# A DESIGN AND ANALYSIS OF COMPUTER EXPERIMENTS-BASED APPROACH TO APPROXIMATE INFINITE HORIZON DYNAMIC PROGRAMMING WITH CONTINUOUS STATE SPACES

by

ASAMA KULVANITCHAIYANUNT

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2014

Acknowledgements

First of all, I would like to express my deepest gratitude to my supervising professors, Dr. Victoria Chen and Dr. Jay Rosenberger, for all their guidance, advice and motivation throughout the journey of my study at UT Arlington. I appreciate their leading me into an exciting and challenging research area. Dr. Chen has been an excellent role model, as a researcher and lecturer. She inspired me with her enthusiasm for research and her ability to write superb technical documents, as well as her communication skills. Dr. Rosenberger is himself very impressive. His knowledge in optimization is unmatched. He provided good insight into the research, encouraged me to think independently, and he kept me motivated to work. Both my advisors cared not only about my studies, but they also cared about my personal life. In Thailand, my country, they say that a teacher is just like a small boat that transfers people to the final destination. They both, Dr. Chen and Dr. Rosenberger, are just like that in my point of view. They gave me the path to make my dream come true.

I also want to acknowledge the wonderful support of my project collaborators from the Department of Electrical Engineering, Dr. Wei-Jen Lee and Mr. Piampoom Sarikprueck. They always provided me valuable comments and advice that were significant to the project. I would also like to thank my committee member Dr. Shouyi Wang for good comments that improved my dissertation research.

For my financial support, I would like to thank the National Science Foundation and the Department of Industrial & Manufacturing Systems Engineering at UT Arlington.

I will always remember my friends from the Center on Stochastic Modeling, Optimization, & Statistics (COSMOS) that made my doctoral study life enjoyable.

Last but not least, I would like to thank my family, for their unflinching faith in my ability. My father is my role model for living. He is the best doctor I have ever seen. He

not only taught me how to walk but also walked with me. My mother keeps telling me that she is so proud of me. That makes me feel that I can go further. I would like to thank my grandmother for bringing some smiles to my face. I would like to take this opportunity to express my thanks to my aunts, Ratchya, Ratanyou and Hathaitip, for their support. I also thank all my other friends and relatives, without their love and support, I would not have gone so far.

April 14, 2014

Abstract

A DESIGN AND ANALYSIS OF COMPUTER EXPERIMENTS-BASED APPROACH TO APPROXIMATE INFINITE HORIZON DYNAMIC PROGRAMMING WITH CONTINUOUS STATE SPACE

Asama Kulvanitchaiyanunt, PhD

The University of Texas at Arlington, 2014

Supervising Professor: Victoria Chen and Jay Rosenberger

Dynamic programming (DP) is an optimization approach that transforms a complex problem into a sequence of simpler sub-problems at different points in stage. The original DP approach used Bellman's equation to compute the "cost-to-go' function. This method is useful when considering a few states and decisions. However, when dealing with high-dimensional data set with continuous state space, the limit called 'curse of dimensionality' obstructs the solution as the size of the state space grows exponentially. Given recent advances in computational power, approximate dynamic programming (ADP) is introduced by not seeking to compute the future value function exactly and at each point of the state space; rather opting for an approximation of the future value function in the domain of the state space. Two main components of ADP method which have been challenged among existing ADP studies are *discretization* of the state space and *estimation* of the cost-to-go or future value function.

The first part of this dissertation research seeks to develop a solution method to solve an infinite horizon dynamic programming called Design and Analysis of Computer Experiment (DACE)-based Approach to ADP. Multivariate Adaptive Regression Splines

(MARS) which is a flexible, nonparametric statistical modeling tool is used to approximate future value functions in stochastic dynamic programming (SDP) problems with continuous state variables. The training data set is updated sequentially based on the conditions. This sequential grid discretization explores the state space and provides a statistically parsimonious ADP methodology which 'adaptively' captures the important variables from the state space. There are 3 different algorithms presented in this dissertation based on the conditions of sampling process of the training data set. Comparisons are presented on a forward simulation with 12 time periods.

The second part of the dissertation research is to develop a batch mode Reinforcement Learning (RL) using MARS as an approximator to solve the same problem with the first part. The main difference between these two methods is the input variables to approximate future value function. In batch mode RL method, the state-action space is used, thus the estimated function (output) is a function of both state and action variables. By contrast, DACE-based ADP used only state variable and the estimated future function is based only on state variables. The study on state-action discretization is presented in this dissertation. Two different designs are used, including Monte Carlo sampling and Sobol' sequence design. Comparisons are presented on the same forward simulation.

The third part is to develop a two-stage framework for Adaptive Design for Controllability of a System of Plug-in Hybrid Electric Vehicle Charging Stations Case Study. The second-stage dynamic control problem is formulated and initially solved by mean value problem using linear programming. After that a DACE approach is used to develop a metamodel of the second stage solution based on the possible solution from the first stage. Then the metamodel will be turned into the first stage and at this point the final solution will be made. DACE helps reduce time-consuming computer models by replacing the loop between first and second stage with a constraint generated from the

gradient of the approximation function. Moreover, the metamodel can give more accessible description to the second stage.

Table of Contents

List of Illustrations

List of Tables

# Chapter 1

# Introduction

## 1.1 Overview of Dynamic Programming

Dynamic programming (DP) was first introduced by Bellman in 1957 as an optimization approach that transforms a complex problem into a sequence of simpler sub-problems at different stages, often represented by time periods [1]. The essential characteristic is the multistage nature of the optimization (or decision-making) procedure. It has been widely used as a tool in variety of study areas, such as engineering, economics, finance, energy and science [2-8]. For example, in economics research, where the problem is dealing with time periods or making a decision regarding to a time period, DP is always seen to be tool to model and solve that problem [2, 9-12]. In the energy sector, the real-time problem or location problem uses DP [4-7], where time or location is referred to as stages. Another example of using DP as a tool is found in finance, a method presented by Yan and Bai [8] uses DP to model and allocate funds between stocks in a stock portfolio, as a result, to maximize income.

In a typical DP problem, a system evolves through a series of consecutive *stages* (or time periods). At each stage the system can be defined by a set of *state variables* (*state* for short). One or more *decisions* must be made at each stage. These decisions may depend on either the stage or the state or both. When a decision is made, a *return* (either profit or cost) is obtained, and the system undergoes a *transition* to the next stage. The return is assumed to be a known single-valued function of the state and decision. The *objective* of the DP formulation is to maximize the total profit or minimize the total cost over all the stages, and the output is the *policy* (series of decisions made in each of the consecutive stages) that achieves the objective [13].

Since the original sequential problem is converted into a collection of small problems (sub-problems), it is important to make sure that the solution to each of the sub-problems is actually optimal in the original problem. In other words, the optimal choice in each stage-wise problem must be globally optimal to justify the transformation. Moreover, future decisions for the remaining stages will constitute an optimal policy with regard to the state resulting from the first decision. Bellman called this the "Principle of Optimality" [1, 13]. Given the different characteristics of the basic elements represented in potential problems, DP can be classified by: discrete or continuous stages, deterministic or nondeterministic (stochastic) transitions, finite or infinite horizon, discrete or continuous states, discrete or continuous decisions. This dissertation mainly focuses on infinite horizon, stochastic dynamic problems with continuous state and decision spaces.

*1.1.1 Stages*

DP is a method for solving complex problems by breaking the problem down into simpler sub-problems. Sub-problems are indexed by stages. The resulting multistage problem is then solved stage by stage. At each stage, the sub-problem is solved as an ordinary optimization problem, and its solution helps to define the characteristics of the next stage problem in the sequence. The stages often represent different time periods in the problem's planning horizon. For example, the problem of determining the daily battery level of a *plug-in hybrid electric vehicle* (*PHEV*) charging station can be stated as a DP problem, where the objective is to minimize a cost function in each time period (each day) or stage. Sometimes the stages do not have time implications. For example, consider the problem of determining the shortest path for a traveler's trip from one city to another city with a certain number of intermediate stops in some optional cities. This problem can be formulated as a DP problem, where stages are defined as the number of the stops made by the traveler.

A DP problem can be defined to have discrete or continuous stages based on the structure of the stage process. Stage is often analogous to time period, but it can also be something else. For example, in supply chain problem, they defined stage as the distance from the start point to the end, and they consider a continuous-stage system in this problem because it allows taking a real-time continuous production/transportation decision in every point in the distance [14]. The PHEV charging stations control problem has discrete time because the system follows the timing of the electricity market which evolves in 15-minute time intervals. A continuous-stage structure is frequently encountered in problems that are addressed by the classical methods of calculus of variations [12-13].

In case of discrete stages, the problem is classified by the problem's time horizon, if the problem has a finite number of stages, then it is DP with finite horizon; otherwise, it is DP with infinite horizon. The solution methods for solve those two different horizons of DP are different [15].

*1.1.2 States*

The states reflect the information required to fully assess the consequences that the current decision has upon future actions on each stage. In the inventory problem by Chen (1999), each stage (time period) has only two variables describing the state which are the inventory level on hand of the single commodity and demand forecasts [16].

States can be defined to be discrete or continuous or mixed. The gambling problem, for example, states are the amount betted in each round where round is referred to stage. In this case, it is discrete state [15]. The initial battery level in the PHEV charging stations control problem is one of the state variables and it does include continuous elements. Thus, the PHEV charging stations control problem has continuous state space.

The specification of the states of the system is very important to a DP problem. Suggestions that motivate the selection of states include: i) The state should provide enough information to make future decisions without considering how the process reached the current state (i.e., Markovian property); and ii) The number of states should be small to reduce the computational effort that may lead to the curse of dimensionality. As a variable, the state can be discrete or continuous or mixed. It may also have high dimension.

Powell [15] defines a state variable as the 'minimally' dimensioned function of history that is necessary and sufficient to compute the decision function, the transition function, and the contribution function. In real situation, if the entire history of states is included up to the current time period, the problem is dealing with computational work. Here, the word 'minimally' is of significance. It can address the issue of the curse of dimensionality associated with the increase in the size of the state space. Thus, the state space should to be as small as possible.

*1.1.3 Decisions and Policies*

Given the state entering a stage, decisions are made in that stage to achieve a desired objective in that stage, subject to any required constraints. These decisions affect the state of the next stage, which affects the decisions in the next stage. In the inventory problem [16] the decision is the amount of the commodity to order at the beginning of each time period. Decision variables can be discrete or continuous or mixed.

A policy is an ordered set of decisions by state and stage. In the inventory problem [16], for example, a policy can be the twelve ordering amounts of the commodity dependent on the inventory levels entering each of the twelve months. An optimal policy is one that maximizes total profit or minimizes total cost over all the stages.

*1.1.4 Transition Function*

A transition function defines the transition of the state variables from the current stage to the next. It is a function of the state, decision, and random variables. function may be categorized as "deterministic" or "nondeterministic." The simplest case is deterministic, which can be illustrated by the shortest-path problem: If the traveler is in a given city, he will move to another city with complete certainty because the distances between each two of the associated cities are known. In this problem, the stage is time that has the traveler has elapsed since the beginning of his travel. The states are possible cities in the current stage and decision are destinations to select for the next stage. Transition function, in this case, is the value (distance) since the beginning of time period to current stage is equal to the return associated with a particular state plus the accumulative return though stage before the current stage. The distances between cities are known, thus it can deterministically make a decision and transform the state in the current stage to a state in the next stage [17]. The A nondeterministic or stochastic case is the one for which the decision outcome is unknown with certainty, but the state and decision variables at each stage are determined as a result of some (assumed) known probability distribution. Transition functions are generally assumed known, but if unknown, they can be approximated by some methods, for example, data mining and statistical modeling [18-22].

We can also classify types of transition function by stationary or not. The stationary system occur when the distribution function of the transition do not change with stage or time. Moreover, in a stationary problem, the return per stage and the randomness statistics are also unchanged from one stage to the next stage [23-24]. For example, the PHEV charging stations control problem has a stationary system because the statistical properties of each state variable do not change over time, as a result, we

have the same transition function for all stages. For the finite-horizon case, transitions can be stationary or nonstationary, and the stages can involve different structure for states, objectives, decisions, etc. For the infinite-horizon case, a steady-state equilibrium policy is desired, hence, the transitions must be assumed stationary

*1.1.5 Future Value or Cost-To-Go Function*

A *stage return* is the return provided by a system in each stage of a process. The *total return* of a process or system depends upon the decisions that are made at each stage. In this dissertation, we only focus on the case of discrete-time. Based on series of discrete stages, there are two types of DP, finite and infinite horizon. In a finite horizon problem, the total return is some combination of the stage returns (e.g., a sum or product), which are accumulated as the process moves from state to state, or equivalently from stage to stage. For the infinite-horizon case, a steady-state equilibrium policy is desired, hence, the transitions must be assumed stationary. In each stage, iterative algorithm is required to solve for a return but when the process reach the steady-state equilibrium, then the return at the steady -state point will become the total return.

The purpose of solving a DP problem is to find the optimal total return. It should also be noticed that it is possible for the return functions to vary from stage to stage. In the inventory problem [16], the stage return can be the ordering and inventory-carrying cost for a month or each time period, and the total return can be the total of that cost for all the months.

Given the different characteristics of the basic elements, there can be various types of DP: deterministic or stochastic transitions, finite or infinite horizon, discrete or continuous states and decisions. The key to a DP solution is the *future value function* or *cost-to-go function* that provides the optimal return to operate the system from a given stage to the end of the time horizon corresponding to the states in that given stage. It can

also solve for the optimal control function that corresponds to the optimal policy and is based on “the principle of optimality."

### 1.2 Main Advantages and Limitations

One of the main advantages of DP is that it transforms a single high-dimensional optimization problem into a sequence of small optimization problems which can be solved in sequence. Another important advantage of DP is that it determines the absolute (global) optima rather than relative (local) optima.

However, certain limitations of DP are still noticeable. The first limitation is that it assumes a fully observed system which means that it allows the system be able to make decisions based on the full knowledge of state space. In fact, in practice, the accurate and comprehensive process knowledge of complex nonlinear control system is rarely known *a priori*. In the case of a non-fully observed system, the partially observable Markov decision process (POMDP) can provide an elegant framework that indirectly observes the states of the system via a set of noisy observations. But with a complex problem, POMDPs takes long time to solve [25]. The second disadvantage is the “curse of dimensionality": the required DP calculations become cost-prohibitive, as the number of states and decisions increase, with an exponential growth in the computation with respect to the dimension of state space [15]. When DP has a high-dimensional state space with continuous-state problems, especially stochastic problems, both of the above limitations become intractable for the classical solution methods. In addition, the original or ‘exact’ DP methodology is relevant when the state of the system is defined completely by a finite set of discrete variables. It breaks down when considering infinite horizon with continuous state space DP. These limitations have motivated the research community to look and move beyond the classical Bellman’s approach to formulate and solve those difficult DP problems. Thus, approximate dynamic programming (ADP) was introduced.

## 1.3 Motivation

The original DP approach provided by Bellman [1] is used to compute the "cost-to-go' function and stores each value for each stage in the state space. This method is useful when considering a few states and decisions. However, when dealing with a high-dimensional, continuous state space, the limitation called the 'curse of dimensionality' prohibits solution. This kind of DP problem requires proper discretization of the continuous state space and future value function approximation. With advances in computational power, numerical methods have been developed by many researchers to implement the original DP method, and has yielded the ADP family of dynamic programming [15, 26].

One of the most popular methods in ADP is called reinforcement learning (RL) [6], which was originally inspired by the trial and error process in the psychology of animal learning [27-29]. This approach is flexible because it does not need a state transition model to find the optimal policy because it directly maps the states to actions (decisions) [30]. RL performs well in discrete environments of small dimensionality [31]. It mostly focuses exclusively on steady-state problems with sets of discrete decisions [15]. However, sufficient exploration for high-dimensional, complex problems can require extremely large and impractical sample sizes in RL. The state and decision spaces can be provided via sampling [33-37]. In such a case of vary large or continuous problems, RL has to be combined with techniques that allow generalization over an information space (state-action space) [38]. Some parametric function approximators are applied to represent either value functions or policies, including neural network [30, 33] Gaussian stochastic processes [39-40], extreme randomized trees [38, 41], and kernel regression [40, 42].

In the case of an infinite horizon DP problem, the main motivation of this kind of DP is to achieve faster convergence to the steady-state optimal solution and simplify the problem of function approximation. The concept of RL is often applied to solve infinite horizon DP with different types of leaning methods, such as heuristic DP by Werbos [36-37, 43-44], and Tree-based batch mode *Q*-iteration by Ernst et al. [45]. RL is simple because it does not require a transition model. However, it converges slowly, and it may require too many trial processes to learn an optimal control strategy [30]. Bertsekas and Castanon (1989) propose an iterative aggregation approach to solve infinite horizon DP problems [46]. However, this method can cause some errors due to inappropriate aggregation [47]. Some researchers approximate the value function by adding some functions, such as nonlinear programming shape-preservation [9], and basis functions [48]. However, those methods perform well only in low-dimensional state spaces. Powell presents the idea of a post-decision state to solve infinite horizon DP. He defines the post-decision state as the state of the system after making a decision but before new information has arrived. This method is able to solve high-dimensional state and action space with several correlated exogenous processes. The value function is approximated by some methods, such as, linear approximation [4] and piecewise linear approximation [5-6, 49]. However, the limitation of this method is that the function needs to be either linear or concave (convex if minimizing).

A simple and natural way to address a continuous state space is by forming a finite grid of discretization points in the state space and using some approximators such as multilinear or spline interpolation to estimate the value function [50-51]. Unfortunately, the full grid grows exponentially in the dimension of the state space and consequently is not practical for high-dimensional problems. In order to reduce the number of discretization points and the corresponding computational effort, the study by Chen et al.

[52] developes statistical design of experiments (DoE) concept to sample the state space and flexible statistical modeling to approximate the optimal value function over the continuous state space. This approach is analogous to a design and analysis of computer experiments (DACE) [53-54] approach, where the computer experiment is the stage-wise optimization. Orthogonal arrays (OAs), Latin hypercubes, and number theoretic methods (NTMs) have been employed to more efficiently sample the continuous state space of a DP problem [52-56]. Corresponding to the use of efficient discretization methods for the state space, statistical modeling methods, including multivariate adaptive regression splines (MARS) and neural networks (NN) have been applied [52-56]. The existing literature on solving continuous-state DP using DACE have been applied to several finite-horizon problems including a nine-dimensional inventory forecasting problem [16, 55, 57], a 20-dimensional wastewater treatment system [35-36], a 30-dimensional water reservoir [58], and a high-dimensional non-stationary ground level ozone pollution ADP problem [61-62].

A DACE-based approach does not exist in the infinite horizon DP literature. This motivates research to develop a DACE ADP approach that can handle high-dimensional, continuous-state infinite-horizon dynamic programming. According to research proposed by Fan [63] and Sahu [64], a sequential framework for state space discretization together with a statistical modeling tool to estimate the cost-to-go or future value functions can solve finite horizon dynamic problems successfully. This motivates research to develop a sequential DACE ADP approach to approximate an infinite horizon DP. In addition, the second task of this research is motivated by Pilla [65], which used a DACE approach to build a metamodel for the expected value function of the second-stage for solving a two-stage stochastic programming problem.

## 1.4 Research Overview

The ultimate objective of this dissertation research is to develop new methodology based on a statistical perspective that employs a sequential DACE-based approach to approximate an infinite horizon, continuous-state stochastic dynamic programming problem. The proposed method uses DACE to address continuous state spaces by employing experimental design to generate the space for state variables and then for each and use a statistical model to fit these state variables and their responses to provide a continuous approximation of future value function. This research uses NTMs to sample the state space and MARS as statistical tool to model a metamodel of the future value function. A two-sided t-test for 45-degree line correspondence is considered as one of stopping rules of the DP algorithm to check for the convergence. The inventory forecasting problem by Chen [16] is implemented to demonstrate the approach. The results are compared by forward simulation over a finite set of time periods.

The other objective of this dissertation is to develop a two-stage framework that integrates the first and second-stage system for adaptive design for controllability of a system of PHEV charging stations using a DACE approach. In this project, the PHEV charging stations control problem is the second stage. One of the contributions of this dissertation is to formulate the control problem. For preliminary study, the problem is formulated as mean value problem and solved using linear programming. A DACE approach is employed to generate the metamodel of the second-stage given possible first-stage solutions. In this task, DACE is used to replace the loop between first and second stage with a constraint generated from the gradient of the approximation function.

A literature review i presented in Chapter 2, including some methods to solve ADP, such as *Q*-learning, using the post-decision state, and sequential DACE. The proposed method with three different algorithms is presented in Chapter 3. A nine-

dimensional inventory forecasting problem is applied to test the proposed method. Chapter 4 presents batch mode reinforcement learning (RL) using MARS as an approximator. The comparison with the proposed method in Chapter 3 is also presented. The problem of controlling a system PHEV charging stations is presented in Chapter 5, including the formulation of the PHEV charging stations control problem with preliminary results using the mean value problem, followed by using DACE to estimate the expected value function of the control problem (second-stage) in two-stage framework. Conclusions and future work are presented in Chapter 6.

# Chapter 2

## Review of Literature

### 2.1 Modeling Dynamic Programs

The original (deterministic) Bellman's equation can be written as

$$V_t(s_t) = \min_{u_t}\left(c_t(s_t, u_t) + V_{t+1}(s_{t+1})\right) \quad \text{.....(2.1)}$$

where $s_{t+1}$ is the state we transition into if we are currently in state $s_t$, and $u_t$ is the decision taken at stage *t*. This problem is deterministic problem.

The classical form of DP assumes that the state space consists of a finite number of states [44]. It also assumes that $V_{t+1}$ is known and is used to compute $V_t(s_t)$.

Stochastic dynamic programming (SDP) is the version of DP whose goal is to minimize an expected "cost" when the randomness is included in the problem. A typical SDP formulation for a finite-horizon with continuous state is written as

$$\begin{aligned} &\min_{u_1,\ldots,u_T} E\left\{\sum_{t=1}^{T} c_t(s_t, u_t, \varepsilon_t)\right\} \\ &\text{s.t.} \quad s_{t+1} = f(s_t, u_t, \varepsilon_t) \quad \text{for } t = 1,\ldots,T-1 \text{ and} \\ &\qquad (s_t, u_t) \in \Gamma_t \quad \text{for } t = 1,\ldots,T, \end{aligned} \quad \text{.....(2.2)}$$

where $T$ is the time horizon; $s_t$ is the vector of state variables; $u_t$ is the vector of decision variables; $\varepsilon_t$ is the uncertainty; $c_t(.)$ is the known cost function, and $f_t(.)$ is the multivariate transition function used to determine $s_{t+1}$. Constraints on the decision and state are represented by $\Gamma_t$.

*2.1.1 Finite-Horizon Model*

The idea of finite horizon is to determine what to do right now when we model the problem over horizon $T$ [15]. The finite horizon stochastic DP formulation in recursive form is written as,

$$V_t(x_t) = \min_{u_t} E\{c_t(s_t, u_t, \varepsilon_t) + V_{t+1}(s_{t+1})\}$$
$$\text{s.t.} \quad s_{t+1} = g(s_t, u_t, \varepsilon_t) \quad \text{for } t = 1,\ldots,T-1 \text{ and} \qquad \text{.....(2.3)}$$
$$(s_t, u_t) \in \Gamma_t \quad \text{for } t = 1,\ldots,T,$$

The state vector describes the state of the system at the beginning of stage *t*. The decision vector is the variable we can control to minimize present plus future cost. The transition function of the state $g_t(\cdot)$ is assumed to be known. However, in many real problems the transition function is unknown, but it is estimated by some methods, for example, data mining and statistical models [18-22]. The transition function can be the same (stationary) or different for each stage (non-stationary). The future value function or cost-to-go function provides the minimal cost of the system from stage *t* through stage $T$, given the system is in state $s_t$ entering stage *t.* The goal is to find the future value functions $V_t(\cdot)$ and the optimal decision variables $u_t^*$. To solve finite horizon DP, the backward algorithm is applied. The future value function at the last stage $V_T(\cdot)$ is solved first, and we continue to solve backward until $V_1(\cdot)$ is obtained [15].

*2.1.2 Infinite-Horizon Model*

For infinite horizon DP, the problem will be solved over an *infinite* time *horizon*. The infinite horizon problem is of particular interest to address steady-state properties in a Markov process. It studies problems where the parameters of the contribution function, transition function, and the process governing the exogenous information process do not

vary over time, although they may vary in cycles (e.g. an infinite horizon model of energy storage from a solar panel may depend on time of day) [15]. We solve the problem until the system reaches the steady-state equilibrium.

In a version of stochastic infinite-horizon DP, such a problem is usually formulated as a discounted model that takes the long-run return of each stage into account, but the return that is received in the future (or from a previous stage) is usually geometrically discounted according to a discount factor. Discount factors are important in infinite-horizon DP in which they determine how the reward is counted. The natural discount factor for example in a Tetris game problem is 1, since the received rewards have the same importance, independently of when received [64]. But for a problem in which the horizon is long enough to affect the time value of money, a discount factor ( $\gamma$ ), which is a number in between 0 and 1, is introduced to the problem [15, 66].

We can think of a steady-state problem as one without the time dimension, assuming the limit exists. Thus, we have $V(s) = \lim_{t\to\infty} V_t(s_t)$ [15]. The steady-state optimality equations can be expressed as

$$V(s) = \min_{u\in U}\left\{c(s,u,\varepsilon) + \gamma \sum_{s\in S} P(s'|s,u)V(s')\right\} \qquad .....(2.4)$$

where $V(s)$is the total return with state $s \in S$ and decision $u \in U$ . The discount factor, $\gamma$ , is in between 0 and 1.

One also can define the cost-to-go function for the discounted infinite horizon model as

$$V(s) = \min_{u\in U} E\left\{\sum_{t=1}^{\infty} \gamma^t c(s_t, u_t, \varepsilon_t)\right\} \qquad .....(2.5)$$

Thus, an infinite horizon stochastic DP formulation in recursive form is written as,

$$V_t(s_t) = \min_{u_t} E\{c_t(s_t, u_t, \varepsilon_t) + \gamma V_{t+1}(s_{t+1})\}$$
$$\text{s.t.} \quad s_{t+1} = g(s_t, u_t, \varepsilon_t) \quad \text{for } t = 1,2,... \qquad \text{.....(2.6)}$$
$$(s_t, u_t) \in \Gamma_t \quad \text{for } t = 1,2,...$$

The transition function is the same for each stage as it is in a stationary problem. The system seeks steady-state optimality.

## 2.2 Approximate Dynamic Programming

The characteristics of three variables causing the curses of dimensionality include: the state variables, the decision variables, and the exogenous information variables [15]. The original DP approach by Bellman in 1957 is a mathematical programming method for optimizing a system changing over time [1]. However, the solutions are only possible for small problems or under very limiting restrictions (linear dynamics, quadratic cost, Gaussian random variables). Thus it may not be able to mitigate the curses of dimensionality in DP. Puterman [67] shows that the original Bellman's equation can solve the discrete and small state space smoothly. Given recent advances in computational power, approximate dynamic programming (ADP) methods have become a practical method to find good solutions [15, 26]. ADP is both a modeling methodology and an algorithmic framework for solving multi-stage stochastic optimization problems. Most of the literature has focused on the problem of approximating the future value to overcome the problem of multidimensional state variables. Thus, the original stochastic Bellman's equation (1) becomes,

$$\hat{V}_t(s_t) = \min_{u_t}\left(c_t(s_t, u_t, \varepsilon_t) + \hat{V}_{t+1}(s_{t+1})\right) \qquad \text{.....(2.7)}$$

where $\hat{V}_t(s_{t+1})$ is an approximation of the value function at time *t*+1.

Some researchers refer to ADP as neuro-dymamic programming (NDP) and reinforcement learning (RL). Bertsekas and Tsitsiklis [68] address these curses of

dimensionality by introducing an NDP method using artificial neural networks (ANNs) as the function approximator, and solve a discrete state dynamic problem. This method creates algorithms aiming at overcoming two main problems of classical DP algorithms, namely the curse of dimensionality and the transition probabilities requirement. The curse of dimensionality is conquered through use of parameterized function approximators that approximate the value function in a similar way to statistical regression. These algorithms rely on output generated by simulators in their computation, rather than explicit transition probabilities. However, the discussion by Lee and Lee [69] states that NDP is more of an off-line based learning, like reinforcement learning, and it assumes that large amounts of data can be collected from simulation trajectories obtained with "good" suboptimal policies. Thus, this method may be difficult to apply to continuous state variables. They also pointed out some limitations of NDP including error in approximation due to continuous state and decision space, costly on-line learning, and data quantity limitation.

RL is commonly studied in computer science. In RL, there is typically a small number of low-dimensional discrete decisions in deterministic models [15]. Names of RL algorithms depend on their algorithms, such as *Q*-learning and SARSA [15, 30, 35, 37, 38, 40-41].

*2.2.1 Q-Learning*

The main advantage of *Q*-Learning is that we are solving a complex problem that we cannot model but are able to observe behaviors directly. We do not need to know an explicit transition function, so we refer to it as model-free. In this case we may make a decision by observing the results of the decision from a physical process or exogenous process. At each state, we make a decision or action that maximizes the function $Q(s,u)$, which is formulated as

$$Q(s,u) = r(s,u) + \gamma \max_{u'} \left(Q(s',u')\right) \quad \text{.....(2.8)}$$

where $r(s,u)$ is an immediate reward, $\gamma$ is a relative value of delayed vs. immediate rewards (between 0 and 1), $s'$ is the new state after making decision $u$, and $u'$ is an action or decision in state $s'$. The return value of being in a state *s* is

$$V(s) = \max_{u} \left(Q(s,u)\right) \quad \text{.....(2.9)}$$

The original *Q*-Learning performs well when solving problems without a model, but it is difficult to apply to problems with high-dimensional state and action spaces [15]. Moreover, the usual way to observe state-action pairs is by constructing a table [15]. To overcome the over generalization of a state-action space, some parametric function approximators or regression tools are applied [15, 30, 35, 37-38, 40-41, 45]. Thus, equation (5) and (6) become

$$\hat{Q}(s,u) = r(s,u) + \gamma \max_{u'} \left(\hat{Q}(s',u')\right) \quad \text{.....(2.10)}$$

$$\hat{V}(s) = \max_{u} \left(\hat{Q}(s,u)\right) \quad \text{.....(2.11)}$$

Ernst et al. [38, 45] presents tree-based batch mode *Q*-iteration, which uses tree-based supervised learning methods to approximate the $Q$ function. They introduce four-tuples $(s_t, u_t, r_t, s_{t+1})$ and denote $F$ as the set $\left\{\left(s_t^l, u_t^l, r_t^l, s_{t+1}^l\right)\right\}_{l=1}^{|F|}$ of available four-tuples. The sampling process is used to generate $F$. They call this problem the ***batch mode*** RL, as the set of transitions is used to produce the control policy in a single step. The state-action pair is an input variable, and $Q(s,u)$ associated with the pair is an output variable. A training set is used to approximate $\hat{Q}(s,u)$. They compare results of several types of tree-based supervised learning methods, including Classification And

Regression Tree (CART), K-dimensional Tree (Kd-tree), Tree Bagging, Extra-Trees, and Totally Random Trees. The two best results are from using Extra-Trees and Tree Bagging.

*2.2.2 The Post-Decision State Variable*

Powell presents the idea of post-decision [4-6, 15, 49] to solve high-dimensional state and action space with several correlated exogenous processes. He defines post-decision state as the state of the system after making a decision but before new information has arrived and can be expressed as $s_t^u = s^{M,u}(s_t, u_t) = s_t + \Delta_t(u_t)$ where $\Delta_t(s_t)$ is the expected change of the state between *t* and *t*+1, the decision $s_t$. The state before making a decision, sometimes, called *pre-decision state*. The next pre-decision state is $s_{t+1} = s_t^u + \varepsilon_{t+1}(u_t) = s_t + \Delta_t(u_t) + \varepsilon_{t+1}(u_t)$. When making decisions, we use

$$\hat{v}_t(s_t^n) = \min_{u_t}\left(c_t(s_t^n, u_t) + \hat{V}_t^{n-1}(s_t^{u,n})\right), \quad t = 0,\ldots,T-1 \qquad \text{.....(2.12)}$$

where the superscript *n* is the iteration counter and $\hat{v}_t^n(s_t^n)$ is the new observed value for visiting state $s_t^n$ or a sample of the value of being in state $s_t^n$. Moreover, $\hat{v}_t^n(s_t^n)$ is also a sample of the value that put us in state $s_{t-1}^{u,n}$ since the transition from $s_{t-1}^{u,n}$ to $s_t^n$ requires only the realization of random exogenous information. Thus, we can update the estimate of the approximate value function around the post-decision state $\hat{V}_{t-1}^n(s_{t-1}^{u,n})$ with the new observation $\hat{v}_t^n(s_t^n)$, as

$$\hat{V}_{t-1}^n(s_{t-1}^{u,n}) = (1-\alpha_{n-1})\hat{V}_{-1t}^{n-1}(s_{t-1}^{u,n}) + \alpha_{n-1}\hat{v}_t^n(s_t^n), \qquad \text{.....(2.13)}$$

where $\alpha_{n-1}$ is a smoothing factor (stepsize) between 0 and 1.

The approximate value function $\hat{V}$ is obtained by the value function approximations method such as linear approximations [65], piecewise linear approximations [5-6, 49], and cutting planes method [15]. In any case, convexity and concavity must be confirmed [15].

Powell also states in his book [15] that a state-action pair in *Q*-learning is a form of a post-decision state, but a post-decision state is more compact.

2.3 Finite-Horizon with Continuous-State Dynamic Programming Solution Methods

The research for continuous-state DP has focused on methods helping to reduce computational efforts from the curse of dimensionality. Johnson et al. [50] addresses the continuous state space by forming a finite grid of discretization points in the state space and using linear and spline interpolation. They compare numerical solution methods using multilinear, Hermite gradient DP, and tensor-product cubic spline interpolation on a four-reservoir problem. They show that cubic splines required fewer grid levels in each dimension. Thus it reduces computational time. However, their method is based on a full grid points (i.e., a full factorial experimental design), which grows exponentially with the number of dimensions.

Chen [16] and Chen et al. [52] apply experimental design and a statistical model to high-dimensional continuous-state stochastic dynamic programs. The proposed method utilizes orthogonal array (OA) experimental designs and multivariate adaptive regression splines (MARS), where OAs are special subsets of full factorial experimental designs that grow only polynomially with the number of dimensions. This approach uses a statistical perspective that can be seen to be analogous to design and analysis of computer experiments DACE [53-54] and is most appropriate for reducing the computational effort for high-dimensional problems. The results achieve good accuracy compared to using a full factorial design with tensor-product cubic spline interpolation by

Johnson et al. [50]. Cervellera et al. [55] introduced the use of an alternate experimental design, Latin hypercubes, and neural networks, which is another statistical modeling method tool similar to Chen's approach, and obtained comparable results to using OAs and MARS on a nine-dimensional inventory forecasting problem and an eight-dimensional water reservoir problem. Cervellera et al. [55] and Wen [70] study experimental designs number-theoretic methods (NTMs) and successfully solve a thirty-dimensional water reservoir problem. Fan [63] creates an adaptive value function approximation (AVFA) approach using number-theoretic methods (NTMs) with feed-forward neural networks (NNs) to solve finite horizon with continuous-state DP. Lastly, Sahu [64] implements AVFA with MARS. Both Fan and Sahu study the inventory forecasting problem by Chen [16] and utilize the 'sequential' concept from DACE to make the grid 'only fine enough' for the 'efficient' discretization and used MARS methods to approximate future value functions [64].

