*n50))

zero_buy = (zero[0,iteration-1] + p1[0,iteration-1])/2

zero_nobuy = (zero[0,iteration-1] + n1[0,iteration-1])/2

p1_buy = (p1[0,iteration-1] + p2[0,iteration-1])/2

p1_nobuy = (p1[0,iteration-1] + zero[0,iteration-1])/2

p2_buy = (p2[0,iteration-1] + p3[0,iteration-1])/2

p2_nobuy = (p2[0,iteration-1] + p1[0,iteration-1])/2

p3_buy = (p3[0,iteration-1] + p4[0,iteration-1])/2

p3_nobuy = (p3[0,iteration-1] + p2[0,iteration-1])/2

p4_buy = (p4[0,iteration-1] + p5)/2

p4_nobuy = (p4[0,iteration-1] + p3[0,iteration-1])/2

n1_buy = (n1[0,iteration-1] + zero[0,iteration-1])/2

```
n1_nobuy   = (n1[0,iteration-1]   + n2[0,iteration-1])/2

n2_buy     = (n2[0,iteration-1]   + n1[0,iteration-1])/2
n2_nobuy   = (n2[0,iteration-1]   + n3[0,iteration-1])/2

n3_buy     = (n3[0,iteration-1]   + n2[0,iteration-1])/2
n3_nobuy   = (n3[0,iteration-1]   + n4[0,iteration-1])/2

n4_buy     = (n4[0,iteration-1]   + n3[0,iteration-1])/2
n4_nobuy   = (n4[0,iteration-1]   + n5)/2

if zero_buy > zero_nobuy:
    pc_buy_z_tmp = 0.9
else:
    pc_buy_z_tmp = 0.1

if p1_buy > p1_nobuy:
    pc_buy_p1_tmp = 0.9
else:
    pc_buy_p1_tmp = 0.1

if p2_buy > p2_nobuy:
    pc_buy_p2_tmp = 0.9
```

```
else:
    pc_buy_p2_tmp = 0.1

if p3_buy > p3_nobuy:
    pc_buy_p3_tmp = 0.9
else:
    pc_buy_p3_tmp = 0.1

if p4_buy > p4_nobuy:
    pc_buy_p4_tmp = 0.9
else:
    pc_buy_p4_tmp = 0.1

if n1_buy > n1_nobuy:
    pc_buy_n1_tmp = 0.9
else:
    pc_buy_n1_tmp = 0.1

if n2_buy > n2_nobuy:
    pc_buy_n2_tmp = 0.9
else:
    pc_buy_n2_tmp = 0.1
```

```
        if n3_buy > n3_nobuy:
            pc_buy_n3_tmp = 0.9
        else:
            pc_buy_n3_tmp = 0.1

        if n4_buy > n4_nobuy:
            pc_buy_n4_tmp = 0.9
        else:
            pc_buy_n4_tmp = 0.1

        if pc_buy_z == pc_buy_z_tmp and pc_buy_p1 == pc_buy_p1_tmp and pc_buy_p2 ==
pc_buy_p2_tmp and pc_buy_p3 == pc_buy_p3_tmp and pc_buy_p4 == pc_buy_p4_tmp and
pc_buy_n1 == pc_buy_n1_tmp and pc_buy_n2 == pc_buy_n2_tmp and pc_buy_n3 ==
pc_buy_n3_tmp and pc_buy_n4 == pc_buy_n4_tmp:
            break

        else:
            pc_buy_z  = pc_buy_z_tmp
            pc_buy_p1 = pc_buy_p1_tmp
            pc_buy_p2 = pc_buy_p2_tmp
            pc_buy_p3 = pc_buy_p3_tmp
            pc_buy_p4 = pc_buy_p4_tmp
            pc_buy_n1 = pc_buy_n1_tmp
```

```
            pc_buy_n2 = pc_buy_n2_tmp
            pc_buy_n3 = pc_buy_n3_tmp
            pc_buy_n4 = pc_buy_n4_tmp

    avf_final = np.array([[prod, n4_buy, n4_nobuy, n3_buy, n3_nobuy, n2_buy, n2_nobuy,
n1_buy, n1_nobuy, zero_buy, zero_nobuy, p1_buy, p1_nobuy, p2_buy, p2_nobuy, p3_buy,
p2_nobuy, p4_buy, p4_nobuy]])

    Final_Data.append(avf_final)

Final_Data = np.array(Final_Data).reshape([11,19])
```

REFERENCES

[1] Kall, Peter, Stein W. Wallace, and Peter Kall. *Stochastic programming*. Chichester: Wiley, 1994.

[2] White, C. C. *Markov decision processes*. Springer US, 2001.

[3] Puterman, Martin L. *Markov Decision Processes.: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[4] Udo, Godwin J. "Neural networks applications in manufacturing processes." *Computers & industrial engineering* 23.1-4 (1992): 97-100.

[5] Guo, Xiaoxiao, et al. "A prediction-based inventory optimization using data mining models." *2014 Seventh International Joint Conference on Computational Sciences and Optimization*. IEEE, 2014.

[6] Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[7] Silver, David. "Lecture 2: Markov Decision Processes." *UCL. Retrieved from www0. cs. ucl. ac. uk/staff/d. silver/web/Teaching_files/MDP. pdf* (2015).

[8] Howard, Ronald A. "Dynamic programming and markov processes." (1960).

[9] Li, Yuxi. "Deep reinforcement learning: An overview." *arXiv preprint arXiv:1701.07274* (2017).

[10] Leiserson, Charles Eric, et al. *Introduction to algorithms*. Vol. 6. Cambridge, MA: MIT press, 2001.

[11] Mahadevan, Sridhar, and Georgios Theocharous. "Optimizing Production Manufacturing Using Reinforcement Learning." *FLAIRS Conference*. 1998.

[12] Qu, Shuhui, et al. "Optimized adaptive scheduling of a manufacturing process system with multi-skill workforce and multiple machine types: An ontology-based, multi-agent reinforcement learning approach." *Procedia CIRP* 57 (2016): 55-60.

[13] Du, Shichang, et al. "Markov modeling and analysis of multi-stage manufacturing systems with remote quality information feedback." *Computers & Industrial Engineering* 88 (2015): 13-25.

[14] Paternina-Arboleda, Carlos D., and Tapas K. Das. "Intelligent dynamic control policies for serial production lines." *Iie Transactions* 33.1 (2001): 65-77.

[15] Mao, Hongzi, et al. "Resource management with deep reinforcement learning." *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. ACM, 2016.

[16] Silver, David. "Lecture 1: Introduction to Reinforcement Learning." *Google DeepMind* (2015).

[17] Birge, John R., and Francois Louveaux. *Introduction to stochastic programming*. Springer Science & Business Media, 2011.

[18] Silver, David. "Lecture 3: Planning by Dynamic Programming." *Google DeepMind* (2015).

VITA

Eakeen Muhammad Haque is from Dhaka, Bangladesh. He earned his Bachelor of Science in Mechanical Engineering degree from Khulna University of Engineering and Technology in 2016. He started his Master's degree in Lamar University, Beaumont, Texas in 2017. In 2018, he transferred to the University of Texas at San Antonio. He is working under Dr. Alaeddini in the field of manufacturing. After the completion of this Master's, he plans to put his knowledge and learnings in the industry sector.