STATISTICS-BASED APPROACHES TO STOCHASTIC OPTIMAL CONTROL PROBLEMS

by

AIHONG WEN

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

December 2005

To my husband Pingsheng and my daughter Jennifer.

ACKNOWLEDGEMENTS

No matter how hard I tried, I would not find adequate or appropriate words in a dictionary to thank my supervising professor Dr. Victoria C.P. Chen, for constantly motivating, encouraging, helping, understanding, and inspiring me and being patient with me. Her dedication to research, passion about teaching, and consideration as a mentor have established a model for my future career. Also, I thank her for introducing me to the study of data mining used in this disertation, which I found to be a very interesting field.

I wish to thank my academic advisors Dr. H.W. Corley, Dr. Chien-Pai Han, Dr. Farhad Kamangar, and Dr. Donald H. Liles for offering great classes during the entire course of my doctoral studies, showing interest in my research, and advising and taking time to serve on my dissertation committee. I especially would like to thank Dr.Corley for all his help beginning from the day I applied to UTA. He made my journey to UTA and my stay here full of care and encouragement. My special thanks also go to Dr. Cristiano Cervellera (Institute of Intelligent Systems for Automation - ISSIA-CNR National Research Council of Italy), Dr. Jay Rosenberger, Dr. Seoung Bum Kim, and Dr. Brian Huff for their generous instructions whenever I directed questions to them.

I am grateful to all the teachers who taught me during the years I spent in school, first in Jiangxi, China, then in Shanghai,China, and finally in Texas, USA.

I would like to extend my appreciation to my wonderful friends in the COSMOS lab – Huiyuan Fan, Heesu Hwang, Venkata Pilla, Prattana Punnakitikashem (instant helper), Dachuan Shih, Sheela Siddappa, Tai-Kuan Sung, Panita Suebvisai, Durai Sundaramoorthi (baby-sitter is one of thousands names I wanted to give to him), Prashant Tarun, and Siriwat Visoldilokpun. I am also grateful to the friends I met in the USA – Homer Burks, Ann Chen, Songhee Gardner, Hui-Chiao Jen, Jie Liu, Dung Nguyen,

Chang Song, Resto Sunarto, Don and Ingrid Thomas, and Jawahar Veera. They were always there for me and letting me resort with comfort when I had difficulties.

Finally, I would like to express my deep gratitude to my husband Pingsheng (Peter) Zeng and my daughter, Jennifer Zeng, who have been with me all my way and sacrificed far more in this endeavor than I have myself. I often drew inspiration from them, too. I am also extremely grateful to all the members of my family and family-in-law in China for their unflagging love and support given in many ways. I am extremely fortunate to be so blessed.

November 7, 2005

ABSTRACT

STATISTICS-BASED APPROACHES TO STOCHASTIC OPTIMAL CONTROL PROBLEMS

Publication No. _____

Aihong Wen, Ph.D.

The University of Texas at Arlington, 2005

Supervising Professor: Victoria C.P. Chen

This dissertation develops practical approaches to *stochastic optimal control* problems in the absence of Linear system transition functions, Quadratic cost functions, and/or Gaussian random disturbances (*LQG hypotheses*). In such type of problems, an analytical solution is impossible and numerical synthesis techniques have to be applied. The "classic" algorithms existing in the literature suffer from the problem of the "*curse of dimensionality*," an exponential increase of computation time and memory requirements as the dimension of the problem grows.

The *statistics-based* numerical approaches are presented as the main tools for mitigating the "curse of dimensionality" phenomenon. Two approaches are explored: (A) *stochastic dynamic programming* (SDP), which approximates the future value functions and solves the recursion relation backwards in stages, and (B) *stochastic gradient* (SG), which approximates the control functions by linear combinations of certain basis functions containing free parameters, and optimizes the parameters through iterations over a sequence of realizations of the system's random variables. The research presented in this dissertation views the approximation of future value functions in SDP and the approximation of control functions in SG via a computer experiments perspective, and integrates

statistical methods from the area of *design and analysis of computer experiments* (DACE) into SDP and SG approaches to enable numerical solution to large SOC problems.

Recent developments in DACE make it possible to approximate high-dimensional, complex input-output relationships with moderate computation time and memory requirements. A statistical perspective of future value function approximation in high-dimensional, continuous-state SDP was first presented using orthogonal array (OA) experimental designs and multivariate adaptive regression splines (MARS) statistical models. This work utilizes number theoretic methods (NTMs) and artificial neural networks (ANNs) as alternatives to OAs and MARS respectively, and introduces the statistical perspective to SG approach. Comparisons consider the differences in methodological objectives, model accuracy and numerical solutions are presented.

Three problems are tested: a nine-dimensional inventory forecasting problem, an eight-dimensional water reservoir network management problem, and a thirty-dimensional water reservoir network management problem. This last application is the largest water reservoir problem solved in the literature.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 The Stochastic Optimal Control Problem

Stochastic Optimal Control (SOC) considers the problem of optimizing the control of a stochastic dynamic system in the sense of minimizing the cost or maximizing the benefit resulted from the control scheme. The system evolves through stages, typically over time. The *state variables* describe the state of the system at a given stage, and the *control variables* are the decisions made over time to control the system. These variables may be subject to certain *constraints*. *Control functions* (control laws, or controllers) specify the relationship between state variables and control variables. The evolution of the states over time is defined by a *state transition equation* that is subject to uncertainty modeled by random variables (random disturbances), giving the system its stochastic nature. General desrciption of SOC can be found in Bryson and Ho (1975), Lewis and Syrmos (1995) and Brogan (1991).

SOC problems are important because they can model applications in many fields such as inventory forecasting, revenue management, water resources management, ozone pollution, etc. Two applications will be tested in this dissertation: an inventory forecasting problem and a water reservoir network management problem. In the inventory forecasting problem, the "cost" is the actual cost involved in holding inventory, having to backorder items, etc., and the constraints involve capacities for holding and ordering items. The state variables include the current inventories and forecasts for demand. The state transition equation updates these inventories and forecasts given customer demand and the arrival of ordered items. The demands from customers are random, and the number of items to be ordered at each stage must be chosen to minimize the inventory costs over several time stages. In a water reservoir network management problem, inflows from external sources (like rivers and rain) bring stochasticity to the system. The

states are the volumes of water in each basin and the external inflows to each basin. The dynamics for the single basin can be modeled by a state transition equation where the amount of water at the beginning of certain stage reflects the flow balance between the water that enters (upstream releases and stochastic inflows) and the water that is released during the previous stage. The amount of water to be released during a given time stage from each reservoir is chosen to minimize some possibly nonlinear cost (or maximize benefit) function related to the releases (e.g., power generation), while satisfying proper constraints, e.g., maximum pumpage capacities or target water level in each basin at the beginning of each stage.

In general, time can be modeled continuously or discretely (i.e., discrete stages of time), and the time horizon can be infinite or finite. In the case of infinite-horizon problems, the SOC solution obtained will be "steady-state" (not varying over time). State and control variables can be discrete or continuous. The space defined by the state variables dictates the size of the problem. If time is modeled with a finite set of discrete time stages and the state space is finite (i.e., the system can only take on a finite set of states), then in theory it is possible to enumerate all the *trajectories* of the states over time and find the optimal sequence of controls for each trajectory. However, this is only computationally practical for small state spaces. For example, if we consider only five states and three time stages, there will have 5*5*5 = 125 different trajectories. If the state space is continuous (or nearly continuous), which is the case with volumes of water or concentrations of ozone, then enumeration is not an option.

This dissertation focuses on the SOC problem formulation with *continuous states*, *continuous control variables*, *discrete time stages*, and a *finite horizon*. This is a practical formulation for the following reasons:

1. Many applications have continuous state and control spaces. Even discrete (numerical) spaces are quite large for real problems and are more efficiently represented by a near-continuous space.
2. Problems that can be mathematically formulated using continuous-time equations require time discretization for an actual implementation on a computer
3. In practice, planning is usually conducted over a finite stage of time, e.g., a year. In addition, solutions to infinite-stage SOC problems can be constructed in a cyclic fashion.

It is well known that SOC problems can be solved analytically only either in very simple cases, or when the assumption that the system transition functions are Linear, the cost functions are Quadratic, and the random disturbances are Gaussian (commonly referred to as *LQG hypotheses*). The absence of the LQG hypotheses renders analytical synthesis of the optimal controllers and state estimators impossible. Since the LQG hypotheses are often violated in realistic problems, it is necessary to develop approximate synthesis techniques.

Different approaches exist in the literature to provide approximate techniques for the synthesis of optimal controllers and state estimators. Among these are Dynamic Programming (DP) (Bertsekas 2000) and the Ritz method (Ritz 1909). However, the "classic" algorithms that can be found in literature suffer from the problem of the "*curse of dimensionality*," which is an exponential increase of computation time and memory requirements as the dimension of state space grows.

This dissertation develops and studies *statistics-based* approximate approaches as the main tools for mitigating the "curse of dimensionality" phenomenon, thus creating new *practical* methods for solving SOC problems. Two types of SOC solution approaches will be explored: (A) Stochastic Dynamic Programming (SDP), which employs a recursion relation to solve the SOC problem backwards in time, and (B) Stochastic Gradient

(SG), which iterates through a sequence of realizations of the system's random variables. The research presented in this dissertation will integrate statistical methods from *Design and Analysis of Computer Experiments* (DACE) into SDP and SG approaches to enable numerical solution to large SOC problems.

The first approach, solving SOC by SDP, is based on the high-dimensional continuous-state SDP work of Chen et al. (1999) and Chen (1999), which utilized *Orthogonal Array* (OA) experimental designs to *discretize* the continuous state space and a statistical model, *Multivariate Adaptive Regression Spline* (MARS) to *approximate* a function over the continuous state space. Their *statistical perspective* enabled the first truly practical numerical solution approach for high dimensional, continuous-state SDP problems. Their work is expanded in this dissertation by considering alternatives to OA discretization and MARS models. Referring to the area of statistical *design of experiments*, efficient discretizations constructed from OA-based Latin Hypercubes (OA-LHs) (Tang 1993) and Number-Theoretic Methods (NTMs) are studied. In particular, NTMs (also called Low-Discrepancy sequences or Quasi-Monte Carlo sequences, Hua and Wang 1981, Fang and Wang 1994, Niederreiter 1992) are attracting more and more attention due to their good "*space filling*" properties. Referring to the area of *statistical modeling*, *Artificial Neural Network* (ANN) models are considered in place of MARS. Although MARS has been applied in several areas (e.g., Carey and Yee 1992, Griffin et al. 1997, Kuhnert et al. 2000), ANN models are much more prevalent in all areas of engineering (e.g., Smets and Bogaerts 1992, Labossiere and Turkkan 1993, Abdelaziz et al. 1997, Chen and Rollins 2000, Li et al. 2000, Liu 2001, Pigram and Macdonald 2001, Kottapalli 2002). Given their popularity, an ANN-based SDP solution method would be more accessible to engineers.

The second approach solves the SOC through approximating the control functions. In this approach, the sequences of realizations of the stochastic variables must be generated, samples have to be drawn from initial state space if the requirement is to provide

control scheme for any initial state vector, and an appropriate model must be chosen to estimate the control functions. These are the analogs to the continuous state space discretization and model estimation in the SDP approach. Parisini and Zoppoli (1994) and Cervellera (2001) have used ANN as the control function approximator, while this dissertation introduces the *statistical perspectives* to their method.

1.2 Research Overview

The objectives of this work can be stated as:

1. SOC approaches via SDP:
   (a) Utilization of NTMs and OA-LHs to discretize the continuous state space.
   (b) Utilization of ANNs to approximate functions over the continuous state space.
   (c) Comparisons among (OA, OA-LH, NTM)/(MARS, ANN).
2. SOC approaches via SG
   (a) Utilization of experimental designs to generate the realizations of the stochastic variables and sample vectors from initial vector space.
   (b) Utilization of statistical modeling to approximate the control function.
   (c) Comparisons of solution quality of the algorithm with/without experiemental designs involved.
3. Application of the two SOC approaches to inventory forecasting and water reservoir network management problems.
4. Comparison of statistics-based SOC/SDP and SOC/SG approaches:
   (a) Computational requirements.
   (b) Solution quality.

Figure (1.1) shows the connections between research areas for the proposed work. Chapter (2) presents the SDP and SG algorithms, Chapter (3) discusses the related techniques in DACE and how they can be used in SDP and SG algorithms, Chapter (4)

describes the test applications, inventory forecasting and water reservoir network management problems, and shows the computational results for the test problems. Conclusions and furture work are presented in Chapter (5).

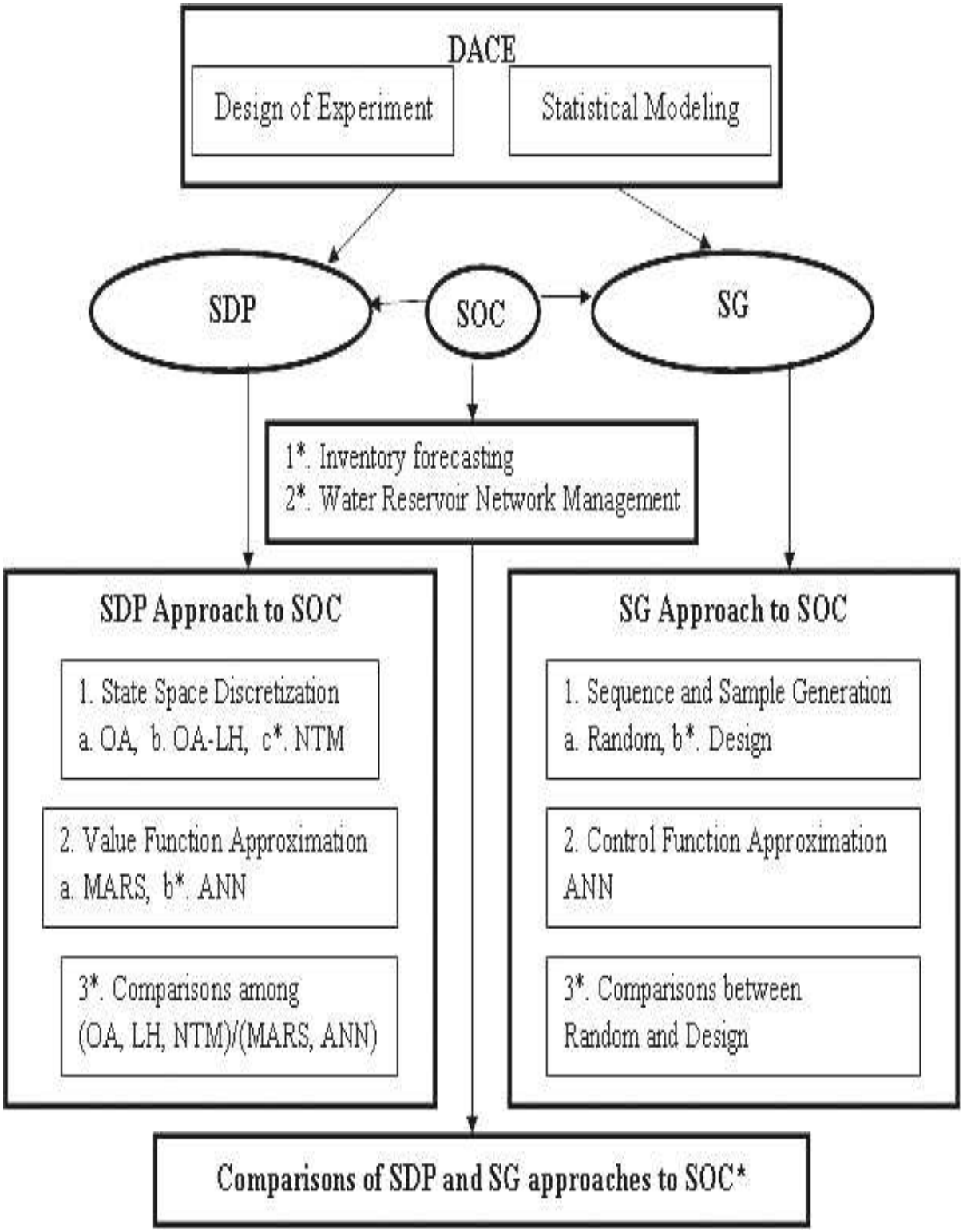

Figure 1.1. Diagram illustrating the connection between research areas. New contributions are indicated by asterisks (*).