The concept of RL is popular in ADP research [27-28, 60]. For continuous problems, RL generalizes over a state-action space [38]. Ernst et al. [38, 45] present batch mode RL using a *Q*-iteration algorithm with tree-based supervised learning methods to approximate the *Q*-function [38, 45]. They compare results of several types of approximators including CART, Kd-tree, Tree Bagging, Extra-Trees and Totally Random Trees and found that Extra-Trees and Tree Bagging are the two best, respectively, in terms of accuracy. They do not diverge to infinity, but they do not ensure convergence of the algorithm either. The convergence property is reached by the Totally Randomized Trees, but its performance is not as good as Extra-Trees and Tree Bagging. They also compare their proposed methods with *Q*-learning combined with a piecewise-constant grid and found that the proposed method performed better [38]. Ernst et al. [45] use *batch mode* RL with Extra-Trees to solve the problem of controlling the academic benchmark

electric power system, which is a finite horizon case. They suggest that the proper way to solve this kind of problem is to combine *batch mode* RL with model-based techniques, such as *Model Predictive Control* (MPC) [45]. However, in that case, it may require more computational effort and make a problem more complicated.

Using post-decision state variables is one of the most attractive methods to solve ADP. He et al. [49] use post-decision state variables with pricewise linear approximators to solve for optimal dosage decisions in controlling ovarian hyperstimulation in a finite-horizon DP. The results indicate that this method can obtain policies as good as using *Markov decision processes* (MDP) benchmark in terms of accuracy but with lower computational time. However, this method uses lookup-tables that need to discretize the continuous state space and the state transitions, which may introduce discretization errors.

### 2.4 Infinite-Horizon with Continuous-State Dynamic Programming Solution Methods

The most challenging part of infinite-horizon DP with continuous state space is convergence, since the problem needs to reach the steady-state equilibrium. In brief, the motivation of this kind of dynamic problem is to achieve faster convergence to an optimal solution and to simplify the problem of function approximation. Werbos [43] apply the concept of RL to infinite-horizon DP and proposes a critic structure called "heuristic DP". This method uses the parametric structure called actor (or action network) to approximate the control law, and another parametric structure called critic (or critic network) to approximate the value function. Critic training does not require a system model for its calculations, but actor training, on the other hand, requires finding the derivatives of a system model with respect to the control variables. Thus, in practice, heuristic DP converges slowly. To improve the convergence, Werbos [36-37] proposes an alternative referred to as *Dual Heuristic Programming* (DHP). DHP uses the critic to approximate the

derivatives of the value function with respect to the state variable. The actor is used to approximate the control law, as in all other adaptive critic approaches. It requires fully-specified model based algorithms to train the critic and actor. Later on, Ferrari and Stengel [71] show that the DHP method could be more efficient than heuristic DP. However, due to the use of derivative information, the relationships for updating the control and value-derivative functional are more involved and, hence, introduce additional computation. Furthermore, since the DHP critic approximates a vector functional, the problem of function approximation is more challenging. Werbos [44] present "*Action-Dependent Heuristic DP*" (ADHDP) with *Q*-learning method to ADP approaches. This method does not require a model of the system. In ADHDP/*Q*-learning methods, one can train a critic to match targets of the value function based on the $Q$ recurrence equation, using exactly the same procedure as in heuristic DP. The actor training is simplified, in that since the control variables are inputs to the critic, the derivatives of the value function with respect to the control variables are obtained directly from back-propagation through the critic. Thus ADHDP/*Q*-learning methods use no system models in the training process. Other ADP research based on the RL perspective includes the following. Anderson et al. [72-73] propose an RL method with robust control theory to guarantee the stability even during training. Saeks et al. [74] work with a variety of adaptive critic and adaptive DP implementations where the adaptive critic was developed and apply to a control of a hybrid electric vehicle, which is a real-time application. Si et al. [75] study direct neuro-DP, developed a model-independent to ADHDP approach, and successfully apply it to a wireless network call admission control in a large scale helicopter control problem. The principle disadvantage of RL is that it requires many trials (repeated experiences) to learn an optimal control strategy, especially if the system starts with a poor initial policy [76].

Other ADP research includes the following. Bertsekas and Castanon [46] propose a class of iterative aggregation approaches to solving infinite horizon DP. This method is to interject aggregation iterations in the course of the usual successive approximation method. The principle of aggregation-disaggregation is to approximate by solving a smaller system of equations obtained by lumping together the states of the original system into a smaller set of aggregate states. Zhang and Baras [47] argue that inappropriate aggregation methods can cause large aggregation errors. They derive an adaptive aggregation scheme method to calculate the value function, which guarantees reduction in aggregation errors and increases the speed of convergence [62]. Cai et al. [9] present a *Nonlinear Programming* (NLP) method, called DPNLP, to solve the infinite horizon DP problems. This method uses shape-preserving approximation methods to approximate the optimal value function by adding some extra degree of freedom. DPNLP solves the deterministic or stochastic DP problem with one or two continuous state variables and several continuous control variables without the curse-of-dimensionality of the action space. DPNLP can solve DP problems with many continuous control variables easily and quickly, however, it can only solve one or two continuous state variables. The post-decision state variable approach is another way to solve infinite horizon DP. It uses of a forward algorithm where there is no need to loop though all possible states in the next time step in order to estimate the value of the current state, just like in *Q*-learning. However, it requires approximating the expectation of the value function by visiting the states with enough frequency to make a good decision [77]. Ryzhov and Powell [78] represent uncertainty of the value function using a Bayesian model with correlated belief. In this way, a decision made at a single state can provide with information about many states and make each individual observation more powerful. However, the performance

of the proposed method with an infinite-horizon problem is not very good, especially within an online objective setting.

### 2.5 Statistical Approach to Approximate Dynamic Programming

The statistical approach to ADP uses statistical models to approximate the future value function when the problem is defines as continuous state (or decision) space. Foufoula-Georgiou and Kitanidis [79] use multi-linear interpolation in their proposed multilinear Hermite gradient DP to solve discrete time linearly constrained stochastic optimal control problems. Johnson et al. [50] use tensor-product cubic spline interpolation for solving a four-reservoir problem and show that cubic splines required fewer grid levels in each dimensions, hence it reduces computational time. Chen [16] and Chen et al. [52] observe that the methods presented by Foufoula-Georgiou and Kitanidis [79], and Johnson et al. [50] are based on a full grid of points and statistically equivalent to a full factorial experimental design. Chen et al. [52] proposes an approach based on an Orthogonal Array (OA), which is the special subset of full factorial experimental designs, and used a Multivariate Adaptive Regression Splines (MARS) as a statistical tool to approximate inventory forecasting SDP. Chen also shows that while the number of points in a full grid discretization grows exponentially with the number of dimensions, OAs grows only polynomially with the number of dimensions. This greatly reduces the computational effort for high-dimensional problems. The result achieves good accuracy compared to using a full factorial design with tensor-product cubic spline interpolation by Johnson et al. [50]. Cervellera et al. [58] use an alternative experimental design, number-theoretic methods (NTMs) and successfully solve a thirty-dimensional water reservoir problem. They also conclude that dimensionality issues can be mitigated by employing neural approximation with efficient discretizations of the state space. Cervellera et al. [55] introduce the use of another alternate experimental design, Latin Hypercube (LH), and an

alternate statistical modeling method, neural networks into Chen's [16] approach, and obtaine comparable results to the application of OAs and MARS to a nine-dimensional inventory forecasting problem and an eight-dimensional water reservoir problem. Cervellera et al. [80] propose a methodology called semilocal approximate minimization (SLAM), which introduces a semi-local approach based on kernel functions to approximate the solution of *T*-stage stochastic optimization (TSO) problems, which is a typical paradigm of Markovian decision processes. The approach is characterized by less demanding computational requirements and seeks to exploit the properties of semi-local approximation through kernel models and efficient sampling of the state space. Kelley and Kolstad [10] present an algorithm for infinite horizon models in environmental economics and policy, relying on a neural net approximation of the value function within an iterative version of the Bellman equation. They use "discrete grid methods" to generate the design and approximate the value function using neutral networks. They claim that the algorithm is capable of becoming arbitrarily accurate, unlike the linear-quadratic method where the accuracy is bounded by the accuracy of the approximation of the return function. However, one concern of the method presented by Kelley and Kolstad is the size of state space. Moreover, with a high-dimensional problem, the NN has more observations to fit the underlying function to increase the accuracy, thus, computational time increases in order to estimate parameters over a large data set. Cervellera et al. [81] compare the neural networks approximation with a semi-local approach based on kernel functions (SLK) [50]. The results show that in terms of both accuracy and computational times, SLK is more attractive in higher-dimensional contexts due to its more advantageous computational requirements. Fan [63] successfully solves the nine-dimensional inventory forecasting a finite horizon dynamic programming with continuous-state problem described in Chen [16] by creating an adaptive value function

approximation (AVFA) approach using number-theoretic methods (NTMs) with feed-forward neural networks (NNs). Sahu [64] proposes the same approach with MARS approximation instead of NNs Both studies utilize the 'sequential' concept from Design and Analysis of Computer Experiments (DACE) to make the efficient discretization (fine grid). Sahu's study also tests various guidelines for setting the number of basis functions for a MARS approximation.

## 2.6 Two-Stage Framework

A two-stage framework was first introduced by Dantzig [82] and Beale [83]. In two-stage stochastic programs with recourse, traditionally, a set of decisions have to be made a *priori* when related environmental information is not completely available in the first-stage (or master) problem, called here-and-now decision. These decisions are called the first-stage decisions. Given the first-stage decisions, the later stage decision variables (also called *recourse variables*) can be decided [82-84]. In optimization algorithms, the second-stage decisions (or solution) are sent to the first-stage. At this time we need to solve for the first-stage decisions again along with the second-stage decisions. If the optimality is obtained, then stop, otherwise we need to solve for new second-stage decisions given the new set of the first-stage decisions. The deterministic equivalent of the two-stage stochastic programming problem with recourse can be written as

$$\min_{x \in X} c^T x + E\left[Q(x)\right] \qquad \text{.....(2.14)}$$

where

$$Q(x) = \min_{y \in Y} \left\{ q^T y : Tx + Wy = h,\ y \geq 0 \right\}. \qquad \text{.....(2.15)}$$

Equation (2.14) is the first-stage objective function, which includes the expectation of the second-stage objective function $Q(x)$, where *x* is the first-stage decision vector and *y* is the second-stage decision vector. Note that we consider the deterministic version so the

random events do not occur in the model. Parameters *c* and *q* are the known objective coefficient vector of *x* and *y,* respectively. Parameters *T* and *W* are matrices, specifying the second-stage linear constraints on *x* and *y*, respectively with the right-hand side vector *h*.

One of the most popular methods is based on building an outer linearization of a recourse cost function and a solution of the first-stage (master) problem plus this linearization. It is a cutting plane technique introduced by Benders [85], called the L-shaped or Bender's decomposition method [84]. However, for complicated problems, the iterative cut can be very slow to converge because of the loop between first and second-stage. Moreover, this method works only with the deterministic problem.

Modified versions of the cutting plane method are numerous. Birge and Louveaux [86] propose a multicut algorithm that allows the L-shaped method to be extended to include multiple cuts for the objective in each iteration. The regularized decomposition method due to Ruszczynski [87-88] is developed in such a way that the next solution of the master problem is not too different from the previous first-stage solution. It poses the problem as a non-smooth optimization problem and adds a regularizing term in the objective function, where the initial first-stage solution is the incumbent solution that is updated, as necessary, after finding optimality. Hooker and Ottosson [89] extend the Benders decomposition framework on a mixed-integer linear program so that constraint logic programs can be used as subproblems to generate cuts that are added to a master problem. Hooker [90] uses logic-based Benders decomposition to solve several multi-stage planning and scheduling problems. Trukhanov et al. [91] introduce an adaptive multicut method that generalizes the single cut and multicut methods. The proposed method adjusts the aggregation level of the optimality cuts in the master program.

Among this research, the loop between first-stage and second-stage is still visible. Chen [16] proposes a discretization scheme based on design of experiment technique using orthogonal array design to create a finite set of points of the first-stage decision and use a statistical model to estimate the surface of the recourse function. Pilla [65] and Pilla et al. [92-93] develop a multivariate adaptive regression splines cutting plane approach for solving a two-stage stochastic program to solve airline fleet assignment problem. There are two phases in this method, DACE phase and Optimization Phase. The approximation of the expected recourse function is achieved by MARS approximation over a discretized first-stage decision space based on a Latin hypercube design in DACE Phase. An approximate second-stage recourse function then is optimized using a cutting plane algorithm. Compared with traditional L-shape method, this proposed method performs faster.

In this dissertation, we only focus on building the metamodel to represent the expected value function of the second-stage.

## 2.7 Insight for Research

The original DP approach derived by Bellman [1] is proved to perform well only for small problems or under very limiting restrictions. Given advances in computational power nowadays, APD exists. For continuous state spaces, DP usually requires proper discretization, by forming a finite grid of discretization points of the continuous state space, and uses some methodology to approximate the future value (or cost-to-go) function. However, the full grid grows exponentially in dimension of the state space, hence, it is not practical for high-dimensional problems. Reinforcement learning (RL) ADP is one of the most popular methods in modern study of implementing DP [30, 35-37, 38, 40-41, 43, 45, 71, 72-78]. RL is flexible because it does not require an explicit state transition model. The sufficient exploration of the state and decision spaces may be

achieved via sampling. In contract, to obtain the sufficient exploration for high-dimensional, complex problems can require extremely large and impractical sample sizes. Moreover, trial-and-error is required in RL and may be costly when training the high dimensional data set. Another option for solving infinite horizon ADP is aggregation. However, an inappropriate aggregation may lead to large aggregation errors [46-47]. Lastly, using post-decision state variables does not perform very well for infinite horizon DP [78].

According to studies by Chen [16], Chen et al. [18-20, 57], Cervellera et al. [29, 55, 80-81], Wen [70], Fan et al. [20, 58], Fan [63], and Sahu [64], they all successfully combine sequential DACE approach for future value function approximation to approximate DP. However, these studies focus on finite-horizon DP. Thus, the proposed method in this dissertation will extend the AVFA for infinite horizon ADP for infinite horizon case with continuous state spaces. The proposed method is compared to batch mode Reinforcement Learning (RL) using MARS as an approximator proposed by Ernst et al. [38, 45].

The second major contribution of this dissertation is to use DACE to build metamodels representing the expected value function of the second-stage problem in a two-stage framework for a plug-in hybrid electric vehicle (PHEV) charging stations case study. The case study seeks adaptive system design for controllability, where the first stage optimizes system design and the second stage optimizes system control. According to studies by Pilla [65], and Pilla et al. [92-93], a DACE approach can be successfully used to develop a metamodel of the second stage solution based on an experimental design over possible solutions from the first stage. DACE helps control the execution of time-consuming computer models by replacing the loop between first and second stage with a constraint generated from the gradient of the approximation function.

The final solution will be made at the first stage with the expected value function from the second stage. The first stage solution is not made in this dissertation but the general formulation is presented in Chapter 5.

# Chapter 3

## Using Design and Analysis of Computer Experiments to Approximate Infinite Horizon Dynamic Programming With Continuous State Space

The proposed methodology adopts a sequential approach for state-space exploration inspired by concepts from design and analysis of computer experiments (DACE) by Sacks et al. [53] and the concept of adaptive value function approximation (AVFA) by Fan [62]. The proposed method presented in this section is based on multivariate adaptive regression splines (MARS) modeling to achieve *'statistical parsimony*' in data-driven (adaptive) future value function approximation. The stopping criteria for an infinite horizon dynamic programming is also studied and discussed in this section.

### 3.1 Motivation

Central to dynamic programming (DP) is the 'cost-to-go' or 'future value' function, which is obtained via solving Bellman's equation. ADP is introduced to solve a high-dimensional DP problem, especially with a continuous state space. However, the usefulness of the ADP algorithm is limited by its computational cost. Complexity of the model adds to the computational cost, as does the exploration of the state space. Thus, the usefulness of the whole ADP algorithm hinges on finding the approximation model with optimal complexity using minimum state-space exploration.

*3.1.1 Bias-Variance Tradeoff*

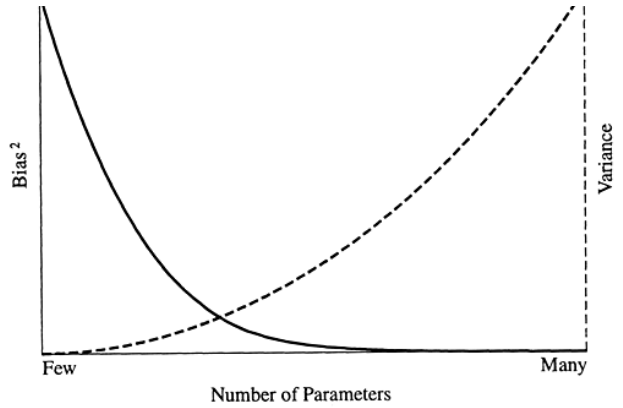



Figure 3.1 Schematic Illustration of Bias-Variance Tradeoff [94]

According to the figure 3.1, it has been observed that the increase in complexity of the model reduces the bias, which measures how good the true function is approximated, but it increases the variance in the predictive performance of the model, which makes the quality of prediction poor in terms of reliability [94]. The aspect of 'model complexity' has traditionally been quantified by the *'degrees of freedom* (df)' measure, which is essentially the number of parameters in the model. In the context of multivariate adaptive regression splines (MARS), the degrees of freedom (df) measure is the number of linearly independent *basic functions* (BF) in the model.

*3.1.2 Generalization Error*

Consider a regression function, $y = f(x) + \varepsilon$, where $\varepsilon$ is the random error independent of $x$ and with mean zero, and $\hat{f}$ is an estimate of the function $f$. Define the loss function for measuring the error of the estimation as

$$L(f, \hat{f}) = \left(f(x) - \hat{f}(x)\right)^2 \qquad \text{.....(3.1)}$$

The *generalization error* is defined as the expected loss or expected prediction error over an independent test sample. This is also referred to as *test error*. This expectation

averages anything that is random including the randomness in the training sample that produced $\hat{f}$ . Specifically, the test error is given by the following over an independent test sample.

$$E\left[L(f,\hat{f})\right]=E\left[\left(f(x)-\hat{f}(x)\right)^2\right]=E\left[\left(y-\hat{f}(x)\right)^2\right] \quad \text{.....(3.2)}$$

The *training error* is defined as the average loss or average training error over the training sample, given by the following.

$$\frac{1}{N}\sum_{i=1}^{N}L(f,\hat{f})=\frac{1}{N}\sum_{i=1}^{N}L\left(f(x_i),\hat{f}(x_i)\right)=\frac{1}{N}\sum_{i=1}^{N}\left(y_i-\hat{f}(x_i)\right)^2 \quad \text{.....(3.3)}$$

The generalization error can be decomposed into bias and variance, such that *Test Error* = *Bias*$^2$ + *Variance*, but it is untrue in the case of training error [94]. If we continue to increase the model complexity, the training error decreases monotonously, but not the test error.

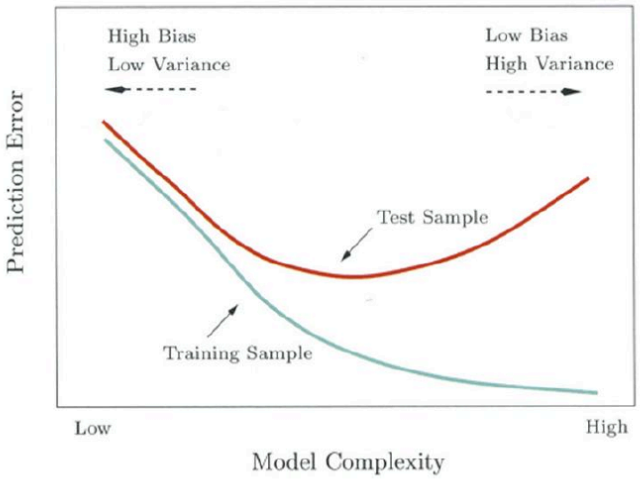



Figure 3.2: Illustration of Interplay between Training Error And Test Error

Source: T. Hastie, R. Tibshirani , J. H. Friedman [94]

When we increase the model complexity too much, the model adapts too closely to the training data and loses the *generalization ability*, and the test error increases in the independent testing data. When the model becomes complex, it can capture more complex underlying structure in the data, which decreases the bias of the estimating function, but it also increases the variance of the estimating function. As long as the decrease in the bias component more than compensates for the increase in the variance component, the estimation error in the testing data set decreases. Beyond certain model complexity, the rate of decrease in bias is more than offset by the rate of increase in variance, and the estimation error increases in the training dataset. In the continuum of model complexity, there is an optimal model complexity that gives the minimum test error.

This research follows the successful study by Fan [63] to seek to incorporate the test error in the model selection criterion in developing a data-driven algorithm to build flexible statistical models of optimal complexity.

*3.1.3 Consistency Trace*

*Consistency* is defined as the asymptotic convergence of an estimator to the object of estimation. Analytical studies reveal that most nonparametric regression algorithms, which include multivariate adaptive regression splines (MARS), are consistent for approximating any regression function and the rate of convergence, depending on the particular algorithm and the underlying function it seeks to approximate [96]. This is a reassuring property of the MARS approximation that we seek to exploit. The observation of Fan [63] that the nonparametric regression function approximations follow a *consistency trace* is of practical significance.

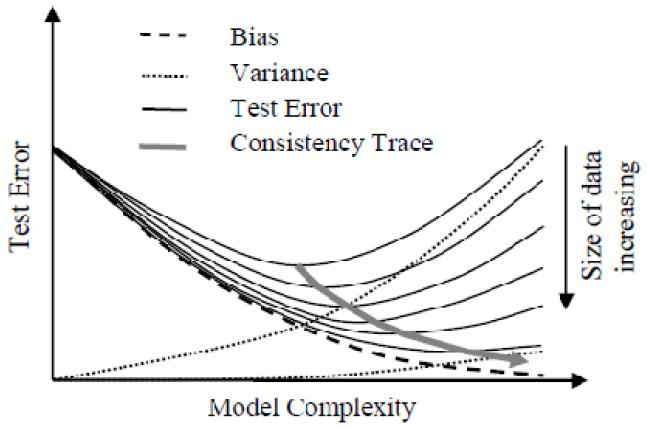



Figure 3.3: Consistency Trace for Adaptive Value Function Approximation

Source: Fan [63]

From figure 3.3, the adaptive value function approximation algorithm seeks to slowly grow both the data sample and the approximation model complexity, to follow a consistent trace. It also shows that the increase in model complexity and the size of training data can force the regression function approximation models to follow the consistency trace. This research seeks to exploit this observation regarding the consistency trace to develop data-driven stopping rules of the data sample incrementing for optimal model complexity.

## 3.2 Multivariate Adaptive Regression Splines (MARS)

MARS, introduced by Friedman [95], yields an adaptive continuous approximation that does not impose any structural assumption on the data. It fits basis functions composed of single or products of truncated linear functions, with optional smoothing, using linear least squares estimation. MARS models can capture complex nonlinearity in the data and provide a data-driven and adaptive modeling method for approximation of the future value functions. MARS is flexible and can automatically

model interactions between variables. Its disadvantage is it does well with only quantitative predictor variables.

MARS is an *adaptive* procedure because the selection of basis functions is data-based and specific to the problem at hand. The MARS procedure for estimating any arbitrary regression function consists of a forward stepwise algorithm to select certain spline basis functions followed by a backward stepwise algorithm to delete basis functions until the best set of basis functions is found that has the lowest *generalized cross validation* (GCV) error among all the possible sets of basis functions. The MARS forward stepwise algorithm is used to create the basis functions of the MARS model. The forward stepwise algorithm loops through the possible choices for basis functions, composed of covariates and *knot* locations, in selecting the next two basis functions to add to the model. The forward stepwise algorithm stops when *M*max basis functions have been selected, where *M*max is a user-specified constant. The MARS approximation approaches interpolation as the number of basis functions increases, but there is a tradeoff between *M*max and computational time. An improper *M*max may cause problems of over-fitting or under-fitting. In general, the backward stepwise algorithm is used to prevent over-fitting by decreasing the complexity of the model without degrading the fit to the data. However, the backward algorithm is computationally intensive, and precious computational effort cannot be wasted within an DP setting. Consequently, Tsai and Chen [97] presented automatic stopping rules to provide an efficient way of choosing *M*max in approximating the future value function. Sahu [64] has provided recommendations on setting a proper *M*max.

*3.2.1 Structure of MARS*

The MARS approximation has the form

$$\hat{f}(x) = \beta_0 + \sum_{m=1}^{M} \beta_m h_m(x), \qquad \text{.....(3.4)}$$

where $x$ is a $v$-variate vector of predictors, $h_m(x)$ is a basis function, $M$ is the number of linearly independent basis functions, and $\beta_m$ is the unknown coefficient for the $m$-th basis function $h_m(x)$. The basis function $h_m(x)$ has the form

$$h_m(x) = \prod_{l}^{L_m} \left[ s_{l,k} \left( x_{v(l,m)} - k_{l,m} \right) \right] \qquad \text{.....(3.5)}$$

where $\left[z\right]_+$ is the hinge function defined as max{0, $z$}, $L_m$ is the order of predictor-predictor interaction in the $m$-th basis function, $x_{v(l,m)}$ is the input variable corresponding to the $l$-th hinge function in the $m$-th basis function, $k_{l,m}$ is the knot value corresponding to the $x_{v(l,m)}$, and $s_{l,m}$ takes the values -1 and +1 corresponding to the pair of basis functions for the each combination of $x_{v(l,m)}$ and $k_{l,m}$. The forward stepwise algorithm in MARS adds basis functions in pairs in each iteration and loops through the possible choices for basis functions ($m$), covariates ($v$) and the knot locations ($k$) to select the next two basis functions to add to the model. The process stops when *M*max is reached.

3.3 General Framework for Solving Continuous Stochastic Dynamic Programming

A typical SDP formulation for an infinite horizon DP is

$$\begin{aligned} & V\left(s_t\right) = \min_{u_t} E_\varepsilon \left\{ c\left(s_t, u_t, \varepsilon_t\right) + \gamma V\left(s_{t-1}\right) \right\} \\ s.t. \quad & s_t = f\left(s_{t-1}, u_t, \varepsilon_t\right), \quad \text{for } t = 1,2,... \\ & u_t \in \Gamma_t, \quad \text{for } t = 1,2,... \end{aligned} \qquad \text{.....(3.6)}$$

where $t$ is the stage (time period).

$s_t$ is the vector of state variables.

$u_t$ is the vector of decision variables.

$\varepsilon_t$ is the uncertain influences.

$c_t(.)$ is the known cost function.

$f_t(.)$ is the transition function.

$\Gamma_t$ is the set of constraints.

$\gamma$ is the discount factor.

$V(.)$ is the future value function.

The following value function iteration approach is one of the most widely used algorithms in dynamic programming, which is solved as a forward DP and seeks a steady-state or infinite-horizon solution [15]. The future value function $V(.)$ represents the minimal cost of operation at each stage $t$ given the system is in state $s_t$ and entering stage $t$. The goal at each stage is to compute the future value function $V(.)$ and find the optimal decision $u_t^*$. The goal of the problem (an infinite horizon DP) is find the future value function at steady-state equilibrium solution. The stopping criteria of DP stage iteration or time period iteration is presented in the next section. The DP is solved forward recursively from the first DP stage iteration until reaching the steady-state set point. At the end of this exercise, the value function $V_k(.)$ and the optimal policy $u_k^*$ at the last iteration (steady-state), $k$, become the solution of problem.

This traditional recursive solution framework for SDP becomes computationally expensive as the number of state points for each stages increases. In the presence of continuous state variables, this traditional recursive framework completely breaks down.

Discretization of the state space coupled with some interpolation technique is adopted to approximate the future value function for any state point. The algorithm for solving an infinite horizon continuous SDP problem is as follows:

Step 0: Initialization:

- Choose $N$ discretization points in the state space $\left\{s_{i,t}\right\}_{i=1}^{N}$ for the $t$-th stage; where $t = 1,2,...$ and $s_{i,t} \in R^n$.
- Set the initial estimated value function $\hat{V}_0$=0.
- Assume the discount factor $\gamma \in [0,1]$.
- Set $k$=0.
- Set *t*=1

Step 1: The stage of an infinite horizon DP:

- Set $k \leftarrow k+1$.
- For each discretization point $s_{i,t}\{i = 1..N\}$ solve for $V_k\left(s_{i,t}\right) = \min_{u_{i,t}} E\left\{C\left(s_{i,t},u_{i,t},\varepsilon_{i,t}\right)\right\} + \gamma \hat{V}_{k-1}\left(f\left(s_{i,t-1},u_{i,t-1},\varepsilon_{i,t-1}\right)\right)$ where $k$ is the $k$-th iteration.
- Approximate $V_k\left(s_t\right)$ with $\hat{V}_k\left(s_t\right)$ for all $s_t \in R^n$.

Step 2: The stopping condition:

- Check the convergence of $\hat{V}_k\left(s_t\right)$with the criteria discussed in section 3.5. If fail, set $t = t+1$ and go back to Step 1.

This discretization of the state space makes the recursive solution framework work for continuous-state SDP. The approximation or interpolation is required to de-discretize any inference over the state space. The quality of the approximation is enhanced with the

increase in fineness of the discretization grid. However, the computational cost of the recursive solution process increases with the increase in the number of discretization points. This tradeoff between the quality of approximation and the cost associated with it has motivated this research to seek ways to find the optimal discretization that would give the optimal quality of approximation. The use of sequential DACE in the finite-horizon ADP work of Fan [63] and Sahu [64] to create an adaptive value function approximation algorithm facilitates the creation of sequential DACE approach for infinite-horizon ADP. The research in this dissertation follows the work of Fan [63] and Sahu [64] to employ a sequential DACE approach for approximating the future value function for an infinite horizon dynamic program to achieve the goal of optimal discretization and optimal quality of approximation.

## 3.4 Proposed Framework

The objective of this research is to develop an algorithm that uses a sequential DACE approach to approximate the future value function of an infinite horizon dynamic program that follows the consistency trace. Based on Chen et al. [52] and Sahu's [64] research, MARS is the nonparametric statistical modeling method of choice because of its consistency. The *M*max guideline recommended by Sahu is used in this research, which is *M*max equal to $\left\lfloor \frac{2n+c}{2+c} \right\rfloor$, where $n$ is the number of data points in the training data set, and $c$ is the penalty parameter set to the default value of 3.

The method in this dissertation implements researches by Chen et al. [52], Fan [63] and Sahu [64] which all focus on solving finite horizon DP with continuous-state by developing an AVFA for infinite horizon ADP algorithm. There are two main components in this algorithm. The first one is *DP stage iteration*, which is the outer loop of the algorithm. The objective of this loop is to achieve an acceptable approximation of the

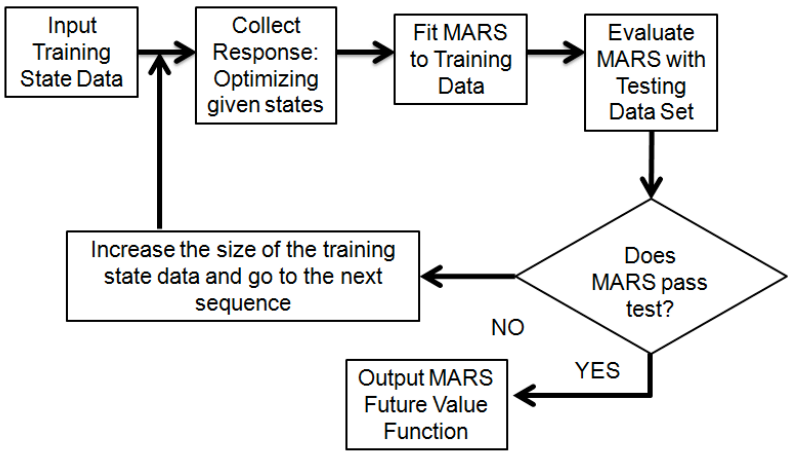
steady-state optimal value function. The second component is the *Data loop*, which conducts computer experiments to gain information for building the value function approximation using the concept of DACE.

In the Data loop presented in Figure 3.4 (a), the proposed method uses a sequential DACE approach for future value function approximation [8, 64]. The computer model is run at sample state points determined by the experimental design, and the output responses are used to fit the *metamodel* that approximates the future value function. In DACE metamodeling, sample points can be generated in batch, and a statistical model is constructed based on the whole batch. In a sequential DACE approach, sample points are selected sequentially; the metamodel is updated sequentially with new sample points; the performance of the metamodel is evaluated each time the metamodel is updated, and the sampling process is stopped as soon as the performance of the metamodel meets the set stopping criteria.

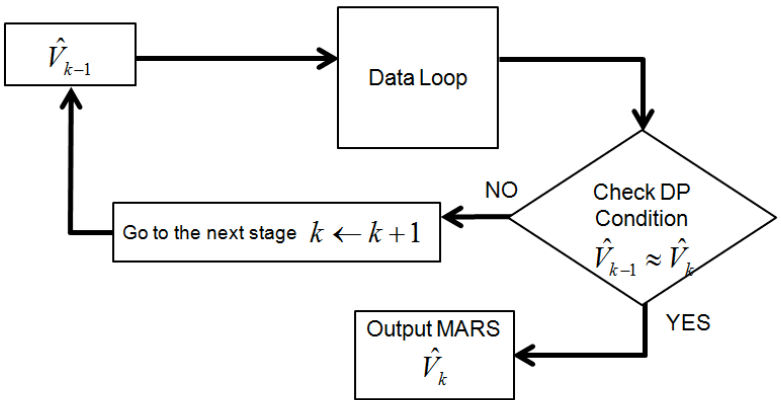
*3.4.1 Flowchart of the Proposed Framework*

At each iteration of the DP, the Data loop runs inside the DP stage iteration loop presented in Figure 3.4 (b). The algorithm of the Data loop starts with the input of the initial size of the training data taken from a low-discrepancy quasi-random sequence and fits a model of optimal complexity for the size of the training dataset. At the end of the sequential step, the model with optimal complexity is evaluated in a fixed testing dataset, and stopping criteria of the Data loop is checked for compliance. The change in $R^2$ in the test response is used as the stopping criterion. If the stopping criterion has not been met, then the algorithm proceeds to the next sequential step with a pre-specified increase in the size of the training dataset. The sequential data iterations continue until the stopping criterion of the Data loop is met. After that, the stopping criterion of the DP stage iteration

is then checked. The stopping rule of the DP stage iteration is subjected to the convergence of the value function and will be present in the next section.




(a)




(b)

Figure 3.4: Flowchart of Proposed Sequential Algorithm (a) Data Loop (b) DP Stage Iteration

## 3.5 DP stage iteration Stopping Condition

There is no limit time period or stage in a case of an infinite horizon DP. The solution of this kind of problem is obtained when it reaches steady-state equilibrium point or convergence of the value function. In order to ensure the convergence, we need a setting rule. In this dissertation, three different stopping rules are discussed as follows.

### *3.5.1 L-Infinity Norm*

The stopping conditions are required to decide at which DP stage iteration (i.e., for which value of $k$ ) the process can be stopped. A simple way to stop the process is to define *a priori* a maximum number of iterations [15, 26, 45]. According to Powell [15], the convergence criterion is defined by

$$\left\|V_k - V_{k-1}\right\| < \phi(1-\gamma)/2\gamma \qquad \text{.....(3.7)}$$

where $\left\|V\right\|$ is the max-norm defined by

$$\left\|V\right\| = \max_s \left|V(s)\right| \qquad \text{.....(3.8)}$$

Thus, the stopping criteria is reached when the maximum change in the value of being in any state is lower than setting right-hand side of the equation (3.7), where $\gamma$ is the discount factor, and $\phi$ is a specified error tolerance.