CHAPTER 2

THE SOC PROBLEM AND SOLUTION ALGORITHMS

2.1 The SOC Problem Formulation

Suppose there are T stages in the system. Define at the beginning of stage t, $t = 0, \ldots, T-1$:

$\boldsymbol{x}_t \in R^n$	the vector of state variables for stage t.
$\boldsymbol{u}_t \in R^m$	the vector of control (or decision) variables for stage t.
$\gamma_t : R^n \to R^m$	the control function for stage t.
$\boldsymbol{\epsilon}_t \in R^l$	the random vector for stage t with known probability distribution which models the stochasticity in the system.
$c_t : R^{n+m+l} \to R$	the known cost function for stage t.
$\boldsymbol{U}_t \subset R^{n+m}$	the nonempty set of constraints on $\boldsymbol{u}_t$ which may depend on $\boldsymbol{x}_t$
$f_t : R^{n+m+l} \to R^n$	the state transition function for stage t.
$c_T : R^n \to R$	the known cost function for the last stage.

Note that as stated in Chapter 1, this work considers the SOC problems in absence some or all LQG hypotheses. Assume the random "disturbances" $\boldsymbol{\epsilon}_t$ are mutually independent and the control functions are of "closed-loop", i.e., the control vector at each stage to be a function of the current state vector:

$$\boldsymbol{u}_t = \gamma_t(\boldsymbol{x}_t), t = 0, \ldots, T-1.$$

Then a general T-stage SOC problem is to find a control law $\pi = (\gamma_0^*, \gamma_1^*, \ldots, \gamma_{T-1}^*)$ that minimizes the overall cost:

$$\begin{aligned} \min_{\gamma_0(\boldsymbol{x}_0),\ldots,\gamma_{T-1}(\boldsymbol{x}_{T-1})} \quad & \underset{\boldsymbol{\epsilon}_0,\ldots,\boldsymbol{\epsilon}_{T-1}}{E} \left\{ \sum_{t=0}^{T-1} c_t(\boldsymbol{x}_t, \gamma_t(\boldsymbol{x}_t), \boldsymbol{\epsilon}_t) + c_T(\boldsymbol{x}_T) \right\} \\ \text{s.t.} \quad & \boldsymbol{x}_0 = \tilde{\boldsymbol{x}} \\ & \boldsymbol{x}_{t+1} = f_t(\boldsymbol{x}_t, \gamma_t(\boldsymbol{x}_t), \boldsymbol{\epsilon}_t), t = 0, 1, \ldots, T-1, \\ & \gamma_t(\boldsymbol{x}_t) \in \boldsymbol{U}_t, t = 0, 1, \ldots, T-1. \end{aligned} \tag{2.1}$$

where $\tilde{\boldsymbol{x}}$ is a given initial state belonging to a suitable state space $\boldsymbol{X}_0$.

The problem (2.1) appears quite formidable since the cost functional must be minimized over a class of functions of the current state. This issue together with the complexity of the cost functional, state transition functions and possible non-Gaussian random disturbances make the use of calculus of variations optimization techniques impossible in almost every case. The algorithms shown below present two possible approximation approaches.

2.2 SDP Algorithm for Solving SOC Problems

Dynamic Programming (DP) is a mathematical technique that has been used for many years by engineers, mathematicians and social scientists in a variety of contexts. Bellman (1957) first developed DP into a systematic tool for optimization. More general background can be found in Puterman (1994) and Bertsekas (2000), and applications can be found in Gal (1979), Shoemaker (1982), White (1985, 1988), Foufoula-Georgiou and Kitanidis (1988), Culver and Shoemaker (1997), Chen, Günther, and Johnson (2002), and Tsai et al. (2004). In Stochastic Dynamic Programming (SDP), uncertainty is modeled in the form of random realizations of stochastic system variables, and the estimated *expected* "cost" ("benefit") is minimized (maximized).

The SDP technique decomposes a SOC problem into a sequence of minimization problems that are carried out over the control space rather than solving for the control functions. Define state variables, decision variables, stochastic variables, transition functions and cost functions as those for SOC, a SDP objective is to minimize the expected cost through stage T:

$$\begin{aligned} \min_{\boldsymbol{u}_0,\ldots,\boldsymbol{u}_{T-1}} \quad & \underset{\boldsymbol{\epsilon}_0,\ldots,\boldsymbol{\epsilon}_{T-1}}{E} \left\{ \sum_{t=0}^{T-1} c_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\epsilon}_t) + c_T(\boldsymbol{x}_T) \right\} \\ \text{s.t.} \quad & \boldsymbol{x}_0 = \tilde{\boldsymbol{x}} \\ & \boldsymbol{x}_{t+1} = f_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\epsilon}_t), t = 0, 1, \ldots, T-1, \\ & (\boldsymbol{x}_t, \boldsymbol{u}_t) \in \boldsymbol{U}_t, t = 0, 1, \ldots, T-1, \end{aligned} \tag{2.2}$$

where $\tilde{\boldsymbol{x}}$ the given initial state.

It has been proven by Bertsekas (2000) that the general SOC problem (2.1) is equivalent to the SDP problem (2.2). The two formulations are nearly identical, except that in SDP the minimization is taken directly over the control vectors instead of over the sequence of unknown control functions. Recall that in the SOC formulation, the control vector is determined by the control function: $\boldsymbol{u}_t = \gamma_t(\boldsymbol{x}_t)$. This relationship also exists in SDP, but is suppressed in equation (2.2), since the optimal policy need not be stored if the "future value functions," defined below, are stored.

In SDP, the *future value function* is defined for a specified stage t as the optimal expected cost to operate the system from the *current* stage through the *end* of the time horizon, given $\boldsymbol{x}_t$, the system state at the beginning of stage t. Denoted as $F_t(\boldsymbol{x}_t)$ in continuous-state SDP, the future value function at stage t is:

$$\begin{aligned} F_t(\boldsymbol{x}_t) = \min_{\boldsymbol{u}_t,\ldots,\boldsymbol{u}_{T-1}} \quad & \underset{\boldsymbol{\epsilon}_t,\ldots,\boldsymbol{\epsilon}_{T-1}}{E} \left\{ \sum_{\tau=t}^{T-1} c_\tau(\boldsymbol{x}_\tau, \boldsymbol{u}_\tau, \boldsymbol{\epsilon}_\tau) + c_T(\boldsymbol{x}_T) \right\} \\ \text{s.t.} \quad & \boldsymbol{x}_{\tau+1} = f_\tau(\boldsymbol{x}_\tau, \boldsymbol{u}_\tau, \boldsymbol{\epsilon}_\tau), \tau = t, \ldots, T-1, \\ & (\boldsymbol{x}_\tau, \boldsymbol{u}_\tau) \in \boldsymbol{U}_\tau, \tau = t, \ldots, T-1. \end{aligned} \tag{2.3}$$

For $t = T-1, \ldots, 0$, we can solve $F_t(\boldsymbol{x}_t)$ recursively backward in time starting at stage $T-1$ and ending at stage 0:

$$F_t(\boldsymbol{x}_t) = \min_{\boldsymbol{u}_t} \underset{\boldsymbol{\epsilon}_t}{E} \left\{ c_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\epsilon}_t) + F_{t+1}(\boldsymbol{x}_{t+1}) \right\} , \tag{2.4}$$

where $F_T(\boldsymbol{x}_T) = c_T(\boldsymbol{x}_T)$, which corresponds to a cost imposed at the end of the horizon, and $F_0(\tilde{\boldsymbol{x}})$ corresponds to the optimal objective value of the problem (2.2). During the calculations of $F_0(\tilde{\boldsymbol{x}})$ the optimal control law $\pi^* = (\gamma_0^*(\tilde{\boldsymbol{x}}), \gamma_1^*(\boldsymbol{x}_1), \ldots, \gamma_{T-1}^*(\boldsymbol{x}_{T-1})$ is simultaneously obtained from the computation of (2.4) for every $\boldsymbol{x}_t$ and t.

From equation (2.4), it is clear that when solving for F_t, the value of F_{t+1} at *any* point $\boldsymbol{x}_{t+1}$ must be able to be estimated. This may be quite difficult and expensive since the minimization must be carried out for *each* value of $\boldsymbol{x}_t$, which is belonging to a continuous state space. Furthermore, the future value function equation can be analytically solved only when the state equation is linear and the cost function is quadratic (*LQ hypotheses*). In a typical numerical solution, the state space is discretized and the minimization is carried out for a *finite* number of states $\boldsymbol{x}_t$. The discretization issue has "historically" been the source of the most serious shortcoming of the DP method: the so-called *"curse of dimensionality,"* an exponential increase of computation time and memory requirements as the dimension of problem grows. In fact, the first discretization method introduced by Bellman (1957), and almost the only one used for a long time, has been the "full uniform" grid, i.e., the uniform discretization of each component of the state space in the same number of values. Also, the most commonly used architectures for the approximation of the future value functions, even the most "advanced" (for example, the use of splines in Johnson et al. (1993)), explicitly require this kind of discretization. In recent years, the approximate solution of the DP equations has been approached in a more general and computationally feasible way. For example, the neuro-dynamic programming method introduced by Bertsekas and Tsitsiklis (1996), where the

future value function is approximated by means of neural networks, which do not require the training set to be a full uniform grid. However their attention is more focused on discrete-state infinite-horizon problems, and there is no need to discretize a state space that is finite, so the problem of an efficient discretization is not actually faced. The breakthrough in computational intractability was presented by Chen et al. (1999) and Chen (1999) who viewed the approximation of the future value function a statistical perspective, and replaced the full grid with a statistical experimental design based on an OA of strength three. This effectively created a polynomial time algorithm for high dimensional problems. The SDP algorithm for SOC developed here will extend their method by investigating alternate statistical techniques.

The general SDP algorithm is as following:

SDP Algorithm

1. For each time period $t = T-1, \ldots, 0$: Choose a set of N points P_t in the state space for $\boldsymbol{x}_t$.
2. In period $T-1$:
   (a) For each point $\boldsymbol{x}_{T-1} \in P_{T-1}$, solve
   $$F_{T-1}(\boldsymbol{x}_{T-1}) = \min_{\boldsymbol{u}_{T-1}} \underset{\boldsymbol{\epsilon}_{T-1}}{E} [c_{T-1}(\boldsymbol{x}_{T-1}, \boldsymbol{u}_{T-1}, \boldsymbol{\epsilon}_{T-1}) + c_T(\boldsymbol{x}_T)].$$
   (b) Then approximate $F_{T-1}(\boldsymbol{x}_{T-1})$ by $\hat{F}_{T-1}(\boldsymbol{x}_{T-1})$, over the state space for $\boldsymbol{x}_{T-1}$, using the N function outputs obtained for F_{T-1} from step 2a.
3. In each period $t = T-2, \ldots, 0$:
   (a) For each point $\boldsymbol{x}_t \in P_t$, solve
   $$\tilde{F}_t(\boldsymbol{x}_t) = \min_{\boldsymbol{u}_t} \underset{\boldsymbol{\epsilon}_t}{E}[c_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\epsilon}_t) + \hat{F}_{t+1}(f_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\epsilon}_t)]. \tag{2.5}$$
   (b) Then approximate $\tilde{F}_t(\boldsymbol{x}_t)$ with $\hat{F}_t(\boldsymbol{x}_t)$, over the state space for $\boldsymbol{x}_t$, as in step 2b.

Again it shows that minimization problem in steps 2a and 3a must be solved N times, thus computational requirements are proportional to N. The techniques of efficient discretization and approximation which enable the above algorithm to have moderate computational requirement will be discussed in Chapter (3).

With $\hat{F}_0(\boldsymbol{x}_0)$, $\hat{F}_2(\boldsymbol{x}_2)$, ..., $\hat{F}_{T-2}(\boldsymbol{x}_{T-2})$, $\hat{F}_{T-1}(\boldsymbol{x}_{T-1})$ in disposal, for any $\boldsymbol{x}_0 \in \mathbf{X_0}$, the optimal control scheme can be obtained using the "reoptimizing policy," based on the forward solution of the value function equations (as described in Chen 1999, Tejada-Guibert et al. 1993). This represents the "on-line" forward phase of the optimization procedure: at the beginning of each stage, optimal control vectors are computed by equation (2.5) using the future value functions approximations obtained "off-line." Then, such optimal vectors are fed into the state transition function, which provides the state vector for the next stage according to the actual random vector, and the new control vector can be computed in the same way.

2.3 SG Algorithms to SOC Problems

Instead of formulating a SOC problem as a SDP problem and solving by approximating future value functions, one can approximate the control functions directly. For SOC problems, the SG method is based on two steps. First, the control functions are constrained to take on the structure of *linear combinations* of basis functions containing free parameters to be optimized, thus the functional optimization problem is reduced to a *nonlinear programming problem.* Then the resulting nonlinear programming problem is solved by a stochastic approximation algorithm. The detailed procedure is described below:

For $t = 0, \ldots, T-1$, preassign the j-th component of the control function $\gamma_t(\boldsymbol{x}_t)$ to have the structure

$$\hat{\gamma}_{tj}(\boldsymbol{x}_t, \boldsymbol{\omega}_{tj}) = \sum_{i=1}^{\nu_t} \beta_{tij} \phi(\boldsymbol{x}_t, \kappa_{ti}), j = 1, \ldots, m \tag{2.6}$$

where $\phi(\cdot, \cdot)$ is a basis function parameterized by κ_{ti}. The coefficients β_{tij} of the linear combination and the coefficients κ_{ti} form the set of "free parameters" that must be optimized, and are represented by the vector $\boldsymbol{\omega}_{tj}$.

The classical Ritz method (Ritz 1909) used fixed basis functions for ϕ, i.e., only the β_{tij} are "free parameters." This limits the flexibility of the approximation, and the number of "free parameters" has to grow exponentially with the dimension of the function to be approximated in order to achieve a desired accuracy. Parisini and Zoppoli (1994) and Cervellera (2001) extended the Ritz method by utilizing basis functions consisting unknown parameters, and showed that the total number of parameters grows moderately to achieve the same accuracy. They borrowed the term "network" from the parlance of neural networks, and named function like (2.6) as an "approximation network" when they benefit from suitable density properties. Furthermore, they termed such networks as nonlinear (linear) approximating networks if their basis functions contain (do not contain) free parameters. Actually, one can show functions as in (2.6) with *sigmoidal* basis functions can represent feedforward neural networks with one hidden layer and linear output activation units.