### *3.5.2 Change in the Future Value Function Average*

In systems theory, a system in a steady-state has numerous properties that are unchanging in time. In this case, the simple way to check is the change in the future value function average [15, 98-100]. This change should be very small. However, in a high-dimensional problem, this condition alone may not be adequate because it considers only the average value, which is only a measure of central tendency [101]. The change in the future value function average can be stated as:

$$\frac{\left|\bar{V}_t - \bar{V}_{t-1}\right|}{\bar{V}_{t-1}} \qquad \text{.....(3.9)}$$

Where $\bar{V}_t$ is the future value function average at time *t*.

3.5.3 *Confidence Interval T-test for 45-Degree Line Correspondence* [102]

Correlation analysis can be used to ensure the convergence of the future value functions. A linear regression model with one predictor variable is used to find the relation between the future value function approximation of two different stages. The future value function approximation of the previous DP stage iteration is assigned as the predictor. The future value function of the current DP stage iteration is put as the response. The general model of linear regression can be stated as follows:

$$Y_i = \beta_0 + \beta_1 X_i + \varepsilon_i \qquad \text{.....(3.10)}$$

where

$Y_i$ is the value of the response variable in the $i$ th trial.

$\beta_0$ is the $Y$ intercept of the regression line.

$\beta_1$ is the slope of the regression line.

$X_i$ is the value of the predictor variable in the $i$ th trial.

$\varepsilon_i$ is a random error term in the $i$ th trial.

The objective in simple linear regression is to generate the best line between the two variables. Then the slope and intercept are estimated. The best line, or fitted line, is the one that minimizes the distances of the points from the line, as shown in Figure 3.5. The fitted regression line can show the actual ratio for the correspondence between predictor and response. A one-to-one correspondence means that the value of the predictor gives about the same results as the response. The line of one-to-one correspondence should

make a 45-degree angle through the origin, and the formula for this line would have a slope coefficient of 1 and a *y*-intercept or constant term equal to 0 (passing the origin). We refer to this line as *45-degree line correspondence.*

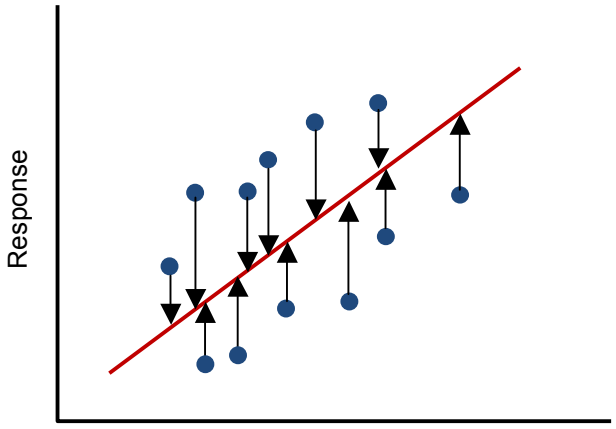



Figure 3.5: Example of Functional Relation

A two-sided t-test is conducted with a 95 percent confidence interval for both $\beta_1$ and $\beta_0$.

The t-test for $\beta_1$ has two alternatives:

$$H_0 : \beta_1 = 1$$
$$H_1 : \beta_1 \neq 1 \qquad \text{.....(3.11)}$$

An explicit test of the alternatives (3.11) is based in the test statistic:

$$t^* = \frac{b_1 - 1}{s\{b_1\}} \qquad \text{.....(3.12)}$$

where $b_1$ is a point estimators of $\beta_1$, which is calculated by (3.13), and $s\{b_1\}$ is a standard error estimator from (3.14)

$$b_1 = \frac{\sum (X_i - \bar{X})(Y_i - \bar{Y})}{\sum (X_i - \bar{X})^2} \qquad \text{.....(3.13)}$$

$$s\{b_1\} = \sqrt{\frac{MSE}{\sum (X_i - \bar{X})^2}} \qquad \text{.....(3.14)}$$

The decision rule with this test statistic for controlling the level of significance at $\alpha$ is:

$$\text{if } |t^*| \le t(1-\alpha/2; n-2), \text{conclude } H_0 \qquad \text{.....(3.15)}$$

$$\text{if } |t^*| > t(1-\alpha/2; n-2), \text{conclude } H_1 \qquad \text{.....(3.16)}$$

where $n$ is the size of the data set.

The 1-$\alpha$ confidence limits for $\beta_1$ are:

$$b_1 \pm t(1-\alpha 2/; n-2)s\{b_1\} \qquad \text{.....(3.17)}$$

where $t(1-\alpha 2/; n-2)$ denotes the $(\alpha/2)100$ percentile of the $t$ distribution with $n$-2 degrees of freedom. The t-test for $\beta_0$ has two alternatives:

$$\begin{aligned} H_0 &: \beta_0 = 0 \\ H_1 &: \beta_0 \neq 0 \end{aligned} \qquad \text{.....(3.18)}$$

An explicit test of the alternatives (3.18) is based in the test statistic:

$$t^* = \frac{b_0}{s\{b_0\}} \qquad \text{.....(3.19)}$$

where $b_0$ is a point estimators of $\beta_0$, which is calculated by (3.20), and $s\{b_0\}$ is an estimator for the standard error, which is from (3.21)

$$b_0 = \frac{1}{n}\left(\sum Y_i - b_1 \sum X_i\right) = \bar{Y} - b_1 \bar{X} \qquad \text{.....(3.20)}$$

$$s\{b_0\} = \sqrt{MSE\left[\frac{1}{n} + \frac{\bar{X}^2}{\sum (X_i - \bar{X})^2}\right]} \qquad \text{.....(3.21)}$$

The decision rule with this test statistic for controlling the level of significance at $\alpha$ is the same as (3.15) and (3.16).

The 1-$\alpha$ confidence limits for $\beta_0$ are:

$$b_0 \pm t(1-\alpha 2/;n-2)s\{b_0\} \quad \text{.....(3.22)}$$

This research developed alternative stopping rules for the value iteration approach. The condition of the change in the future value function, and the 95% confidence interval t-test for 45-degree line correspondence are used as the stopping criteria. More discussion on why these conditions can be set as a stopping rule is presented in the next section. The comparison with the L-infinity norm stopping rule and the proposed stopping rules is shown in Appendix A.

## 3.6 Application to Inventory Forecasting Problem

This research seeks to study the performance of the resulting sequential ADP methodology based on MARS modeling on an infinite horizon nine-dimensional stochastic inventory forecasting problem [16]. The choice of this particular SDP problem is for easy comparison and benchmarking, as this nine-dimensional stochastic inventory forecasting problem has been extensively studied and frequently used for performance comparison by researchers in the past [16, 63-64].

### *3.6.1 Overview of Stochastic Inventory Forecasting Problem*

The nine-dimensional stochastic inventory forecasting problem [16] is concerned with optimal order quantities for three items over two forecast periods given the inventory level and demand forecast for each item.

The state variables include:

- The inventory level of item $i$ at the beginning of time period $t$, $\left(I_t^{(i)}\right)$.

- The forecast determined at the beginning of time period $t$ to predict the demand of item $i$ in the current time period $t$, $\left(D_{(t,t)}^{(i)}\right)$.
- The forecast determined at the beginning of time period $t$ to predict the demand of item $i$ in next time period $t+1$, $\left(D_{(t,t+1)}^{(i)}\right)$.

The state vector at the beginning of stage is represented by

$$x_t = \left(I_t^{(1)}, I_t^{(2)}, I_t^{(3)}, D_{(t,t)}^{(1)}, D_{(t,t)}^{(2)}, D_{(t,t)}^{(3)}, D_{(t,t+1)}^{(1)}, D_{(t,t+1)}^{(2)}, D_{(t,t+1)}^{(3)}\right)^T$$

The decision variables are the amounts of item $i$ ordered in period $t$, the decision vector at the beginning of stage is represented by

$$\boldsymbol{u}_t = \left(u_t^{(1)}, u_t^{(2)}, u_t^{(3)}\right)$$

The transition functions are modeled using the multiplicative Martingale model of forecast evolution (See Chen [16] for details). The constraints on the decision variables (amounts ordered) and the state variables (inventory levels) are placed in the form of capacity constraints. The transition functions are given by

$$\begin{aligned} I_{t+1}^{(i)} &= I_t^{(i)} + u_t^{(i)} - (D_{(t,t)}^{(i)} \cdot \varepsilon_{(t,t)}^{(i)}), \\ D_{(t+1,t+1)}^{(i)} &= (D_{(t,t+1)}^{(i)} \cdot \varepsilon_{(t,t+1)}^{(i)}), \\ D_{(t+1,t+2)}^{(i)} &= (\mu_{t+2}^{(i)} \cdot \varepsilon_{(t,t+2)}^{(i)}) \end{aligned} \qquad \text{.....(3.23)}$$

where $\mu_t^{(i)}$ is the mean demand for item $i$ in period $t$, and $\varepsilon_{(t,t+k)}^{(i)}$ is the change in forecast for the time $t+k$ from the forecast made in period $t$.

The objective function is a cost function involving inventory holding costs and backorder costs. The cost function is typically V-shaped and is represented as

$$c_v(x_t, u_t) = \sum_{i=1}^{3} \left(h_i \left[I_{t+1}^{(i)}\right]_+ + \pi_i \left[-I_{t+1}^{(i)}\right]_+\right) \qquad \text{.....(3.24)}$$

where $h_i$ is the holding cost parameter for item $i$, and $\pi_i$ is the backorder cost parameter for item $i$. For optimization purposes, a smoothed version of the cost function has been used (See Chen [16] for details).

*3.6.2 Computational Setup*

MATLAB software was used to code the algorithm along with "ARESLab: Adaptive Regression Splines toolbox for MATLAB" developed by Jekabsons [103]. The experiments are run on Windows 7 Home Premium with Intel(R) Core(TM) i7 @ 2.80 GHz 8.00 GB RAM system. The fmincon function from Matlab is used for optimization.

For any stage iteration, the training data used to build the future value function approximation models came from a nine-dimensional Sobol' sequence [104]. As in the work by Fan [63] and Sahu [64], the sampling starts at 50 points and increments 50 more points for each iteration in the Data loop until the stopping criteria of the Data loop is met. The testing data consisted of 250 design points from a nine-dimensional Halton sequence [105].

After obtaining the output at the steady-state point, the result is then tested in a forward simulation. There are three algorithms proposed in this dissertation, which will be tested in the same simulation for comparison.

## 3.7 Proposed Algorithms

*3.7.1 Algorithm-I*

Algorithm-I uses the original MARS algorithm presented by Friedman [95] with *M*max recommended by Sahu [64]. The stopping rule for the Data loop is the change in $R^2$ when adding more states to the sample space. The change in $R^2$ must be lower than or equal to 0.005. The stopping rules for DP stage iteration use the T-test for 45-degree line and a percent change of the future value function average of 5% or lower, where the average is calculated over the 250 test values of the approximated future value function

3.7.1.2 Result and discussion

The CPU time of Algorithm-I is 19,971 seconds (5.55 hours). DP stage iteration stops at DP stage iteration 25, where the percent change of the future value function average is 0.5%, and it passes a two-sided t-test for 45-degree line correspondence. The result in Figure 3.5 shows the number of training data points at each DP stage iteration. The average number of training data points is 250. According to Figure 3.6, the DP stage iterations 2 and 5 required 700 and 400 training data points to meet the criteria of the Data loop, respectively. As a result, they require more elapsed time compared with other DP stage iterations.

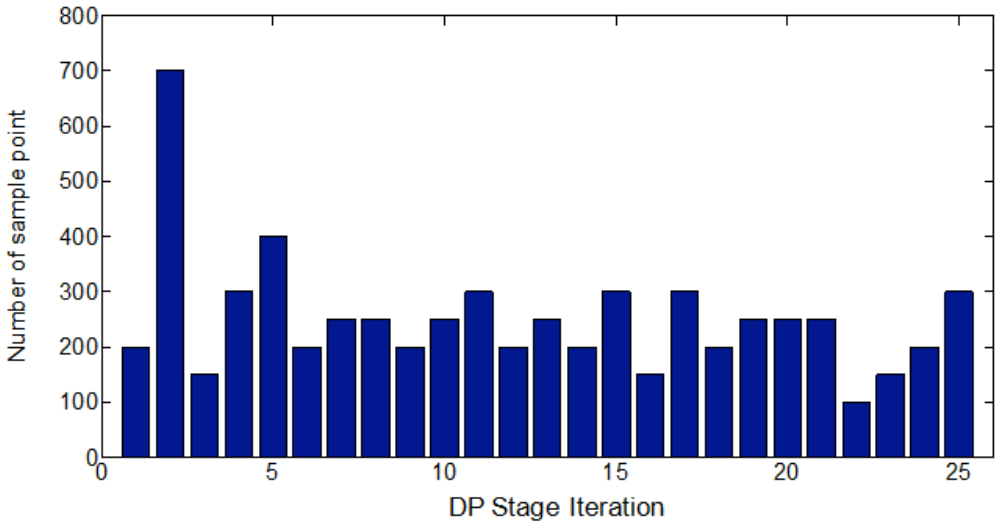



Figure 3.6: Number of Sample Points in Each DP Stage Iteration of Algorithm-I

Figure 3.7 shows the $R^2$ of the testing data set in each DP stage iteration of Algorithm-I. Overall, $R^2$ values of the testing data set are more than 0.9, except the one in stages 3, 4 and 5. Figure 3.8 presents the percent change in the future value function average, which is one of the DP stage iteration stopping criteria. The stopping rule is when the percent change in the future value function average is lower than or equal to

5%. From Figure 3.8, at the DP stage iteration 5, the percent change in the future value function average is dropped to 4%. If the stopping rule of the DP stage iteration were the percent change in future value function average alone, the process would stop at this point. However, Figure 3.7 shows that at the DP stage iteration 5, the $R^2$ is 0.70, which is the lowest $R^2$ of the system. The result in Figure 3.9 presents the $R^2$ from the final future value function at the steady-state (true value), which is calculated by equation (3.26). This figure measures how close the future value functions at each stage are to the true value function at the steady-state equilibrium. The plot shows the trend of the convergence when incrementing DP stage iteration, the fraction of variance explained by the true value is closer to 1. However, at DP stage iteration 5, the $R^2$ from the true value function does not follow the trend.

$$MSE_{from\ true} = \sum_{i=1}^{N} \frac{\left(\hat{Y}_i - Y_i^{ture}\right)}{N} \qquad \text{.....(3.25)}$$

$$R^2 = 1 - \frac{MSE_{from\ true}}{\text{Variance}_{\text{true}}} \qquad \text{.....(3.26)}$$

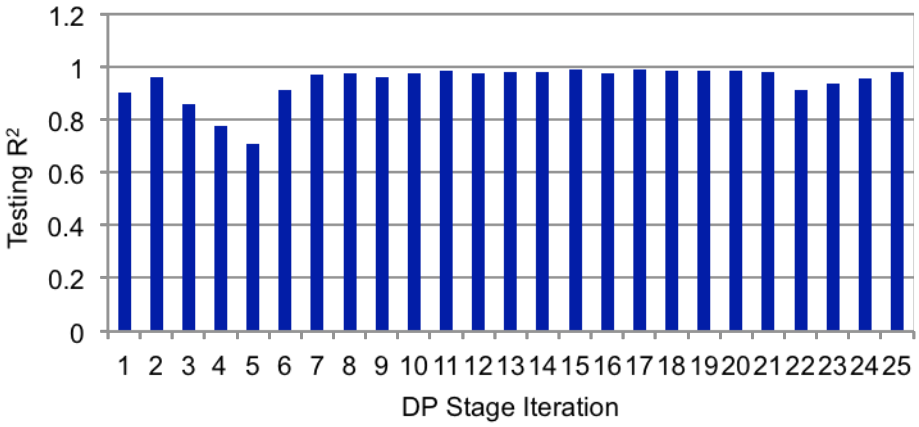



Figure 3.7: $R^2$ of the Testing Data Set in Each DP Stage Iteration of the Algorithm-I

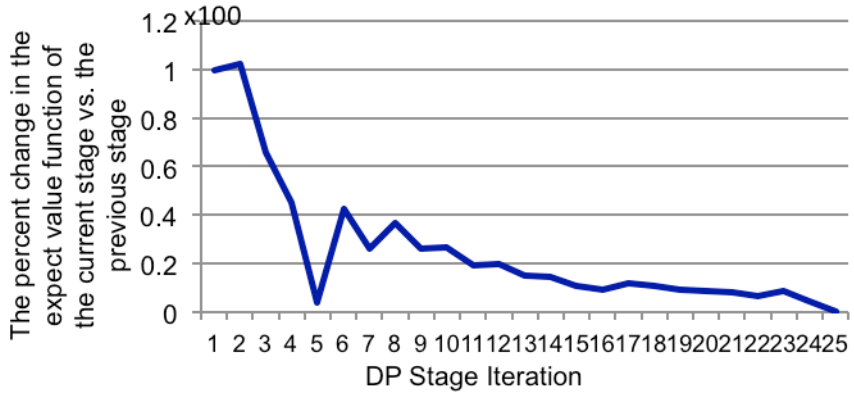



Figure 3.8: The Percent Change in the Future Value Function Average at Each DP Stage Iteration of Algorithm-I

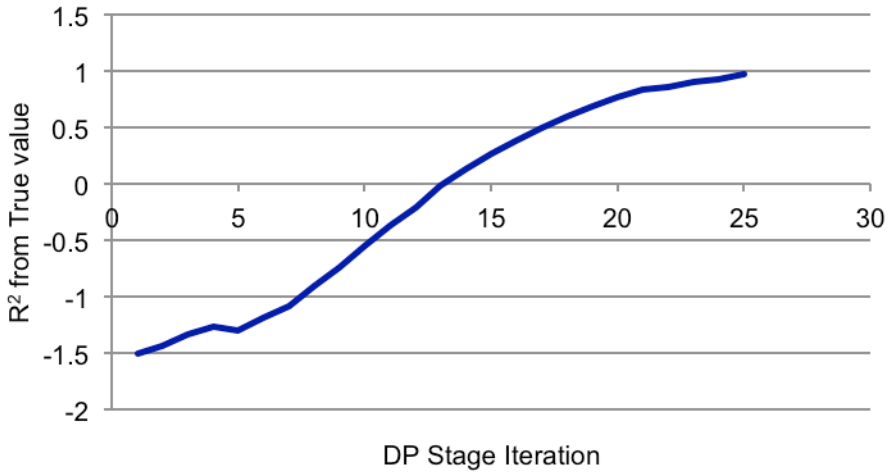
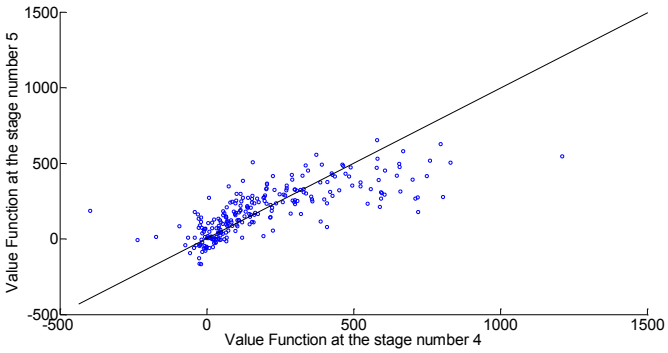


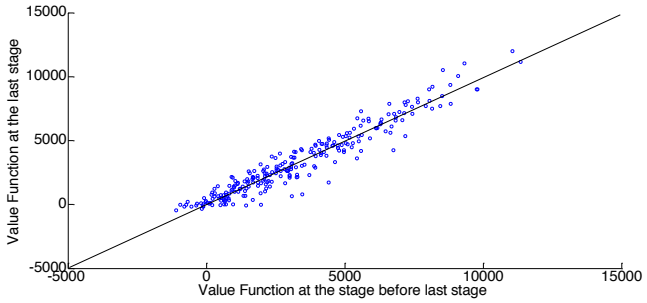

Figure 3.9: The $R^2$ of the Future Value Function in Each DP Stage Iteration vs. the Final Value Function at the Steady-state of the Algorithm-I

Figure 3.10(a) shows the scatter plot of the value function at stage 4 vs. the value function at stage 5. From the plot it shows no sign of convergence, and there are some outliers. On the other hand, Figure 3.10(b) shows the scatter plot of the value function at the steady-state vs. the value function at the stage before it. The plot follows the 45-degree line correspondence. It can be concluded from the 95 percent confidence

interval t-test that the scatter plot of the future value functions at DP stage iteration 24 and DP stage iteration 25 are following the 45-degree line correspondence.




(a)




(b)

Figure 3.10: (a) The Scatter Plot of the Value Function at DP Stage Iteration 4 vs. 5. (b) The Scatter Plot of the Future Value Function at the Steady-State vs. the Future Value Function at the DP Stage Iteration before

Figures 3.7-3.10 show that stage 2 requires more training data points, but the percent change in the future value function average of that DP stage iteration is very bad. Moreover, $R^2$ from the true value final is very high, which indicates that the future value function at this DP stage iteration is very far away from the true future value function.

Moreover, the percent change in the future value function average alone is not enough to confirm convergence.

*3.7.2 Algorithm-II*

Algorithm-II is based upon the performance results of Algorithm-I. From the results of Algorithm-I, DP stage iteration 2 requires a high number of training data points, which means that it requires a long time to reach the Data loop stopping criteria. However, the results show that the future value function at that stage is very far away from the future value function at the steady-state equilibrium point. These results suggested a new stopping rule for the Data Loop as follows:

> Data loop stopping criteria is *minimum* $R^2$ of 0.8 and either *minimum change of the future value function average* of 10% or *maximum change in* $R^2$ of 0.005.

We want an accurate approximation, so we need to consider a high value of $R^2$, which indicates a good fit, and in this case 0.8 is used at a minimum. From Figures 3.8-3.9, the $R^2$ from the true value is good when the DP stage iteration is more than 20, where change of the value function is below 10%. Thus, we considered a *minimum change of the value function* of 10% as one of the conditions of the Data loop.

3.7.2.1 Results

The CPU time of Algorithm-II is 19,243 seconds (5.35 hours). DP stage iteration stops at DP stage iteration 98. The average number of training data points at each stage is 161. The maximum number of training data points is 400 and occurs at stage 19, as shown in Figure 3.10.

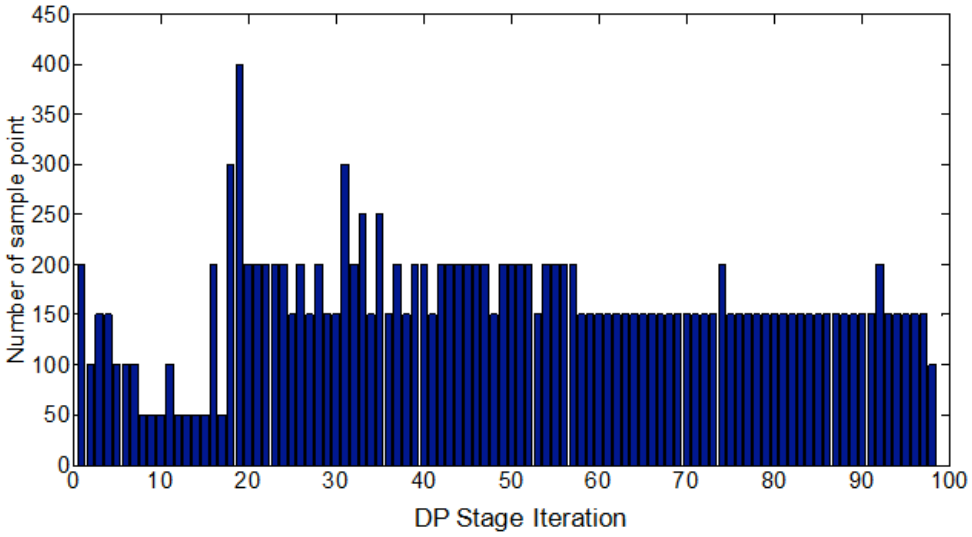



Figure 3.11: Number of Sample Points in each DP Stage Iteration of Algorithm-II

In this version, the $R^2$ of the testing data set is one of the rules in Data loop. The testing $R^2$ at each stage must be at least 0.8. The result from Figure 3.12 shows that at the very beginning of the experiment, the trend of the $R^2$ is unstable. However, after DP stage iteration 17, the value of the $R^2$ is much closed to 1 and looks stable.

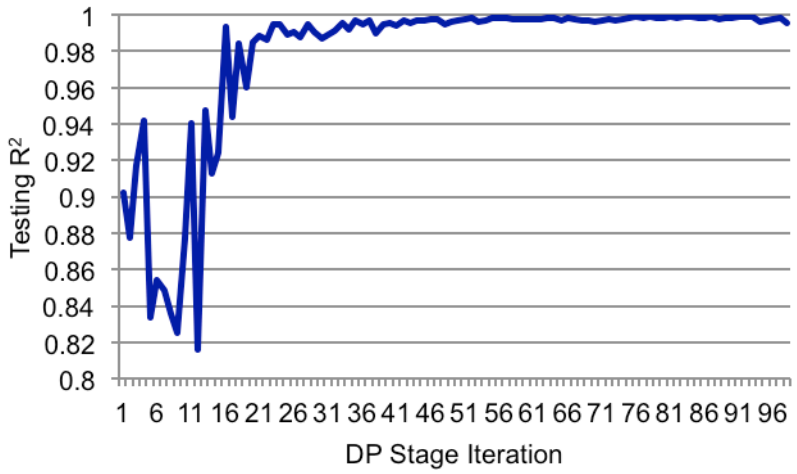
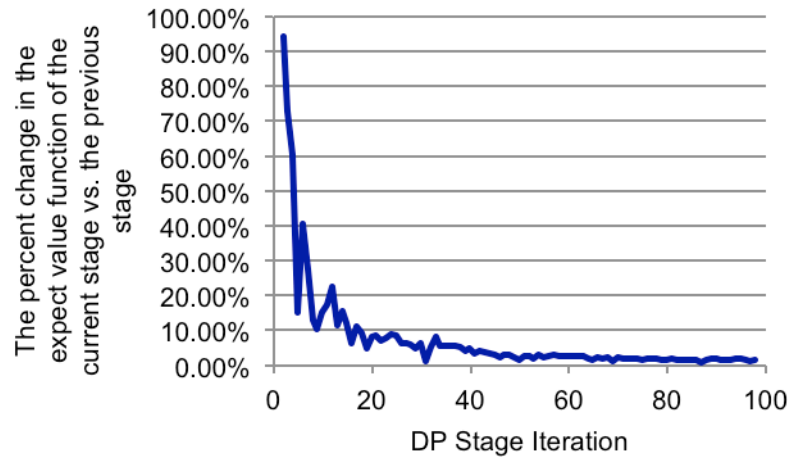


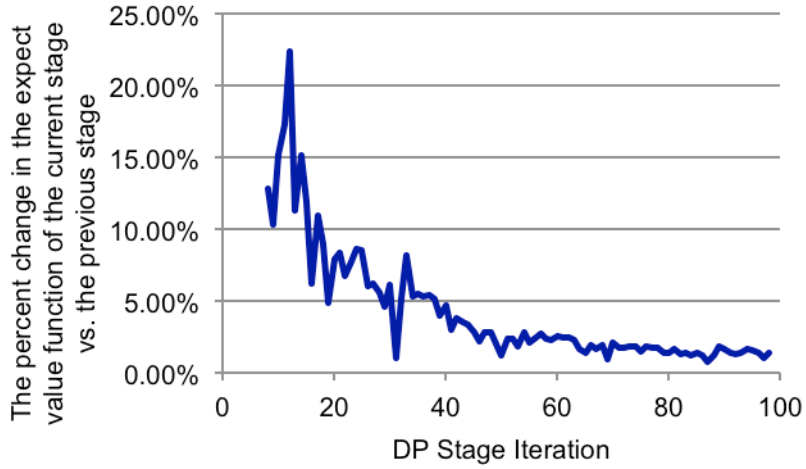

Figure 3.12: $R^2$ of the Testing Data Set in each DP Stage Iteration of Algorithm-II

The results in Figures 3.13 (a) and (b) show the percent change in the future value function average at each DP stage iteration. It can be seen that the change in the future value function tends to reduce at each DP stage iteration. Since stage 18, the percent change is less than 10%, and it goes below 5% since stage 39. At this point, we ignore the change in $R^2$ because the percent change in the future value function average is more than 10%. From Figure 3.14, it can be seen that before DP stage iteration 18, the future value functions are very far away from the future value function at the steady-state equilibrium point (true value). As a result, it requires fewer training data points and requires less CPU time.

From Figure 3.14, the $R^2$ from the true future value function starts to become reasonable from DP stage iteration 78 where the $R^2$ is greater than 0.7.




(a)




(b)

Figure 3.13: (a) The Percent Change in the Future Value Function Average at Each DP Stage Iteration of Algorithm-II Starting from DP Stage Iteration 2, (b) The Percent Change in the Future Value Function Average at Each DP Stage Iteration of Algorithm-II Starting from DP Stage Iteration 8

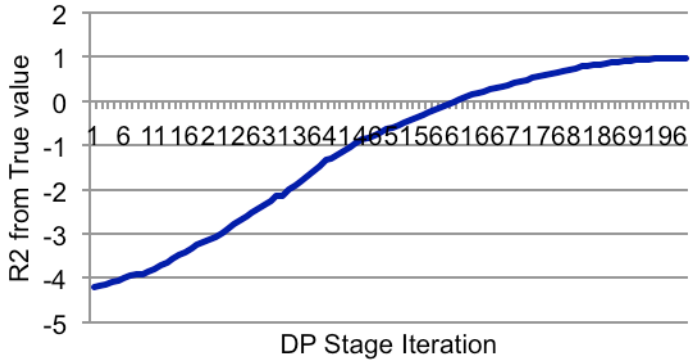



Figure 3.14: The $R^2$ of the Future Value Function in Each DP Stage Iteration vs. the Final Future Value Function at the Steady-state of Algorithm-II

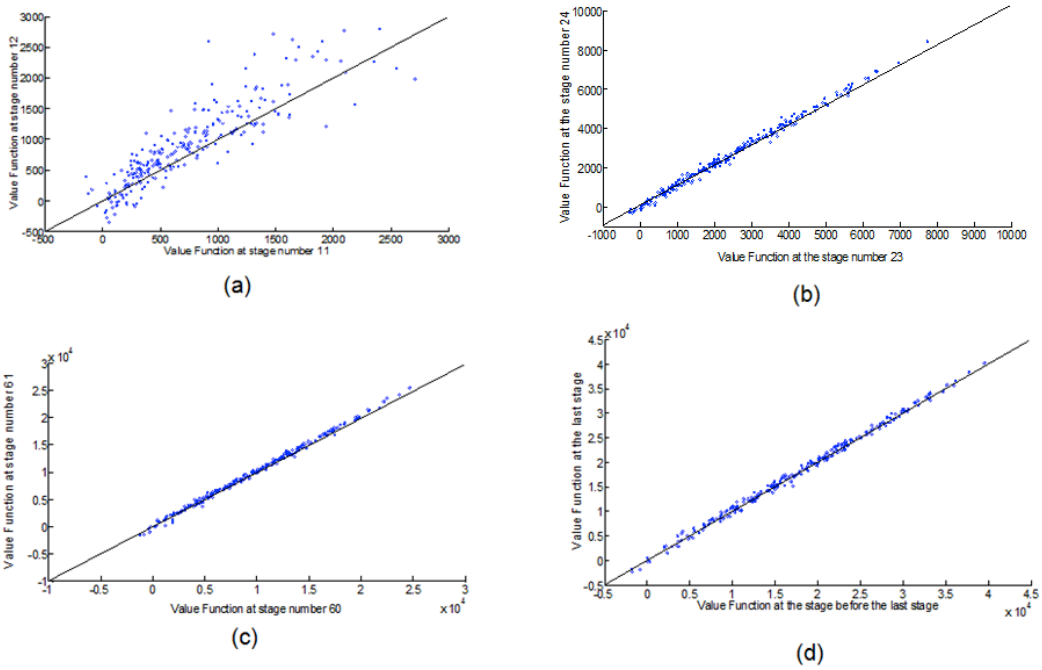



Figure 3.15: (a) The Scatter Plot of the Future Value Functions at DP Stage Iteration 11 vs. 12, (b) at DP Stage Iteration 23 vs. 24, (c) at DP Stage Iteration 60 vs. 61, and (d) at DP Stage Iteration 97 vs. 98

The results in Figure 3.15(a) reveal that at the early iteration the scatter plot is not following the 45-degree line. But later on, the plots look much more linear. Figures 3.15(b) and (c) have some shifts on the right-hand side. It can be concluded from the 95 percent confidence interval t-test that the scatter plot of the future value function at stages 97 and 98 follow the 45-degree line correspondence.

*3.7.3 Algorithm-III*

Algorithm-III constructed in the same way as Algorithm-II but different values of the $R^2$ in the Data loop condition. A new stopping rule for the Data Loop is

> Data loop stopping criteria is *minimum* $R^2$ of 0.75 and either *minimum change of the future value function average* of 10% or *maximum change in* $R^2$ of 0.005.

3.7.3.1 Results

The CPU time of Algorithm-III is 8627 seconds (2.40 hours). The DP stage iteration loop stops at iteration 43. The average number of training data points at each stage is 140. The maximum number of training data points is 250 and occurs at stage 7 and 29, as shown in Figure 3.16.

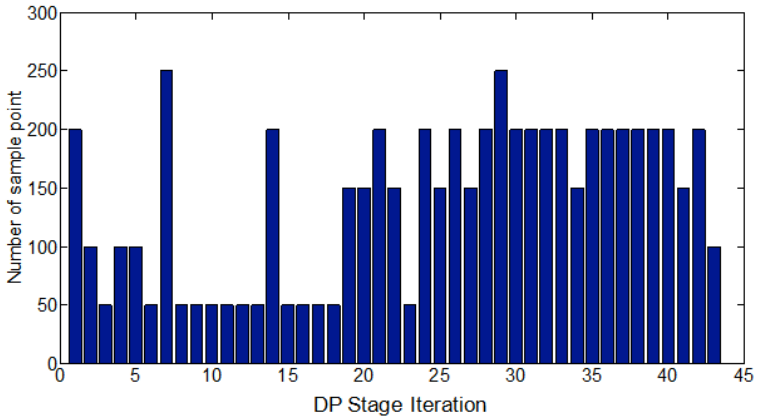



Figure 3.16: Number of Sample Points in each DP Stage Iteration of Algorithm-III

The results from Figure 3.17 show that at the very beginning of the experiment, the $R^2$ is unstable. However, after stage 19, the value of the $R^2$ is closed to 1 and looks more stable.

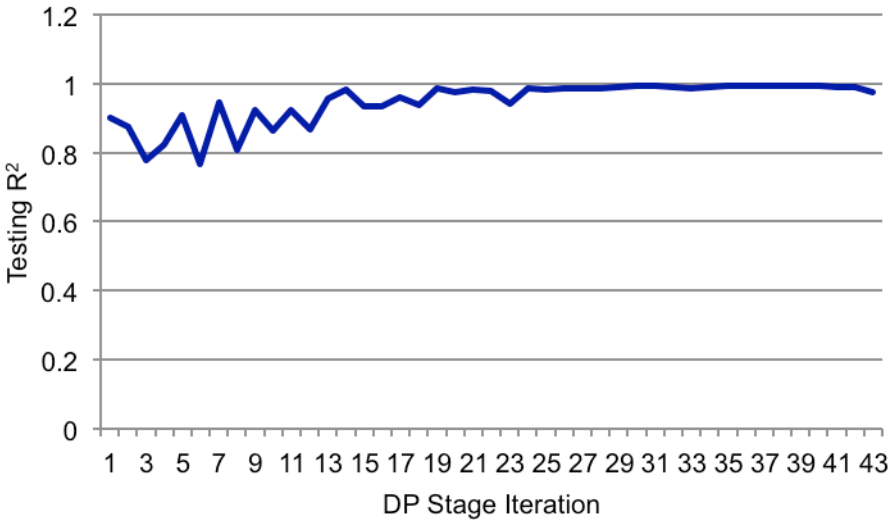
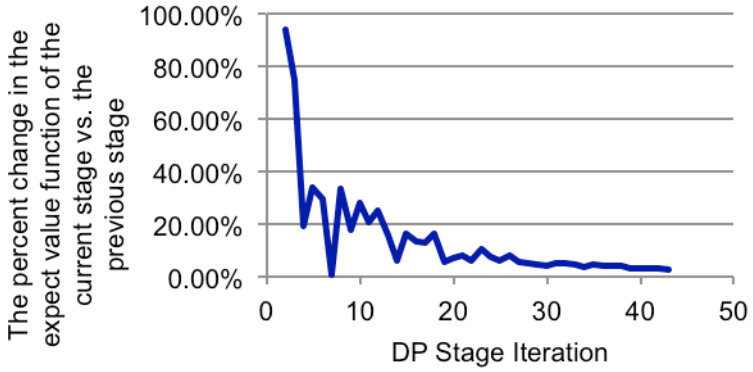


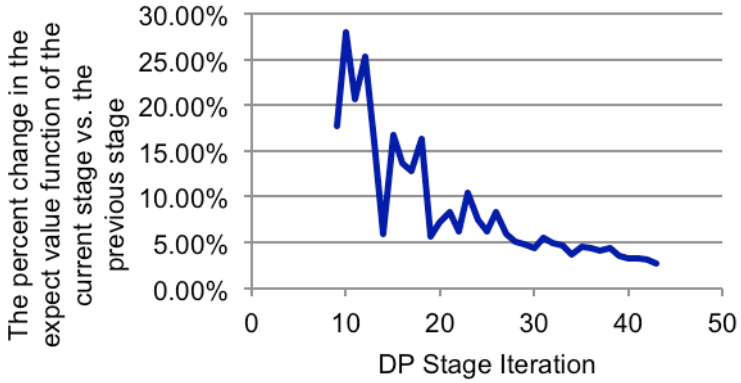

Figure 3.17: $R^2$ of the Testing Data Set in Each DP Stage Iteration of Algorithm-III

The results in Figures 3.18 (a) and (b) show the percent change in the future value function average at each DP stage iteration. It can be seen that the change in the future value function average tends to reduce at each DP stage iteration. Since iteration 19, the percent change is less than 10% and goes below 5% after DP stage iteration 28. However, the percent change in the future value function average drops and goes below 10% at iteration 7. As a result, the change in $R^2$ is considered to guarantee the best fit of the model. However, the future value function at that DP stage iteration does not pass the 95 percent confidence interval t-test of the 45-degree line correspondence. Thus, the sampling process is not stopped at that iteration.

From Figure 3.19, the $R^2$ from the true value of in each stage starts to be reasonable at DP stage iteration 34 where the $R^2$ is greater than 0.7.




(a)




(b)

Figure 3.18: (a) The Percent Change in the Future Value Function Average at Each DP Stage Iteration of Algorithm-III Starting from DP Stage Iteration 2, (b) the Percent Change in the Future Value Function Average at Each Stage Iteration of Algorithm-III Starting from DP Stage Iteration 9

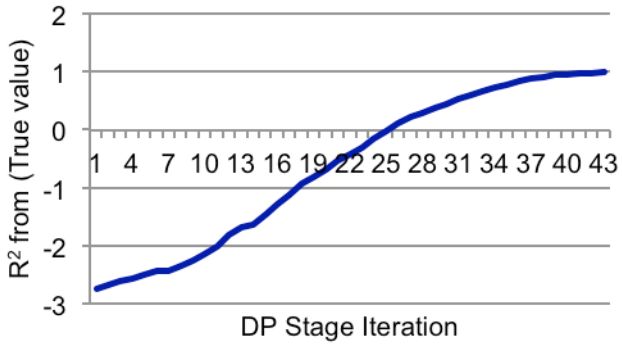



Figure 3.19: The $R^2$ of the Future Value in Each DP Stage Iteration Vs. The Final Future Value Function at the Steady-state of Algorithm-III

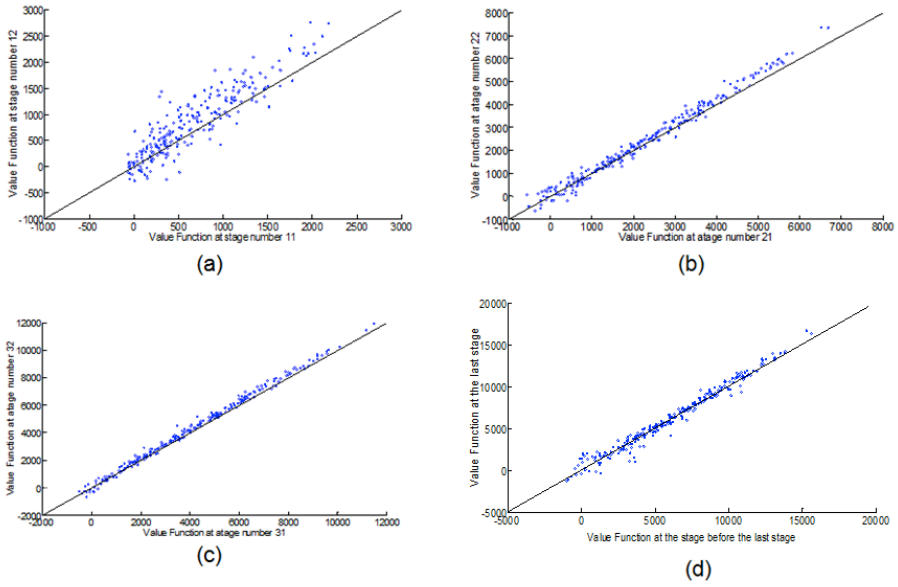



Figure 3.20: (a) The Scatter Plot of the Value Function at DP Stage Iteration 11 vs. 12. (b) at DP Stage Iteration 21 vs. 22, (c) at DP Stage Iteration 31 vs. 32, and (d) at DP Stage Iteration 42 vs. 43

The results in Figure 3.20 reveal that at the early stages the scatter plots are not following the 45-degree line correspondence. However, the plots look more linear later on. Figures 3.20(b) and (c) have some shifts on the right-hand side. It can be concluded

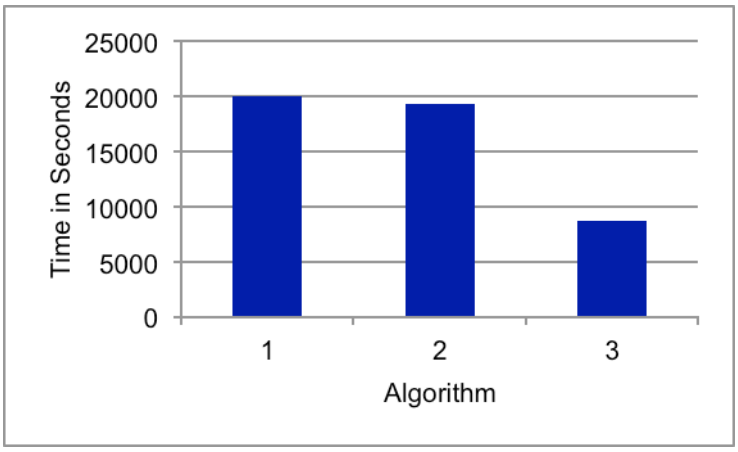
from the 95 percent confidence interval t-test that the scatter plot of the future values at stage 42 and stage 43 has a 45-degree correspondence. Thus, we can consider the function at DP stage iteration 43 in steady-state.

*3.7.4 Comparison of SDP Solution Quality*

An inventory forecasting forward simulation is used to test the SDP solution quality. The metamodel outputs from the ADP algorithms are used to run the forward simulation. A Sobol' sequence [104] is used to generate the initial state space with 100 points of in nine dimensions. Randomness is considered at each state point with 100 points. The simulation is for demonstration purposes and is limited to 12 stages. MATLAB software was used to code the algorithm along with "ARESLab: Adaptive Regression Splines toolbox for MATLAB" developed by Jekabsons [103]. The fmincon function is used for optimization. The total return is the total of the cost for all DP stage iterations. The experiments are run on Windows 7 Home Premium with Intel(R) Core(TM) i7 @ 2.80 GHz 8.00 GB RAM system.

In the Figure 3.21, the CPU times for training the optimal model have been presented for comparison. Algorithm-III performs faster than Algorithm-II as the $R^2$ is lower. Algorithm-II is slightly faster than Algorithm-I, but the convergence of Algorithm-II is done in stage 98 while Algorithm-I stops at stage 44.

The results in Figure 3.22 reveal that Algorithm-II gives the best quality of the SDP solution both in terms of the mean cost and the deviation in mean cost from the minimum, followed by Algorithm-III and Algorithm-I, respectively. Figure 3.22 (b) suggests that the proposed method in Algorithm-II is more promising compared than the other two algorithms.

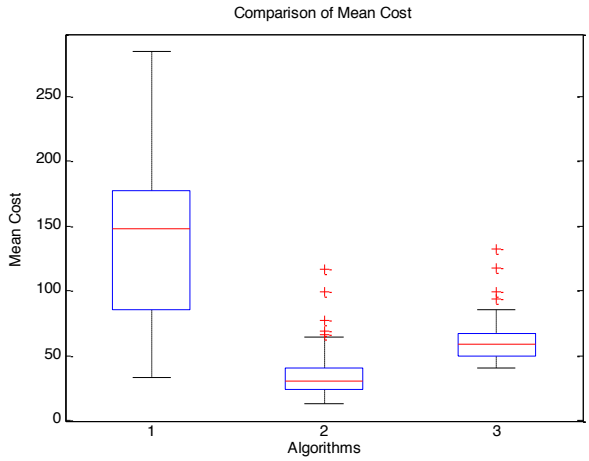



1- Algorithm-I

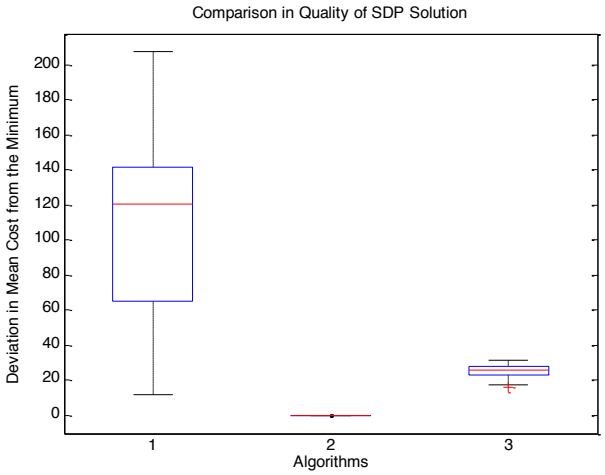
2- Algorithm-II

3- Algorithm-III

Figure 3.21: Comparison of CPU Time Used by Each Algorithm




(a)




(b)

1- Algorithm-I

2- Algorithm-II

3- Algorithm-III

Figure 3.22: (a) Comparison of Mean Cost, (b) Comparison of Quality of SDP Solution

# Chapter 4

## Comparison of Dace-Based Approach to Approximate an Infinite Horizon Dynamic Programming and Batch Mode Reinforcement Learning

Ernst et al. [38, 45] developed a tree-based batch mode reinforcement learning algorithm to determine an infinite-horizon optimal control problem with discounted rewards. Several classical tree-based supervised learning methods (CART, kD-tree, tree bagging) were used as approximators. The review on *Q*-Learning is in section 2.2.1. There is some modification in Tree-Based Batch mode RL in this dissertation and the original version presented by Ernst et al. [45]. First, MARS is used as an approximator with the same stopping rule presented in Chapter 3. Second, the stopping condition for DP stage iteration originally in Ernst's work is based upon the L-infinity norm [38, 45]. In this dissertation though, the percent change in the future value function average and the 95 percent confidence interval t-test of the 45-degree line correspondence are considered. However, the comparison on convergences with the L-infinity norm and the 95 percent confidence interval t-test of the 45-degree line correspondence are discussed in the future work. Results from the proposed method in Chapter 3 and Batch mode RL are compared at the end of this chapter.

### 4.1 Fitted *Q*-Iteration Algorithm

Step 0: Initialization:

- Choose $N$ discretization points in the state space $\{s_{it}\}_{i=1}^{N}$ for the $t$-th stage; where $t = 1,2,...$ and $s_{it} \in R^n$.
- Set the initial estimated value function $\hat{Q}_0 = 0$.
- Assume the discount factor $\gamma \in [0,1]$.
- Set $k$ =0 and *t* =1.

Step 1: The stage of an infinite horizon DP:

- Set $k \leftarrow k+1.$
- Build the training set,

$$\left\{\left(i^l, o^l\right) = \left((s_t^l, u_t^l), c_t^l + \gamma \min_{u \in U} \hat{Q}_{k-1}(s_{t+1}, u)\right)\right\}_{l=1}^{L} \quad \text{.....(4.1)}$$

where $i$ and $o$ represent input and output variables of the training set, respectively, $s_t$ is the state variable at time period $t$, $u_t$ is the decision variable at time period $t$, $c_t$ is the cost function at time period $t$, $L$ is the number of discretization points in state-action space (state-decision space) for the $t$-th stage, and $k$ refers to the $k$-th iteration.

- Approximate $\hat{Q}_k(s,u)$ with the training set.
- Solve for

$$\hat{V}_k = \min_{u \in U} \hat{Q}_k(s_t, u) \quad \text{.....(4.2)}$$

Step 2: The stopping condition:

- Check the convergence of $V_k$ with the criteria discussed in section 3.5. If the stopping criteria fail, set $t \leftarrow t+1$ and go back to Step 1.

*4.1.1 State-Action Space Discretization*

According to the RL literature, a lookup table based upon a discretized state-action space is used, which works only with a small state-action space, especially with discrete variables [15, 30, 35, 37-38, 40-41]. However, when the problem has a continuous or a very large discrete state and/or action spaces, the *Q*-function cannot be represented by a table with one entry for each state-action pair. The best method to generate the state-action space is still an open question in the RL research community.

Moreover, Ernst et al. [45] does not mention about how they came up with a state-action space discretization.

Thus, in this dissertation, we generate the state-action space by using Monte Carlo sampling with some different sizes of the training data set. Moreover, a Sobol' sequence [104] is considered as a state-space generator along with 10 different action spaces generated by Monte Carlo sampling to study the stability of the methodology.

*4.1.2 Computational Setup*

The inventory forecasting problem that is used in Chapter 3 is applied to Batch mode reinforcement learning with MARS. MATLAB software was used to code the algorithm along with "ARESLab: Adaptive Regression Splines toolbox for MATLAB" developed by Jekabsons [103]. The experiments are run on Windows 7 Home Premium with Intel(R) Core(TM) i7 @ 2.80 GHz 8.00 GB RAM system.

For any stage Iteration, the training data used to build the future value function approximation models came from a 12-dimensional state-action space. There is only one loop presented in the algorithm, which is the DP stage iteration with the fixed number of points of the training data set. The testing data set is not included in the algorithm. After obtaining the results, they are tested in the same simulation presented in Chapter 3.

## 4.2 The Algorithms

There are seven tests presented in this dissertation. To test the accuracy and stability of the results, each test contains 10 runs.

*4.2.1 Test-I: 125 Data Points*

In this test, the size of the state-action data set is 125 points. First, the 3-dimensional action space is generated using a full factorial design with 5 levels. Later on, the state space is generated using Monte Carlo sampling with 5 levels replicated 10 times each. Thus, there are 10 runs, and each run contains the same action space but a

different state data set. The 12-dimension state-action spaces then become input variables for the algorithm are presented in section 4.1. The fmincon function from Matlab is used for optimization.

4.2.1.1 Results

Figure 4.1 shows that the range of mean costs among those 10 runs is wide, which means that the results among those runs are not stable. Compared with the results in Chapter 3, which give mean costs value below 150 (the worst case of the proposed method in Chapter 3), none of those runs in this Test-I are comparable.

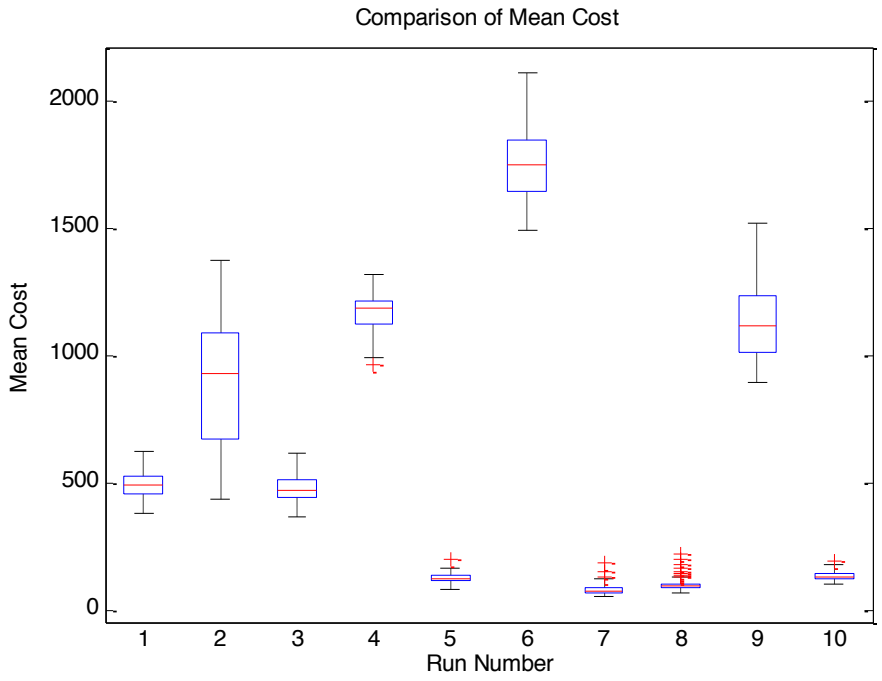



Figure 4.1: Comparison of Mean Cost of each Run in Test-I

Figures 4.2 and 4.3 show the CPU time and number of iterations. The results from these two figures also reveal that with the same number of iterations, the CPU time and mean cost are almost the same, i.e. runs 1 and 3, and runs 5 and 10. Run 7, which

used 62 iterations gives the best result in terms of mean cost; however, the runner up which is run 8 used 16 iterations. Moreover, run 6, which has 56 iterations, yields the worst result in term of mean cost. Thus, a higher number of iterations does not guarantee a better result.

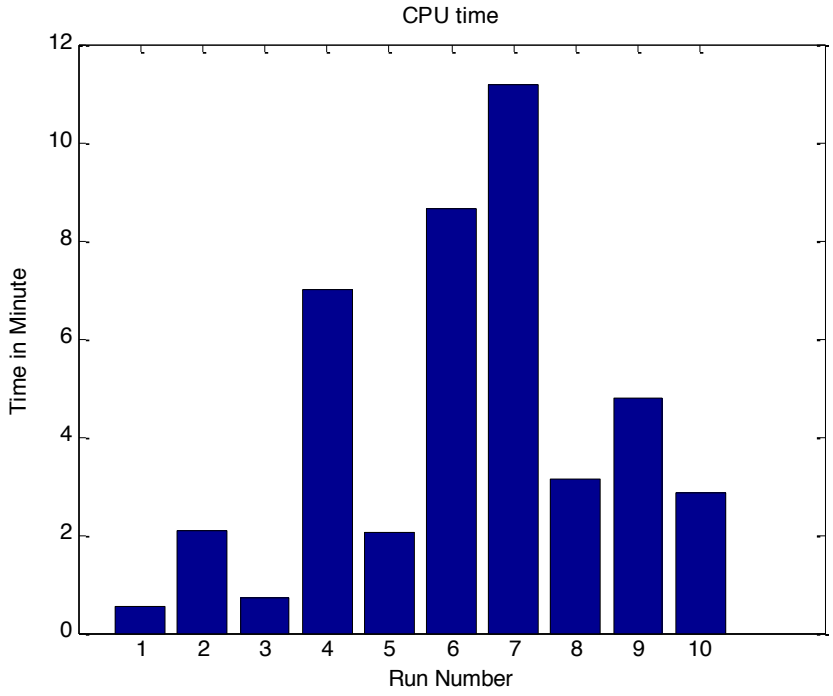



Figure 4.2: CPU Time of Each Run in Test-I

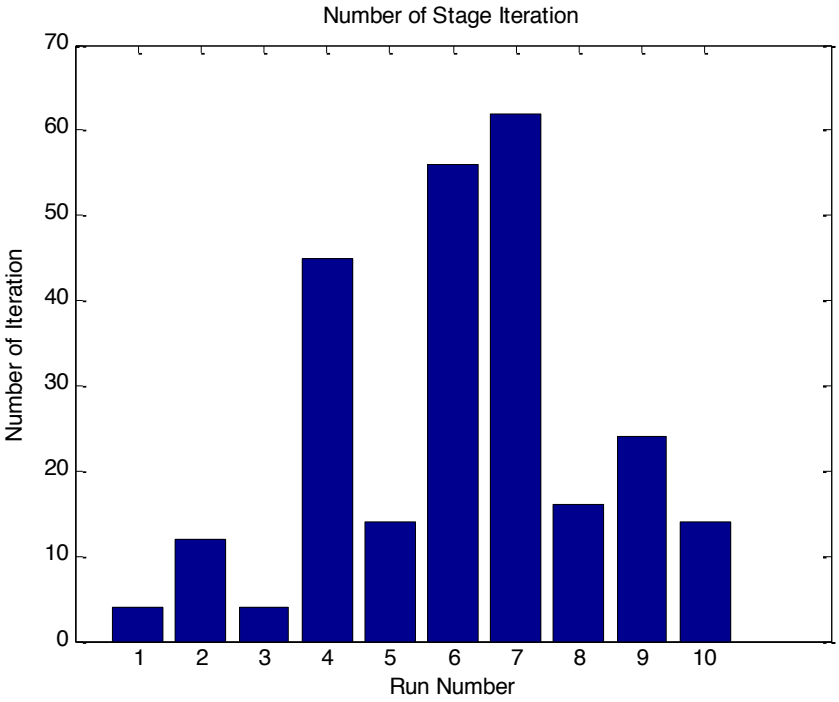



Figure 4.3: Number of Iterations of Each Run in Test-I

*4.2.2 Test-II: 125 Data Points with Look-Up Table Solution*

In this test, the size of the state-action data set is 125 points, which is the same as the one in Section 4.2.1. The difference here from Test-I is that Test-II used the look-up table method to search for the minimum of the *Q*-function.

4.2.2.1 Results

The results from Figure 4.4 reveal inconsistency. Compared with the worst case result in Chapter 3, run 10 is comparable, but it used only 3 iterations, which may not in steady-stage equilibrium. Compared with the results of Test-I, the range of mean costs in Figure 4.4 is wider than those in Figure 4.1, which means that using a continuous optimization technique is more accurate and consistent than using a look-up table.

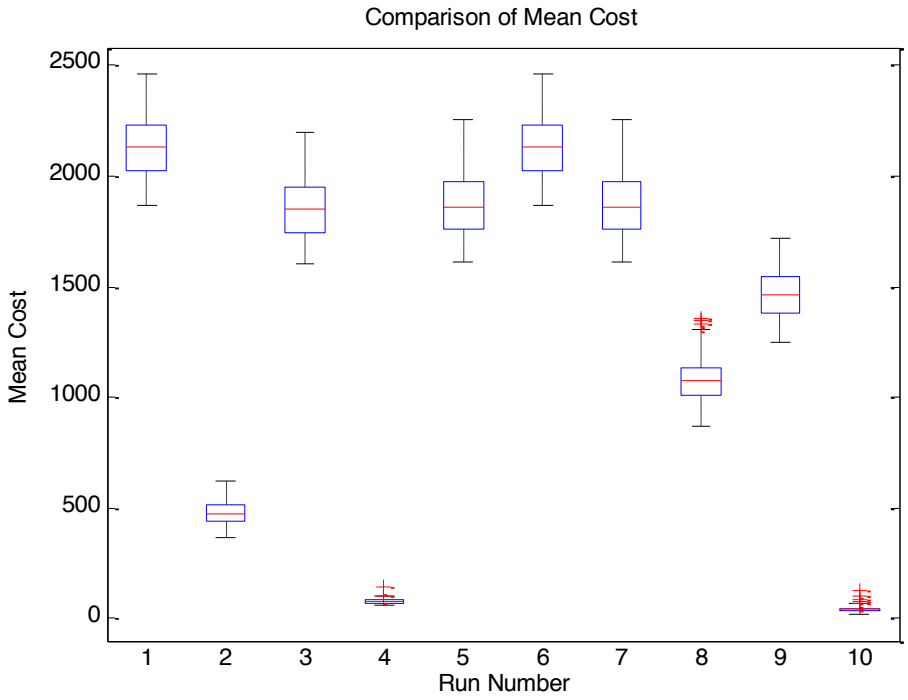



Figure 4.4: Comparison of Mean Cost of Each Run in Test-II

According to Figures 4.5 and 4.6, CPU time and the number of iterations are positively correlated. The results also show that the number of iterations does not guarantee a good result. In comparison with the results for Test-I, the look-up table method used more CPU time but does not give better results than the optimization methods. In summary, using an optimization technique gives better results with lower computational time, compared with the traditional look-up table.

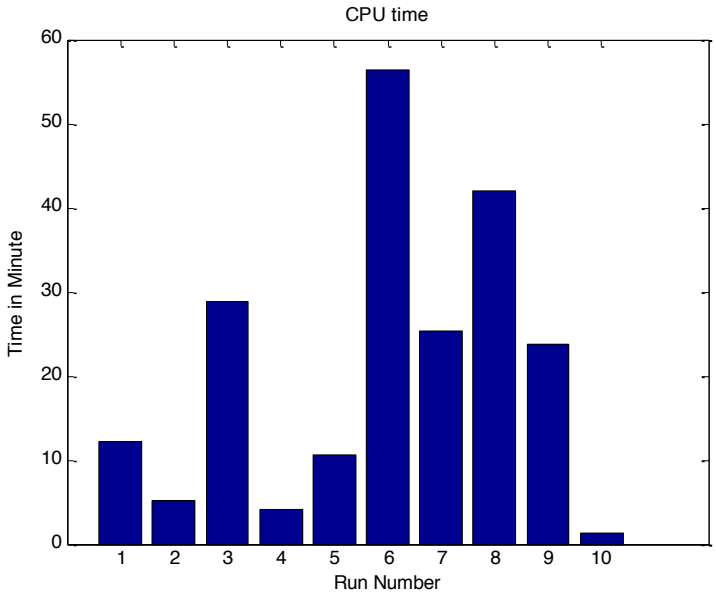



Figure 4.5: CPU Time of Each Run in Test-II

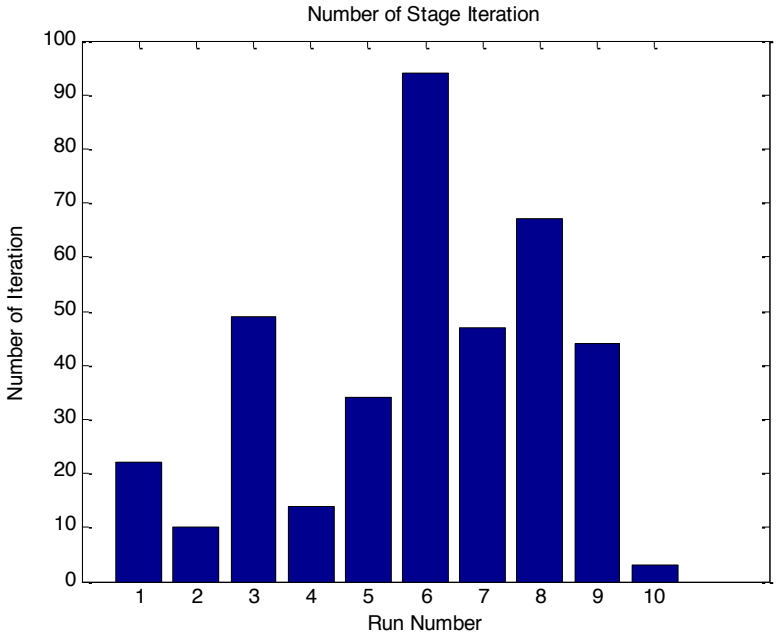



Figure 4.6: Number of Iterations of Each Run in Test-II

*4.2.3 Test-II: 216 Data Points*

In this test, the size of the state-action data set is 216 points. First, the 3-dimensional action space is generated using full factorial design with 6 levels. Later, the state space is generated using Monte Carlo sampling with 6 levels replicated 10 times each, following the same procedure as the one in section 4.2.1.

4.2.3.1 Results

Figure 4.7 shows that the range of mean costs among those 10 runs is shorter than those in Figures 4.1 and 4.4. However, the boxplots are unstable. Moreover, the results in Chapter 3 are substantially better than those in Test-III.

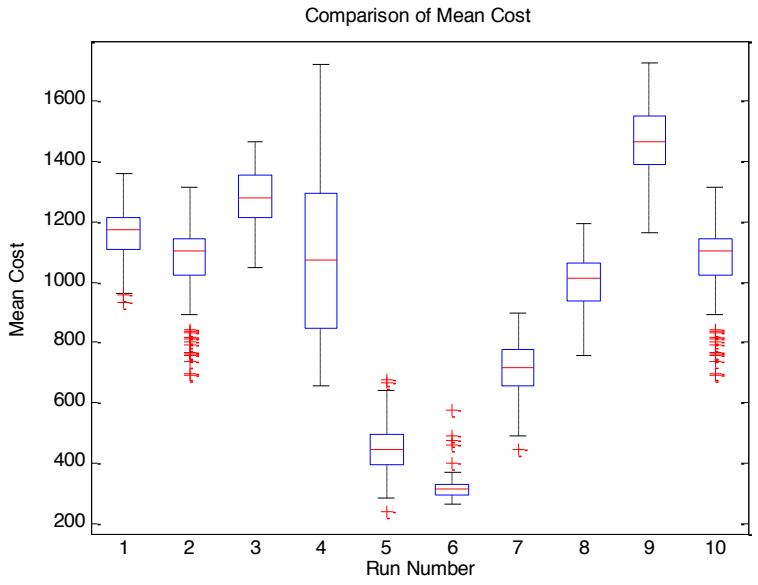



Figure 4.7: Comparison of Mean Cost of Each Run in Test-III

CPU time and number iterations are positively correlated. From Figures 4.7 and 4.9, run 4 and 5 have the same number of iterations, but the mean costs are totally different from each other. Thus, number of iterations does not guarantee an accurate result.

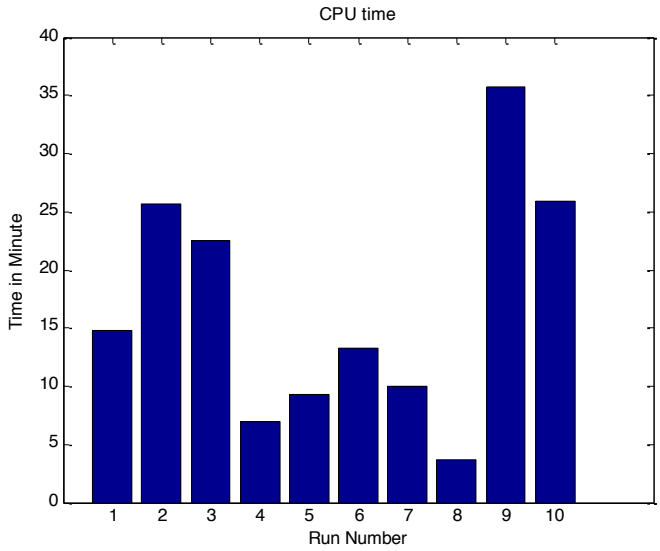



Figure 4.8: CPU Time of Each Run in Test-III

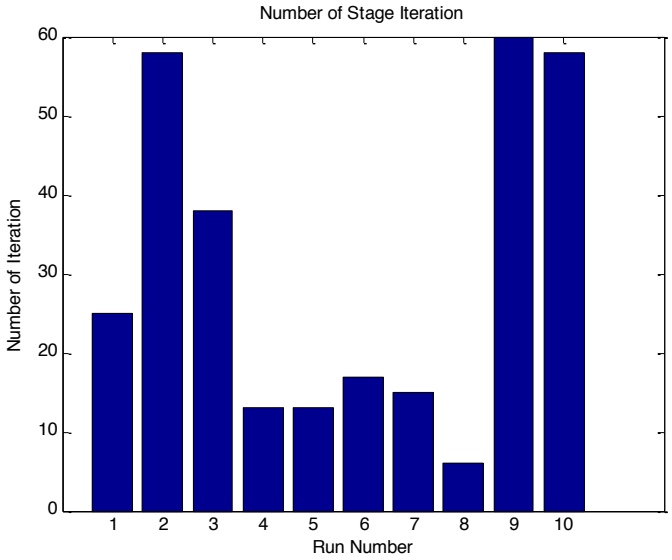



Figure 4.9: Number of Iterations of Each Run in Test-III

*4.2.4 Test-IV: 343 Data Points*

In this test, the size of the state-action data set is 343 points. A full factorial design with 7 levels is used to generate the 3-dimensional action space. Later, the state space is generated using Monte Carlo sampling with 7 levels replicated 10 times each. Then, the 12-dimensional state-action space becomes input for the algorithm presented in Section 4.1 using the optimization routine fmincon from Matlab.

4.2.4.1 Results

The result from Figure 4.10 shows that the range of mean costs among those 10 runs is wider than the one in Figure 4.7. This result indicates that adding more points in the training data set (state-action space) does not give more consistent results. Run number 9 gives the best result in terms of mean cost and is similar to the results in Chapter 3. However, this run uses 9 stage iterations, which may not ensure the convergence of an infinite horizon DP.

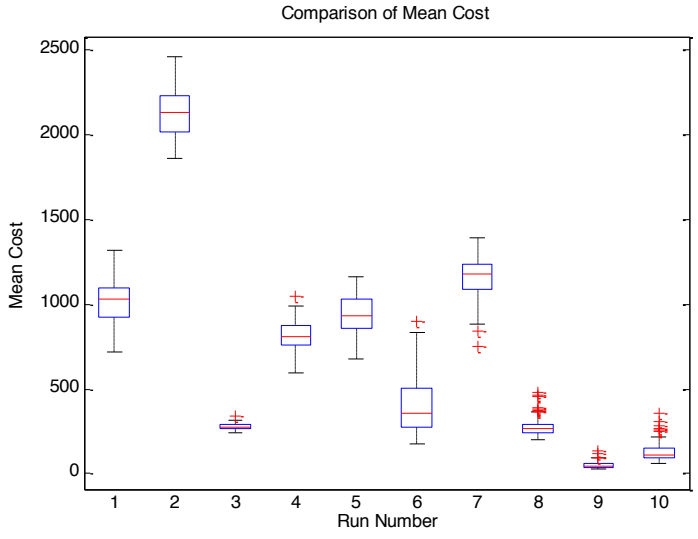



Figure 4.10: Comparison of Mean Cost of Each Run in Test-IV

The results in Figures 4.10 and 4.12 reveal that more iterations do not guarantee better results. Figures 4.11 and 4.12 show that CPU time and the number of iterations are positively correlated.