Denote $\boldsymbol{\omega}_{t\nu_t} = col(\boldsymbol{\omega}_{tj}, j = 1, \ldots, m)$, where ν_t is the number of free parameters for stage t. Now we can rewrite the SOC problem in (2.1) as:

$$\begin{aligned} \min_{\boldsymbol{\omega}_{0\nu_0},\ldots,\boldsymbol{\omega}_{T-1\nu_{T-1}}} \quad & \underset{\boldsymbol{\epsilon}_0,\ldots,\boldsymbol{\epsilon}_{T-1}}{E} \left\{ \sum_{t=0}^{T-1} c_t(\boldsymbol{x}_t, \hat{\gamma}_t(\boldsymbol{x}_t, \boldsymbol{\omega}_{t\nu_t}), \boldsymbol{\epsilon}_t) + c_T(\boldsymbol{x}_T) \right\} \\ \text{s.t.} \qquad \boldsymbol{x}_0 &= \tilde{\boldsymbol{x}} \\ \boldsymbol{x}_{t+1} &= f_t(\boldsymbol{x}_t, \hat{\gamma}_t(\boldsymbol{x}_t, \boldsymbol{\omega}_{t\nu_t}), \boldsymbol{\epsilon}_t), t = 0, 1, \ldots, T-1, \\ (\boldsymbol{x}_t, \hat{\gamma}_t(\boldsymbol{x}_t, \boldsymbol{\omega}_{t\nu_t})) &\in \boldsymbol{U}_t, t = 0, 1, \ldots, T-1. \end{aligned} \tag{2.7}$$

Or,

$$\begin{aligned} \min_{\boldsymbol{\omega}_{0\nu_0},\ldots,\boldsymbol{\omega}_{T-1\nu_{T-1}}} \quad & \underset{\boldsymbol{\epsilon}_0,\ldots,\boldsymbol{\epsilon}_{T-1}}{E} \left\{ \sum_{t=0}^{T-1} c_t(\boldsymbol{x}_t, \boldsymbol{\omega}_{t\nu_t}, \boldsymbol{\epsilon}_t) + c_T(\boldsymbol{x}_T) \right\} \\ \text{s.t.} \qquad \boldsymbol{x}_0 &= \tilde{\boldsymbol{x}} \\ \boldsymbol{x}_{t+1} &= f_t(\boldsymbol{x}_t, \boldsymbol{\omega}_{t\nu_t}, \boldsymbol{\epsilon}_t), t = 0, 1, \ldots, T-1, \\ (\boldsymbol{x}_t, \boldsymbol{\omega}_{t\nu_t}) &\in \boldsymbol{U}_t, t = 0, 1, \ldots, T-1. \end{aligned} \tag{2.8}$$

where the minimization is now over the free parameters $\boldsymbol{\omega}_{0\nu_0}, \ldots, \boldsymbol{\omega}_{T-1\nu_{T-1}}$, thus the functional optimization problem is reduced to a nonlinear programming one. One can solve equation (2.7) or (2.8) using a gradient descent method.

Denote

$$\boldsymbol{\omega}_\nu = col(\boldsymbol{\omega}_{t\nu_t}, t = 0, 1, \ldots, T-1),$$

(Here ν is the total number of free parameters and can be used as part of the measurement of complexity of the algorithm). Redefine

$$\boldsymbol{\epsilon} = col(\boldsymbol{\epsilon}_t, t = 0, 1, \ldots, T-1),$$

and

$$J(\boldsymbol{x}_0, \boldsymbol{\omega}_\nu, \boldsymbol{\epsilon}) = \sum_{t=0}^{T-1} c_t(\boldsymbol{x}_t, \boldsymbol{\omega}_{t\nu_t}, \boldsymbol{\epsilon}_t) + c_T(\boldsymbol{x}_T),$$

with $\boldsymbol{x}_0 = \tilde{\boldsymbol{x}}$ the given initial state and subsequent state vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{T-1}$ defined by the transition functions in equation (2.7). Then a gradient descent method (for example,

steepest descent method) optimizes $E_{\boldsymbol{\epsilon}}\, J(\boldsymbol{x}_0, \boldsymbol{\omega}_\nu, \boldsymbol{\epsilon})$ by updating the parameter vector $\boldsymbol{\omega}_\nu$, at each step k with

$$\boldsymbol{\omega}_\nu(k+1) = \boldsymbol{\omega}_\nu(k) - \alpha(k) \nabla_{\boldsymbol{\omega}_\nu} \underset{\boldsymbol{\epsilon}}{E} \left\{ J(\boldsymbol{x}_0, \boldsymbol{\omega}_\nu(k), \boldsymbol{\epsilon}) \right\}, k = 1, 2, \ldots \tag{2.9}$$

where $\alpha(k)$ is a step size that suitablly decreases to ensure convergence.

Note: Since problem (2.1) and consequently (2.7) and (2.8) involves constraints, while gradient descent methods deal with unconstrained problems, in practice a proper penalty function has to be added to J, so as to transform the constrained optimization problem to an unconstrained one.

The difficulty now lies in calculating the gradient $\nabla_{\boldsymbol{\omega}_\nu} E_{\boldsymbol{\epsilon}} \left\{ J(\boldsymbol{x}_0, \boldsymbol{\omega}_\nu(k), \boldsymbol{\epsilon}) \right\}$, which involves an expected value and does not have an explicit form for a general problem (this is the reason we give the algorithm the name "Stochastic Gradient"). In order to overcome the computational difficulties related to the approximation of the expected value at each iteration step, one can resort to the *Stochastic Approximation* (SA) scheme (Robbins and Monro 1951). A SA algorithm substitutes the gradient of the expected value $\nabla_{\boldsymbol{\omega}_\nu} E_{\boldsymbol{\epsilon}} \left\{ J(\boldsymbol{x}_0, \boldsymbol{\omega}_\nu(k), \boldsymbol{\epsilon}) \right\}$ with the gradient of the single realization $\nabla_{\boldsymbol{\omega}_\nu} J(\boldsymbol{x}_0, \boldsymbol{\omega}_\nu(k), \boldsymbol{\epsilon}(k))$, where $\boldsymbol{\epsilon}(k)$ is randomly drawn in accordance to its probability density function. i.e., function (2.9) becomes

$$\boldsymbol{\omega}_\nu(k+1) = \boldsymbol{\omega}_\nu(k) - \alpha(k) \nabla_{\boldsymbol{\omega}_\nu} J(\boldsymbol{x}_0, \boldsymbol{\omega}_\nu(k), \boldsymbol{\epsilon}(k)), k = 1, 2, \ldots \tag{2.10}$$

It is proved in Robbins and Monro (1951), Benveniste et al. (1990) and Kushner and Yin (1997) that the SA algorithm converges to the optimal solution (with probability one) if sufficient conditions on $\alpha(k)$, J and $\boldsymbol{\epsilon}$ are satisfied.

Specifically, one must compute, at each iteration step, the components of the gradient $\nabla_{\boldsymbol{\omega}_\nu} J(\boldsymbol{x}_0, \boldsymbol{\omega}_\nu(k), \boldsymbol{\epsilon}(k))$, i.e., the partial derivatives

$$\frac{\partial}{\partial \omega_{t,i}} J(\boldsymbol{x}_0, \boldsymbol{\omega}_\nu(k), \boldsymbol{\epsilon}(k)), i = 1, \ldots, \nu_t, t = 0, \ldots, T-1$$

where $\omega_{t,i}$ is the i-th component of the vector $\boldsymbol{\omega}_{t\nu_t}$. One can write

$$\frac{\partial J}{\partial \omega_{t,i}} = \frac{\partial J}{\partial \boldsymbol{u}_t} \frac{\partial \hat{\gamma}_t(\boldsymbol{x}_t, \boldsymbol{\omega}_{t\nu_t})}{\partial \omega_{t,i}}, \tag{2.11}$$

By using some algebra, it can be seen that $\partial J/\partial \boldsymbol{u}_t$ is given, for $t = 0, \ldots, T-1$, by

$$\frac{\partial J}{\partial \boldsymbol{u}_t} = \frac{\partial}{\partial \boldsymbol{u}_t} c_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\epsilon}_t) + \frac{\partial J}{\partial \boldsymbol{x}_{t+1}} \frac{\partial}{\partial \boldsymbol{u}_t} f_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\epsilon}_t). \tag{2.12}$$

Moreover, define $\lambda_t \equiv \partial J/\partial \boldsymbol{x}_t$, then it can be computed backward recursively by

$$\lambda_t = \frac{\partial}{\partial \boldsymbol{x}_t} c_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\epsilon}_t) + \lambda_{t+1} \frac{\partial}{\partial \boldsymbol{x}_t} f_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\epsilon}_t) + \frac{\partial J}{\partial y_t}, \tag{2.13}$$

$$\lambda_T = \frac{\partial}{\partial \boldsymbol{x}_T} c_T(\boldsymbol{x}_T), \tag{2.14}$$

where y_t denotes the actual "input" to the preassigned structure at stage t.

For any given, fixed initial state vector $\boldsymbol{x}_0$, the SG algorithm iterates through the random disturbance sequence $\boldsymbol{\epsilon}$ and can be summarized as:

SG Algorithm-1

1. Initialize $\boldsymbol{\omega}_\nu = \boldsymbol{\omega}_\nu(0)$.
2. For $k = 1, 2, 3, \ldots,$ (alternate (a) and (b) until convergence):
   (a) Forward Pass: Generate $\boldsymbol{\epsilon}(k) = col(\boldsymbol{\epsilon}_t(k), t = 0, 1, \ldots, T-1)$, then beginning from $\boldsymbol{u}_0 = \hat{\gamma}_0(\boldsymbol{x}_0, \boldsymbol{\omega}_\nu(k-1))$, $\boldsymbol{x}_1 = f_1(\boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{\epsilon}_0(k))$, simulate the evolution of the system by computing the control sequence $\boldsymbol{u}_t(k), t = 0, \ldots, T-1$ and the state vectors $\boldsymbol{x}_t(k), t = 1, \ldots, T$, according to equation (2.6) and state transition functions.
   (b) Backward Pass: Compute $\nabla_{\boldsymbol{\omega}_\nu} J(\boldsymbol{x}_0, \boldsymbol{\omega}_\nu(k), \boldsymbol{\epsilon}(k))$ as in equation (2.14), (2.13), (2.12), and (2.11), and update $\boldsymbol{\omega}_\nu(k-1)$ to $\boldsymbol{\omega}_\nu(k)$ by means of equation (2.10).

The iterated steps are named "forward pass" and "backward pass" because these steps are similar to the "backpropagation" method used to train feedforward multi-layer neural networks.

Note that the SDP algorithm can be solved for *any* initial vector in the initial vector space by means of "reoptimization" once one has the approximated future value functions of each stage. Unlike the SDP algorithm, the SG algorithm stated above uses only one *specific, fixed* initial vector in all the iterations; thus, the solution obtained is only for that specific initial vector. In order to obtain the optimal control scheme for the SOC problem with *any unspecified* initial state, or the SOC problem with initial states unknown prior to solving equation (2.7) or (2.8), the information of the initial state space $\boldsymbol{X}_0$ has to be captured by the algorithm.

Suppose samples can be drawn from the initial state space $\boldsymbol{X}_0$, i.e., model the initial vector as a random sample from $\boldsymbol{X}_0$, then one can re-formulate equation (2.7) and (2.8) as:

$$\begin{aligned} \min_{\boldsymbol{\omega}_{0\nu_0},\ldots,\boldsymbol{\omega}_{T-1\nu_{T-1}}} \quad & \underset{\boldsymbol{x}_0\in\boldsymbol{X}_0}{E}\ \underset{\boldsymbol{\epsilon}_0,\ldots,\boldsymbol{\epsilon}_{T-1}}{E}\left\{\sum_{t=0}^{T-1} c_t(\boldsymbol{x}_t,\hat{\gamma}_t(\boldsymbol{x}_t,\boldsymbol{\omega}_{t\nu_t}),\boldsymbol{\epsilon}_t)+c_T(\boldsymbol{x}_T)\right\} \\ \text{s.t.} \quad & \boldsymbol{x}_0 = \tilde{\boldsymbol{x}} \\ & \boldsymbol{x}_{t+1} = f_t(\boldsymbol{x}_t,\hat{\gamma}_t(\boldsymbol{x}_t,\boldsymbol{\omega}_{t\nu_t}),\boldsymbol{\epsilon}_t), t=0,1,\ldots,T-1, \\ & (\boldsymbol{x}_t,\hat{\gamma}_t(\boldsymbol{x}_t,\boldsymbol{\omega}_{t\nu_t})) \in \boldsymbol{U}_t, t=0,1,\ldots,T-1. \end{aligned} \tag{2.15}$$

Or,

$$\begin{aligned} \min_{\boldsymbol{\omega}_{0\nu_0},\ldots,\boldsymbol{\omega}_{T-1\nu_{T-1}}} \quad & \underset{\boldsymbol{x}_0\in\boldsymbol{X}_0}{E}\ \underset{\boldsymbol{\epsilon}_0,\ldots,\boldsymbol{\epsilon}_{T-1}}{E}\left\{\sum_{t=0}^{T-1} c_t(\boldsymbol{x}_t,\boldsymbol{\omega}_{t\nu_t},\boldsymbol{\epsilon}_t)+c_T(\boldsymbol{x}_T)\right\} \\ \text{s.t.} \quad & \boldsymbol{x}_0 = \tilde{\boldsymbol{x}} \\ & \boldsymbol{x}_{t+1} = f_t(\boldsymbol{x}_t,\boldsymbol{\omega}_{t\nu_t},\boldsymbol{\epsilon}_t), t=0,1,\ldots,T-1, \\ & (\boldsymbol{x}_t,\boldsymbol{\omega}_{t\nu_t}) \in \boldsymbol{U}_t, t=0,1,\ldots,T-1. \end{aligned} \tag{2.16}$$

Thus equation (2.9) becomes

$$\boldsymbol{\omega}_\nu(k+1)=\boldsymbol{\omega}_\nu(k)-\alpha(k)\nabla_{\boldsymbol{\omega}_\nu}\underset{\boldsymbol{x}_0\in\boldsymbol{X}_0}{E}\underset{\boldsymbol{\epsilon}}{E}\left\{J(\boldsymbol{x}_0,\boldsymbol{\omega}_\nu(k),\boldsymbol{\epsilon})\right\}, k=1,2,\ldots \tag{2.17}$$

and equation (2.10) becomes

$$\boldsymbol{\omega}_\nu(k+1) = \boldsymbol{\omega}_\nu(k) - \alpha(k)\nabla_{\boldsymbol{\omega}_\nu} J(\boldsymbol{x}_0(k), \boldsymbol{\omega}_\nu(k), \boldsymbol{\epsilon}(k)), k = 1, 2, \ldots \tag{2.18}$$

The SG algorithm for unspecified initial states iterates through both the samples drawn from $\boldsymbol{X}_0$ and the random sequence $\boldsymbol{\epsilon}$, and it can be stated as:

SG Algorithm-2

1. Choose a set of N points $\boldsymbol{x}_0(1), \boldsymbol{x}_0(2), \ldots, \boldsymbol{x}_0(N)$ in the initial state space $\boldsymbol{X}_0$.
2. Initialize $\boldsymbol{\omega}_\nu = \boldsymbol{\omega}_\nu(0)$.
3. For $k = 0, 1, 2, 3, \ldots, N-1, N, N+1, N+2, \ldots, 2N-1, 2N, 2N+1, \ldots, lN, \ldots,$ (alternate (a) and (b) until convergence):
   (a) Forward Pass: Let $\boldsymbol{x}_0 = \boldsymbol{x}_0((k \bmod N) + 1)$, Generate $\boldsymbol{\epsilon}(k) = col(\boldsymbol{\epsilon}_t(k), t = 0, 1, \ldots, T-1)$, then beginning from $\boldsymbol{u}_0 = \hat{\gamma}_0(\boldsymbol{x}_0, \boldsymbol{\omega}_\nu(k-1))$, $\boldsymbol{x}_1 = f_1(\boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{\epsilon}_0(k))$, simulate the evolution of the system by computing the control sequence $\boldsymbol{u}_t(k), t = 0, \ldots, T-1$ and the state vectors $\boldsymbol{x}_t(k), t = 1, \ldots, T$, according to equation (2.6) and state transition functions.
   (b) Backward Pass: Compute $\nabla_{\boldsymbol{\omega}_\nu} J(\boldsymbol{x}_0, \boldsymbol{\omega}_\nu(k), \boldsymbol{\epsilon}(k))$ as in equation (2.14), (2.13), (2.12), and (2.11), and update $\boldsymbol{\omega}_\nu(k-1)$ to $\boldsymbol{\omega}_\nu(k)$ by means of equation (2.18).