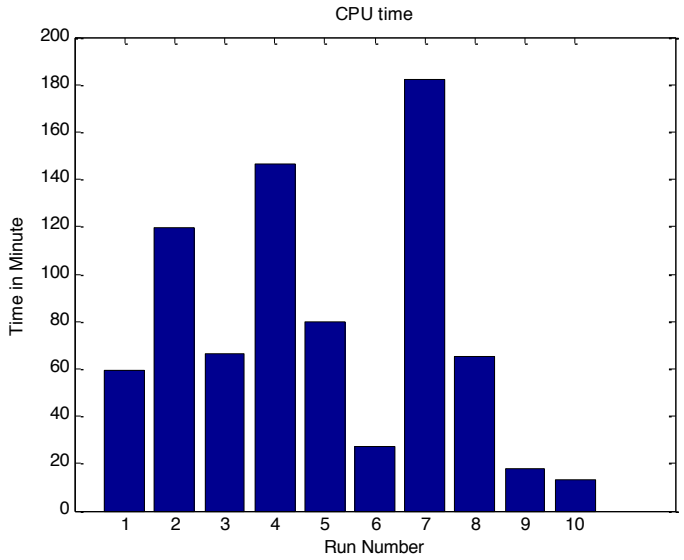



Figure 4.11: CPU time of Each Run in Test-IV

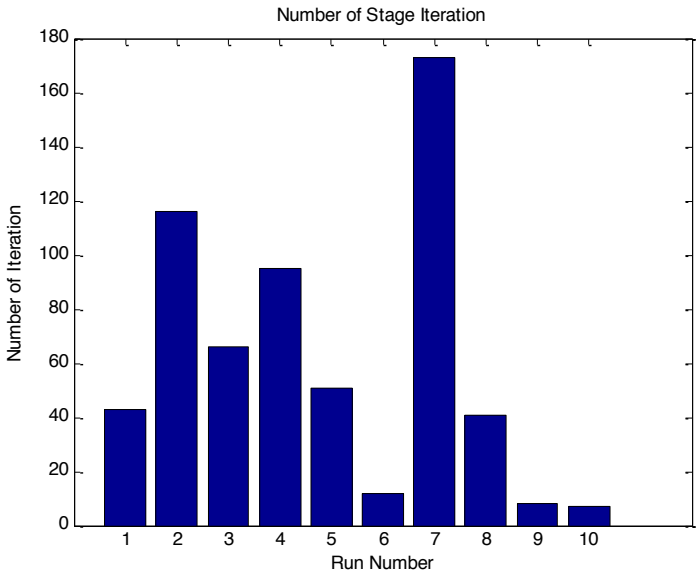



Figure 4.12: Number of Iterations of Each Run in Test-IV

*4.2.5 Test-V: 512 Data Points*

In this test, the size of the state-action data set is 512 points. The 3-dimensional action space is generated using a full factorial design with 7 levels, and the state space is generated using Monte Carlo sampling with 7 levels replicated 10 times each. The state-action space is used as an input in the algorithm with the optimization routine fmincon from Matlab.

4.2.5.1 Results

Figure 4.13 shows a wide spread of each boxplot, indicating inconsistency. Moreover, the range in this figure is wider than the one with 216 data points in section 4.2.3. Furthermore, the results in Chapter 3 are substantially better than those in Test-V.

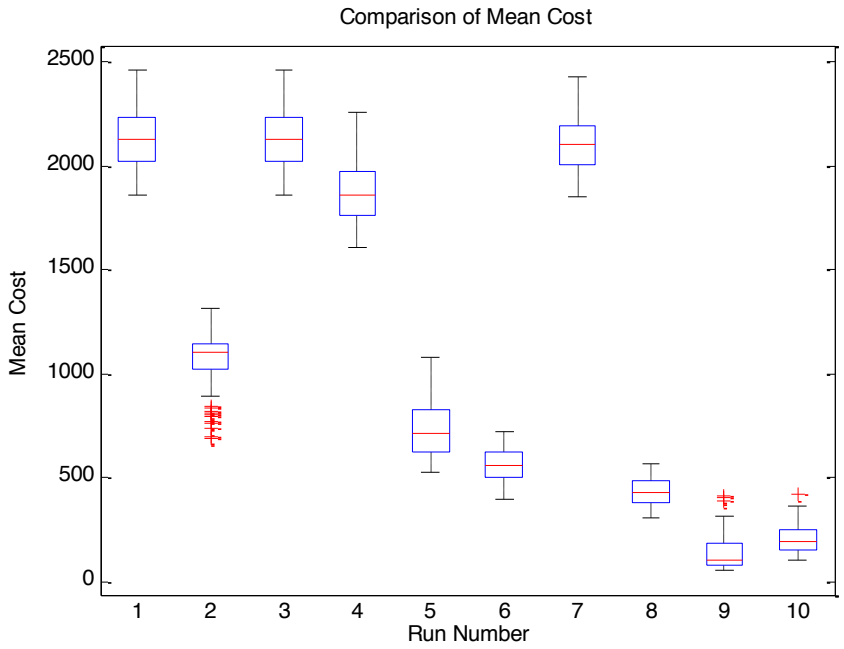



Figure 4.13: Comparison of Mean Cost of Each Run in Test-V

The CPU time and number of iterations are not in the same pattern. From Figures 4.14 and 4.15, the CPU times of runs 6 and 8 are close, but their numbers of

iterations are significantly different. This may be a result of complexity of the function approximation. Runs 2 and 6 are comparable in both CPU time and number iterations as well. However, the result in mean cost of those two runs are significantly different from each other. Thus, the results in Test-V indicate inaccuracy and instability.

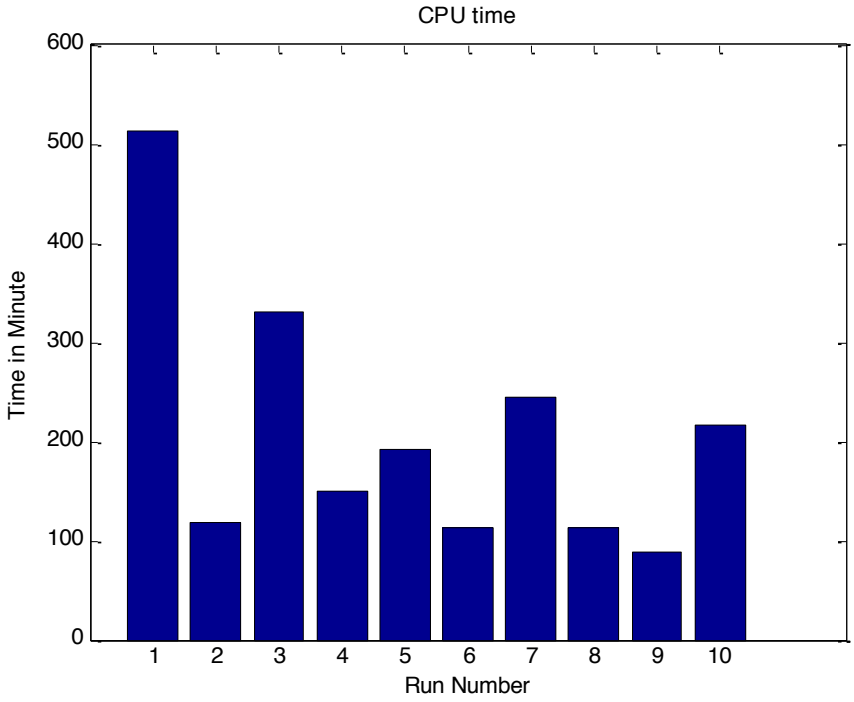



Figure 4.14: CPU time of Each Run in Test-V

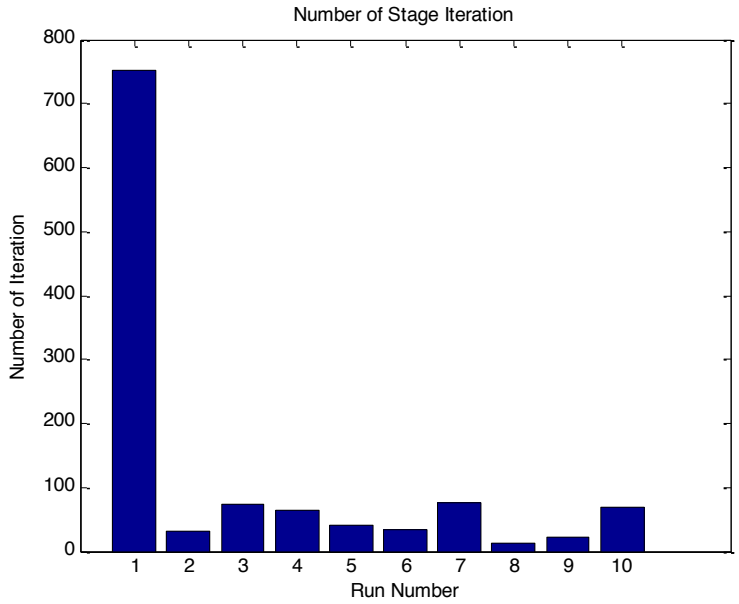



Figure 4.15: Number of Iterations of Each Run in Test-V

*4.2.6 Test-VI: 512 Data Points with State Space Generated by Sobol' Sequence*

In this test, the size of the state-action data set is 512 points. A 9-dimensional state space is generated by a Sobol' sequence design [104]. In this test, only one state space is used, but we match it with 10 different action spaces, so 10 runs are used as input in the algorithm presented in section 4.1, separately. There are two purposes of this test. The first is to study the effect of different action spaces to the solution. The second is to compare the results of using Sobol' sequence and Monte Carlo sampling.

4.2.6.1 Results

Figure 4.16 shows some consistency. Among those 10 runs, 2 runs are noticeably out of the group. Other than them, the boxplots are in the same range. However, the differences of those 2 runs and others are not as significant as the differences of each run in Figure 4.13. As a result, the state space generated by the Sobol' sequence design gives consistent results. When we compare the SPD solution

from Figures 4.13 and 4.16, Monte Carlo sampling gives the better result on average, but the Sobol' sequence design gives better results in the worst case; i.e. run 1 in Figure 4.16 used the same action space as the one in Figure 4.13 (Test-V has only one action space generation). The results in Chapter 3 are significantly better than each of the 10 runs in Figure 4.16. Figures 4.17 and 4.18 show that CPU time and the number of iterations do not necessarily correspond to each other. The results also reveal that the early stage iterations take more CPU time than later iterations.

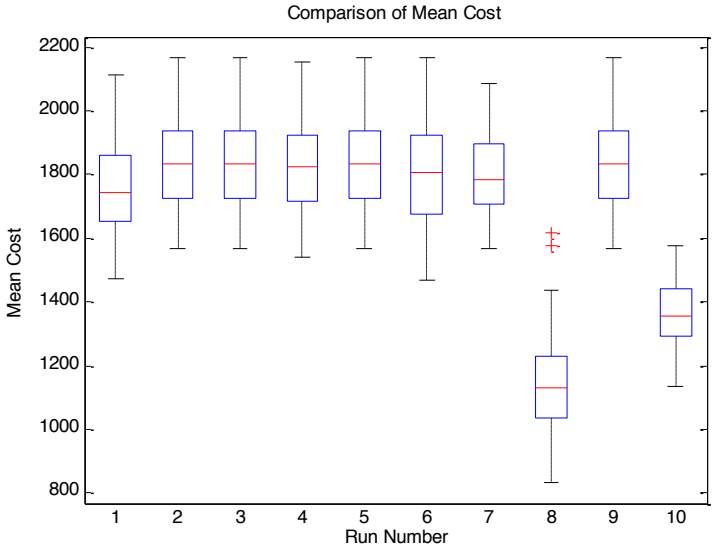



Figure 4.16: Comparison of Mean Cost of Each Run in Test-VI

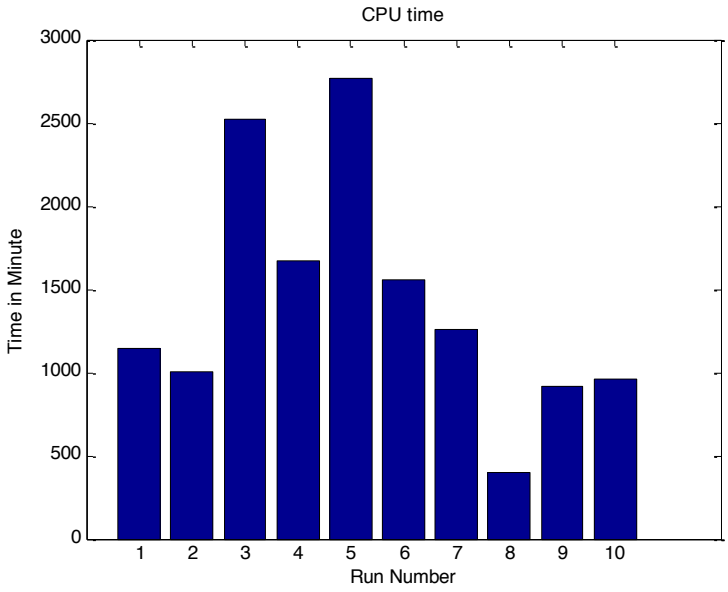



Figure 4.17: CPU time of Each Run in Test-VI

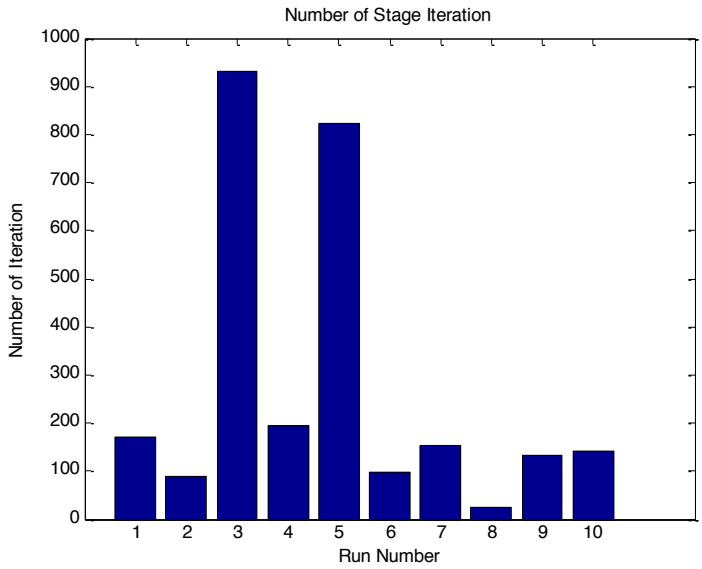


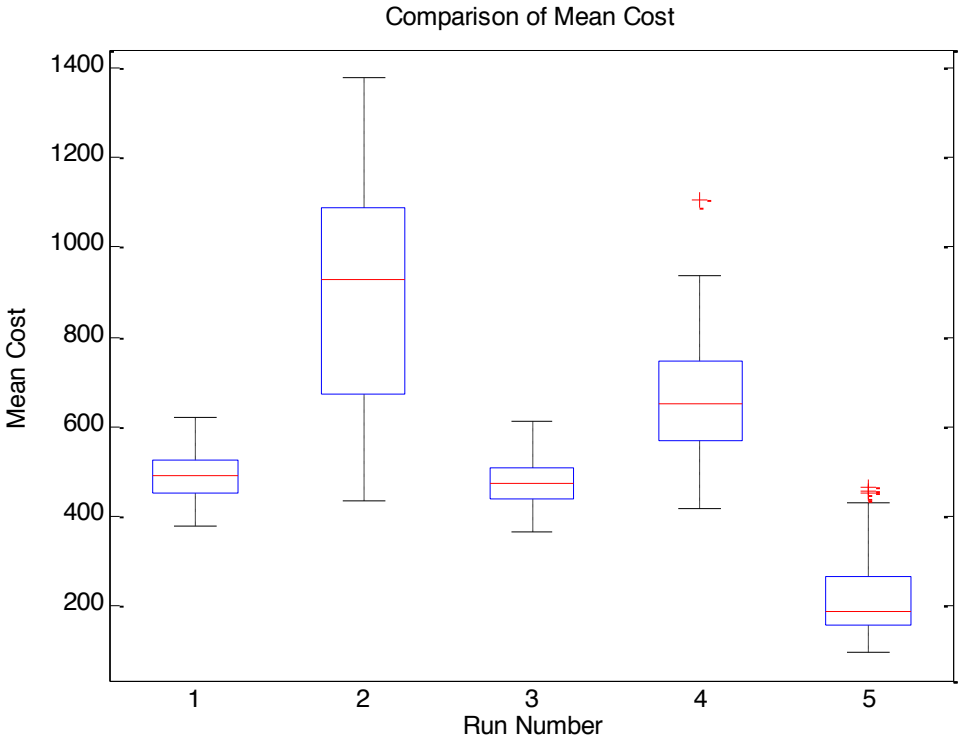

Figure 4.18: DP Stage Iteration of Each Run in Test-VI

*4.2.7 Test-VII: Online Q-learning*

The purpose of the test is to test the amenability of the *Q*-iteration to follow a consistency trace when increasing the size of the input data. The test begins with 125 dimension state-action space (Test-I) and increments by 125 points.

4.2.7.1 Results

From Figure 4.19, the results do not follow the consistency trace with the model generalization performance improving with the simultaneous increase in the size of the training data. The more training data points do not seem to be helpful in obtaining a better result. Meanwhile, it requires more computational time. Run 5 in Figure 4.19 gives a good result with adding more state-action points to the data set, but it used more than twice the CPU time of the summation of the first, second, and third runs. Moreover, when incrementing 125 more points to the fifth run, the number of iterations is more than 1500 iterations and does seem not to reach convergence.




1- Run number 1 of Test-I

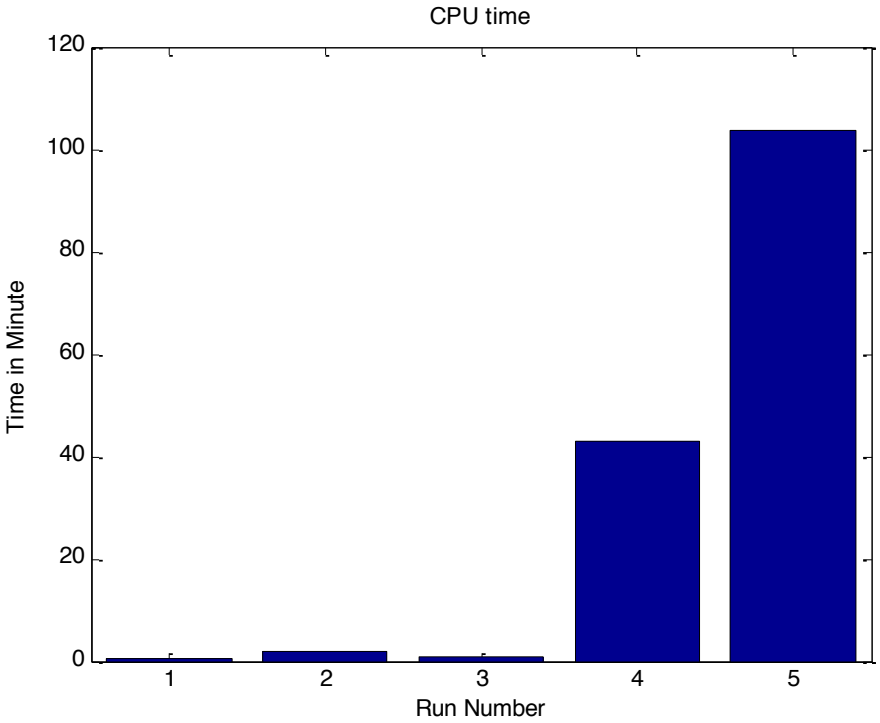
2- Run number 2 of Test-I

3- Run number 3 of Test-I

4- Run number 1 and 2 Combined

5- Run number 1, 2 and 3 Combined

Figure 4.19: Comparison of Mean Cost of Online *Q*-learning




1- Run number 1 of Test-I

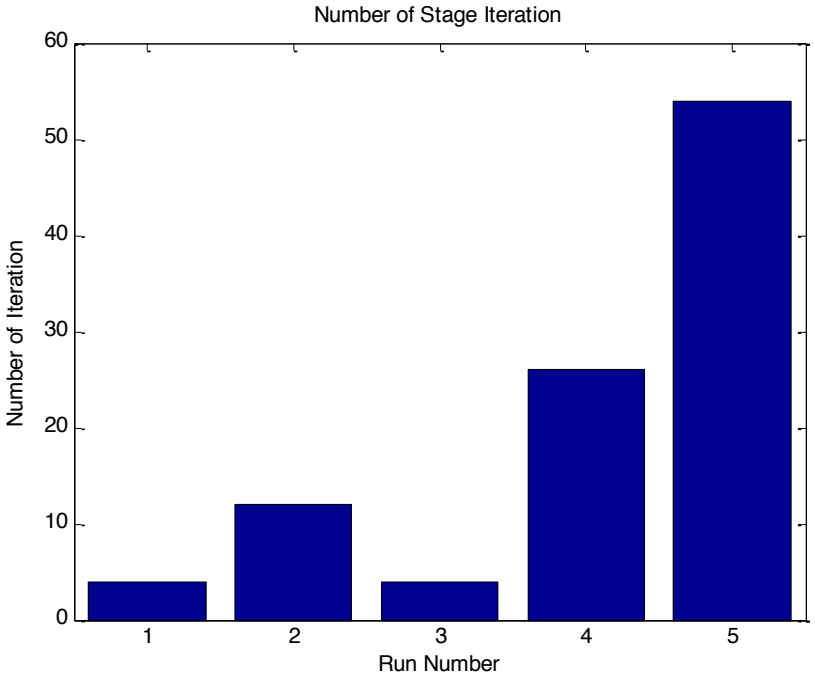
2- Run number 2 of Test-I

3- Run number 3 of Test-I

4- Run number 1 and 2 Combined

5- Run number 1, 2 and 3 Combined

Figure 4.20: CPU time of Each Run of Online *Q*-learning




1- Run number 1 of Test-I

2- Run number 2 of Test-I

3- Run number 3 of Test-I

4- Run number 1 and 2 Combined

5- Run number 1, 2 and 3 Combined

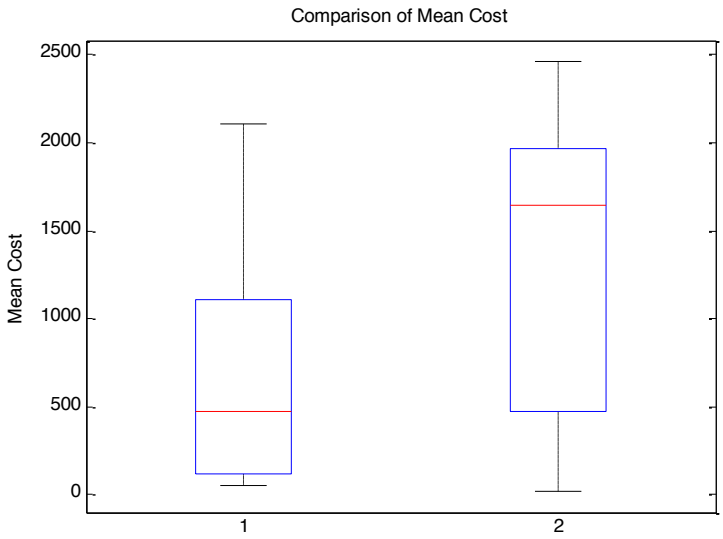
Figure 4.21: DP Stage Iteration of Each Run of Online *Q*-learning

## 4.3 Discussion of Results

### *4.3.1 Comparison between Optimization Technique vs. Look-up Table Method*

To compare the results when using an optimization technique (Test-I) and look-up table method (Test-II) to optimize the *Q*-function when using the same state-action space, the mean costs of all 10 runs of each tests are combined in one single box plot. The results in Figures 4.19 and 4.20 reveal that the optimization technique gives better results compared with the look-up method. The algorithm with the optimization technique

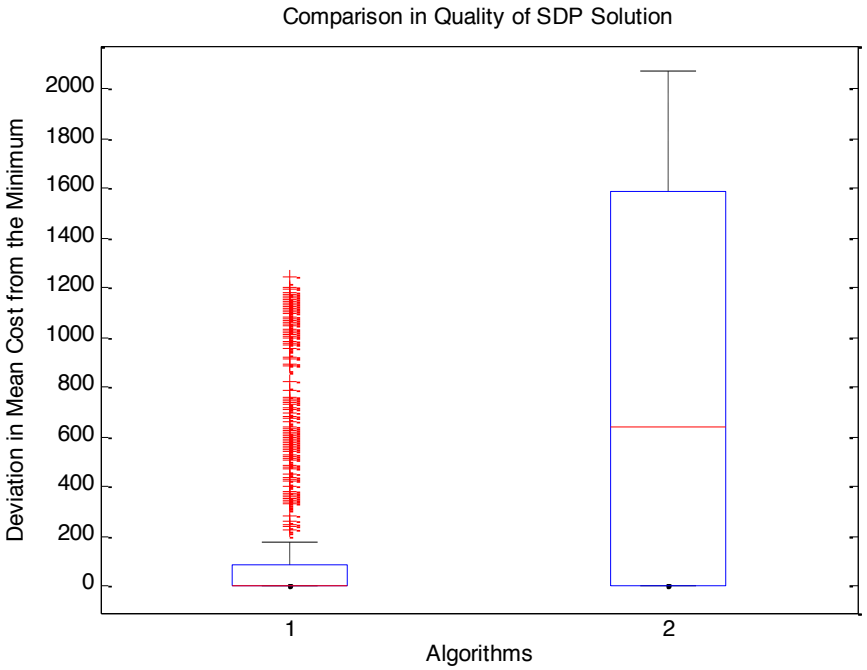
gives a better quality of the SDP solution in terms of deviation in mean cost from the minimum. Moreover, the standard deviation of the mean cost from using the optimization technique is less than that of the look-up table. The look-up table method searches for the minimum value function when the action variables are discrete, while the optimization technique looks for the best result of each possible decision variable. As a result, the optimization technique gives better results and consumes less time at each state set point. It solves the problem once, while the look-up table method looks for the best result based on every point of action space.




1- Optimization Technique

2- Look-up Table Method

Figure 4.22: Comparison of Mean Cost of 125-Point State-Action Space with Optimization Technique vs. Look-up Table Method




1- Optimization Technique

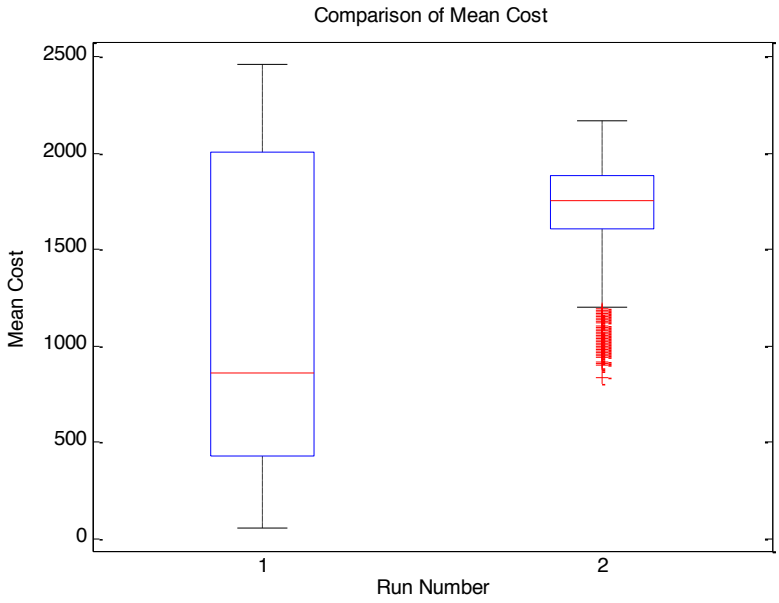
2- Look-up Table method

Figure 4.23: Comparison in Quality of SDP Solution of 125-Point State-Action Spaces with Optimization Technique vs. Look-up Table Method

*4.3.2 Comparison between Monte Carlo sampling and Sobol' Sequence Design*

Test-V and Test-VI have the same action space but different state spaces. Test-V used Monto Carlo Sampling to generation the state space while the other one uses a Sobol' sequence design [104]. The mean cost of the 10 runs of Test-V as well as the 10 runs of Test-VI are combined in to 1 mean cost, separately.

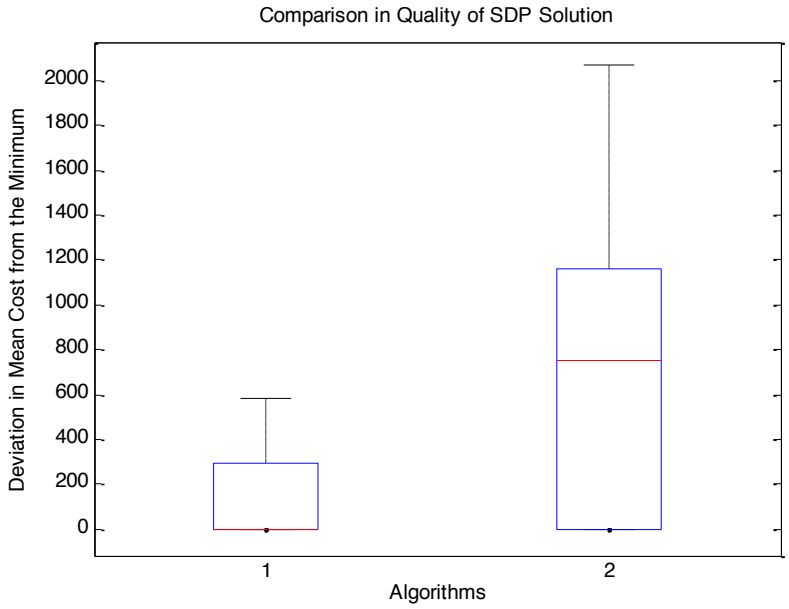
The results in Figure 4.21 and 4.22 reveal that Sobol' sequence design gives more consistent results as the standard deviation of the mean cost is lower. However, the quality of the SDP solution in terms of deviation in mean cost from the minimum was the best for the algorithm with Monte Carlo sampling.




1- Monte Carlo Sampling

2- Sobol' Sequence Design

Figure 4.24: Comparison of Mean Cost of 512-Point State-Action Space with Monte Carlo Sampling vs. Sobol' Sequence Design




1- Monte Carlo Sampling

2- Sobol' Sequence Design

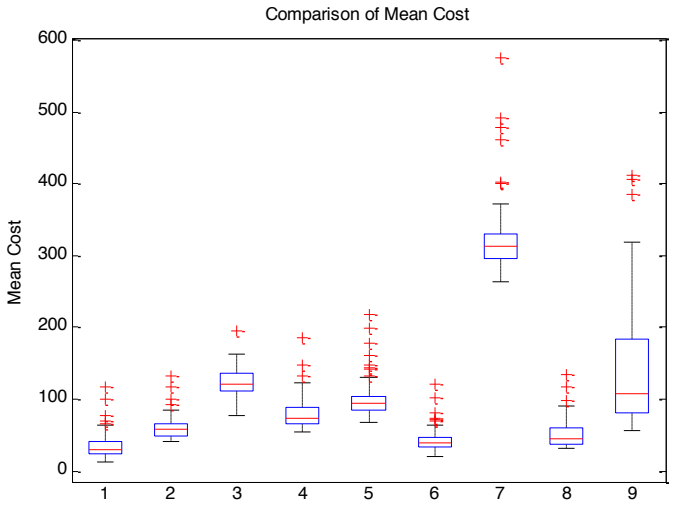
Figure 4.25: Comparison in Quality of SDP Solution of 512-Point of State-Action Space with Monte Carlo Sampling vs. Sobol' Sequence Design

*4.3.3 Comparison between Dace-based Approach to Approximate an Infinite Horizon Dynamic Programming and Batch Mode Reinforcement Learning with MARS*

In this section, the selected runs from each test in this Chapter that have good results in terms of mean costs are compared with the results in Chapter 3 using the same forward simulation.

From Figure 4.27, the best number in terms of the quality of the SDP solution is number 1 which is the Algorithm-II of the proposed method in Chapter 3. Runs 6 and 8 seem to be comparable with number 1, but the number of stage iterations of those runs are low (3 and 8, respectively) indicating that they may not ensure convergence. Number 4, which is *Q*-Iteration algorithm of run 7 of Test-I gives a good result even if the size of

the training dataset is smaller than others. It shows that adding more training data points to the *Q*-iteration algorithm may not help to get a better result. Among 63 runs (3 runs from Chapter 3 and 60 runs from Chapter 4), it can be concluded that sequential DACE APD is better than Batch Mode RL.




1- Algorithm-II of the Proposed Method in Chapter 3

2- Algorithm-III of the Proposed Method in Chapter 3

3- *Q*-Iteration Algorithm of Run Number 5 of Test-I

4- *Q*-Iteration Algorithm of Run Number 7 of Test-I

5- *Q*-Iteration Algorithm of Run Number 8 of Test-I

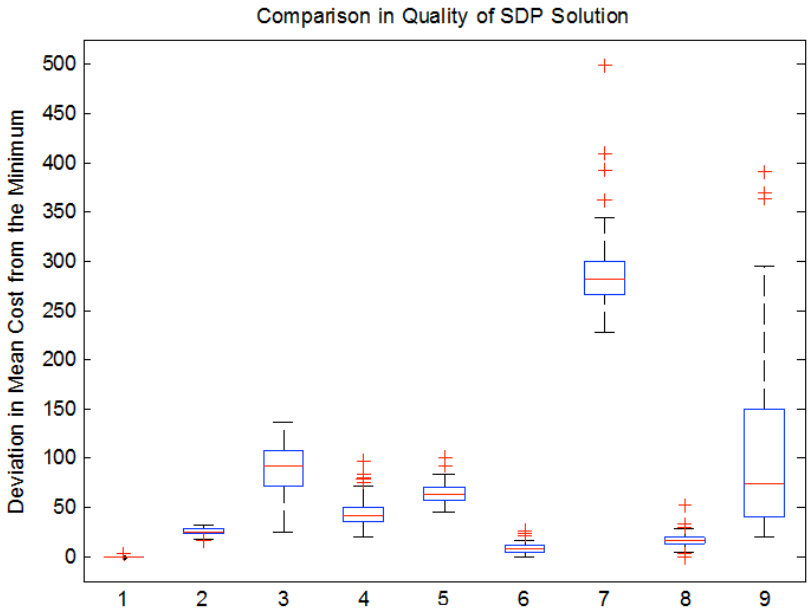
6- *Q*-Iteration Algorithm of Run Number 10 of Test-II

7- *Q*-Iteration Algorithm of Run Number 6 of Test-III

8- *Q*-Iteration Algorithm of Run Number 9 of Test-IV

9- *Q*-Iteration Algorithm of Run Number 9 of Test-V

Figure 4.26: Comparison of Mean Cost




1- Algorithm-II of the Proposed Method in Chapter 3

2- Algorithm-III of the Proposed Method in Chapter 3

3- *Q*-Iteration Algorithm of Run Number 5 of Test-I

4- *Q*-Iteration Algorithm of Run Number 7 of Test-I

5- *Q*-Iteration Algorithm of Run Number 8 of Test-I

6- *Q*-Iteration Algorithm of Run Number 10 of Test-II

7- *Q*-Iteration Algorithm of Run Number 6 of Test-III

8- *Q*-Iteration Algorithm of Run Number 9 of Test-IV

9- *Q*-Iteration Algorithm of Run Number 9 of Test-V

Figure 4.27: Comparison in Quality of SDP Solution

# Chapter 5

## Two-Stage Framework Application to a Controllability of a System of Plug-In Hybrid Electric Vehicle (PHEV*)* Charging Stations

This material is based upon work supported by the National Science Foundation under Grant No. 1128871. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF. The title of this NSF project is “EPAS/AIS Collaborative Research: Adaptive Design for Controllability of a System of Plug-In Electric Vehicle Charging Stations.” The main purpose of the project is to develop a framework that integrates system design and dynamic system control. There are two stages in the project. The first stage (or master problem) addresses the system design problem, while the second stage (or subproblem) addresses the dynamic control problem. The research in this dissertation focused on a dynamic control of a system of plug-in hybrid electric vehicle (PHEV) charging stations (the second stage). A design and analysis of computer experiments (DACE) approach is used to build a metamodel for the expected value function of the second-stage.