The final value of $\boldsymbol{\omega}$ should be saved and for any initial state vector, the corresponding control scheme can be obtained "on-line" utilizing the saved $\boldsymbol{\omega}$ by evolving the system through the state transition function.

The primary concern with the SG algorithm is the accuracy of the solution. Compared to the SDP algorithm, which approximates a $R^n \to R^1$ future value function for each stage, the SG algorithm approximates a $R^n \to R^m$ control function for each stage, which is equivalent to approximating m $R^n \to R^1$ functions. For SG algorithm-2, speed of the convergence is also a concern, since the algorithm iterates not only through the

random disturbance, but also through the samples of the initial state. Similar to the approach using SDP, two key components of the SG algorithm are: (i) generating the sequence of $\boldsymbol{\epsilon}(k)$ according to its probability density function (and for SG algorithm-2, generating representative samples of the initial state space), and (ii) approximating the control functions. Obviously, the speed of convergence depends on the total number of the free parameters, which is given by the chosen structure of the basis functions. The other possible way to speed up the convergence is to use better sampling techniques to generate the random sequence (and better points in the initial state space for the SG algorithm-2) in the forward pass step so that the stochastic space is represented better with fewer points, and consequently, reduce the number of iterations. Parisini and Zoppoli (1994), Baglietto et al. (2001) and Cervellera (2001) employed random number generation and one-hidden-layer feedforward ANN models, which are one type of approximating networks. The present work proposes to introduce a statistical perspective to their method, addressed further in section (3.1).

CHAPTER 3

STATISTICAL PERSPECTIVES AND METHODS

3.1 A Statistical Perspective

A common task in statistics is to estimate the relationship between a *response* variable y and several *factor* variables $\boldsymbol{x} = (x_1, \ldots, x_n)$. This unknown relationship may be represented as

$$E[Y] = f(x_1, \ldots, x_n), \tag{3.1}$$

where the response is represented by a random variable Y and the functional relationship is between the mean response $E[Y]$ and the n factor variables. A variety of statistical modeling methods are available to estimate $f(\cdot)$, including linear regression (Kutner et al. (2005)).

When the factors are controllable, a statistical *experimental design* may be constructed to designate the levels of the factors at which to observe the response. An experimenter would set each factor at a specific level, run the experiment, then observe the resulting response y. After running the experiment several times with the factors set at a variety of levels, the collected data may then be used to estimate the *response surface* over the (quantitative) factor variable space. An experimental design organizes the setting of the factor levels, so as to enable better estimation of the response surface. The levels of the factors are given in an $N \times n$ design matrix, where N is the number of experimental runs. Popular references for response surface models are Box and Draper (1987) and Myers and Montgomery (1995).

The more recent area of Design and Analysis of Computer Experiments (DACE) considers statistical studies conducted via a *computer experiment*. In the design of complex systems, computer experiments are frequently the only practical approach to optimizing the system. Typically, a simulation model of system performance is constructed, and design parameters which affect performance are identified. If a performance measure

is not straightforward to calculate, such as one that involves an integral, then *sampling* via computer experiments may be employed to estimate the measure. If the simulation model is computationally expensive, then the optimization may instead rely on a *metamodel*, i.e., a mathematical model surrogate, which is constructed through two basic tasks: (1) Utilize an experimental design to select sample points in the input space and observe the corresponding responses output from the computer experiment; (2) Fit a statistical model to this data. Methods for the first task may be used to conduct sampling in general. Reviews of both experimental design and statistical modeling methods for computer experiments are given by Chen et al. (2003) and Chen et al. (2005).

In the SDP algorithm, the key response surface is for the future value function $F_t(\boldsymbol{x}_t)$, where $\boldsymbol{x}_t$ can be interpreted as the vector of quantitative factors. Each of the n components of the state vector $\boldsymbol{x}_t$ represents one of the n factors of interest, $x_1, \ldots, x_n$. In order to solve the dynamic program, the n-dimensional state space must be discretized to a finite set of N points. This is equivalent to choosing an experimental design for n factors, in which each point makes up a row in the $N \times n$ design matrix. The optimal value function is computed at each of the N points

$$F_t(\mathbf{x}_{it}) = \min_{\mathbf{u}_t} E\{c_t(\mathbf{x}_{it}, \mathbf{u}_t, \boldsymbol{\epsilon}_t) + F_{t+1}(\mathbf{x}_{i't+1})\},$$

for $i = 1, \ldots, N$, and this optimization may be interpreted as the computer experiment. One can then use these N values to estimate the shape of $F_t(\mathbf{x}_t)$ with some kind of statistical model. When the form of the unknown relationship cannot be specified (e.g., as a quadratic polynomial), then a flexible modeling method and an experimental design that distributes points to evenly cover the factor variable space are preferred.

In the SG algorithm, the statistical perspective is not as straightforward. However, there are two issues that may benefit from statistical methods: (i) representation of the stochastic space (and the initial state space in SG algorithm-2) and (ii) approx-

imation of the control functions. For (ii), a flexible modeling method is again needed; however, the form of the model must be fixed prior to inclusion in the SG algorithm. For (i), the random sequence $\boldsymbol{\epsilon}_t(k)$ in the forward pass step must be generated from its distribution in each iteration. This is equivalent to sampling from a stochastic space. Existing methods of random number generation, such as inversion and rejection methods, transform uniformly distributed random variates to the desired distribution (see Bratley et al. 1987). However, the traditionally generated uniform random variates (so called Pseudo-random numbers) are only a substitute for the true random numbers and tend to show clustering effects (see the following section and Figures (3.1) and (3.2)). Furthermore, for multi-dimensional integration, it has been shown that it requires more points sampled from random uniform distribution generators than from the so called Quasi-random generators to achieve convergence (Hua and Wang 1981, Niederreiter 1992, Fang and Wang 1994). Intuitively, Quasi-random approaches deliberately place the points to achieve good coverage of the space while the completely random approach relies on chance and, consequently, does not guarantee good coverage. Instead of random uniform distribution, Quasi-random points can be used to generate desired random sequences (and replace the random samples drawn from the initial state space in SG algorithm-2), so that the stochastic space can be represented better with fewer points, and consequently, reduce the number of iterations. Experimental designs that fill the a space evenly may be employed for Quasi-random sampling. In the following sections, various approaches for design of experiments and statistical modeling will be described.

3.2 Design of Experiments

This section describes the experimental design approaches that will be employed in this dissertation. Experimental design approaches that fill the factor variable space evenly are called "space-filling." The most basic experimental design that may be considered

to fill space is the full factorial design, which is equivalent to a complete grid of points. Since the full factorial design is considered by statisticians to be too large in practice for high dimensions, a variety of more efficient experimental designs have emerged (see Chen et al. (2003) for a review). For the applications implemented here, no prior knowledge of the form of the functions to be approximated is available, so it is reasonable to choose a "model-independent" design and a design based on the concept of "space-filling" with the general objective of identifying a set of design points that are "uniformly scattered" over the design space. Orthogonal Array (OA), Latin Hypercubes (LH) and Number-Theoretic Method (NTM) are types of designs which follow the above mentioned philosophy.

3.2.1 Orthogonal Array and Latin Hypercubes

OA and LH seek balance among the dimensions. Consider a design with p levels in each of n dimensions. A full grid would consist of $N = p^n$ points. An OA design of strength d is a fractional factorial design with the property that if one looks at any d of the n dimensions, each of the p^d possible combinations of levels of these d variables occurs equally often, say λ times, in the design. Thus, an OA design has $N = \lambda p^d$. In this dissertation, $\lambda = 1$ and strength $d = 2$ or $d = 3$ were chosen. Spatially, when the design points of an OA of strength two (three) with $\lambda = 1$ are projected onto any two (three)-dimensional subspace, each point of the two (three)-dimensional full factorial with p levels in each dimension will be represented exactly once.

OAs are most commonly derived using difference matrices over Galois fields (see Bose and Bush 1952, Hedayat and Wallis 1978) and finite projective geometry (see Rao 1946, Raghavarao 1971 and Chen 2001). For the applications tested in this work, the generator provided by Chen (2001) and corresponding code is used. Particularly, $p = 31$ and $p = 43$ is used to generate $N = 961$ and $N = 1849$ points of strength two OAs, while $p = 11$ and $p = 13$ to generate $N = 1331$ and $N = 2197$ points of strength three OAs.

To guarantee their space-filling quality, Chen (2001) has conducted the study of how the space-filling quality affects an approximation and developed a measure for fast calculation of the "goodness" of the OA designs. A "good", "neutral" and "bad" OA for each of the design size mentioned above is generated, and comparisons will be conducted to identify "good" approximation quality based on these designs.

A Latin hypercube (LH) (Rao 1946) design is mathematically equivalent to an OA of strength one (although such an OA is technically not an orthogonal design). When the points of an LH are projected onto any single dimension, all p levels will be represented exactly once. Thus, one can consider $N = p$ for an LH. High correlations between dimensions are unfortunately possible in an LH design, and hybrid OA-LH designs attempt to overcome this drawback. An OA-LH design takes the points of an OA and utilizes a mapping on the p levels to convert the design into an LH. The OA designs previously mentioned are utilized to generate OA-LH designs.

3.2.2 Number Theoretic Method Designs

In this section the term "sequence" is referred to as"infinite sequence," and "point set" as "finite sequence." An NTM, a.k.a Quasi-Random sequence or Low-Discrepancy (LD) sequence, aims to create uniformly-spaced designs in $I^s \equiv [0,1]^s$ (the domain is nomalized to the closed and bounded unit cube for convenience) by minimizing a "discrepancy" measure (Weyl (1916) and Weyl (1956)), the most common of which is *star-discrepancy*, defined in Niederreiter (1992) as

$$D(N, \mathcal{P}_N) = \sup_{w \in [0,1]^s} \left| \frac{A(\{\boldsymbol{x}_i \leq \boldsymbol{w}\}, \mathcal{P}_N)}{N} - v([\mathbf{0}, \boldsymbol{w}]) \right|, \tag{3.2}$$

where $\mathcal{P}_N$ is a point set with N points in $[0,1]^s$,$\mathcal{P}_N = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N)$, $A(\{\boldsymbol{x}_i \leq \boldsymbol{w}\}, \mathcal{P}_N)$ is a counter function that counts the number of points $\boldsymbol{x}_i \in \mathcal{P}_N$ that satisfy the inequality, and $v([\mathbf{0}, \boldsymbol{w}])$ is the volume of the rectangle $[\mathbf{0}, \boldsymbol{w}] \in [0,1]^s$. In other words, the

"discrepancy" of a points set is the largest difference between the fraction of number of points in a subset of $[0,1]^s$ and the volume of the subset. For an infinite sequence $\mathcal{P}$, the discrepancy of the sequence is the discrepancy of its first N points. Other types of discrepancy are defined in Niederreiter (1992) by using intervals different from $[\mathbf{0}, \boldsymbol{w}]$, for example, *extreme-discrepancy* when the intervals are of the form $[\mathbf{u}, \boldsymbol{v}]$,where $u, v \in [0,1]^s$.

A more general concept of discrepancy in terms of "statistical language" is *F-discrepancy* with respect to the distribution $F(\boldsymbol{w})$ which is given by Wang and Fang (1994) as:

$$D_F(N, \mathcal{P}_N) = \sup_{w \in [0,1]^s} \left| F_N(\boldsymbol{w}) - F(\boldsymbol{w}) \right|,$$

$$F_N(w) = \frac{1}{N} \sum_{i=1}^{N} 1(\boldsymbol{x}_i \leq \boldsymbol{w}),$$

where $1(\boldsymbol{x}_i \leq \boldsymbol{w})$ is the indicator function that equals one when all components of the arguments satify the inequality. Note that $F_N(w)$ is the empirical distribution of the point set $\mathcal{P}_N$ and that $D_F(N, \mathcal{P}_N)$ is the Kolmogorov-Smirnov statistic for testing the goodness of fit of the the distribution of points in $\mathcal{P}_n$ with regards to distribution $F(w)$. When $F(w)$ is the uniform distribution on I^s, the *F-discrepancy* reduces to the *star-discrepancy*.

It is common in the literature to define a low-discrepancy sequence as an infinite sequence $\mathcal{P}$ such that

$$D_N(\mathcal{P}) \leq C_s \frac{(logN)^s}{N}, N \geq 2$$

with constant C_s depending on $\mathcal{P}$ and on dimension s. Therefore, the sole difference among the many types of low-discrepancy sequence is how small the constant term C_s is. Sequences with lower discrepancy are considered to be more "uniformly" scattered.

The widest range of applications, and indeed the historical origin of NTMs, is found in numerical integration in dimension s, with large s. The usual rules need a huge number

of nodes with increasing dimension to obtain good accuracy. NTMs were developed to over come this *"curse of dimensionality"* (Hua and Wang 1981, Halton 1960).

Pseudo-random numbers are often used for the same purpose of filling as uniformly as possible a considered space, but it is well-known that Pseudo-random numbers are only a substitute for true random numbers and tend to show clustering effects. This can be seen intuitively from Figure (3.1) and Figure (3.2), which show the layout of the first 200 points in the 2-dimensional space generated by a Pseudo-random generator and the Hammersley sequence, a particular NTM sequence. Figure (3.3) and Figure (3.4) are the plot of the first 1000 points in the 2-dimensional space generated by a Pseudo-random generator and the Hammersley sequence. It is clear that for Hammersley point set, the points added later appear where there is no existing points, thus giving a pattern that fills up a space evenly. In contrast, the Pseudo-random points do not show this trend.

A multitude of classes of NTMs have been proposed based on the fruitful developments within and between several mathematical disciplines such as algebraic number theory, combinatorial methods, algebraic geometry and ergodic theory. Particular low-discrepancy sequences include good lattice point, good point (Hua and Wang 1981, Fang and Wang 1994), Van der Corput sequence, Halton sequence (Halton 1960), Hammersley sequence (Hammersley 1960), Sobol′ sequence (Sobol′ 1967), Faure sequence (Faure 1982) and Niederreiter sequence (Niederreiter 1992, Niederreiter and Xing 1995, 1996). Hammersley, Sobol′, Faure and Niederreiter-Xing (a particular alternative of Niederreiter sequence) point sets will be studied and tested in this dissertation. Each sequence has alternatives when they are implemented, and a wide field of investigations are still in progress. For instance, the Niederreiter sequence is being further developed by Niederreiter and his colleagues' research. Meanwhile, the ideas of using permutations on the digits of each point and permutations on the generated sequence to reduce discrepancy were first used by applied mathematicians (Papageorgiou and Traub 1997) and investegated

by statisticians (for instance, Owen 1998), the resulting sequence is called a *scrambled sequence*. Only the basic version is introduced in this dissertation.

For any given positive interger b, put $Z_b = \{0, 1, \ldots, b-1\}$, i.e., Z_b is the *least residuals system modulo b*. Every non-negative integer n has a unique *expansion* $n = \sum_{r=0}^{\infty} a_r(n) b^r$ in base b, where $a_r(n) \in Z_b$ for all $r \geq 0$ and $a_r(n) = 0$ for all sufficiently large r. Then $x_n = \sum_{r=0}^{\infty} a_r(n) b^{-r-1}$ is called *radical inverse of the least residual system expansion*. To the author's understanding, this *radical inverse of the least residual system expansion* is the basic ingredient for almost all types of NTM sequences, i.e., almost all methods are based on certain projections of $a_r(n)$ and b^{-r} to generate the nth point in the sequence.