A finite horizon dynamic problem is presented. Based upon the 15-minute updated period of the electricity market price, the objective function is to maximize profit, which is the revenue benefit from selling back to the grid and the charging of the vehicles minus the cost of buying electricity from the grid. The state variables in each 15-minute time period consist of the total wind purchased by the system, solar power generation at each charging station, total demand at each station, and nodal market price at station locations. As an initial solution analysis, the mean value problem is formulated as a deterministic linear program and solved to present potential policies providing insight into the behavior of the system.

## 5.1 Design and Analysis of Computer Experiments Approach

In statistics, a common problem is to model the relationship between input variables and responses. Many methods from statistical experimental design (DoE) and statistical modeling have been developed to address the estimation of the function relationship between the input and its responses [34]. When applied to a two-stage framework, the possible solutions of the first stage become input variables, and the objective solutions (or policies) from the second stage based on the first stage solutions are responses.

The steps involved in DACE are:

- Design of Experiments (DoE) is used to generate the set of sample points offering the first stage space as input to the optimization model, which then provides the corresponding responses.
- An optimization model (computer experiment) of system performance is constructed based on knowledge of how the system operates based on each possible first stage solution.
- A statistical model is used to approximate the second stage value given a first stage solution.

Once the metamodel is obtained, it is passed to the first stage design problem.

## 5.2 Adaptive Design for Controllability of a System of PHEV Charging Stations Case Study

### *5.2.1 The First Stage Master Problem*

The first stage master is the system design function where the objective consists of costs on the design parameters and an expected cost $E_s\left[V(s,x)\right]$ from the second-stage optimal value function over possible initial states. The system design variables

include the locations of the charging stations and the number of slot at each charging station. The general formulation of the first stage design problem is

$$c(x) + E_s\left[V(s,x)\right] \quad \text{.....(5.1)}$$

which satisfies the specifications

$$\text{s.t. } x \in \Gamma_D \quad \text{.....(5.2)}$$

- $c(x)$ is the "cost" objective.
- $x$ is the system design variables.
- $s$ is the control problem state variable.
- $V(s,x)$ is the optimal value function for the second stage dynamic control problem.
- $\Gamma_D$ is the constraint set for the system design variables.

In this dissertation, the main focus is to solve for the $E_s\left[V(s,x)\right]$ from the second stage control problem. Thus, solution method of the first stage design problem is not presented.

*5.2.2 The Second Stage Control Problem*

The controllability module is a dynamic control problem because decisions are made in several time stages, and the optimization problem becomes dynamic and multi-stage. There is at least one transition equation on the problem, which means that the next state of the process depends entirely on the current state of the process and the current decisions taken [106].

At each stage, the system is defined by sets of state variables, which include the market price of energy, solar production of each station, the total wind purchased to the system, and the total demand of each station. When a decision is made, a cost is obtained, and the system undergoes a transition to the next stage. The decision variables in this problem include wind allocation fraction among charging stations, electricity sold

back to the grid from the battery and direct charge, electricity purchased from the grid, demand satisfied by the battery and direct charge, and battery charging level.

The objective is to maximize profit or, equivalently, to minimize operational cost, which is the cost of buying from the grid minus the revenue from selling back to the grid and charging the PHEV both from the battery and the direct charge across all the stations. Following the timing of the electricity market, the system evolves in 15-minute time intervals. We consider a 24-hour time period. As a mean value problem, we assume that the forecasts are perfect. At each time period, each state variable is equal to its estimated value. The objective is given by equation (5.3)

$$\max \sum_{t\in T} \sum_{j\in J} \left( \widetilde{B}_t (g_{tj}^- + R_{tj}) - \widetilde{C}_t g_{tj}^+ \right) + r_t \widetilde{D}_{tj} \qquad \text{.....(5.3)}$$

where $\widetilde{C}_t$ is the market selling price of energy in time period $t$, $\widetilde{B}_t$ is the market buying price of energy in time period $t$, $g_{tj}^+$ is the electricity bought from the grid of station $j$ in time period $t$, $g_{tj}^-$ is the electricity sold back to the grid from the direct charge of station $j$ in time period $t$, $R_{tj}$ is the electricity sold back to the grid from the battery of station $j$ in time period $t$, $r_t$ is the retail price of energy in time period $t$, and $\widetilde{D}_{tj}$ is the total demand in time period $t$ at charging station $j$.

The first constraint set (5.4) includes the battery level transition from period $t$-1 to period $t$ for each station $j$:

$$I_{t,j} = I_{(t-1),j} + BC_{tj} - \frac{R_{tj}}{e_j} - \frac{D_{tj}^{\,2}}{e_j} \qquad \forall j \in J, \forall t \in T \qquad \text{.....(5.4)}$$

where $I_{tj}$ is the battery level of station $j$ at the beginning of time period $t$, $BC_{tj}$ is the battery Charge of station *j* in time period $t$, $D_{tj}^{\,2}$ is the demand satisfied by the battery of

station $j$ in time period $t$, and $e_j$ is the storage efficiency of station $j$. In our computational results, we assume that the storage efficiency $e_j$ is 79.8% [107-108].

The second constraint set (5.5) includes the energy balance for the battery charge at each station.

$$BC_{tj} = \widetilde{W}_t W_{tj} + \widetilde{S}_{tj} + g_{tj}^{+} - g_{tj}^{-} - D_{tj}^{1} \quad \forall j \in J, \forall t \in T \qquad \text{.....(5.5)}$$

where $W_{tj}$ is the fraction of wind allocated to station $j$ in time period $t$, $\widetilde{W}_t$ is the total wind purchased in time period $t$, $\widetilde{S}_{tj}$ is the solar production of station $j$ in time period $t$, $D_{tj}^{1}$ is the demand satisfied by the direct charge of station $j$ in time period $t$.

The total demand consists of the demand satisfied by direct charge and demand satisfied by the battery as shown in constraint set (5.6)

$$\widetilde{D}_{tj} = D_{tj}^{1} + D_{tj}^{2} \qquad \forall j \in J, \forall t \in T \qquad \text{.....(5.6)}$$

The combination of electricity sold back to the grid from the battery and demand satisfied by the battery together is less than or equal to the discharge rate ($dc$) multiplied by the storage efficiency, as shown in constraint set (5.7)

$$R_{tj} + D_{tj}^{2} \le dc * e_j \qquad \forall j \in J, \forall t \in T \qquad \text{.....(5.7)}$$

The battery charge must not be greater than the charge rate ($cr$), and the battery level must be constrained in between the minimum battery level and the battery capacity for each station, as in constraints (5.8) and (5.9), respectively.

$$BC_{tj} \le cr \qquad \forall j \in J, \forall t \in T \qquad \text{.....(5.8)}$$

$$\text{U}nit\_Min_j \le \text{I}_{\text{tj}} \le \ Unit\_size_j \quad \forall j \in J, \forall t \in T \qquad \text{.....(5.9)}$$

The battery level at the last stage is assumed to be equal to the first stage.

$$I_{T,j} = I_{1,j} \ \forall j \in J \qquad \text{.....(5.10)}$$

The fraction of wind allocation, constraint in equation (5.11), is constructed to allocate the total wind production to each station. Lastly, the set of nonnegative constraints is given in (5.12).

$$\sum_{j \in J} W_{tj} = 1 \qquad \forall t = 0,1,\ldots \qquad \text{.....(5.11)}$$

$$I_{tj}, W_{tj}, g^{+}_{tj}, g^{-}_{tj}, BC_{tj}, R_{tj} \quad \geq \quad 0 \quad \forall j \in J, \forall t \in T \qquad \text{.....(5.12)}$$

As an initial solution analysis, the mean value problem is formulated as a deterministic linear program to provide insight into the behavior of the system. In this dissertation, the result on the mean value problem is used to fit with the DACE-Approach.

*5.2.3 Mean Value Problem Results*

The result from MATLAB solving the mean value problem of control for 5 PHEV charging stations over 96 time periods is presented in this section. PHEV charging demand profile in 2012 from Khosrojerdi et al. [109] is used (including demand in Tarrant, Ellis, Dallas (Garland area), Collin and Denton). In this model, we assume that we have a contract with a wind farm (e.g. 30% of wind energy production) and we do not include this cost in the objective function. This simulation is based on January 2012, and the average retail sale price of electricity in the transportation sector in Texas is 10.17 cents per kilowatt-hour [110]. The maximum and minimum battery capacities are 3.6 and 0.72 MWh per slot. The charging rate and discharging rates are 0.6 and 0.075 MWh per slot. In this simulation, we assume that there is only 1 slot per each station.

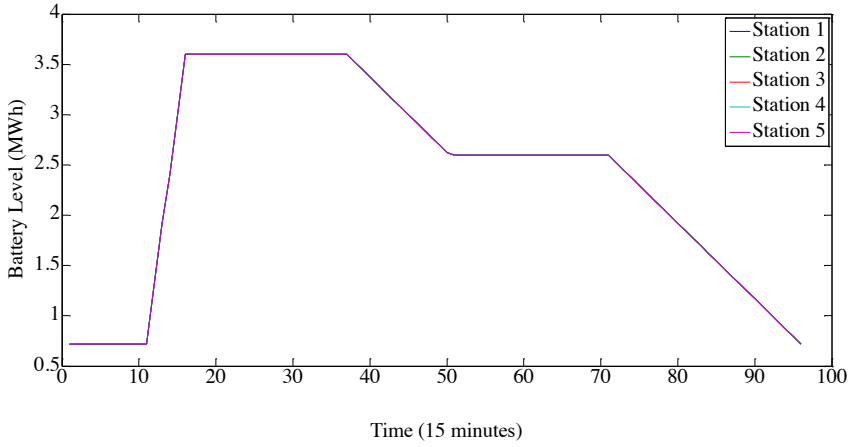



Figure 5.1: Battery Level

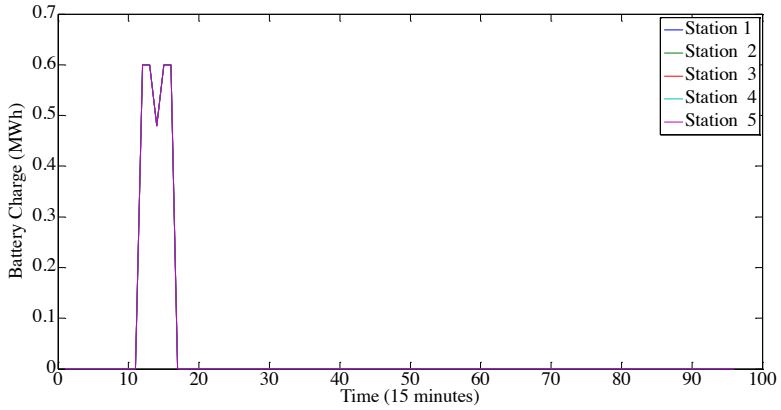



Figure 5.2: Battery Charge

Figure 5.1 shows that the battery level starts to increase at $t$ =12 and reach the maximum level at $t$ = 16. After that, it stays constant until $t$ = 38, and it starts reducing until it $t$ = 50. Then, it reduces again at $t$ = 71 until reaching the minimum at the end of time period. All stations have the same battery level. The battery charge is close to 0 in all time periods, except time periods 11 to 16 as shown in Figure 5.2. Due to a low market price, shown in Figure 5.6, the system increases the battery level even if there is

a small amount of demand in the system at that time. At $t$ = 14, there is a drop due to a change in market price. All stations have the same battery charge.

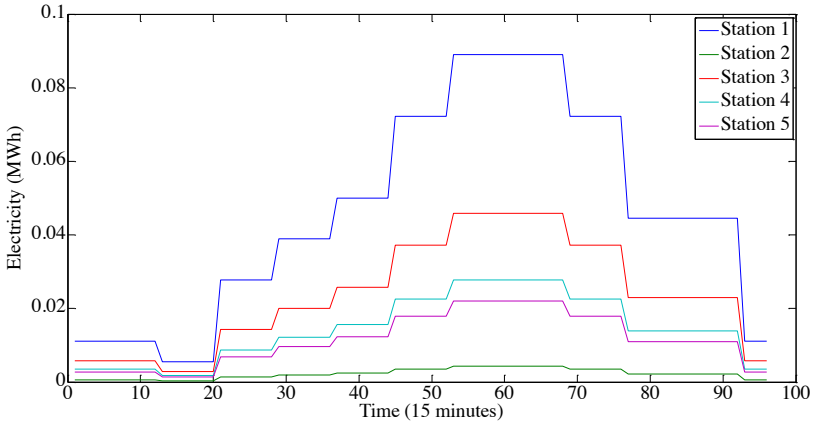



Figure 5.3: Total Demand

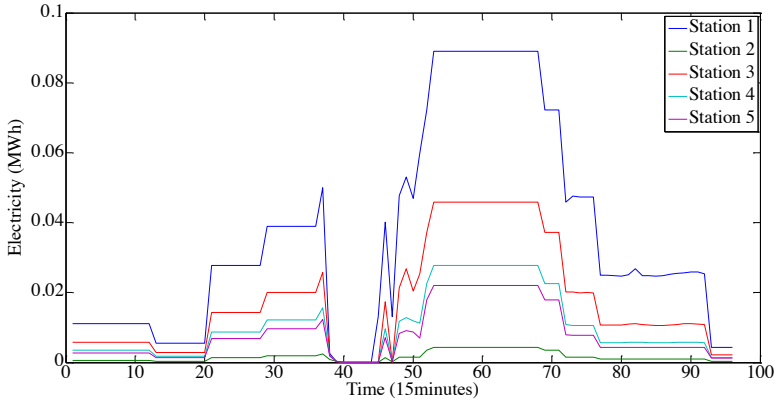



Figure 5.4: Demand Pulled from the Direct Charge

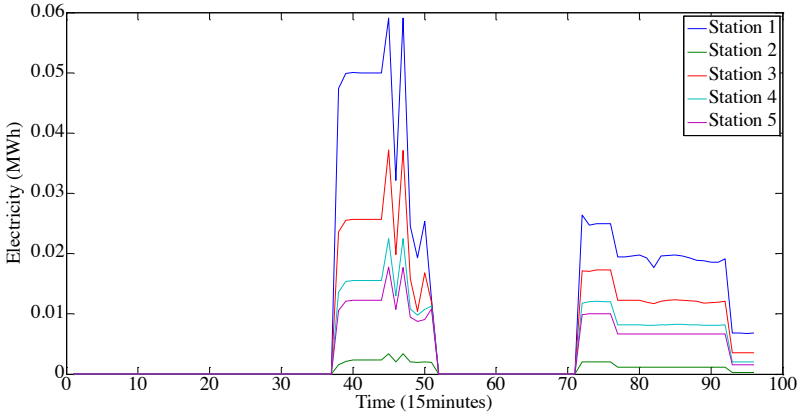



Figure 5.5: Demand Pulled from the Battery

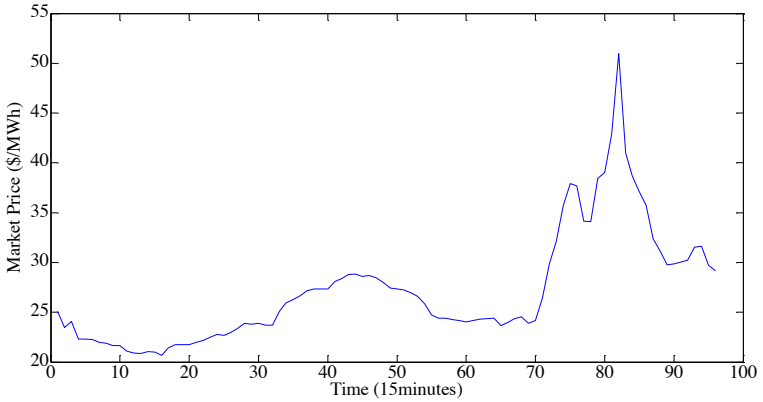



Figure 5.6: Energy Market Price

In this system, there are two ways to serve the total demand. The first way is by the direct charge, as shown in Figure 5.4. The other way is by the battery, which is shown in Figure 5.5. Since the beginning of time period, the demand is satisfied by direct charge until stage $t$ = 38. At that time, the market price is increased. Thus, the system takes advantage by serving the demand by some energy stored in the battery. At time period $t$ = 52, the market price is reduced, and the demand is supplied by the direct charge again. At time period $t$ = 71, the peak market price occurs. Thus, the system decides to serve

the demand by energy stored in the battery as much as it can. However, due to the limit on charging rate and the amount of electricity in the battery, the system still needs to serve some demand through direct charge. The total demands at each station are 17.800, 0.871, 9.166, 5.566 and 4.398 MWh, respectively.

From Figure 5.8, solar generation has a small impact on the system. However, the data input in this simulation uses January data set. The electricity sold back to the grid is mainly generated by wind power, see Figure 5.7. Figure 5.9 shows the allocation of wind generation to each station. The system mainly allocated wind energy to station 1 where the highest demand occurs.

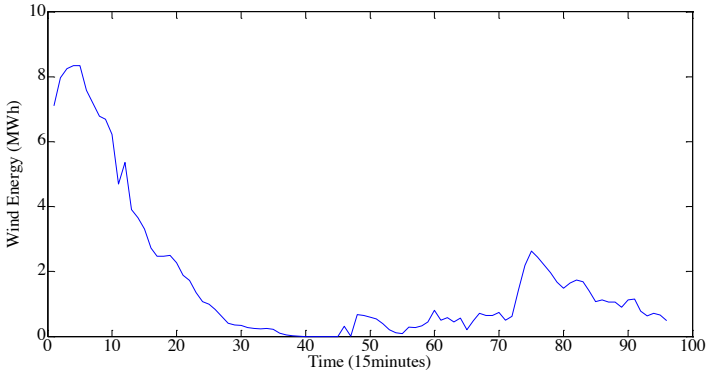



Figure 5.7: Total Wind Purchase to the System

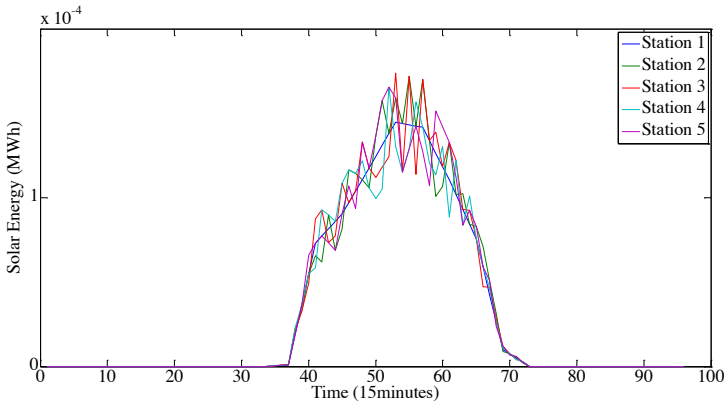



Figure 5.8: Solar Generation

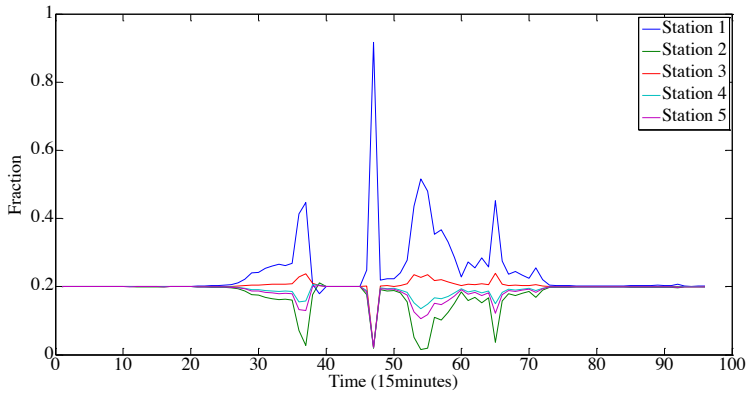



Figure 5.9: Wind Fractional Allocation

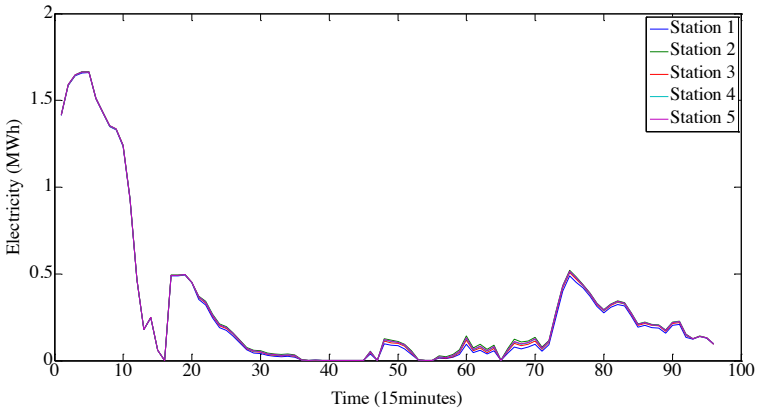



Figure 5.10: The Electricity Sold from Direct Charge

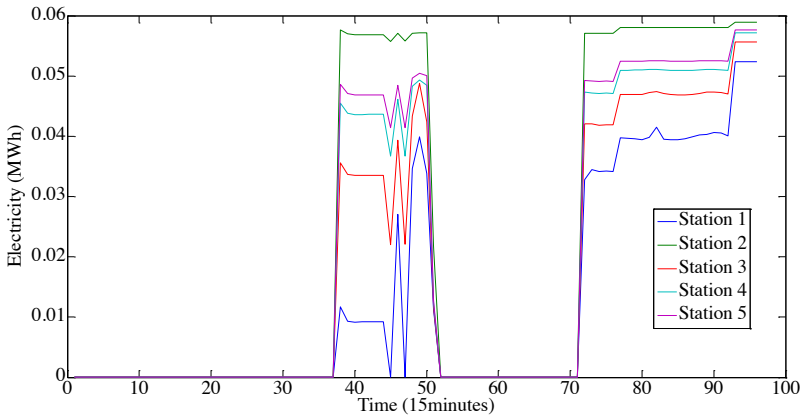



Figure 5.11: The Electricity Sold from the Battery

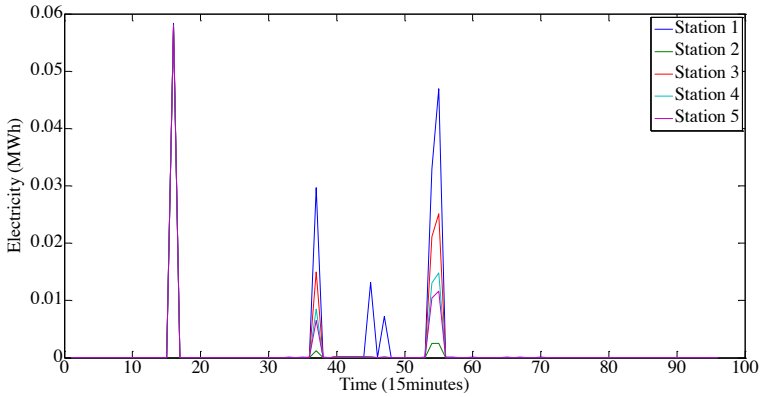



Figure 5.12: The Electricity Bought from the Power Grid

The electricity sold back to the grid from direct charge is similar to the total wind energy purchased to the system, except in time periods between 11 and 16 when the low market price occurs, Figure 5.10. The system decides to sell some energy from the battery back to the grid when the market price is high. However, the demand must be satisfied first. Thus, sometime when we have some demand in the system but the direct charge energy is not enough, it is necessary to purchase some energy from the grid even if the market price is not low, Figure 5.12.

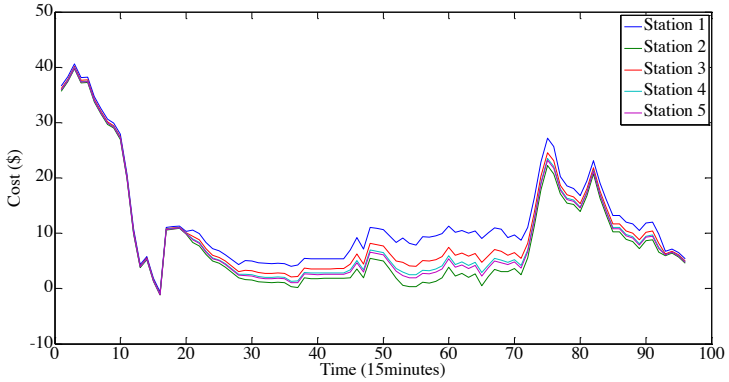



Figure 5.13: The Objective Function Output

The objective function is calculated by equation (5.3) with 5 stations and 96 time periods. The maximum profit over the 96 time periods is $4933.7.

Results from the mean value problem suggest that the system takes advantage of the low market price in the morning and uses direct charge from the wind and the grid to store energy in the battery before peak demand occurs. Once the system has satisfied all demand for the day, the remaining stored electricity is sold back to the grid at the peak market price. It is beneficial to use the direct charge from the wind, the utility grid, and solar to supply demand.

## 5.3 Generating the Experimental Design

The 11 clusters of the power grid (as shown in Figure 5.14) are used as potential station locations since a station location should be closed to the power grid. The total number of full factorial design points with 11 variables is 2048. This number is very large. Fractional factorial designs can be used in these cases to draw out valuable conclusions from fewer runs. In this case the lowest resolution design for a fractional factorial design is resolution-III, which requires 16 observations.

First, the full factorial design is generated. Sixteen observations are carefully selected from the full factorial design based on a fixing defining relation. In this study, every open station is assumed to have to same number of slots. To address the number of slots in the design, a block design is generated. Each block contains 16 observations with different defining relationships. The number of slots is from 1 to 10. Thus, the total number of blocks is 10, and the total number of observation is 160; see Appendix B for more details.

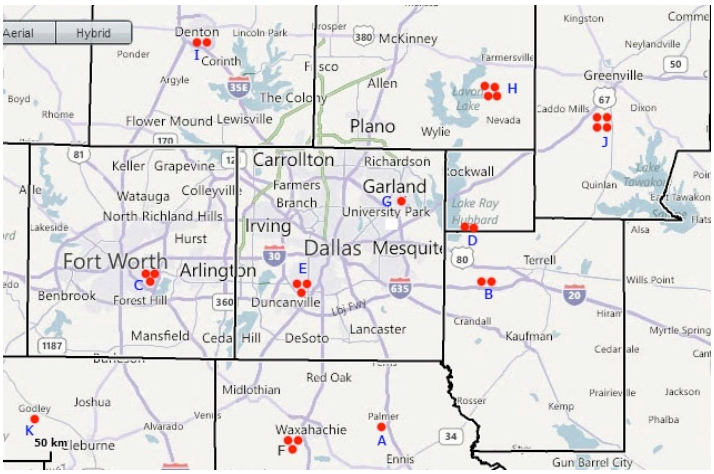



Figure 5.14: 11 Cluster of the Power Grid in DFW [109]

## 5.4 Optimization Model

The results from control problem presented in section 5.2.2 are used as the response given the first stage input variable from section 5.3. However, the benefit of serving demand is not included here. The objective function is to minimizing the operational cost, thus the objective function in equation (5.3) becomes

$$\min \sum_{t \in T} \sum_{j \in J} \left( \widetilde{C}_{tj} g_{tj}^{+} - \widetilde{B}_{t} (g_{tj}^{-} + R_{tj}) \right) \qquad \text{.....(5.13)}$$

The parameters of the simulation include,

- Discharging rate is 75 kW/slot.
- Maximum battery capacity is 3.6 MWh/slot.
- Charging rate is 600 kW /slot.
- Minimum level of battery capacity is 720 kWh/slot.
- 30% of total wind generation is assigned to the system.

A PHEV charging demand profile for all 9 counties in 2012 [109] is assigned to the nodes in Figure 5.14. 2012 historical data of wind, solar, and market price are used in the optimization model [111-113]. Perfect forecasting is assumed in this model.

## 5.5 Statistical Model

To estimate the relationship of the input variables and the response for controllability of PHEV charging stations, multiple linear regression is used. The predictors are binary variables representing the selection of open stations and numerical for the number of slots, and the response is the value function from the control problem.

### *5.5.1 Preliminary Multiple Linear Regression Model*

The multiple linear regression model can be expressed in the following form:

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5 + \beta_6 x_6 + \beta_7 x_7 + \beta_8 x_8 + \beta_9 x_9 + \beta_{10} x_{10} + \beta_{11} x_{11} + \beta_{12} Nslot + \varepsilon \quad .....(5.14)$$

where:

$Y$ = Value function from the deterministic LP for PHEV charging station (minimize controllability cost.)

$x_i$ = The binary variable $\begin{cases} 0 \text{ if station } i \text{ is not open} \\ 1 \text{ if station } i \text{ is open} \end{cases}$ where $i$ = 1,..,11

*Nslot* = Number of slots (*Nslot* is integer number started from 1 to 10)

$\varepsilon$ = Uncontrollable error

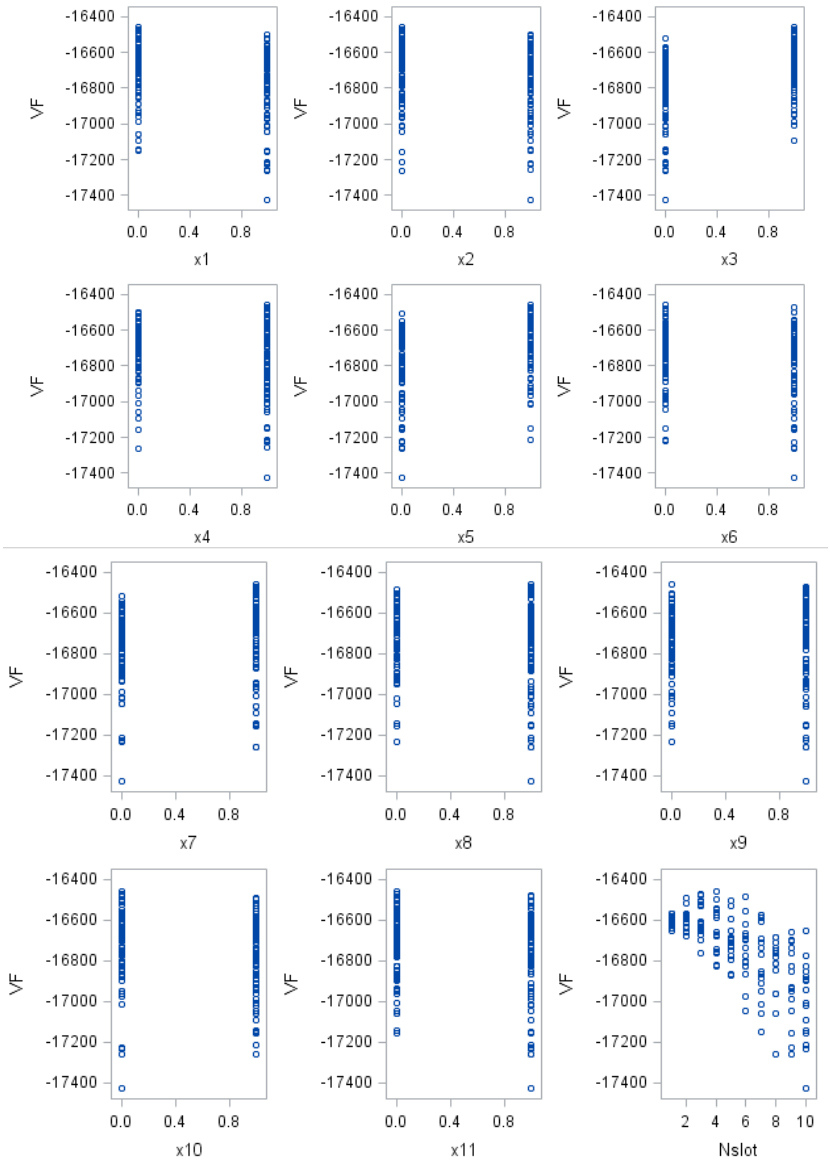



Figure 5.15: Response vs. Predictors Plots

From Figure 5.15, the trends for the binary variables are unclear, which may be attributed to possible interaction effects with the number of slots and each other. The trend for the number of slot seem to be mostly downward with some curvature. This implies that controllability improves as the number of slots increases.

5.5.1.1 Model fit

Using the least squares method, we can find the least squares estimates for the model parameters ( $\beta_i$ ), shown in equation (5.14). Using the parameter estimates shown in Table 5.1, an estimated regression function can be written as follows:

$$\hat{Y} = -16403 - 79.188x_1 - 57.47x_2 + 136.8x_3 - 57.71x_4 + 93.89x_5 - 68.84x_6 + 2.99x_7 - 63.86x_8 - 1.27x_9 - 57.93x_{10} - 51.72x_{11} - 42.73Nslot$$

.....(5.15)

Table 5.1: Analysis of Variance of Preliminary Model

| Source | DF | Sum of Squares | Mean Square | F Value | Pr > F |
|---|---|---|---|---|---|
| Model | 12 | 4896343 | 408029 | 75.36 | <.0001 |
| Error | 147 | 795952 | 5414.63886 | | |
| Corrected Total | 159 | 5692295 | | | |

| | | | |
|---|---|---|---|
| Root MSE | 73.58423 | R-Square | 0.8602 |
| Dependent Mean | -16745 | Adj R-Sq | 0.8488 |
| Coeff Var | -0.43945 | | |

| Parameter Estimates | | | | | | | |
|---|---|---|---|---|---|---|---|
| Variable | DF | Parameter Estimate | Standard Error | t Value | Pr > \|t\| | Type I SS | Variance Inflation |
| Intercept | 1 | -16403 | 22.75826 | -720.75 | <.0001 | 44861724773 | 0 |
| x1 | 1 | -79.18792 | 12.00546 | -6.60 | <.0001 | 265118 | 1.06409 |
| x2 | 1 | -57.47150 | 11.72288 | -4.90 | <.0001 | 177086 | 1.01458 |
| x3 | 1 | 136.80329 | 11.86177 | 11.53 | <.0001 | 818254 | 1.03942 |
| x4 | 1 | -57.71379 | 12.11904 | -4.76 | <.0001 | 216845 | 1.08432 |
| x5 | 1 | 93.89278 | 12.00783 | 7.82 | <.0001 | 301891 | 1.03704 |
| x6 | 1 | -68.83740 | 12.07588 | -5.70 | <.0001 | 322389 | 1.06364 |
| x7 | 1 | 2.98821 | 12.10021 | 0.25 | 0.8053 | 43524 | 1.06472 |
| x8 | 1 | -63.85852 | 11.72205 | -5.45 | <.0001 | 185921 | 1.01111 |
| x9 | 1 | -1.27058 | 11.84612 | -0.11 | 0.9147 | 1141.15708 | 1.03522 |
| x10 | 1 | -57.93194 | 11.99283 | -4.83 | <.0001 | 172666 | 1.06185 |
| x11 | 1 | -51.71785 | 11.77713 | -4.39 | <.0001 | 92225 | 1.02207 |
| Nslot | 1 | -42.73063 | 2.07361 | -20.61 | <.0001 | 2299281 | 1.04824 |

5.5.1.2 Model assumptions

- *Model Form*

To check on the linearity of the model, a plot of residuals ( $e_i$ ) vs. predictors is examined to determine if the model has curvature trends. The plot with *Nslot* in Figure 5.16 shows curvature. Hence, the model form is inadequate and will need to be addressed.

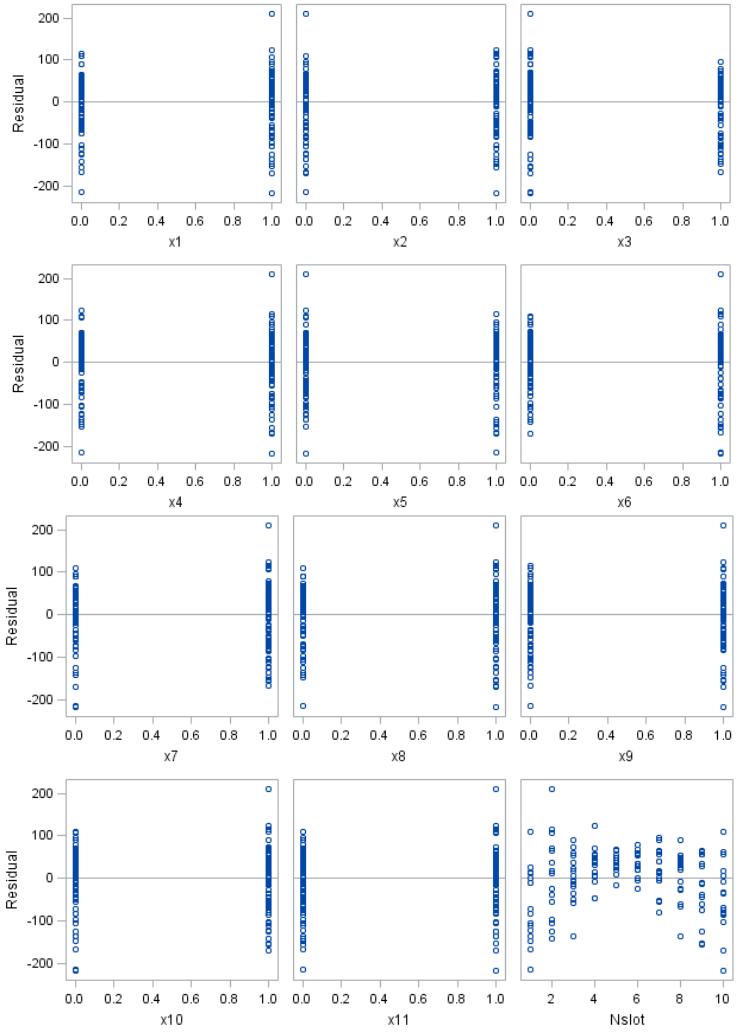



Figure 5.16: Residuals ( $e_i$ ) vs. Predictors Plots

- *Constant variance*

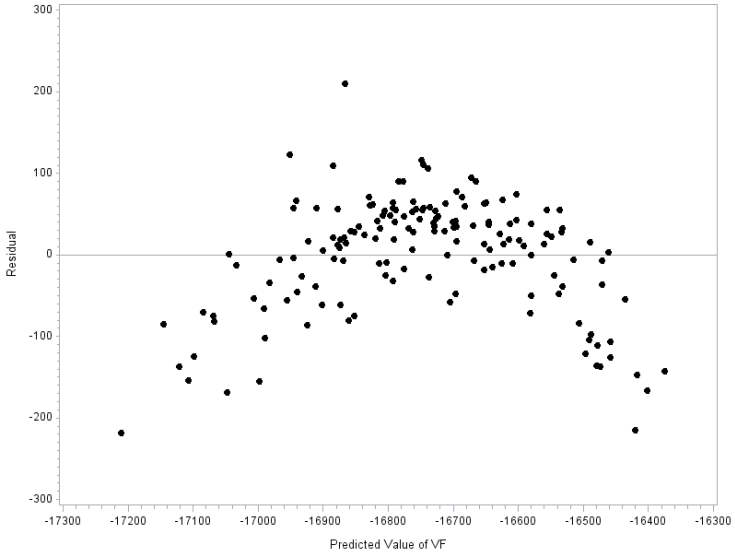



Figure 5.17: Residuals ( $e_i$ ) vs. Predicted Response( $Y_{hat}$ )

Figure 5.17 shows no indication of a “funnel-shape” along the curve. Hence, the constant error variance assumption appears to be satisfied.

- *Normality*

  To check on the normality of the errors, we can examine the normality plot shown in Figure 5.18.

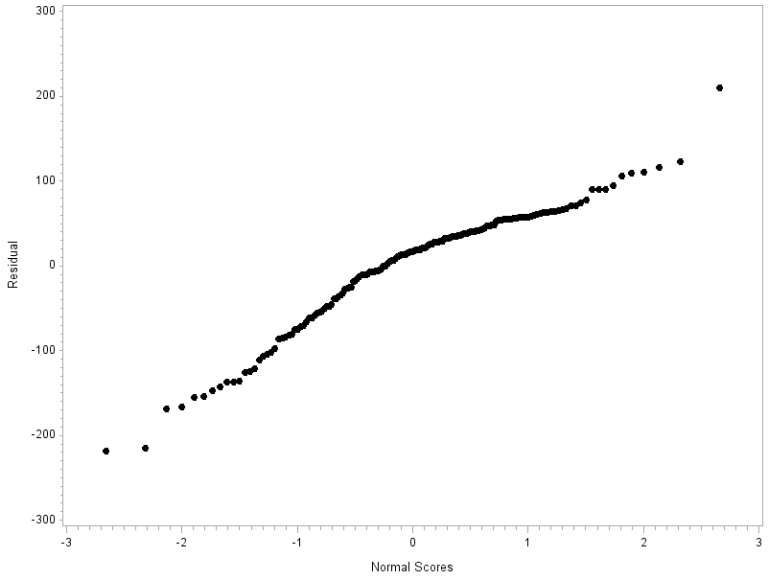



Figure 5.18: Normality Plot

The plot above shows longer tails than the normal distribution. By addressing the issue of curvature, it is possible the normality of the errors will improve.

5.5.1.3 Model summary remedial actions

$R^2$ in this model is 0.86, which high enough to have a good fit. However, there is a curvature trend for Nslot. For the remedial actions, to address the curvature, both the addition of regression spline basis functions and the addition of interaction terms are considered. There are two types of interaction terms. The first type is the interaction between each charging station location and the number of slots. The second type is the interaction between two stations where the distance between those stations is within 30 miles. Stations that are located farther than 30 miles apart are assumed to have no joint influence on controllability. Stepwise regression is used to select model terms at a 0.05 significance levels (alpha).

Salford system data mining and predictive analytics software version 7.0 for multivariate adaptive regression splines was employed to add regression spline basis

functions. The residuals from the stepwise regression model were entered into the software as the output variable data, and Nslot was entered as the input variable. The software identified knot locations, and the appropriate regression spline basis functions were then added to the stepwise regression model.

*5.5.2 Multiple Linear Regression Model with Stepwise Selection*

The multiple linear regression model with interaction terms after stepwise selection can be expressed in the following form:

$$Y = \beta_0 + \beta_1 x_3 + \beta_2 x_4 + \beta_3 x_5 + \beta_4 x_7 + \beta_5 x_9 + \beta_6 x_{10} + \beta_7 \text{int}_1 + \beta_8 \text{int}_2 + \beta_9 \text{int}_3 + \beta_{10} \text{int}_4 + \beta_{11} \text{int}_6 + \beta_{12} \text{int}_8 + \beta_{13} \text{int}_9 + \beta_{14} \text{int}_{10} + \beta_{15} \text{int}_{11} + \beta_{16} x_4 x_7 + \beta_{17} x_5 x_7 + \varepsilon \quad \text{.....(5.16)}$$

where:

$Y$ = Value function from deterministic LP for PHEV charging station (minimize controllability cost.)

$x_i$ = Binary variable $\begin{cases} 0 \text{ if station } i \text{ is not open} \\ 1 \text{ if station } i \text{ is open} \end{cases}$ where $i$ = 1, ..., 11

*Nslot* = Number of slots (*Nslot* is integer number started from 1 to 10)

$\text{int}_i$ = Interaction between $x_i$ and *Nslot*

$x_i x_j$ = Interaction between $x_i$ and $x_j$

$\varepsilon$ = Uncontrollable error

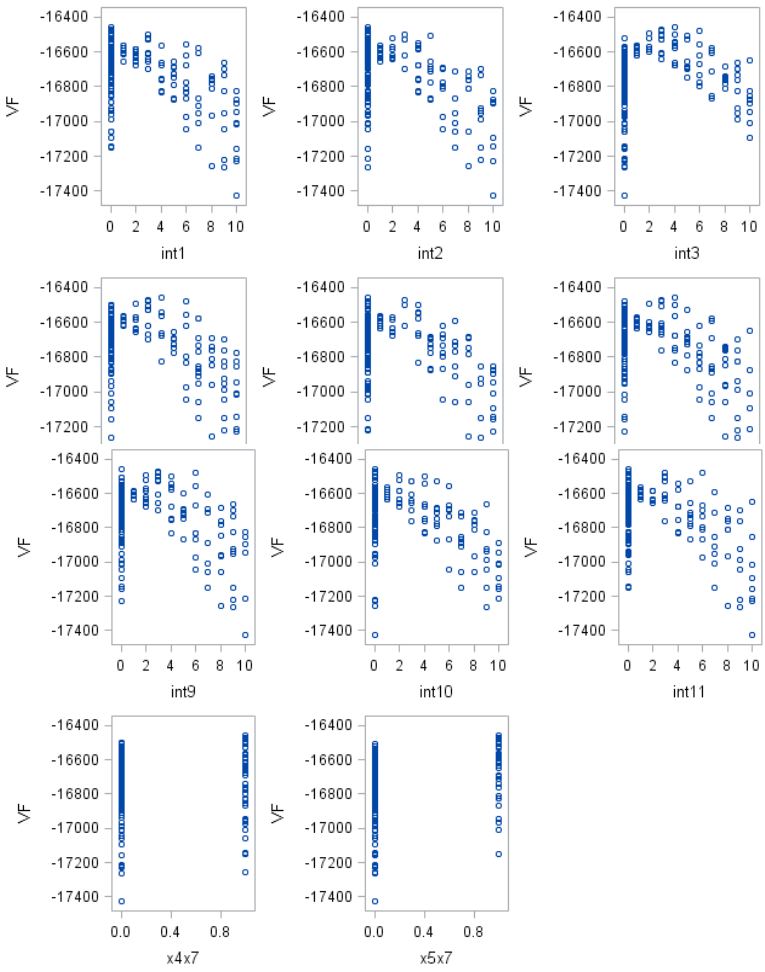



Figure 5.19: Value Function vs. Predictors Plots (Additional Variables)

5.5.2.1 Model fit

Table 5.2: Analysis of Variance of Multiple Linear Regression Model with Stepwise Selection

| Source | DF | Sum of Squares | Mean Square | F Value | Pr > F |
|---|---|---|---|---|---|
| Model | 17 | 5530921 | 325348 | 286.29 | <.0001 |
| Error | 142 | 161374 | 1136.43674 | | |
| Corrected Total | 159 | 5692295 | | | |

| | | | |
|---|---|---|---|
| Root MSE | 33.71108 | R-Square | 0.9717 |
| Dependent Mean | -16745 | Adj R-Sq | 0.9683 |
| Coeff Var | -0.20132 | | |

| Parameter Estimates | | | | | | | |
|---|---|---|---|---|---|---|---|
| Variable | DF | Parameter Estimate | Standard Error | t Value | Pr > \|t\| | Type I SS | Variance Inflation |
| Intercept | 1 | -16658 | 9.83147 | -1694.3 | <.0001 | 44861724773 | 0 |
| x3 | 1 | 33.28160 | 11.15431 | 2.98 | 0.0034 | 854692 | 4.37925 |
| x4 | 1 | 87.30797 | 14.04462 | 6.22 | <.0001 | 140575 | 6.93846 |
| x5 | 1 | 107.10327 | 8.35153 | 12.82 | <.0001 | 340461 | 2.39015 |
| x7 | 1 | 53.91809 | 9.71650 | 5.55 | <.0001 | 54353 | 3.27111 |
| x9 | 1 | 74.78673 | 11.06204 | 6.76 | <.0001 | 20983 | 4.30105 |
| x10 | 1 | 23.24154 | 10.95380 | 2.12 | 0.0356 | 240661 | 4.22059 |
| int1 | 1 | -13.24481 | 0.86404 | -15.33 | <.0001 | 1381005 | 1.27018 |
| int2 | 1 | -10.54077 | 0.87764 | -12.01 | <.0001 | 711963 | 1.25009 |
| int3 | 1 | 17.57749 | 1.73144 | 10.15 | <.0001 | 137560 | 4.91397 |
| int4 | 1 | -16.65515 | 1.72510 | -9.65 | <.0001 | 521778 | 5.36568 |
| int6 | 1 | -12.58907 | 0.88459 | -14.23 | <.0001 | 307116 | 1.28896 |
| int8 | 1 | -12.91885 | 0.85123 | -15.18 | <.0001 | 372187 | 1.18409 |
| int9 | 1 | -14.33202 | 1.76888 | -8.10 | <.0001 | 138218 | 5.05443 |
| int10 | 1 | -15.36016 | 1.71731 | -8.94 | <.0001 | 104421 | 4.80092 |
| int11 | 1 | -11.83942 | 0.90689 | -13.05 | <.0001 | 165463 | 1.31047 |
| x4x7 | 1 | -65.50052 | 12.24244 | -5.35 | <.0001 | 34075 | 4.08509 |
| x5x7 | 1 | -24.34918 | 11.16145 | -2.18 | 0.0308 | 5408.45014 | 3.17628 |

The estimated regression function can be written as follows:

$$\hat{Y} = -16658 + 33.28x_3 + 87.31x_4 + 107.1x_5 + 53.92x_7 + 74.79x_9 + 23.24x_{10} - 13.24\,\text{int}_1 - 10.54\,\text{int}_2 + 17.58\,\text{int}_3 - 16.66\,\text{int}_4 - 12.59\,\text{int}_6 - 12.92\,\text{int}_8 - 14.33\,\text{int}_9 - 15.36\,\text{int}_{10} - 11.84\,\text{int}_{11} - 65.5x_4x_7 - 24.35x_5x_7$$

.....(5.17)

5.5.2.2 Model assumptions

- *Model Form*

To check on the linearity of the model, a plot of residuals ($e_j$) vs. predictors is examined to determine if the model has curvature trends.

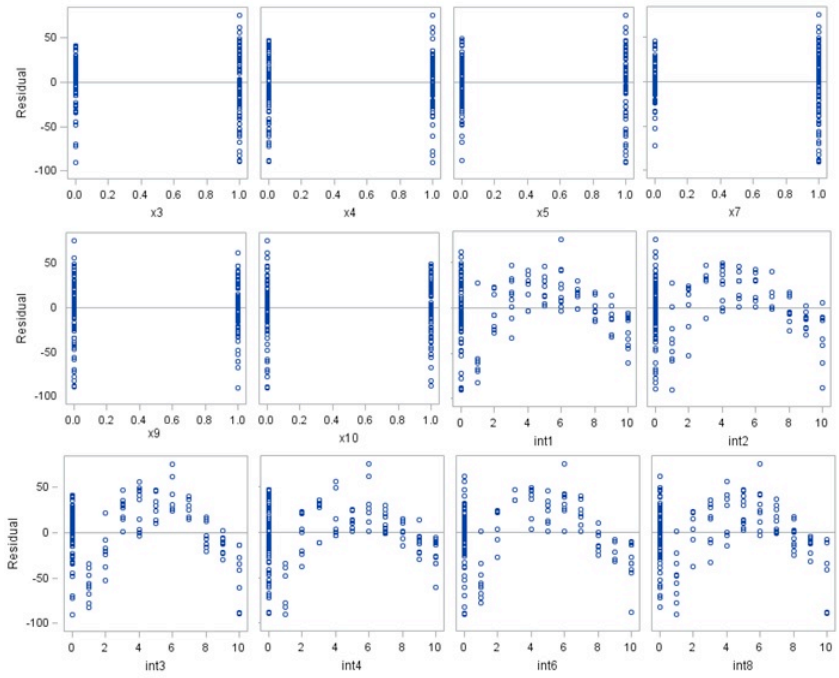



Figure 5.20: Residuals ($e_j$) vs. Predictors Plots

Curvature trend is still visible, so the current model form is still inadequate. Regression spline terms are yet to be added.

- *Constant variance*

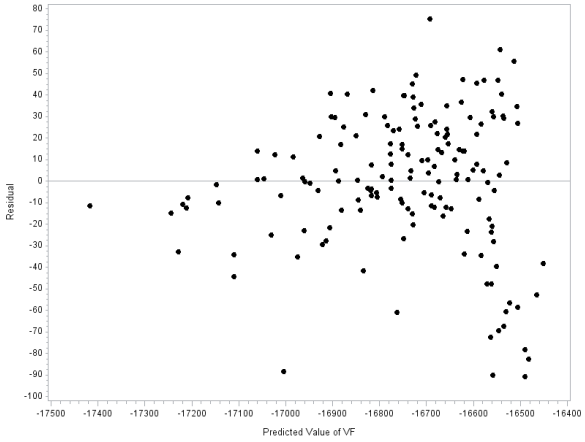



Figure 5.21: Residuals ( $e_j$ ) vs. Predicted Response ( $Y_{hat}$ )

Figure 5.21 shows a "funnel-shape," which indicates possible non-constant error variance. This is assessed again later.

- *Normality*

To check on the normality of the model we can examine the normality plot shown in Figure 5.22. The plot shows a longer left tail and a shorter right tail than the normal distribution. However it does show an improvement from Figure 5.18.

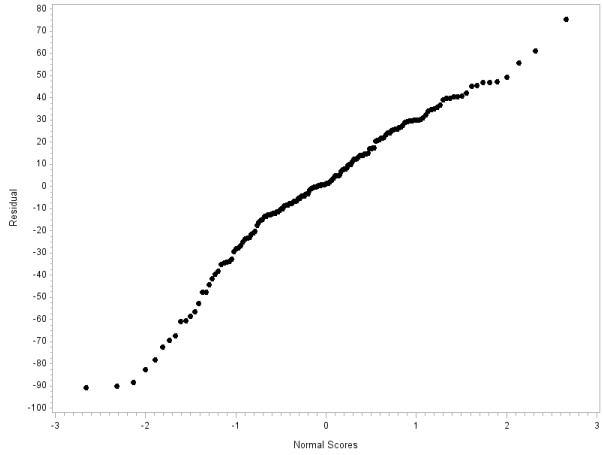



Figure 5.22: Normality Plot

5.5.2.3 Model summary

Several interaction terms were identified as statistically significant. There is an increase in $R^2$ from 0.86 to 0.97. To address the continued presence of curvature due to *Nslot*, the remedial action using regression spline basis functions is implemented next.

*5.5.3 Multiple Linear Regression Model with Stepwise Selection and Basis Functions*

The multivariate adaptive regression splines (MARS) software by Salford Systems was executed using with the residuals from the stepwise regression model and *Nslot*. Knots to model curvature were selected at *Nslot* values of 3, 4, and 6. The resulting multiple linear regression model with stepwise selection and basis functions can be expressed in the following form:

$$\begin{aligned} Y = {} & \beta_0 + \beta_1 x_3 + \beta_2 x_4 + \beta_3 x_5 + \beta_4 x_7 + \beta_5 x_9 + \beta_6 x_{10} + \beta_7 \,\mathrm{int}_1 + \beta_8 \,\mathrm{int}_2 + \beta_9 \,\mathrm{int}_3 + \beta_{10} \,\mathrm{int}_4 \\ & + \beta_{11} \,\mathrm{int}_6 + \beta_{12} \,\mathrm{int}_8 + \beta_{13} \,\mathrm{int}_9 + \beta_{14} \,\mathrm{int}_{10} + \beta_{15} \,\mathrm{int}_{11} + \beta_{16} x_4 x_7 + \beta_{17} x_5 x_7 \\ & + \beta_{18} Nslot + \beta_{19} BF_1 + \beta_{20} BF_2 + \beta_{21} BF_3 + \varepsilon \end{aligned}$$

.....(5.18)

where:

| | | |
|---|---|---|
| $Y$ | = | Value function from deterministic LP for PHEV charging station |
| $x_i$ | = | Binary variable $\begin{cases} 0 \text{ if station } i \text{ is not open} \\ 1 \text{ if station } i \text{ is open} \end{cases}$ where $i$ = 1, ..., 11 |
| *Nslot* | = | Number of slots (*Nslot* is integer number started from 1 to 10) |
| $\mathrm{int}_i$ | = | Interaction between $x_i$ and *Nslot* |
| $x_i x_j$ | = | Interaction between $x_i$ and $x_j$ |
| $BF_1$ | = | Regression spline basis function max(0, *Nslot*-3) |
| $BF_2$ | = | Regression spline basis function max(0, *Nslot*-4) |
| $BF_3$ | = | Regression spline basis function max(0, *Nslot*-6) |
| $\varepsilon$ | = | Uncontrollable error |

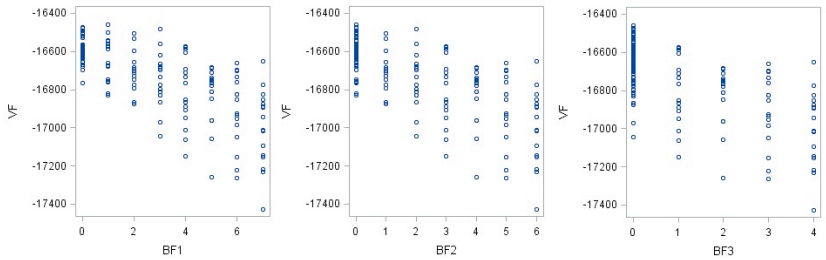



Figure 5.23: Value Function vs. Predictors Plots (Additional Variables)

5.5.3.1 Model fit

From table 5.3, the estimated regression function can be obtained as follows:

$$\hat{Y} = -16721 + 37.01x_3 + 62.73x_4 + 98.72x_5 + 42.74x_7 + 71.32x_9 + 15.7x_{10} - 13.07\,\text{int}_1 - 10.58\,\text{int}_2 + 17.38\,\text{int}_3 - 13.83\,\text{int}_4 - 12.26\,\text{int}_6 - 13.57\,\text{int}_8 - 14.23\,\text{int}_9 - 14.06\,\text{int}_{10} - 11.58\,\text{int}_{11} - 48.89x_4x_7 - 13.54x_5x_7 - 26.71BF_1 - 3.51BF_2 - 18.06BF_3 + 31.04Nslot$$

.....(5.19)

Table 5.3: Analysis of Variance of Multiple Linear Regression Model with Stepwise Selection and Basis Functions

| Source | DF | Sum of Squares | Mean Square | F Value | Pr > F |
|---|---|---|---|---|---|
| Model | 21 | 5627999 | 268000 | 575.21 | <.0001 |
| Error | 138 | 64296 | 465.91403 | | |
| Corrected Total | 159 | 5692295 | | | |

| | | | |
|---|---|---|---|
| Root MSE | 21.58504 | R-Square | 0.9887 |
| Dependent Mean | -16745 | Adj R-Sq | 0.9870 |
| Coeff Var | -0.12891 | | |

| Parameter Estimates | | | | | | | |
|---|---|---|---|---|---|---|---|
| Variable | DF | Parameter Estimate | Standard Error | t Value | Pr > \|t\| | Type I SS | Variance Inflation |
| Intercept | 1 | -16721 | 11.30759 | -1478.7 | <.0001 | 44861724773 | 0 |
| x3 | 1 | 37.01037 | 7.74951 | 4.78 | <.0001 | 854692 | 5.15588 |
| x4 | 1 | 62.73237 | 9.59623 | 6.54 | <.0001 | 140575 | 7.90103 |
| x5 | 1 | 98.71707 | 5.40113 | 18.28 | <.0001 | 340461 | 2.43838 |
| x7 | 1 | 42.73546 | 6.29290 | 6.79 | <.0001 | 54353 | 3.34670 |
| x9 | 1 | 71.32759 | 7.59152 | 9.40 | <.0001 | 20983 | 4.94084 |
| x10 | 1 | 15.69666 | 7.57421 | 2.07 | 0.0401 | 240661 | 4.92218 |
| int1 | 1 | -13.06580 | 0.56611 | -23.08 | <.0001 | 1381005 | 1.32998 |
| int2 | 1 | -10.58134 | 0.57421 | -18.43 | <.0001 | 711963 | 1.30526 |
| int3 | 1 | 17.37705 | 1.24534 | 13.95 | <.0001 | 137560 | 6.20060 |
| int4 | 1 | -13.83443 | 1.27022 | -10.89 | <.0001 | 521778 | 7.09566 |
| int6 | 1 | -12.25512 | 0.57580 | -21.28 | <.0001 | 307116 | 1.33212 |
| int8 | 1 | -13.57423 | 0.56009 | -24.24 | <.0001 | 372187 | 1.25037 |
| int9 | 1 | -14.23461 | 1.21747 | -11.69 | <.0001 | 138218 | 5.84020 |
| int10 | 1 | -14.06472 | 1.22566 | -11.48 | <.0001 | 104421 | 5.96496 |
| int11 | 1 | -11.57745 | 0.58786 | -19.69 | <.0001 | 165463 | 1.34308 |
| x4x7 | 1 | -48.88711 | 7.95112 | -6.15 | <.0001 | 34075 | 4.20302 |
| x5x7 | 1 | -13.54515 | 7.25795 | -1.87 | 0.0641 | 5408.45014 | 3.27601 |
| BF1 | 1 | -26.70870 | 9.90593 | -2.70 | 0.0079 | 15798 | 207.58014 |
| BF2 | 1 | -3.51028 | 9.69036 | -0.36 | 0.7177 | 46314 | 151.24013 |
| BF3 | 1 | -18.06311 | 4.64394 | -3.89 | 0.0002 | 6996.63473 | 14.81211 |
| Nslot | 1 | 31.03887 | 4.00604 | 7.75 | <.0001 | 27970 | 45.46732 |

5.5.3.2 Model assumptions

- *Model Form*

To check on the linearity of the model, a plot of residuals ( $e_i$ ) vs. predictors is examined to determine if the model have some curvature trends.

All plots in Figure 5.24 seem to show reasonable scatter. There is some minor curvature visible in the interaction term plots *Nslot* plot. However, for practical purposes, the model form appears reasonably adequate.

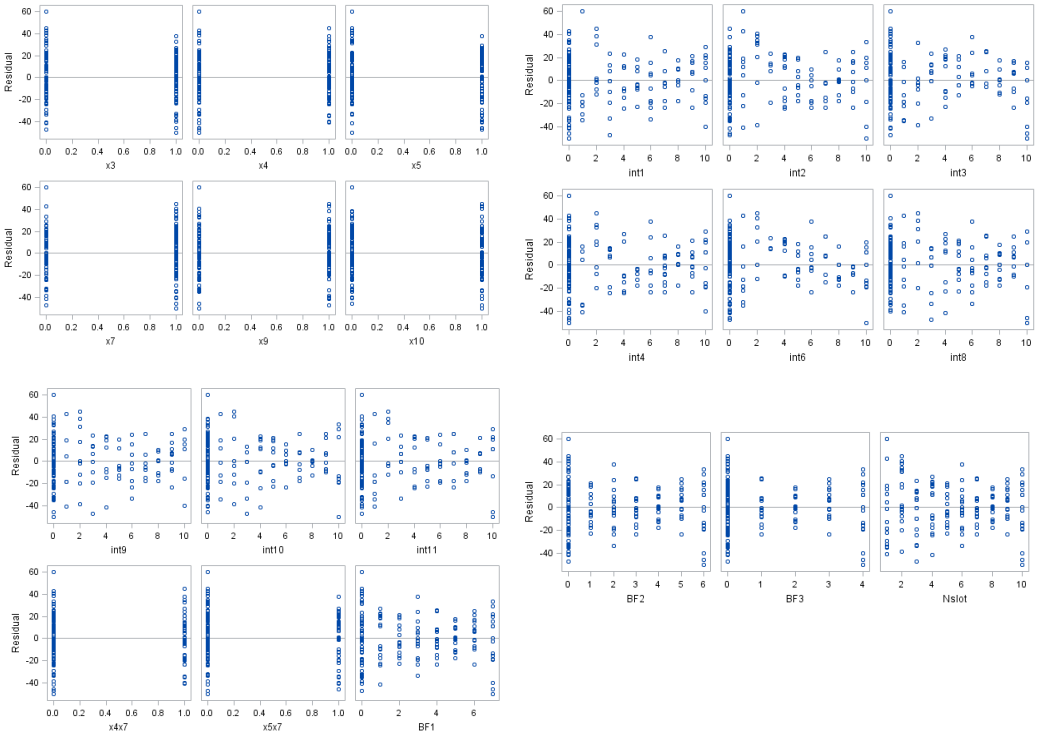



Figure 5.24: Residuals ( $e_i$ ) vs. Predictors Plots

- *Constant variance*

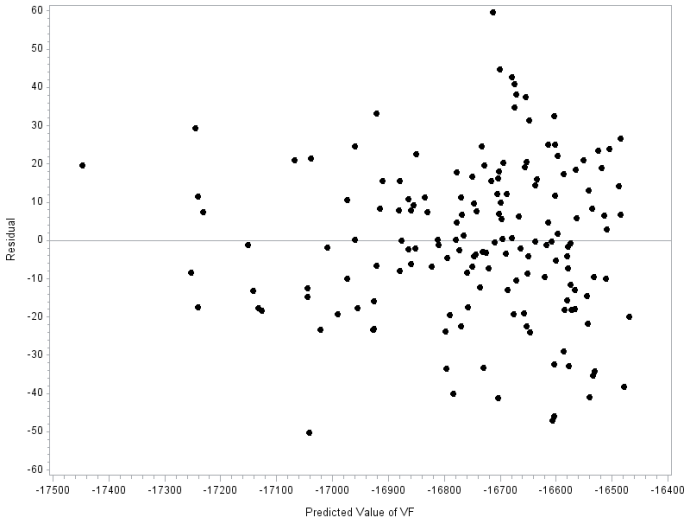



Figure 5.25: Residuals ( $e_i$ ) vs. Predicted Response ( $Y_{hat}$ )

The residuals vs. predicted values of the response is satisfied. Figure 5.25 shows a well scattered plot, indicating the constant error variance assumption is reasonable. To further verify the constant variance assumption, the Modified-Levene test was performed. Group 1 consists of the observations that have a $Y_{hat}$ value equal or less than mean value of $Y_{hat}$ -16744.72394, and group 2 consists of the observations that have a $Y_{hat}$ value greater than -16744.72394.

Table 5.4: T-test for Modified-Levene

| GROUP | N | Mean | Std Dev | Std Err | Minimum | Maximum |
|---|---|---|---|---|---|---|
| 1 | 67 | 13.6208 | 10.6519 | 1.3013 | 0 | 47.6149 |
| 2 | 93 | 17.2273 | 12.8541 | 1.3329 | 0 | 56.9003 |
| Diff (1-2) | | -3.6065 | 11.9835 | 1.9203 | | |

| GROUP | Method | Mean | 95% CL Mean | | Std Dev | 95% CL Std Dev | |
|---|---|---|---|---|---|---|---|
| 1 | | 13.6208 | 11.0226 | 16.2191 | 10.6519 | 9.1041 | 12.8387 |
| 2 | | 17.2273 | 14.5801 | 19.8746 | 12.8541 | 11.2350 | 15.0227 |
| Diff (1-2) | Pooled | -3.6065 | -7.3992 | 0.1863 | 11.9835 | 10.7953 | 13.4679 |
| Diff (1-2) | Satterthwaite | -3.6065 | -7.2863 | 0.0734 | | | |

| Method | Variances | DF | t Value | Pr > \|t\| |
|---|---|---|---|---|
| Pooled | Equal | 158 | -1.88 | 0.0622 |
| Satterthwaite | Unequal | 154.85 | -1.94 | 0.0547 |

| Equality of Variances | | | | |
|---|---|---|---|---|
| Method | Num DF | Den DF | F Value | Pr > F |
| Folded F | 92 | 66 | 1.46 | 0.1077 |

T-test: $H_0$: Means of $d_{i1}$ and $d_{i2}$ populations are equal

$H_1$: Means of $d_{i1}$ and $d_{i2}$ populations are not equal

Assume: $\alpha$ = 0.05 Decision Rule: Reject $H_0$ if $|t^*| > t(1-\alpha/2; n-2)$

$t(1-0.05/2; 160-2) = t(0.975, 158) = 1.975$

Since, $t^* = -1.88$, $|t^*| = 1.88 < t(0.975; 158) = 1.975$

Conclusion: It is failed to reject $H_0$. Therefore, non-constant error variance is not detected, and together with the plot, it is concluded that the constant error variance assumption is reasonable.

- *Normality*

To check on the normality of the model we can examine the normality plot shown in Figure 5.26. The plot above shows extremely strong linearity, indicating a strong match between the residuals and normality.

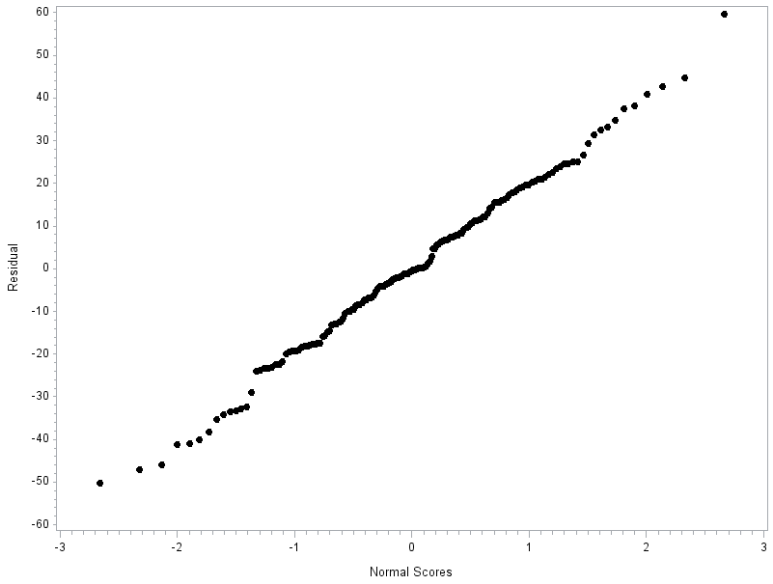



Figure 5.26: Normality Plot

Test for normality $H_0$ : Normality is OK

$H_1$ : Normality is violated

The decision rule is to reject $H_0$ if $\hat{\rho} < c$

Table 5.5: Normality Test

| Simple Statistics | | | | | | | |
|---|---|---|---|---|---|---|---|
| Variable | N | Mean | Std Dev | Sum | Minimum | Maximum | Label |
| e | 160 | 6.1618E-12 | 20.10916 | 9.8589E-10 | -50.17021 | 59.75040 | Residual |
| enrm | 160 | 0 | 0.99259 | 0 | -2.66059 | 2.66059 | Normal Scores |

| Pearson Correlation Coefficients, N = 160<br>Prob > \|r\| under H0: Rho=0 | | |
|---|---|---|
| | e | enrm |
| e | 1.00000 | 0.99781 |
| Residual | | <.0001 |
| enrm | 0.99781 | 1.00000 |
| Normal Scores | <.0001 | |

From table 5.5, let $\alpha = 0.01$, $\hat{\rho} = 0.99781$

$c\ (\alpha, n)$ = c(0.01,160) ~ 0.982

Since $\hat{\rho}$ > $c$ (0.99781 > 0.982), then we fail to reject $H_0$. Together with the normality plot, it is concluded that the assumption of normally distributed errors is reasonable.

5.5.3.3 Model summary

The R-squared of the final model is 0.9887, which implies a very good fit. The model has constant variance and normality. There is minor curvature still visible in some of the residual vs. predictor plots; however, a more complex model to closely address this curvature is impractical.