3.2.2.1 Hammersley Sequence

For $s \geq 2$, the nth element of the first dimension of the Hammersley point set is of the form $x_n^1 = \frac{n}{N}$, $n = 0, 1, 2, \ldots$, where N is the total number of points in the point set. The elements for the remaining dimensions are determined as follows:

Choose $s-1$ bases $b_2, b_3, \ldots, b_s$, such that $gcd(b_i, b_j) = 1$, if $i \neq j$. For each dimension, put $Z_{b_i} = \{0, 1, \ldots, b_i - 1\}, i = 2, 3, \ldots, s$. for non-negative integer n, find the unique expansion $n = \sum_{r=0}^{\infty} a_r^i(n) b_i^r$ in base b_i, where $a_r^i(n) \in Z_{b_i}$ for all $r \leq 0$ and $a_r^i(n) = 0$ for all sufficiently large r. Then $x_n^i = \sum_{r=0}^{\infty} a_r^i(n) b_i^{-r-1}$ is the nth element of the ith dimension. The Van der Corput sequence is a one-dimensional sequence generated by the above radical inverse of the least residual system expansion, and the Halton sequence (Halton 1960) is actually a Hammersley sequence without the first dimension.

Example: choose $b_2 = 2$ and $b_3 = 3$ as bases for the 2nd and 3rd dimension. The first 4 points for a three-dimensional Hammersley point set of 1000 points are:

Point 1:

$n = 0$

$\Rightarrow x_1^1 = \frac{0}{1000} = 0$

$n = 0 = 0 \times 2^0 + 0 \times 2^1 + \ldots$

$\Rightarrow x_1^2 = 0 \times 2^{(-0-1)} + 0 \times 2^{(-1-1)} + \cdots = 0$

$n = 0 = 0 \times 3^0 + 0 \times 3^1 + \ldots$

$\Rightarrow x_1^3 = 0 \times 3^{(-0-1)} + 0 \times 3^{(-1-1)} + \cdots = 0$

Point 2:

$n = 1$

$\Rightarrow x_2^1 = \frac{1}{1000}$

$n = 1 = 1 \times 2^0 + 0 \times 2^1 + \ldots$

$\Rightarrow x_2^2 = 1 \times 2^{(-0-1)} + 0 \times 2^{(-1-1)} + \cdots = \frac{1}{2}$

$n = 1 = 1 \times 3^0 + 0 \times 3^1 + \ldots$

$\Rightarrow x_2^3 = 1 \times 3^{(-0-1)} + 0 \times 3^{(-1-1)} + \cdots = \frac{1}{3}$

Point 3:

$n = 2$

$\Rightarrow x_3^1 = \frac{2}{1000} = \frac{1}{500}$

$n = 2 = 0 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + \ldots$

$\Rightarrow x_3^2 = 0 \times 2^{(-0-1)} + 1 \times 2^{(-1-1)} + 0 \times 2^{(-2-1)} + \cdots = \frac{1}{4}$

$n = 2 = 2 \times 3^0 + 0 \times 3^1 + 0 \times 3^2 + \ldots$

$\Rightarrow x_3^3 = 2 \times 3^{(-0-1)} + 0 \times 3^{(-1-1)} + 0 \times 3^{(-2-1)} + \cdots = \frac{2}{3}$

Point 4:

$n = 3$

$\Rightarrow x_4^1 = \frac{3}{1000}$

$n = 3 = 1 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + \ldots$

$\Rightarrow x_4^2 = 1 \times 2^{(-0-1)} + 1 \times 2^{(-1-1)} + 0 \times 2^{(-2-1)} + \cdots = \frac{3}{4}$

$n = 3 = 0 \times 3^0 + 1 \times 3^1 + 0 \times 3^2 + \ldots$

$\Rightarrow x_4^3 = 0 \times 3^{(-0-1)} + 1 \times 3^{(-1-1)} + 0 \times 3^{(-2-1)} + \cdots = \frac{1}{9}$

Thus the 1000-point 3-dimensional Hammersley point set in base 2 and 3 consists of: $(0, 0, 0)$, $(\frac{1}{1000}, \frac{1}{2}, \frac{1}{3})$, $(\frac{1}{500}, \frac{1}{4}, \frac{2}{3})$, $(\frac{3}{1000}, \frac{3}{4}, \frac{1}{9})$, ...

3.2.2.2 Sobol′ Sequence

The Sobol′ sequence uses base 2 and a one-dimensional Sobol′ sequence can be generated by the following three steps.

Step 1: Initialize a set of *direction numbers* $v_1, v_2, \ldots$: Each v_i is a binary fraction that can be written as $v_i = m_i/2^i$ where m_i is an odd integer, and $0 < m_i < 2^i$. To obtain v_i:

1. Choose a polynomial with coefficients chosen from $0, 1$, which is a primitive polynomial in the field F_2. Thus, one might choose, say, $Q \equiv x^d + a_1 x^{d-1} + a_2 x^{d-2} + \cdots + a_{d-1} x + 1$ where each a_i is 0 or 1 and Q is a primitive polynomial of degree d in F_2. (Provided Q is primitive, the choice of polynomial is otherwise arbitrary.)
2. Choose the values of $m_1, m_2, \ldots, m_d$, which are odd and $m_i < 2^i$. (These values can be chosen freely provided that each m_i is odd and $m_i < 2^i$.)
3. For $i = d+1, d+2, \ldots$, calculate $m_i = 2a_1 m_{i-1} \oplus 2^2 a_2 m_{i-2} \oplus \cdots \oplus 2^{d-1} a_{d-1} m_{i-d+1} \oplus 2^d m_{i-d} \oplus m_{i-d}$, where $\oplus$ denotes a bit-by-bit exclusive-OR operation.
4. Calculate $v_i = m_i/2^i$ for $i = 1, 2, \ldots, d, d+1, d+2, \ldots$.

Step 2: For the nth element in the sequence, calculate the *gray code* for n. (Note: gray code: An ordering of binary numbers such that only one bit changes from one entry to

the next. Gray codes are not unique. e.g., one gray code of 3 bits might be: 000, 010, 011, 001, 101, 111, 110, 100.)

1. Write $n = \ldots e_3 e_2 e_1$, which is the binary representation of n.
2. The gray code for n can be obtained from the binary representation of n using graycode(n)$= \cdots g_3 g_2 g_1 = \cdots e_3 e_2 e_l \oplus \cdots e_4 e_3 e_2$.

Step 3: Obtain the values of x_n from $x_n = g_1 v_1 \oplus g_2 v_2 \oplus \cdots$.

Example: Generate the 23rd point of the one-dimensional Sobol′ sequence.

Step 1: Initialize a set of direction numbers $v_1, v_2, \ldots$:

1. Choose the primitive polynomial $x^3 + x + 1$ of degree 3, so $d = 3, a_1 = 0, a_2 = 1$.
2. Initialize values $m_1 = 1, m_2 = 3$, and $m_3 = 7$.
3. For $i = 4, 5, \ldots,$ calculate

$$\begin{aligned} m_i &= 2a_1 m_{i-1} \oplus 2^2 a_2 m_{i-2} \oplus \cdots \oplus 2^{d-1} a_{d-1} m_{i-d+1} \oplus 2^d m_{i-d} \oplus m_{i-d} \\ &= 2 \times 0 \times m_{i-1} \oplus 2^2 \times 1 \times m_{i-2} \oplus 2^3 m_{i-3} \oplus m_{i-3} \\ &= 4m_{i-2} \oplus 8m_{i-3} \oplus m_{i-3}. \end{aligned}$$

Then,

$$\begin{aligned} m_4 &= 12 \oplus 8 \oplus 1 \\ &= 1100 \oplus 1000 \oplus 0001 \quad \text{in binary} \\ &= 0101 \quad \text{in binary} \\ &= 5, \end{aligned}$$

$$\begin{aligned} m_5 &= 28 \oplus 24 \oplus 3 \\ &= 11100 \oplus 11000 \oplus 00011 \quad \text{in binary} \\ &= 00111 \quad \text{in binary} \\ &= 7, \end{aligned}$$

$m_6 = 20 \oplus 56 \oplus 7$

$= 010100 \oplus 111000 \oplus 000111$ in binary

$= 101011$ in binary

$= 43,$

and so on.

4. Calculate $v_i, i = 1, 2, \ldots,$:

   $v_1 = m_1/2^1 = 1/2 = 0.1$ in binary,

   $v_2 = m_2/2^2 = 3/4 = 0.11$ in binary,

   $\ldots$

   $v_5 = m_5/2^5 = 7/2^5 = 0.00111$ in binary.

Step 2: Calculate the Gray code for 23:

$n = 23 = 10111$ in binary,

$\text{graycode}(23) = 10111 \oplus 01011$

$= 11100.$

Step 3: Obtain the value of x_{23}:

$x_{23} = v_3 \oplus v_4 \oplus v_5$

$= 0.11100 \oplus 0.01010 \oplus 0.00111$

$= 0.10001$ in binary

$= 17/32.$

To generalize this procedure to s dimensions, choose any s different primitive polynomials, calculate s different sets of direction numbers as explained above, and then generate each component $x_n^i, i = 1, 2, \ldots, s$ of the quasi-random vector separately using the corresponding set of direction numbers.

3.2.2.3 Niederreiter Sequence: (t, m, s)-net and (t, s)-sequence

Some basic definitions for this family of sequences are:

1. An *elementary interval* in base b, $(b \geq 2)$: a sub-interval of I^s of the form

$$E \equiv \prod_{i=1}^{s} [c_i b^{-d_i}, (c_i + 1) b^{-d_i}]$$

with $0 \leq c_i \leq b^{d_i}$ and $d_i \geq 0$.

2. A (t, m, s)-*net* in base b: a point set $\mathcal{P}$ of b^m points in I^s such that $A(E, \mathcal{P}) = b^t$ for every elementary interval E in base b with $V(E) = b^{t-m} (0 \leq t \leq m)$, where $V(E)$ is the volume of E.

3. A (t, s)-*sequence* in base b: an infinite sequence X in I^s such that for all $k \geq 0$ and $m > t$, the point set $\{X_{kb^m}, \cdots, X_{(k+1)b^m - 1}\}$ is a (t, m, s)-net.

One can see that t is the quality parameter of the point set in that smaller values of t imply greater equidistribution. In other words, elementary intervals that "should" have b^t points do have b^t points. When $t = 0$, every elementary interval in base b of volume b^{-m} contains exactly one point of the point set, i.e., every elementary interval that "should" have one point does have one point of the point set. The Sobol′ sequence is a (t,s)-sequence in base 2, and Faure sequence discribed in the following subsection is a (0,s)-sequence. The general construction principle of (t, s)-sequences, introduced by Niederreiter (1992) and slightly modified by Tezuka (1995), is to choose the following :

1. R, a commutative ring with cardinality b, and set of digits in base b, $B = \{0, 1, \ldots, b-1\}$,

2. Bijections Ψ_r from B to R for $r \geq 1$ interger, with $\Psi_r(0) = 0$ for all sufficiently large r,

3. Bijections $\lambda_{i,j}$ from R to B for i, j intergers such that $1 \leq i \leq s$ and $j \geq 1$, with $\lambda_{i,j}(0) = 0$ for all sufficiently large j and all i,

4. The elements $C^i_{j,r} \in R$ for integers i, j, r as above.

Expand $n \geq 0$ in base b as $n = \sum_{r=1}^{\infty} a^i_r(n) b_i^{r-1}$, and set $X^i_n = \sum_{j=1}^{\infty} x^i_{n,j} b^{-j}$ where $x^i_{n,j} = \lambda_{i,j}(\sum_{r=1}^{\infty} C^i_{j,r} \Psi_r(a_r(n)))$. Then the sequence $\mathcal{P} = (X_n)$, with coordinates X^i_n as

above, is a (t, s)-sequence in base b if the coefficients $C^i_{j,r}$ are well chosen. The matrices $C^i = (C^i_{j,r})_{j\geq 1, i\geq 1}$ are called the *generator matrices* of $\mathcal{P}$. Usually the bijections Ψ_r and $\lambda_{i,j}$ are omitted in the practical implementation. For example, if we take $R = F_b$, we see R and B have same elements, except that R has certain elements as zero and unit elements, and two operations. Then we can simply choose Ψ_r and $\lambda_{i,j}$ to be bijections of transforming elements in R or B to it's counterpart in B and R. Then Ψ_r and $\lambda_{i,j}$ can be omitted in the explicit expression of $x^i_{n,j}$. The problem is therefore to correctly choose generator matrices.

Niederreiter(1988) gives a method to obtain generator matrix as follows: Choose $R = F_b$. For the finite field F_b, let $G = F_b(x^{-1})$ be the field of formal Laurent series $S = \sum_{r=w}^{\infty} a_r x^{-r}$ in x^{-1} , with $a_r \in F_b$ and w an arbitrary interger. Let $p_1, p_2, \ldots, p_s \in F_b(x)$ be s pairwise coprime polynomials over F_b, with $degree(p_i) = e_i \geq 1$ and let $g_{i,j} \in F_b(x^{-1})$ be polynomials such that $gcd(p_i, g_{i,j}) = 1$, with $1 \leq i \leq s$ and $j \geq 1$. Then the expansion, in which $0 \leq k < e_i$ and $w \geq 0$,

$$\frac{x^k g_{i,j}(x)}{(p_i(x))^j} = \sum_{r=w}^{\infty} a^{(i)}(j, k, r) x^{(-r)}$$

gives the generator matrices $C^{(i)}$ of the general principle by:

$$C^{(i)}_{j,r} = a^{(i)}(q+1, u, r), j \geq 1, r \geq 1, j = qe_i + u + 1, 0 \leq u < e_i.$$

Further, the element $a^{(i)}(q+1, u, r)$ was defined by element v in F_b. Following this the steps to calculate $C^{(i)}_{j,r}$ are:

1. Choose a suitable monic polynomial $p(x)$ with coefficients in F_b. Let $degree(p(x)) = e \geq 1$. Set $j = 0, q = -1, u = e$.
2. Increment j. If $u = e$, do step 3; otherwise, go to step 4.
3. Increment q and set $u = 0$. Calculate first

$$b(x) = (p(x))^{q+1} = x^m - b_{m-1}x^{m-1} - \cdots - b_0,$$

here $degree((b(x)) = m = e(q+1)$.

Then calculate v_l as follows:

$v_l = 0$, for $0 \le l \le m-2$,

$v_l = 1$, for $l = m-1$,

$v_l = \sum_{k=1}^{m} b_{m-k} v_{l-k}$, for $m \le l \le K+e-2$.

4. For $0 \le r \le K-1$, set $c_{j,r} = v_{r+u}$. Increment u. If $j < K$, go to step 2; otherwise, stop.

Note: The arithmetic are in terms of modulo b since coeffients of $p(x)$ and $b(x)$ are elements of F_b. For clarity, suppose just one dimension is discussed and then all index i are omitted. And suppose it is chosen not to allow more than K base-b digits when n is represented in base b. This means of course that only less than bK values of a sequence in base b can be generated. But this limitation is of little practical concern when it is implemented by computer, since one can choose K large enough.

Example: Choose $b = 3, K = 4, p(x) = x^2 + 1$, so $e = 2$.

The algorithm for calculating the elements proceeds as follows:

1. The first time through step (3) $j = 1, q = 0, u = 0$, and

$$b(x) = (p(x))^1 = x^2 + 1 = x^2 - 2$$

so $m = 2$, $b_0 = 2$ and $b_1 = 0$. Hence $v_0 = 0$, $v_1 = 1$, and from $v_{r+2} = 2v_r$, $r = 0, 1, \ldots$, one can find $v_2 = v_4 = 0$, $v_3 = 2$. Now $c_{1,r} = v_r$, so $c_{1,0} = 0$, $c_{1,1} = 1$, $c_{1,2} = 0$, $c_{1,3} = 2$.

2. The second time through the loop, $u = 1$ and step 3 is omitted. Step 4 uses the same values of v to find $c_{2,r} = v_{r+1}$, so $c_{2,0} = 1$, $c_{2,1} = 0$, $c_{2,2} = 2$, and $c_{2,3} = 0$.