## 5.6 Discussion on the Final Model

In the two-stage framework, DACE was developed to replace time-consuming computer models or expensive physical experiments by replacing the loop between first and second stage with a constraint generated from the gradient of the approximation function [53-54]. It also allows more complicated recourse functions if appropriate.

The final model or metamodel gives some accessible descriptions to the control problem based on the design variables. The objective function is to minimize the operational cost. The profit is only by selling the electricity back to the grid, as in the benefit of serving demand is not included. Negative parameter estimators in equation (5.19) means that the system pays less in term of operational cost when there is less demand, and the system could benefit by selling the surplus energy back to the grid at the peak market price. For example, at node E, which is located in Dallas, there is a lot of demand in the system, so the system has little excess electricity to sell back to the grid. Thus, the system needs to buy some electricity from the grid to satisfy the demand.

Some stations that are located nearby would have joint influence on controllability. From equation (5.19), it is beneficial to build at both Stations E and G, and Stations D and G together.

# Chapter 6

## Summary and Future Work

The prospect of the proposed sequential DACE algorithms in enabling a nonparametric statistical modeling method for solving an infinite horizon approximate dynamic programming framework seems promising. The proposed sequential algorithm exploits the propensity of MARS modeling to follow the consistency trace, whereas the MARS modeling algorithm provides an explicit complexity measure (the number of basis functions) that can be controlled directly during the model building stage.

In chapter 3, three different stopping conditions of the DP stage iteration are described, and two of them are presented in the proposed algorithms. A two-sided t-test for 45-degree line correspondence is considered as one of stopping rules of the DP algorithm to check for the convergence to help find a near optimal solution quickly. From Figure A-2, it shows that the algorithm that passed the t-test (both origin point and slope) yielded better results from the forward simulation. The results in Chapter 3 did not include the L-infinity norm as one of the stopping conditions, but the simulation results in Appendix A show that the L-infinity norm may not be a good choice for a stopping rule. Future research can be done to study the stopping rule to ensure convergence. In theory, at the steady-state equilibrium, the future value function has converged. The results from the forward simulation in Appendix A shows that at the point where it passed a two-sided t-test for the 45-degree line correspondence rule, the algorithm may not be at steady-state. From Figure A-1, the steady-state equilibrium is reached after stage iteration 1000. In practice, the steady-state point may take a long time to be reached, but it may come up with little change in the result. It can be seen from the results that the mean cost at DP stage iteration 98 is close to the one at the real steady stage point but takes much less time to find. Thus, DP stage iteration 98 can be considered as a good place to stop.

Future research will study the width of $b_0$ and $b_1$ from equation (3.13) and (3.20) as a condition to ensure convergence.

The results of the sequential algorithms in Chapter 3 show that at the very beginning of the DP stage iterations, where the value function is far away from the true value function at the steady-state, we may consider a small number of state points. However, a good fit must be confirmed by a high value of the $R^2$. The percent change in the future value function average (as defined in Chapter 3) and $R^2$ can be used as a condition to stop exploring the state space. In this way, it can reduce computational time and give a good result more quickly. The results in this dissertation suggested that the Data loop condition when *minimum* $R^2$ is 0.8 and either *minimum change of the future value function average* is 10% or *maximum change in* $R^2$ is 0.005 gives the best result compared with the other two algorithms. Future research can be done to better values of *$R^2$, change in $R^2$, and change of the future value function average.*

In Chapter 4, batch mode reinforcement learning with MARS is presented. The main questions of this *Q*-iteration algorithm are how to generate the state-action space and how many training data points. Monte Carlo sampling is used to generate the state space, and a full factorial design is used to generate the action space. The results from the forward simulation presented in Chapter 4 show that adding more state-action data points does not improve the performance or give a better result. Moreover, the online mode reinforcement learning is presented to test the amenability of *Q*-iteration to follow a consistency trace when increasing the size of the input data. The results of the online mode fail to follow a consistency trace. Sobol' sequence design is the other method used to generate a state space. The results reveal that different action spaces have only a small impact on the results when using the same state space generated by Sobol' sequence. Moreover, in the worst case, Sobol' sequence design gives better result, but

Monte Carlo sampling gives the better result on average. Another comparison done in Chapter 4 is how to solve for the future *Q*-function. The results reveal that an optimization technique gives better results with lower computational time, compared with the traditional look-up table. Future research can be done to use the *Q*-function (state-action data set as an input) instead of *V*-function (state data set as an input) in the proposed algorithm presented in Chapter 3. In addition, the future work can study the effect of different action spaces to the results when using a state space generated by Monte Carlo sampling. Another interesting direction for future work is to find a good sampling sizes for the training sets as the input of the *Q*-function approximation.

Chapter 5 presents an application of a control problem for a system of PHEV charging stations. The main purpose of this NSF supported project is to develop a two-stage framework that integrates the first-stage system design problem and the second-stage dynamic system control problem. In this dissertation, a design and analysis of computer experiments (DACE) approach is applied to build a metamodel for the expected value function of the second stage for solving a two-stage framework problem. After that, the expected value function of the second stage will be included in the objective of the first-stage problem. As an initial solution analysis, the mean value problem is formulated as a deterministic linear program and solved. The results give potential policies that provide insight into the behavior of the system, i.e. how can the system take advantage of peak or low market price and when is the best time to charge the battery. The metamodel gives an assessable description to the control problem based on the design variables; for example, which station yields more profit and which two stations should be built together. In the future work, this problem will be formulated as an infinite-horizon stochastic dynamic programming, because it is assumed to have stationary system dynamics and many time periods. The proposed method in Chapter 3 can be applied to this application.

The future work can study more accurate metamodels for the expected value function of the second stage with more complex first stage design variables.

# Appendix A

## Comparison on the Stopping Conditions to Ensure the Convergence of An Infinite Horizon Dynamic Programming

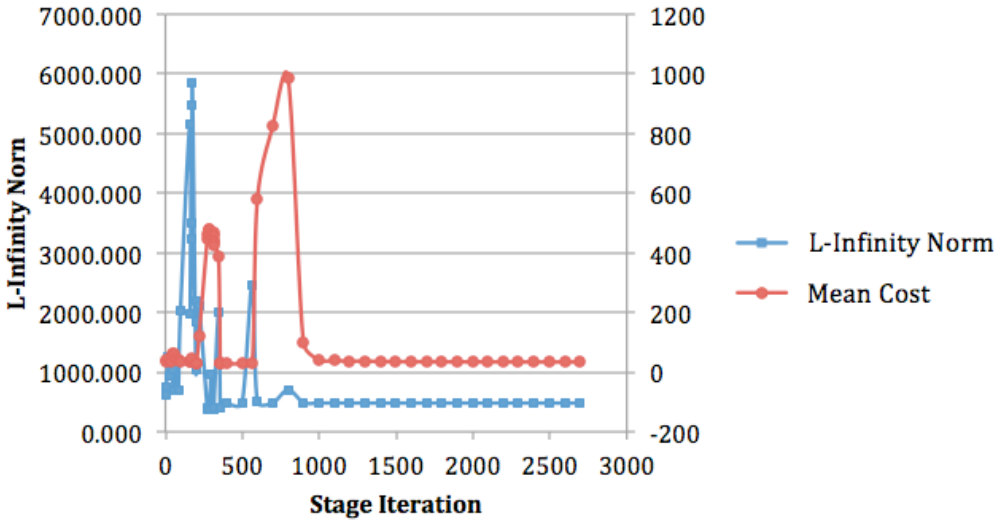
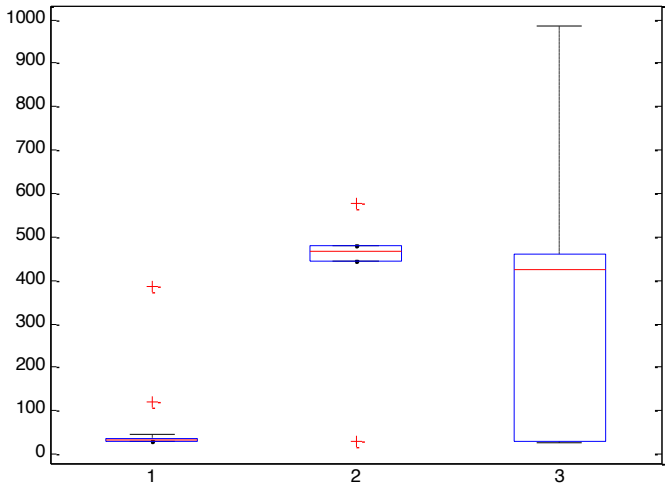



Figure A-1: L-Infinity Norm and Mean Cost vs. Stage Iteration

Figure A-1 is the results of running the output from Algorithm-II in forward simulation discussed in Chapter 3. There are three important issues appearing in Figure A-1. First, at the beginning of DP stage iteration, the values of the L-Infinity Norm were high and unstable. At those periods, the values of the mean cost in the forward simulation are inconsistent. Second, when the values of the L-Infinity Norm dropped, the values of mean cost went up. At DP iteration around 400, the value function iteration seemed to be at steady state equilibrium as the values of the L-Infinity Norm were low and stable as well as the values of mean cost. However, after DP stage iteration 520, the mean cost went up to the peak until the problem reaches steady state point after DP stage iteration 980. The mean cost at the true steady state equilibrium is 30.

Using a confidence interval t-test for 45-degree line correspondence, the problem stopped at DP stage iteration 98 where the mean cost is 33.5. In theory, we want to solve the system until we reach the steady state point where the value function does not (or slightly) change anymore. But in practical, we want to get the steady state point and want

to speed up the computational time. Even if the DP stage iteration 980 gives a better result from the forward simulation, when we compare the number of iterations and computational time, stage iteration number 98 is more attractive (each DP stage iteration used 196.36 seconds in average).




1- Pass both t-tests, 2- Pass either one, 3- Do not pass both t-tests

Figure A-2: Boxplot from the Result of the T-test

Figure A-2 presents Boxplots where number 1 is the mean cost of DP iterations that pass the confidence interval t-test for 45-degree line correspondence. The other two boxplots are ones that failed the t-test. From the plots, they show that boxplot number 1 gives the better result in terms of the mean cost and a consistent result. There are some outliers visible in the plot but they are 10% of the total and their values are still less than the other two mean values. Thus, it can be concluded that using confidence interval t-test for 45-degree line correspondence as an infinite DP condition helps ensure a good result.

# Appendix B

## Resolution-III Fractional Factorial Designs for PHEV Charging Station Case Study

## (Partial Design)

The resolution-III with 11 variables requires 16 observations. To address the number of slot variables, from 1 to 10 slots, a block design is generated. Each block contains 16 observations with different defining relations. Thus, the total number of block is 10, and the total number of observation is 160. Tables 1 and 2 are examples of the block design generated from different defining relation (I). The last column (Nslot) is the number of slots, which is the same at every open station.

Table B.1: Block-I Defining Relation I=DHK=1

| Run | A | B | C | D | E=ABC | F=BCD | G=ACD | H=ABD | J=ABCD | K=AB | L=AC | Nslot |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | |
| 2 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | |
| 3 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | |
| 4 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | |
| 5 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | |
| 6 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | |
| 7 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | |
| 8 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 9 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | |
| 10 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | |
| 11 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | |
| 12 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | |
| 13 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | |
| 14 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | |
| 15 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | |
| 16 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | |

Table B.2: Block-II Defining Relation I=AEGH=1

| Run | A | B | C | D | E=ABC | F=BCD | G=ACD | H=ABD | J=ABCD | K=AB | L=AC | Nlost |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | |
| 2 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | |
| 3 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | |
| 4 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | |
| 5 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | |
| 6 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | |
| 7 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | |
| 8 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 2 |
| 9 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | |
| 10 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | |
| 11 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | |
| 12 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | |
| 13 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | |
| 14 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | |
| 15 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | |
| 16 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | |

References


1. R. E. Bellman. *Dynamic Programming.* Princeton, NJ: Princeton University Press; 1957.
2. I. King. A Simple Introduction to Dynamic Programming in Macroeconomic Models. Economics Department, The University of Auckland. 2002.
3. D. P. Bertsekas. Dynamic Programming: An Overview. In *Proceedings of Sixth International Conference on Chemical Process Control*, (J. B. Rawlings, B. A. Ogunnaike, and J. W. Eaton, eds.) 1995.
4. W. R. Scott and W. B. Powell. Approximate Dynamic Programming for Energy Storage with New Results on Instrumental Variables and Projected Bellman Errors. Princeton University, *Technical Report.* 2013.
5. R. N. Anderson, W. B. Powell and W. Scott. Adaptive Stochastic Control for the Smart Grid. *Proceeding of IEEE V.99, Issue 6*. 2011.
6. J. M. Nascimento, and W. B. Powell. An Optimal Approximate Dynamic Programming Algorithm for The Lagged Asset Acquisition Problem. *Mathematics of Operations Research* 34(1), 210237. 23. 2009.
7. M.P. O'Keefe and T. Marke. Dynamic Programming Applied to Investigate Energy Management Strategies for a Plug-In HEV. Conference Paper. *The 22nd International Battery, Hybrid and Fuel Cell Electric Vehicle Symposium and Exhibition*, Yokohama, Japan 2006.
8. E. Yan, F. Bai. Application of Dynamic Programming Model in Stock Portfolio under the Background of the Subprime Mortgage Crisis. Intern*ational Journal of Business and management*, vol. 4, No.3, March 2009.

9. Y. Cai, K. L. Judd, T. S. Lontzek, V. Michelangeli, and C. Su. Nonlinear Programming Method for Dynamic Programming. *National bureau of economic research*, Cambridge, MA. 2013.

10. D. L. Kelly and C. D. Kolstad. Solving Infinite Horizon Growth Models with an Environmental Sector. *Computational Economics*, Society for Computational Economics, vol. 18(2), p. 217-31, October 1999.

11. S. C. Graves, H. Meal, S. Dasu, Y. Qin. Two-Stage Production Planning in A Dynamic Environment. (S. Axsater, C. Schneeweiss, E. Silver, eds.). *Multi-Stage Production Planning and Control*. Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, 9-43. 1986.

12. M. Carter. Mathematical Economics. *The MIT Press*, 2001.

13. L. Cooper, and M. W. Cooper. *Introduction to Dynamic Programming*. Pergamon Press, NY. 1981.

14. P. Berling and V. martinez-de-Albeniz. Optimal Expediting Decisions in a Continuous-Stage Serial Supply Chain. IESE Business School, University of Navarra, Spain, 2011.

15. W. B. Powell. Approximate Dynamic Programming. *Wiley,* New York, 2007.

16. V. C. P. Chen. Application of MARS and Orthogonal Arrays to Inventory Forecasting Stochastic Dynamic Programs. *Computational Statistics and Data Analysis*, 30, pp. 317-341. 1999.

17. H. A. Taha. Operation Research: An Introduction 7th Edition. *Pearson Education International.* 2003.

18. Yang, Z., V. C. P. Chen, M. E. Chang, T. E. Murphy, and J. C. C. Tsai. Mining and Modeling for a Metropolitan Atlanta Ozone Pollution Decision-

Making Framework. *IIE Transactions*, Special Issue on Data Mining, 39(6), pp. 607–615. 2007.

19. Z. Yang,V. C. P. Chen, M. E. Chang, M. L. Sattler, and A. Wen. A Decision-Making Framework for Ozone Pollution Control. *Operations Research*, 57(2), pp. 484–498. 2009.
20. H. Fan, P. K. Tarun, V. C. P. Chen, D. T. Shih, J. M. Rosenberger, S. B. Kim, and D. Bergman. Data-Driven Optimization for Minimizing the Environmental Impact of Airport Deicing Activities. *COSMOS Technical Report* 12-06. 2012.
21. N. V. Sule, V. C. P. Chen, and M. L. Sattler. A Decision-Making Framework for Assessing Control Strategies for Ground Level Ozone. *Atmospheric Environment*, 45, pp. 4996–5004. 2011.
22. C. F. Lin, A. K. LeBoulluec, L. Zeng, V. C. P. Chen, and R. J. Gatchel. An Adaptive Pain Management Framework. *Health Care Management Science*. 2013.
23. M. Scott. Applied Stochastic Processes in Science and Engineering. The University of Waterloo. 2013.
24. G. Lindgren, H. Rootzen, and M. Sandsten. Stationary Stochastic Processes for Scientists and Engineers. *CRC Press*. Taylor & Francis Group. 2013.
25. M. hauskrecht. Value-Function Approximations for Partially Observable Markov Decision Processes. *Journal of Artificial Intelligence Research* 13 (2000) 33-94. 2000.
26. J. Si, A. G. Barto, W. B. Powell, D. Wunsch. *Handbook of Learning and Approximate Dynamic Programming.* New York, NY: Wiley. 2004.

27. P. J. Werbos. Beyond Regression: New Tools for prediction and Analysis in the Behavioral Sciences, PhD thesis. Harvard University. 1974.

28. P. J. Werbos. Back Propagation and Neurocontrol: A Review and Prospectus. *Neural Networks* pp. 209-216. 1989.

29. C. Cervellera, V. C. P. Chen, A. Wen. Optimization of a Large-Scale Water Reservoir Network by Stochastic Dynamic Programming with Efficient State Space Discretization. *European Journal of Operational Research*; 171:1139-1151. 2006.

30. R. Sutton, and A. Barto. Reinforcement Learning: An Introduction. Cambridge, MA: *MIT Press*. 1998.

31. F. Guenter, M. Hersch, S. Calinon, and A. Billard. Reinforcement Learning for Imitating Constrained Reaching Movements. *RSJ Advanced Robotics*, Special Issue on Imitative Robots, vol. 21, num. 13, p. 1521-1544. 2007.

32. J. Si, A. G. Barto, W. B. Powell, D. Wunsch. Handbook of Learning and Approximate Dynamic Programming. New York, NY: *Wiley*. 2004.

33. P. J. Werbos. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences, Ph.D. dissertation, Committee on Applied Mathematics, Harvard University, 1974.

34. J. M. Lee, J. H. Lee. Approximate Dynamic Programming Strategies and Their Applicability for Process Control: A Review and Future Directions. *International Journal of Control, Automation, and System*; 2(3):267-278. 2004.

35. L. P. Kaelbling, M. L. Littman, A. W. Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research;* 4:237-285. 1996.

36. P. J. Werbos. Approximate Dynamic Programming for Real-time Control and Neural Modeling. In: *Handbook of Intelligent Control*, (eds., D. A. White and D. A. Sofge). New York, NY: Van Nostrand Reinhold. p. 493-525. 1992.

37. P. J. Werbos. Using ADP to Understand and Replicate Brain Intelligence: The Next Level design. In: *Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning* (ADPRL 2007). p. 209-216. 2007.

38. D. Ernst, M. Glavic, F. Capitanescu, L. Wehenkel. Reinforcement Learning Vs. Model Predictive Control: A Comparison on a Power System Problem. *IEEE Transactions on Systems, Man, and Cybernetcis-Part B: Cybernetics*; 39(2): 517-529. 2009.

39. C. E. Rasmussen, C. Williams. Gaussian Processes for Machine Learning. Cambridge, MA: *MIT Press.* 2006.

40. C. E. Rasmussen, M. Kuss. Gaussian Processes in Reinforcement Learning. Advances in Neural Information Processing Systems. Cambridge, MA: *MIT Press*. 2004.

41. A. Guez, R. D. Vincent, M. Avoli, and J. Pineau. Adaptive Treatment of Epilepsy via Batch-mode Reinforcement Learning. *In Proceedings of the 20th Conference on Innovative Applications of Artificial Intelligence* (IAAI). Chicago, IL. 2008.

42. D. Ormoneit and S. Sen. Kernel-Based Reinforcement Lerning. *Mach Learn.,* vol. 49, no. 2/3, pp. 161-178, Nov. 2002.

43. P. J. Werbos. Advanced Forecasting Methods for Global Crisis Warning and Models of Intelligence." *General Systems Yearbook*, 22, pp. 25-38. 1977.

44. P. J. Werbos. ADP: Goals, Opportunities and Principles. *Handbook of Learning and Approximate Dynamic Programming (IEEE Press series on Computation Intelligence)*, (J. Si, A. G. Barto, W. B. Powell and D. Wunsch, eds.). NY: *Wiley-IEEE Press*. pp. 3-44. 2004.

45. D. Ernst, P. Geurts and L. Wehenkel. Tree-based Batch Mode Reinforcement Learning. *Journal of Machine Learning Research* 6 503-556, 2005.

46. D. P. Bertsekas, and D. A. Castanon. Adaptive Aggregation Methods for Infinite Horizon Dynamic Programming. *IEEE Transactions on Automatic Control*, Vol. 34, No. 6. 1989.

47. C. Zhang and J. S. Baras. A New Adaptive Aggregation Algorithm for Infinite Horizon Dynamic Programming. *Center for Satellite and Hybrid Communication Networks Technical Report 2001-5,* 2001.

48. Z. Wen, L. J. Durlofsky, B. Van Roy, and K. Aziz. Approximate Dynamic Programming for Optimizing Oil Production. Chapter 25 in *Reinforcement Learning and Approximate Dynamic Programming for Feedback Control*, edited by F. L. Lewis and D. Liu, Wiley-IEEE Press, 2012.

49. M. He, L. Zhao, and W. B. Powell. Approximate Dynamic Programming Algorithms for Optimal Dosage Decisions in Controlled Ovarian Hyperstimulation. *European Journal of Operational Research*, Volume 222, Issue 2, 16 October 2012, Pages 328-340. 2012.

50. S. A. Johnson, J. R. Stedinger, C. A. Shoemaker, Y. Li, and J. A. Tejada-Guibert. Numerical Solution of Continuous-State Dynamic Programs Using Linear and Spline Interpolation." *Operations Research*, 41, pp. 484-500. 1993.

51. W. Whitt. Approximations of Dynamic Programs I. *Mathematics of Operations Research*; 3:231-243. 1978.

52. V. C. P. Chen, D. Ruppert, and C. A. Shoemaker. Applying Experimental Design and Regression Splines to High-Dimensional Continuous-State Stochastic Dynamic Programming." *Operations Research*, 47, pp. 38-53. 1999.

53. J. Sacks, W. J. Welch, T. J. Mitchell and H. P. Wynn. Design and Analysis of Computer Experiments. *Statistical Science*, 4, pp. 409-423, 1989.

54. V. C. P. Chen, K. L. Tsui, R. R. Barton, and M. Meckesheimer. Design, Modeling, and Applications of Computer Experiments. *IIE Transactions*, 38, pp. 273–291. 2006.

55. C. Cervellera, A. Wen, and V. C. P. Chen. Neural Network and Regression Spline Value Function Approximations for Stochastic Dynamic Programming." *Computers and Operations Research*, 34, pp. 70-90. 2007.

56. V. C. P Chen, K. L. Tsui, R. R. Barton, and J. K. Allen. A review of design and modeling in computer experiments. In *Handbook of Statistics: Statistics in Industry*, (R. Khattree and C. R. Rao, eds.), 22, Elsevier Science, Amsterdam, pp. 231-261. 2003.

57. P. K. Tarun, V. C. P. Chen, H. W. Corley, F. Jiang. Optimizing Selection of Technologies in a Multiple Stage, Multiple Objective Wastewater Treatment System [Internet]. *Journal of Multi- Criteria Decision Analysis.* 2011.

58. H. Y. Fan, V. C. P. Chen. Adaptive Value Function Approximation for Continuous-State Stochastic Dynamic Programming. *COSMOS Technical Report.* 2010.

59. J. C. C. Tsai, V. C. P. Chen, J. Chen, M. B. Beck. Stochastic Dynamic Programming Formulation for a Wastewater Treatment Decision-Making Framework. *Annals of Operations Research*, Special Issue on Applied Optimization under Uncertainty; 132: 207-221. 2004.

60. S. Dreyfus. Richard Bellman on the Birth of Dynamic Programming. *Operations Research* 50 (1), pp. 48–51. 2002.

61. Z. Yang, V. C. P. Chen, M. E. Chang, M. L. Sattler, A. Wen. A Decision-Making Framework for Ozone Pollution Control. *Operations Research*; 57(2): 484-498. 2009.

62. B. Ariyajunya, V. C. P. Chen, S. B. Kim. Orthogonalized Dynamic Programming State Space for Efficient Value Function Approximation. In *Proceedings of the 2010 IE Research Conference.* Cancun, Mexico: 2010.

63. H. Fan. Sequential Frameworks For Statistics-Based Value Function Representation In Approximate Dynamic Programming. PhD Dissertation, University of Texas at Arlington. 2008.

64. S. Sahu. Multivariate Adaptive Regression Spline Based Framework for Statistically Parsimonious Adaptive Dynamic Programming. PhD Dissertation, The University of Texas at Arlington. 2011.

65. V. L. Pilla. Robust Airline Fleet Assignment. PhD Dissertation, The University of Texas at Arlington. 2006.

66. M. Petrik, and B. Scherrer, Biasing Approximate Dynamic Programming with a Lower Discount Factor. *NIPS,* page 1265-1272. Curran Associates, Inc., 2008.

67. M. L. Puterman. Markov Decision Processes. *Wiley*, New York, 1994.

68. D.P. Bertsekas and J.N. Tsitsiklis. *Neuro-Dynamic Programming*, *Athena Scientific*. 1996.

69. J. Lee, and J. H. Lee. Approximate Dynamic Programming Strategies and Their Applicability for Process Control: A Review and Future Directions. *International Journal of Control*, Automation, and Systems, vol. 2, no. 3, pp. 263-278: 2004.

70. A. Wen. Statistics-Based Approach to Stochastic Optimal Control Problems, PhD Dissertation, The University of Texas at Arlington. 2005.

71. S. Ferrari, and R. F. Stengel. Model-Based Adaptive Critic Designs. *Handbook of Learning and Approximate Dynamic Programming (IEEE Press series on Computation Intelligence)*, (J. Si, A. G. Barto, W. B. Powell and D. Wunsch, eds.). NY: *Wiley-IEEE Press*. pp. 65-95. 2004.

72. C. W. Anderson, M. Kretchmar, P. Young, and D. Hittle. Robust Reinforcement Learning using Integral-Quadratic. *Handbook of Learning and Approximate Dynamic Programming (IEEE Press series on Computation Intelligence)*, (J. Si, A. G. Barto, W. B. Powell and D. Wunsch, eds.). NY: *Wiley-IEEE Press*. pp. 337-358. 2004.

73. C. W. Anderson, D. Hittle, M. Kretchmar, and P. Young. Robust Reinforcement Learning for Heating, Ventilation, and Air Conditioning Control of Buildings." *Handbook of Learning and Approximate Dynamic Programming (IEEE Press series on Computation Intelligence)*, (J. Si, A. G. Barto, W. B. Powell and D. Wunsch, eds.). NY: *Wiley-IEEE Press*. pp. 517-534. 2004.

74. R. Saeks, C. Cox, J. Neidhoefer, P. Mays, and J. Murray. Adaptive Critic Control of A Hybrid Electric Vehicle. *IEEE Transaction on Intelligent Transportation Systems*, 3(4). 2002.

75. J. Si, D. Liu, and L. Yang. Direct Neural Dynamic Programming. *Handbook of Learning and Approximate Dynamic Programming (IEEE Press series on Computation Intelligence)*, (J. Si, A. G. Barto, W. B. Powell and D. Wunsch, eds.).NY: *Wiley-IEEE Press*. pp. 125-151. 2004.

76. S. D. Whitehead, R. S. Sutton, and D. H. Ballard. Advances in Reinforcement Learning and Their Implications for Intelligent Control. Department of Computer Science, University of Rochester, Rochester NY. 2007.

77. J. C. Medina, and R. F. Benekohal. Q-learning and Approximate Dynamic Programming for Traffic Control – A Case Study for an Oversaturated Network*. in Proceedings of the Boar Annual Meeting,* 2012*.*

78. I. O. Ryzhov and W. B. Powell. Approximate Dynamic Programming With Correlated Bayesian Beliefs. *in Proceedings of the 48th Allerton Conference on Communication, Control, and Computing.* 2010.

79. E. Foufoula-Georgiou,and P. K. Kitanidis. Gradient Dynamic Programming for Stochastic Optimal Control of Multidimensional Water Resources Systems. *Water Resources Research*, 24, pp. 1345-1359. 1988.

80. C. Cervellera, D. Macciò, and M. Muselli. Functional Optimization Through Semilocal Approximization Minimization. *Operations Research*, 58(5), pp. 1491-1504. 2010.

81. C. Cervellera, and D. Macciò. A Comparison of Global and Semi-local Approximization in T-stage stochastic optimization. *European Journal of Operational Research*, 208(2), pp. 109-118. 2011.

82. G. B. Dantzig, and P. Wolfe. Decomposition Principle of Linear Programs, *Operations Research 8(1),* 101-111. 1960.

83. E. Beale. On Minimizing a Convex Function Subject to Linear Inequalities. *The Royal Statistical Society 17b,* 173-184. 1955.

84. J. R. Birge, and F. Louveaux. Introduction to Stochastic Programming, 2nd Edition. SpringerSeries in Operations Research and Financial Engineering. 2011.

85. J. F. Benders. Partitioning Procedures for Solving Mixed-Variable Programming Problems. *Numerische Mathematik 54, 238–252. 1962.*

86. J.R. Birge, and F.V. Louveaux. A Multicut Algorithm for Two-Stage Stochastic Linear Programs. *European Journal of Operational Research 34, 384–392. 1988.*

87. A. Ruszczynski. A Regularized Decomposition Method for Minimizing a Sum of PolyHedral Functions. *Mathematical Programming 35, 309-333. 1986.*

88. A. Ruszczynski. Regularized Decomposition of Stochastic Programs: Algorithmic Techniques and Numerical Results. Working Paper. Department of Management Science, Rutgers University, Newark, NJ. 1993.

89. J. N. Hooker, and G. Ottosson. Logic-based Benders Becomposition. *Mathematical Programming 96 33-60. 2003.*

90. J.N. Hooker. Planning and Scheduling by Logic-Based Benders Decomposition. *Operations Research 55 588-602. 2007.*

91. S. Trukhanov, L. Ntaimo, and A. Schaefer. Adaptive Multicut Aggregation for Two-Stage Stochastic Linear Programs with Recourse. *European Journal of Operational Research* 206 395–406. 2010.

92. V. L. Pilla, J. M. Rosenberger, V. C. P. Chen, and B. C. Smith. A statistical computer experiments approach to airline fleet assignment. *IIE Transactions*. 40(5): 524-537. 2008.

93. V. L. Pilla, J. M. Rosenberger, V. C. P. Chen, N. Engsuwan, and S. Siddappa. A multivariate adaptive regression splines cutting plane approach for solving a two-stage stochastic programming fleet assignment model. *European Journal of Operational Research.* v. 216. issue 1. pp. 162–171. 2012.

94. T. Hastie, R. Tibshirani, J. Friedman. The Elements of Statistical Learning. *Springer*. New York, 2002.

95. J. H. Friedman. “Multivariate Adaptive Regression Splines” (with discussion), *Annals of Statistics*, 19, pp. 1–67, 1991.

96. S. Geman, E. Bienenstock, and R. Doursat,R. Neural networks and the bias/variance dilemma. *Neural Computation* 4, 1–58, 1992.

97. J. C. C. Tsai and V. C. P. Chen. Flexible and robust implementations of multivariate adaptive regression splines within a wastewater treatment stochastic dynamic program. *Quality and Reliability Engineering International*, 21, pp. 689-699, 2005.

98. G. L. Nemhauser. Introduction to Dynamic Programming. New York, NY: *Wiley*. 1966.

99. D. P. Bertsekas. Dynamic Programming and Optimal Control. Belmont, MA: *Athena Scientific*. 2001.

100. D. P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. II.* Belmont, MA: *Athena Scientific*, 2007.

101. J. Bibby. Aziomatisations of average and a further generalization of monotonic sequence. *Glasgow Mathematical Journal*, vol. 15, pp. 63-65. 1974.

102. M. H. Kutner, C. J. Nachtsheim, J. Neter and W. Li. Applied Linear Statistical Models. *McGraw-Hill.* 2005.

103. G. Jekabsons. ARESLab: Adaptive Regression Splines toolbox for Matlab/Octave, 2012. [Online]. Available: http://www.cs.rtu.lv/jekabsons/

104. I. M. Sobol. The distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics,* 7, pp. 784-802. 1967.

105. J. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2, pp. 84-90. 1960.

106. A. H. Ronald. Dynamic Programming and Markov Processes. *John Wiley & Sons*. 1960.

107. D. Wetz, “Energy Storage Needs and Options," Renewable Energy Sources, The University of Texas at Arlington., Arlington, TX, 2010.

108. S. Vazquez, S. M. Lukic, E. Galvan, L. G. Franquelo, and J. M. Carrasco. Energy Storage Systems for Transport and Grid Application. *IEEE Trans. Industrial Electronics*, vol.57, No.12, pp.3881-3895, 2010.

109. A. Khosrojerdi, M. Xiao, P. Sarikpruech, J. K. Allen and F. Mistree. Designing a System of Plug-in Hybrid Electric Vehicle Charging Stations. *Proceedings of the ASME 2013 International Design Engineering Technical*

*Conferences and Computers and Information in Engineering Conference*. Portland, Oregon, 2013.

110. *U.S. Department of Energy*. Electric Power Monthly March 2012 (with Data for January 2012). March, 2012. [Online]. Available: http://www.eia.gov/cneaf/electricity/epm/epm_sum.html

111. *National Renewable Energy Laboratory*. National Solar Radiation Data Base. August, 2012. [Online]. Available: http://rredc.nrel.gov/solar/old_data/nsrdb/

112. A. Miller and B. Lumby. Utility Scale Solar Power Plants: A Guide for Developers and Investors. *International Finance cooperation (IFC).* 2012.

113. *Electric Reliability Council of Texas*. ERCOT History. 2002. [Online]. Available: http://www.ercot.com/about/profile/history/

114. F. Huang, "Optimization of PHEV charging station. MS Thesis, Department of Electrical Engineering, University of Texas at Arlington, Arlington, TX, 2010.

115. J. Pineau, M. G. Bellemare, A. J. Rush, A. Ghizaru, S. A. Murphy. Constructing Evidence-Based Treatment Strategies using Methods from Computer Science. *Drug and Alcohol Dependence*; 88:S52-S60. 2007.

116. R. Demir. An Approximate Dynamic Programming Approach to Discrete Optimization. PhD Thesis. Department of Sloan School of Management. Massachusetts Institute of Technology, Sloan School of Management, Operations Research. 2000.

117. D. C. Heath, P. L. Jackson. Modeling the Evolution of Demand Forecasts with Application to Safety Stock Analysis in Production Distribution-Systems. *IIE Transactions* 26(3) 17-30. 1994.

## Biographical Information

Asama Kulvanitchaiyanunt received a B.S. in Chemical Engineering from Chulalongkorn University, Bangkok, Thailand, and a MS in Industrial Engineering from Lehigh University, Bethlehem, PA. She is a member of the Center on Stochastic Modeling Optimization and Statistics at the University of Texas at Arlington and an active member of INFORMS, IEEE, and IIE. During her doctoral studies at UTA, she has worked as a Graduate Research Associate with Dr. Victoria Chen, Dr. Jay Rosenberger and Dr. Wei-Jen Lee in the National Science Foundation (NSF) supported project on an adaptive design for controllability of a system of Plug-in Hybrid Electric Vehicle (PHEV) charging stations which is a part of her doctoral dissertation. Her research interests include mathematical programming, dynamic programming, data mining, machine learning, and optimization in energy application. In the future, she intends to engage herself in research activity in the areas of application of optimization and data mining to solve numerous real-world problems such as facility location, decision making in energy application and transportation.