3. The third time through the loop, step 3 has $j = 3$. Set $q = 1$, $u = 0$ and

$$b(x) = (p(x))^2 = x^4 + 2x^2 + 1 = x^4 - x^2 - 2,$$

Now $m = 4$, $b_0 = 2$ and $b_1 = b_3 = 0$, $b_2 = 1$. Hence, $v_0 = v_1 = v_2 = 0$, $v_3 = 1$, and from $v_{r+4} = v_{r+2} + 2v_r, r = 0, 1, \ldots$, one can find $v_4 = 0$. Now $c_{3,r} = v_r$, so $c_{3,0} = 0$, $c_{3,1} = 0$, $c_{3,2} = 0$, $c_{3,3} = 1$.

4. On the final pass through the loop the values of v remain unchanged, and $c_{4,r} = v_{r+1}$, so $c_{4,0} = 0$, $c_{4,1} = 0$, $c_{4,2} = 1$, $c_{4,3} = 0$.

Suppose x_7 is going to be generated, then $n = 7$(in decimal)$= 0021$(in base 3), so $a_0 = 1$, $a_1 = 2$ and $a_2 = a_3 = 0$. Thus $x_{7,1} = c_{1,0}a_0 + c_{1,1}a_1 + c_{1,2}a_2 + c_{1,3}a_3 = 2$. Similarly $x_{7,2} = 1$, $x_{7,3} = 0$ and $x_{7,4} = 0$. Hence $x_7 = \sum_{j=1}^{\infty} x_{7,j}d^{-j} = x_{7,1}3^{-1} + x_{7,2}3^{-2} + x_{7,3}3^{-3} + x_{7,4}3^{-4} = 0.77\ldots$(in decimal).

3.2.2.4 Faure Sequence

Faure sequence (Faure 1982) is a $(0, s)$-sequence in base b, where b is the smallest prime greater than or equal to s. The generator matrices $C^{(i)}, i = 1, 2, \cdots, s$ are powers of the Pascal matrix M modulo b :

$$C^{(i)} = M^{i-1} = \left[\binom{r}{j}(i-1)^{r-j}\right]_{r\geq 0, j\geq 0} \; mod\ b.$$

A more general alternative of Faure sequence is the generalized Faure sequence, which has the generator matrices $C^{(i)} = A^{(i)}M^{(i-1)}$ where $A^{(i)}$ is an arbitrary nonsigular lower triangular matrix over F_b, $i = 1, 2, \cdots, s$. The generalized Faure sequence is computed as follows:

1. Expand the interger n in base b: $n = \sum_{r=0}^{\infty} a_r(n)b^r$.
2. The ith coordinator of x_n is:

$$x_n^i = \sum_{k=0}^{\infty}\sum_{d=0}^{\infty} c_{k,d}^{(i)} a_d(n) b^{-k-1}.$$

The element of the generator matrix, $c_{k,d}^{(i)}$ is:

$$c_{k,d}^{(i)} = i^{d-k} c_{k,d},$$

where $c_{k,d}$ is an element of the Pascal matrix,

$$c_{k,d} = \begin{cases} \frac{d!}{k!(d-k)!} & k \leq d \\ 0 & k > d. \end{cases}$$

3.3 Statistical Modeling

For flexible modeling in high dimensions, recent developments in statistics have created multiple methods for different settings (for a review, see Chen et al. 2003). Two methods are very competitive for high dimensional problems: MARS and ANN models (only feedforward neural networks are studied here, and "ANN" in this dissertation is used interchangeablly with "FF1ANN", which is a feedforward one-hidden-layer neural network). Comparisons between MARS and ANNs for empirical modeling have been conducted by De Veaux et al. (1993) and Pham and Peat (1999), and in both studies MARS performed better overall. Although it is obvious that ANN models can be employed to approximate functions, the existing literature does not provide sufficient studies to predict how they will compare to MARS in the SDP setting. In continuous-state SDP, value functions are typically well-behaved (e.g., convex) and the SDP data involve little noise. Previous comparisons involved statistical data with considerable noise. It is expected that ANN models will perform better in low noise situations. However, ANN models can add extraneous curvature, a disadvantage when a smooth convex approximation is desired. MARS, on the other hand, constructs a parsimonious approximation, but its limited structure, although quite flexible, could affect its approximation ability, which would be most noticeable in low noise situations. For SG algorithms however, ANN has the advantage of having a "fixed" structure once giving the number of hidden layers and the number of hidden nodes. To use MARS for control function approximation, some pre-processing work must be developed. In the present work, only ANNs are employed for SG algorithms.

3.3.1 Multivariate Adaptive Regression Splines

MARS (Friedman 1991) is an adaptive procedure for regression based on traditional statistical thinking with a forward stepwise algorithm to select terms in a linear model followed by a backward procedure to prune the model. It uses expansions in piecewise linear basis functions of the form $(x-t)_+$ and $(t-x)_+$. The "+" means positive part, so

$$(x-t)_+ = \begin{cases} x-t, & x > t, \\ 0, & x \leq t, \end{cases}$$

and

$$(t-x)_+ = \begin{cases} t-x, & x < t, \\ 0, & x \geq t. \end{cases}$$

Each function is piecewise linear, with a knot at the value t (one of the objectives in MARS model building is to select appropriate knots). The two functions are termed as a reflected pair and they are actually linear splines. The idea is to form reflected pairs for each factor x_j with chosen knots (e.g., at each observed value x_{ij} of that input). Denote the set of knots for factor x_j as T^j, the collection of basis function is

$$Z = \left\{(x_j - t^j)_+, (t^j - x_j)_+\right\}, t^j \in T^j, j = 1, 2, \ldots, n.$$

The model-building strategy is like a forward stepwise linear regression, but use functions from the set Z and their products. The result model has the form

$$f(\boldsymbol{x}) = \beta_0 + \sum_{m=1}^{M} \beta_m h_m(\boldsymbol{x}) \tag{3.3}$$

To construct the functions h_m, start with only the constant function $h_0(\boldsymbol{x}) = 1$ in the model, and all functions in the set Z are candidate functions to be added to the model. Two terms are added to the model at a time. At step k, $2k-1$ terms are in the model, and at step $k+1$, the terms of the form below are candidates to be added to the model:

$$\hat{\beta}_{2k} h_l(\boldsymbol{x}) \cdot (x_j - t^j)_+ + \hat{\beta}_{2k+1} h_l(\boldsymbol{x}) \cdot (t^j - x_j)_+, l = 1, 2, \ldots, 2k-1.$$

The term that produces the largest decrease in training error is chosen to be added. Here $\hat{\beta}_{2k}$ and $\hat{\beta}_{2k+1}$ are coefficients estimated by least square, along with all the other $2k-1$ coefficients in the model. The process is continued until the model set contains some preset maximum number of terms, or other specified conditions regarding training error are met.

At the end of this process a large model of the form (3.3) is obtained. This model typically over fit the data, and so a backward deletion procedure can be applied. the term whose removal causes the smallest increase in residual square error is deleted from the model at each stage. Cross-validation can be used to estimate the best total number of terms of the model. One can achieve continuous derivative of the model by replacing cubic or quintic functions that coincide with the piecewise-linear fit at "side knots" (Chen 1999, Chen et al. 1999). The other useful option in the MARS procedure is to set an upper limit on the order of interaction to aid in the interpretation of the model. For the applications in this work, the order of interaction is set to be 2 or 3.

The advantages of MARS model are: (i) Ability to operate locally. The basis functions are zero over part of their range, and when they are multiplied together, the result is nonzero only over the small part of the input space where all component functions are nonzero. As a result, the regression surface is built up parsimoniously, using nonzero component locally only when they needed, thus "spend" parameters carefully. This is especially important in high dimensional setting. (ii) Computational efficiency. Suppose there is N knots for an input x_j, MARS allows one to try every knot in only $O(N)$ operations. (iii) The forward modeling strategy in MARS is hierarchical, in the sense that multiway products are build up from products involving terms already in the model. The philosophy here is that a higher-order interaction will likely only exist if some of its lower-order "footprints" exist as well. This need not be true, but is a reasonable working assumption and avoids the search over an exponentially growing space of alternative.

3.3.2 Artificial Neural Networks

ANNs are mathematical modeling tools widely used in many different fields (for general background and applications see Lippmann 1987, Haykin 1999, Bertsekas and Tsitsiklis 1996, Cherkassky and Mulier 1998; for statistical perspectives see White 1989, Barron et al. 1992, Ripley 1993, or Cheng and Titterington 1994; for approximation capabilities see Barron 1993). From a purely "analytical" perspective, an ANN model is a nonlinear parameterized function that computes a mapping of n input variables into (typically) one output variable, i.e., n predictors and one response. The basic architecture of ANNs is an interconnection of *neural nodes* that are organized in layers. The input layer consists of n nodes, one for each input variable, and the output layer consists of one node for the output variable. In between there are *hidden layers* which induce flexibility into the modeling. *Activation functions* define transformations between layers, the simplest one being a linear function. Connections between nodes can "feed back" to previous layers, but for function approximation, the typical ANN is *feedforward* only with one hidden layer and uses "sigmoidal" activation functions (monotonic and bounded). A comprehensive description of various ANN forms may be found in Haykin (1999).

In the present work, a FeedForward One-hidden-layer ANN (**FF1ANN**) is utilized with a "hyperbolic tangent" activation function in the hidden layer: $\text{Tanh}(x) = (e^x - e^{-x})/(e^x + e^{-x})$. Define n as the number of input variables, H as the number of hidden nodes, v_{ih} as weights linking input nodes i to hidden nodes h, w_h as weights linking hidden nodes h to the output node, and θ_h and ϑ as constant terms called "bias" nodes (like intercept terms). Then our ANN model form is:

$$\hat{g}(\boldsymbol{x}; \boldsymbol{w}, \boldsymbol{v}, \boldsymbol{\theta}, \vartheta) = \left(\sum_{h=1}^{H} w_h Z_h + \vartheta \right), \tag{3.4}$$

where for each hidden node h

$$Z_h = \text{Tanh}\left(\sum_{i=1}^{n} v_{ih}x_i + \theta_h\right). \tag{3.5}$$

The parameters v_{ih}, w_h, θ_h, and ϑ are estimated using a nonlinear least squares approach called *training*. For simplicity, the entire vector of parameters is referred to as $\boldsymbol{w}$ below.

The training employed here is a batch processing method (as opposed to the more common on-line processing) that minimized the squared error loss lack-of-fit (LOF) criterion:

$$\text{LOF}(\hat{g}) = \sum_{j=1}^{N}[y_j - \hat{g}(\boldsymbol{x}_j, \boldsymbol{w})]^2,$$

where N is the number of data points on which the network is being trained, a.k.a., the training data set. Nonlinear optimization techniques used to conduct the minimization are typically gradient descent-based methods. *Backpropagation method* (see Rumelhart et al. 1986, Werbos 1994), is used here. The backpropagation method computes the gradient of the output with respect to the outer parameters, then "backpropagates" the partial derivatives through the hidden layers, in order to compute the gradient with respect to the entire set of parameters. This is completed in two phases: (i) a forward phase in which an input $\boldsymbol{x}_j$ is presented to the network in order to generate the various outputs (i.e., the output of the network and those of the inner activation functions) corresponding to the current value of the parameters, and (ii) a backward phase in which the gradient is actually computed on the basis of the values obtained in the forward phase. After all N samples are presented to the network and the corresponding gradients are computed, the descent algorithm step is completed by upgrading the vector of parameters $\boldsymbol{w}$ using the steepest-descent method:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \nabla \text{LOF}(\boldsymbol{w}_k),$$

where $\nabla \text{LOF}(\boldsymbol{w}_k)$ is the gradient of the LOF function with respect to the vector $\boldsymbol{w}_k$. Implementation of this approach is very simple, since the computation of the gradient

requires only products, due to the mathematical structure of the network. Unfortunately, backpropagation presents many well-known disadvantages, such as entrapment in local minima. As a consequence, the choice of the stepsize α_k is crucial for acceptable convergence, and very different stepsizes may be appropriate for different applications.

The best training algorithm for FF1ANNs, in terms of rate of convergence, is the Levenberg-Marquardt method (first introduced in Hagan and Menhaj 1994), which is an approximation of the classic second-order (i.e. it makes use of the Hessian matrix) Newton method. Since the LOF criterion is quadratic, one can approximate its Hessian matrix via backpropagation as

$$K[\boldsymbol{e}(\boldsymbol{w})]^T K[\boldsymbol{e}(\boldsymbol{w})]$$

where

$$\boldsymbol{e}(\boldsymbol{w}) = [y_1 - \hat{g}(\boldsymbol{x}_1, \boldsymbol{w}), \ldots, y_N - \hat{g}(\boldsymbol{x}_N, \boldsymbol{w})]^T.$$

and $K[\boldsymbol{e}(\boldsymbol{w})]$ is the Jacobian matrix of $\boldsymbol{e}(\boldsymbol{w})$. Each row of the matrix contains the gradient of the j-th single error term $y_j - \hat{g}(\boldsymbol{x}_j, \boldsymbol{w})$ with respect to the set of parameters $\boldsymbol{w}$.

The update rule for the parameter vector at each step is:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - [K[\boldsymbol{e}(\boldsymbol{w}_k)]^T K[\boldsymbol{e}(\boldsymbol{w}_k)] + \lambda I]^{-1} [K\boldsymbol{e}(\boldsymbol{w}_k)]^T \boldsymbol{e}(\boldsymbol{w}_k)^T.$$

When λ is large, the method is close to gradient descent with a small stepsize. When $\lambda = 0$ it becomes a standard Newton method with an approximated Hessian matrix (Gauss-Newton method), which is more accurate near the minima. After each successful step (i.e., when the LOF decreases) λ is decreased, while it is increased when the step degrades the LOF.

Other than the selection of a model fitting approach, the most important decision in implementing an ANN model is the choice of architecture. For a FF1ANN, the number

of nodes in the hidden layer, H, determines the architecture. Unfortunately, the appropriate number of hidden nodes is never clear-cut. Existing theoretical bounds on the accuracy of the approximation are non-constructive or too loose to be useful in practice. Thus, it is not possible to define "a priori" the optimal number of hidden nodes, since it depends on the number of samples available and on the smoothness of the function being approximated. Too few nodes would result in a network with poor approximation properties, while too many would "overfit" the training data (i.e. the network merely "memorizes" the data at the expense of generalization capability). Therefore, a tradeoff between these two possibilities is needed. There are many rules of thumb and methods for model selection and validation in the literature, as well as methods to avoid or reduce overfitting (see Haykin 1999), but the choice of a good architecture requires several modeling attempts and is always very dependent on the particular application.

For the applications in this work, the appropriate number of hidden nodes were identified by exploring many architectures and checking the average cost using a test data set from the simulation.

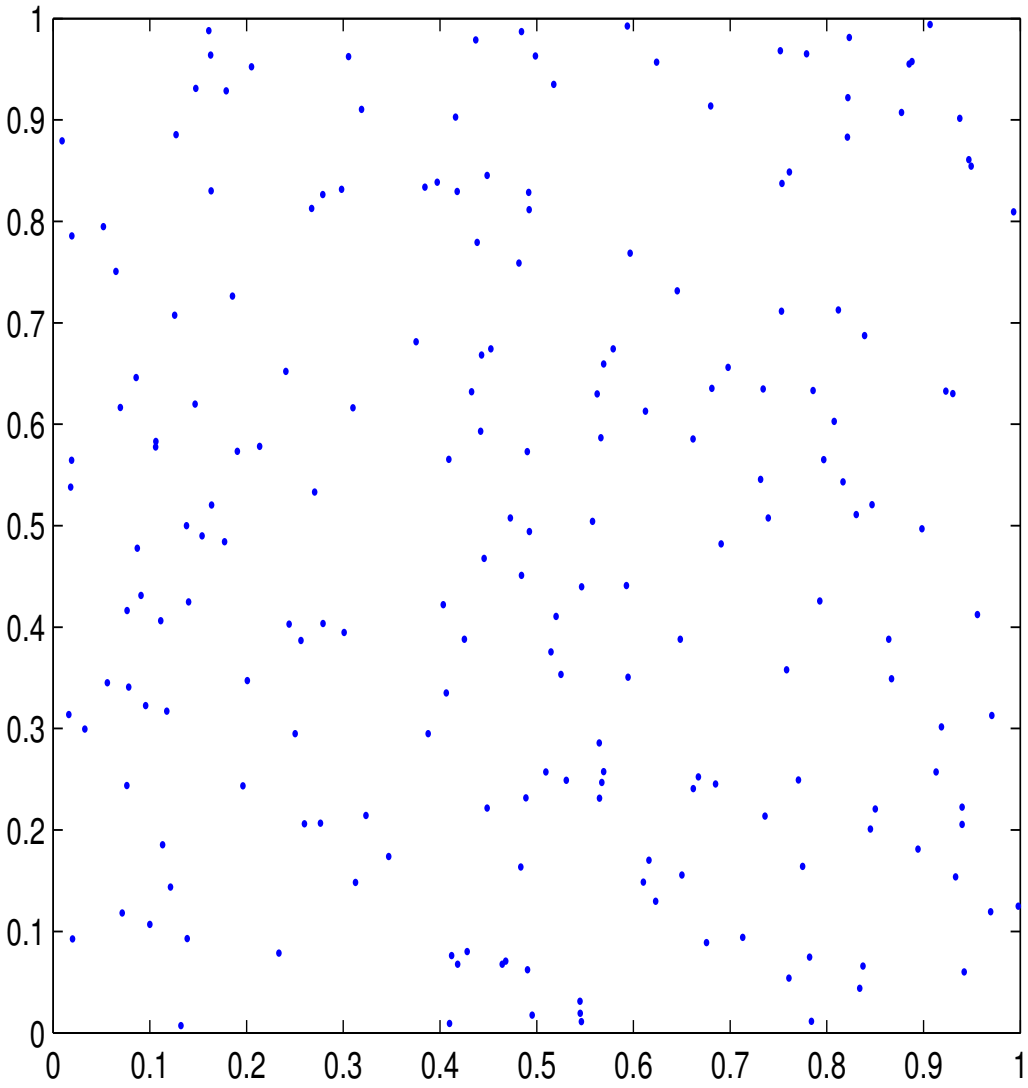

Figure 3.1. The first 200 Pseudo-random points.

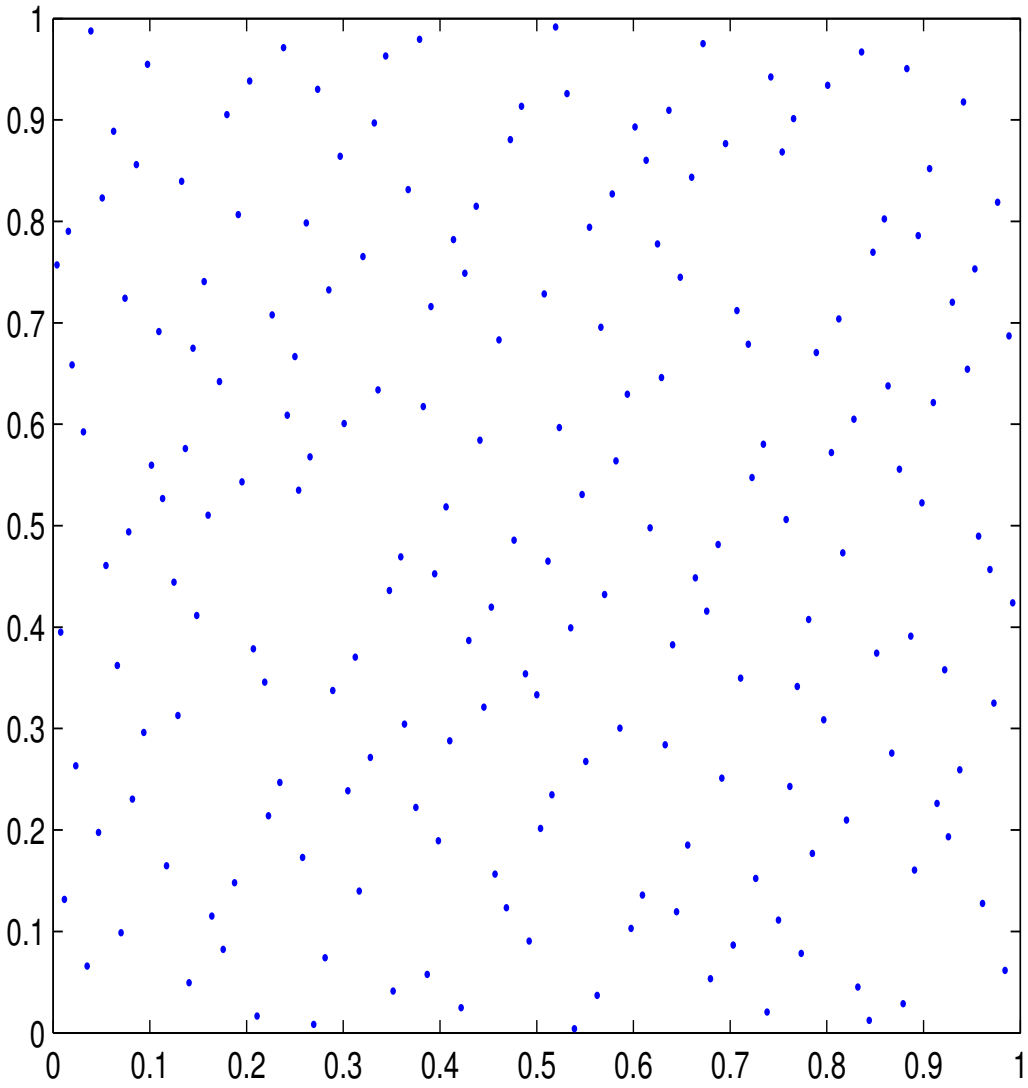

Figure 3.2. The first 200 Hammersley points.

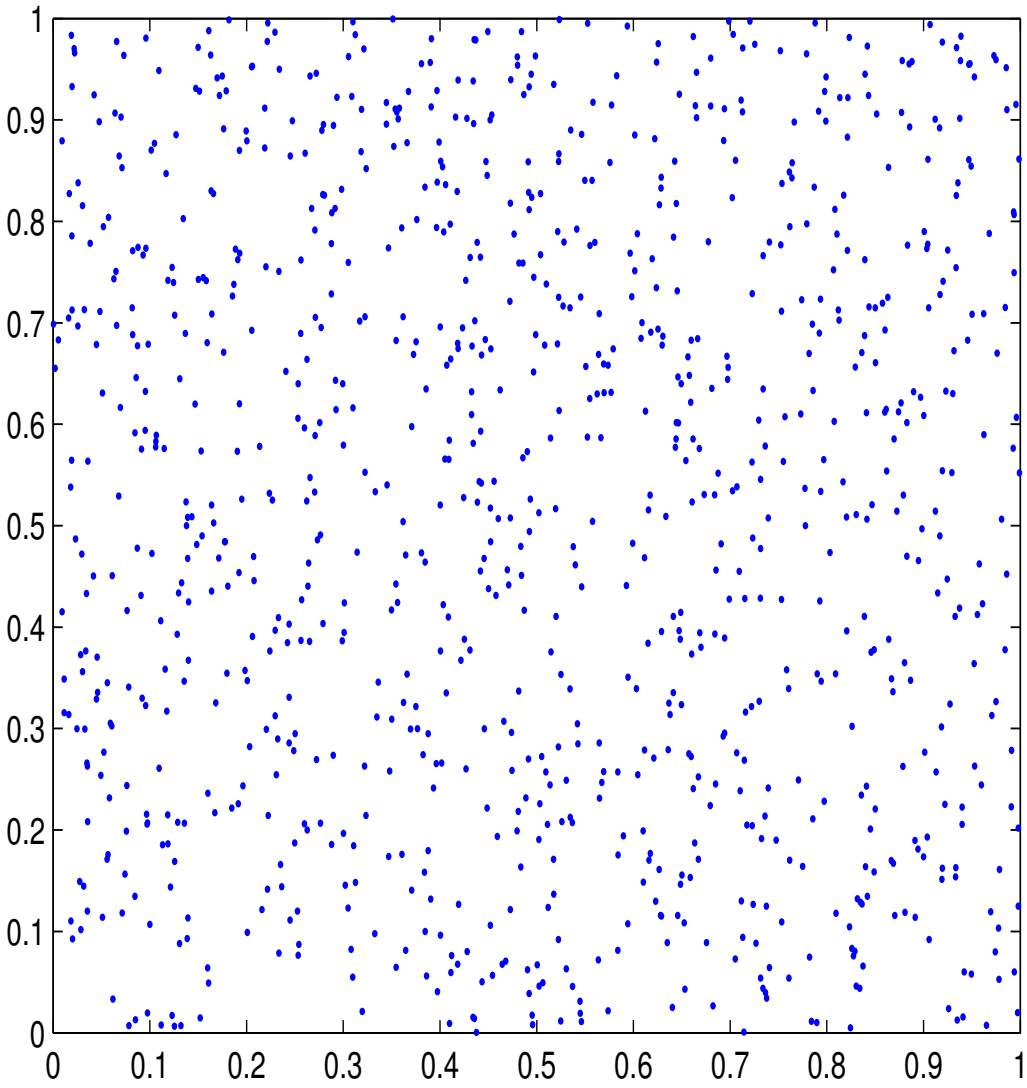

Figure 3.3. The first 1000 Pseudo-random points.

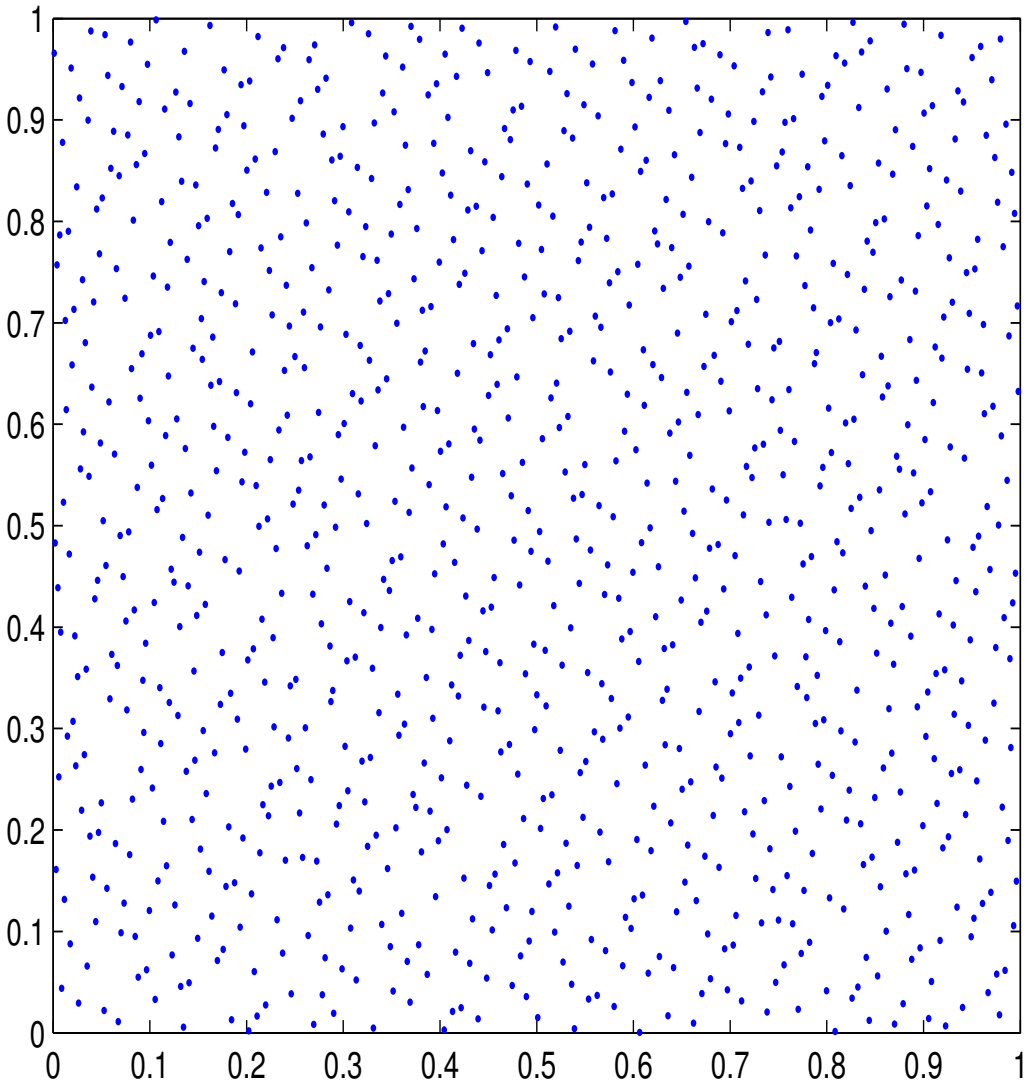

Figure 3.4. The first 1000 Hammersley points.

CHAPTER 4

THREE TEST PROBLEMS AND COMPUTATIONAL RESULTS

To test the proposed approaches to SOC problems, a nine-dimensional inventory forecasting problem and two water reservoir network management problems, one with eight dimensions and one with thirty-dimensions, are constructed. The inventory forecasting problem (Chen 1999) is known to have significant interaction effects between inventory levels and forecasts due to the capacity constraints, while the water reservoir network management problems has simpler structure and are more typical of continuous-state SOC problems. To the best of the author's knowledge, the thirty-dimensional problem (Cervellera et al. 2005-1) is the problem with largest dimension ever addressed in at least the reservoirs management literature without introducing restrictive hypotheses on the cost and/or the model.

4.1 Inventory Forecasting Problem

For an inventory problem, the inventory levels are inherently discrete since items are counted, but the range of inventory levels for an item is too large to be practical for a discrete DP solution. Thus, one can relax the discreteness and use a continuous range of inventory levels. The inventory forecasting problem uses forecasts of customer demand and a probability distribution on the forecasting updates to model the demand more accurately than traditional inventory models (see Hadley and Whitin 1963).

4.1.1 State and Decision Variables

The *state variables* for inventory forecasting consist of the inventory levels and demand forecasts for each item. For the nine-dimensional problem discussed in this dissertaion, three items ($N_I = 3$) and two forecasts for each are considered. At the beginning of stage t, let $I_t^{(i)}$ be the inventory level for item i at the *beginning of the*

current stage. Let $D^{(i)}_{(t,t)}$ be the forecast made at the *beginning of the current stage* for the demand of item i occurring over the stage. Similarly, let $D^{(i)}_{(t,t+1)}$ be the forecast made at the *beginning of the current stage* for the demand of item i occurring over the *next* stage $(t+1)$. Then the *state vector* at the *beginning of stage* t is:

$$\boldsymbol{x}_t = \left(I_t^{(1)}, I_t^{(2)}, I_t^{(3)}, D^{(1)}_{(t,t)}, D^{(2)}_{(t,t)}, D^{(3)}_{(t,t)}, D^{(1)}_{(t,t+1)}, D^{(2)}_{(t,t+1)}, D^{(3)}_{(t,t+1)}\right)^T.$$

The *decision variables* control the order quantities. Let $u_t^{(i)}$ be the amount of item i ordered at the beginning of stage t, then the *decision vector* is $\boldsymbol{u}_t = (u_t^{(1)}, u_t^{(2)}, u_t^{(3)})^T$. Each component of $\boldsymbol{u}_t$ is a function of the state variables, i.e, $u_t^{(i)} = \gamma_t^{(i)}(\boldsymbol{x}_t), i = 1, 2, 3$, where $\gamma_t^{(i)}(\cdot)$ is an unknown function. Put in a compact way, one can write $\boldsymbol{u}_t = \boldsymbol{\gamma}_t(\boldsymbol{x}_t)$. For simplicity, it is assumed that orders arrive instantly.

4.1.2 State Transition Function

The *transition function* for stage t specifies the new state vector to be $\boldsymbol{x}_{t+1} = f_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\epsilon}_t)$, where $\boldsymbol{x}_t$ is the state of the system (inventory levels and forecasts) at the beginning of stage t, $\boldsymbol{u}_t$ is the amount ordered in stage t, and $\underline{\epsilon}_t$ is the stochasticity associated with updates in the demand forecasts. The evolution of these forecasts over time is modeled using the multiplicative Martingale Model of Forecast Evolution, for which the stochastic updates are applied multiplicatively and assumed to have a mean of one, so that the sequence of future forecasts for stage $t+k$ forms a martingale (Heath and Jackson 1991). See Chen (1999) for details on the stochastic modeling.

4.1.3 Objectives and Constraints

The *constraints* for inventory forecasting are placed on the amount ordered, in the form of capacity constraints, and on the state variables, in the form of state space bounds. For this test problem, the capacity constraints are chosen to be restrictive,

forcing interactions between the state variables, but are otherwise arbitrary. The bounds placed on the state variables in each stage are necessary to ensure accurate modeling over the state space.

The *objective function* is a cost function involving inventory holding costs and backorder costs. The typical V-shaped cost function common in inventory modeling is:

$$c_v(\boldsymbol{x}_t, \boldsymbol{u}_t) = \sum_{i=1}^{n_I} (h_i[I_{t+1}^{(i)}]_+ + \pi_i[-I_{t+1}^{(i)}]_+), \tag{4.1}$$

where $[q]_+ = \max\{0, q\}$, h_i is the holding cost parameter for item i, and π_i is the backorder cost parameter for item i. The *smoothed* version of equation (4.1), described in Chen (1999), is employed here.

4.2 Water Reservoir Network Management Problem

Optimal operation of water reservoir networks has been studied extensively in the literature (e.g., Archibald et al. 1997, Foufoula-Georgiou and Kitanidis 1988, Gal 1979, Lamond and Sobel 1995, Turgeon 1981; Yakowitz 1982 provides an excellent survey). This section first presents a water reservoir network consisting of 4 reservoirs which leads to an eight-dimensional model and later points out the modifications of the model which leads to a 10 reservoirs and thirty-dimensional one. A graph can be used to depict a water reservoir network with nodes representing water basins and edges linking the nodes representing the water release direction(see Figure 4.1 for the configuration of the 4-reservoir network). In typical models, the inputs to the nodes are water released from upstream reservoirs and stochastic inflows from external sources (e.g., rivers and rain), while the outputs correspond to the amount of water to be released during a given time stage (e.g., a month).

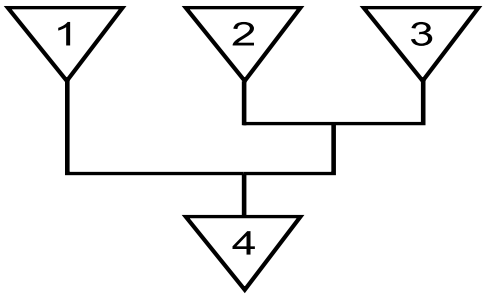

Figure 4.1. Reservoir network.

4.2.1 State and Decision Variables

In a realistic representation of external inflow dynamics, the amount of (random) water flowing into the reservoirs from external sources during a given stage t depends on the amounts of past stages. The realistic modeling of Gal (1979) is followed and an autoregressive linear model of order one is used. Thus, the ith component of the random vector $\boldsymbol{\xi}_t$, representing the stochastic external flow into reservoir$i$, has the form

$$\xi_t^{(i)} = a_t^{(i)} \xi_{t-1}^{(i)} + b_t^{(i)} + c_t^{(i)} \epsilon_t^{(i)} \ , \tag{4.2}$$

where $\epsilon_t^{(i)}$ is a random correction that follows a standard normal distribution. The coefficients of the linear combinations $(\boldsymbol{a}_t, \boldsymbol{b}_t, \boldsymbol{c}_t)$ actually depend on t, so that it is possible to model proper external inflow behaviors for different stages. The actual values of the coefficients used for the model are based on the one-reservoirreal model described in Gal (1979) and extended to the multi-reservoir network.

Given equation (4.2), the *state vector* consists of the water level and stochastic external inflow for each reservoir:

$$\boldsymbol{x}_t = \left(L_t^{(1)}, L_t^{(2)}, L_t^{(3)}, L_t^{(4)}, \xi_{t-1}^{(1)}, \xi_{t-1}^{(2)}, \xi_{t-1}^{(3)}, \xi_{t-1}^{(4)} \right)^T \ , \tag{4.3}$$

where $L_t^{(i)}$ is the amount of water in reservoir i $(i = 1, 2, 3, 4)$ at the beginning of stage t, and $\xi_t^{(i)}$ the stochastic external inflow into reservoir i during stage t. The *decision vector* is $\boldsymbol{u}_t = (u_t^{(1)}, u_t^{(2)}, u_t^{(3)}, u_t^{(4)})^T$, where $u_t^{(i)}$ is the amount of water released from reservoir i

during stage t. Same as it is for inventory forecasting model, $u_t^{(i)}$ is a function of current state variables, and one can write $\boldsymbol{u}_t = \boldsymbol{\gamma}(\boldsymbol{x}_t)$.

4.2.2 State Transition Function

The *transition function* for stage t updates the water levels and external inflows. The amount of water at stage $t+1$ reflects the flow balance between the water that enters (upstream releases and stochastic external inflows) and the water that is released. In order to deal with unexpected peaks from stochastic external inflows, each reservoir has a floodway, so that the amount of water never exceeds the maximum value $W^{(i)}$. Thus, the new water level is

$$L_{t+1}^{(i)} = \min \left\{ L_t^{(i)} + \sum_{j \in U^{(i)}} u_t^{(j)} - u_t^{(i)} + \xi_t^{(i)} \ , \ W^{(i)} \right\} \ ,$$

where $U^{(i)}$ the set of indexes corresponding to the reservoirs which release water into reservoir i. The stochastic external inflows are updated using equation (4.2). Like the inventory forecasting model, the two equations above can be grouped in the compact transition function $\boldsymbol{x}_{t+1} = f_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\epsilon}_t)$.

4.2.3 Objectives and Constraints

The *constraints* for water reservoir network management are placed on the water release $u_t^{(i)}$ in the form:

$$0 \leq u_t^{(i)} \leq \min \left\{ L_t^{(i)} + \sum_{j \in U^i} u_t^{(j)} \ , \ R^{(i)} \right\} \tag{4.4}$$

where $R^{(i)}$ the maximum pumpage capability for reservoir i. This implies nonnegativity of L_{t+1}^i. The *objective function* is intended to maintain a reservoir's water level close to a target value $\tilde{L}_t^{(i)}$ and maximize benefit represented by a concave function $benf$ of the water releases (e.g., power generation, irrigation, etc.). For maintaining targets,

the intuitive cost would be $|L_t^{(i)} - \tilde{L}_t^{(i)}|$, which is a V-shaped function. To facilitate numerical minimization, a smoothed version, denoted by $l(L_t^{(i)}, \tilde{L}_t^{(i)})$, that is analogous to the smoothed cost function for the inventory forecasting problem is used. For modeling the benefits, a nonlinear concave function $benf$ of the following form (for $z \in [0, +\infty)$) is used:

$$benf(z, \mu, \beta) = \begin{cases} (\beta+1)z & 0 \le z \le \frac{1}{3}\mu \\ \beta z + \frac{1}{2}\mu - \frac{45}{\mu^2}(z - \frac{2}{3}\mu)^3 - \frac{153}{2\mu^2}(z - \frac{2}{3}\mu)^4 & \frac{1}{3}\mu \le z \le \frac{2}{3}\mu \\ \beta z + \frac{1}{2}\mu & z \ge \frac{2}{3}\mu \end{cases} .$$

Such a function models a benefit which becomes relevant for "large" values of the water releases, depending on suitable parameters μ and β. In our example, we used $R^{(i)}$ for μ and a small number 0.1 for β. Then, the total cost function at stage t can be written as

$$c_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\epsilon}_t) = \sum_{i=1}^{4} l(L_t^{(i)}, \tilde{L}_t^{(i)}) - \sum_{i=1}^{4} q^{(i)} benf(u_t^{(i)}, \mu^{(i)}, \beta) \ , \tag{4.5}$$

where the $q^{(i)}$,$i = 1, 2, 3, 4$ are weights chosen to balance the benefits against the costs.

For the thirty-dimensional water reservoir model (Cervellera et al. 2005-1), the water reservoir network consists of 10 water basins, and the autoregressive system used to model the random inflow behaviour is of order 2. Therefore, equation(4.2) becomes

$$\xi_t^{(i)} = a_t^{(i)} \xi_{t-1}^{(i)} + b_t^{(i)} \xi_{t-2}^{(i)} + c_t^{(i)} + d_t^{(i)} \epsilon_t^{(i)} \ , \tag{4.6}$$

and the state vector becomes

$$\boldsymbol{x}_t = \left(L_t^{(1)}, L_t^{(2)}, \ldots, L_t^{(10)}, \xi_{t-1}^{(1)}, \xi_{t-1}^{(2)}, \ldots, \xi_{t-1}^{(10)}, \xi_{t-2}^{(1)}, \xi_{t-2}^{(2)}, \ldots, \xi_{t-2}^{(10)} \right)^T \ . \tag{4.7}$$

The index for equations of control vector, constraints and cost function are changed correspondingly.

4.3 Computational Results

For SDP algorithm, all problems described above are tested, and various state space discretization-future value function approximation combinations are implemented to find the best solutions and conduct comparisons of computational requirements. The major attention is paid to whether ANN and NTM are promising alternatives to MARS and OA/OA-LH for SDP, and consequently for SOC. For SG algorithms, only the eight-dimensional water reservoir problem is tested. Several aspects of the SG algorithms are discussed and some comparisons to the SDP algorithm are presented.

4.3.1 SDP Solutions

All problems are solved over three stages ($T = 3$). in SDP algorithm, first an experimental design has to be generated to discretize the state space, and a statistical model is chosen for the future value function approximation; then the approximated future value functions are obtained "off-line" by the SDP algorithm listed in section (2.2) for each discretization-approximation combination; finally "on-line" simulations using the "reoptimizing policy," based on the forward solution of the value function equations as described in Tejada-Guibert et al. (1993) and Chen (1999), are conducted to find the optimal control scheme for a given initial state vector and random sequences.

In this dissertation, OAs, OA-LHs and NTMs are implemented in the SDP algorithm. For the inventory forecasting problem, nine-dimensional designs are generated. For the water reservoir problems, eight-dimensional and thirty-dimensional designs are generated. For the nine-dimensional inventory forecasting problem and eight-dimensional water reservoir problem, strength three ($d = 3$) OAs described in Chen (2001) are utilized, and the OA-LH designs are based on these strength three OAs. Specifically, the strength three OAs with $\lambda = 1$, the smaller size OAs with $p = 11$ levels (thus $N = \lambda p^d = 1 * (11)^3 = 1331$) and the larger size OAs with $p = 13$ levels (thus

$N = \lambda p^d = 1 * (13)^3 = 2197$) in each dimension. For each size ($N = 1331$ and $N = 2197$), one "good," one "neutral," and one "poor" OA is selected based on space-filling measures described in Chen (2001). Two realizations of OA-LH designs are generated for each OA. Thus, a total of 6 OAs and 12 OA-LH designs are generated. For water reservoir network managment problem, only one OA-LH realization is used. The thirty-dimensional OAs for the thirty-dimensional water reservoir problem have strength two ($d = 2$). The smaller size OA has $p = 31$ levels (thus $N = \lambda p^d = 1 * (31)^2 = 961$) and the larger size OAs has $p = 43$ levels (thus $N = \lambda p^d = 1 * (43)^2 = 1849$) in each dimension. The space-filling measures are ignored here for thirty-dimensional water reservoir problem. For each size OA, one realization of OA-LH is generated.

The NTMs tested in this dissertation are Hammersley, Sobol′, Faure and Niederreiter-Xing (NX) point sets. The numbers of points in the NTM points sets are chosen to be comparable to those of OAs and OA-LHs. Specifically, the Faure point sets are downloaded from the website http://www.sequences.com/download_pkzip.html and are subsets of 101-dimensional Faure sequences. It is pointed out by the website owner that the number of points must be used in sequential multiples of 101 to achieve low discrepancy. Thus, this dissertation uses 1313 points for the smaller size point set and 2222 points for the larger size point set. The Sobol′ point sets are generated from the C++ code obtained from http://ldsequences.sourceforge.net, and the NX sequences are constructed from the generator obtained from http://www.dismat.oeaw.ac.at/pirs/niedxing.html. Ideally, Sobol′ and NX point sets will have 2^m points to achieve low descrepancy. We used 1248 and 2197 points for the smaller and larger size point sets of eight and nine dimensions respectively (since 2496 is too far from 2197). The Hammersley point sets are generated by a Matlab code developed by the author, and have 1331 points and 2197 points respectively. Thus we have a total of 8 NTM designs for the nine-dimensional inventory forecasting

problem and eight-dimensional water reservoir problem. For the thirty-dimensional water reservoir problem, only Sobol′ and NX point sets of $N = 961$ and $N = 1849$ are tested.

In summary, the nine-dimensional inventory forecasting problem tested 6 OAs, 12 OA-LHs and 8 NTMs, a total of 26 designs; the eight-dimensional water reservoir problem tested 6 OAs, 6 OA-LHs and 8 NTMs, a total of 20 designs; and the thirty-dimensional water reservoir problem tested 2 OAs, 2 OA-LHs and 4 NTMs, a total of 8 designs.

To use MARS for future value function approximation, a few parameters must be chosen in advance: highest order of interactions, I, number of knots, $Tknot$, and maximum number of terms in the model, M_{max}. The selection of these parameters is heavily dependent on the application and implementer's experience. For this work, some results utilizing OAs and OA-LHs are obtained from Chen (1999) and Cervellera et al. (2005-2), thus the selection of parameters integrates the experiences of research for those papers and trial-and-error practice (a preliminary study is conducted for the eight-dimensional water reservior network management problem by simulating the solutions for a small set of test data). All the parameters are finally selected as below (MARS is not implemented for the thirty-dimensional water reservoir problem):

1. Selection of I:

   For the inventory forcasting problem, it is known there exists significant interaction effects between inventory levels and forecasts due to the capacity constraints, so I is set to 3. For the water reservoir managment problem, 2 and 3 are tried at preliminary study stage, and $I = 2$ is found to give better results. The preliminary study for the water reservior network management problem agrees with the intuition that water reservoir model has simpler structure than the inventory forecasting model.

2. Seclection of $Tknots$:

When OAs of level p are used for discretization, knots can only be placed at the inner interger value of the interval $(0, p)$, i.e., the largest number of eligible knots in each dimnesion is $Tknots = p - 2$, and this largest number is chosen for both the inventory forecasting problem and the water reservoir network management problem. Thus $Tknots = 9$ when using the smaller size OAs, and $Tknots = 11$ when using the larger size OAs. OA-LH and NTM point sets do not have this restriction, so for inventory forcasting problem with smaller size OA-LHs, $Tknots = 9, 17, 35$ is tested, while for the larger size OA-LHs, $Tknots = 11, 17$ is tested, and $Tknots = 35$ for both smaller size and larger size NTMs is tested. For the water reservoir network management problem, $Tknots = 50$ for all OA-LHs and NTMs is tested.

3. Selection of M_{max}:

   For the inventory forecasting problem, $M_{max} = 200, 300$ and $M_{max} = 300, 400$ are tested for the smaller and the larger size OAs and OA-LHs, respectively, and $M_{max} = 100, 200, 300, 400, 500$ are tested for all NTMs. For the water reservoir network management problem, $M_{max} = 100, 200, 300, 400, 500$ are tested in the preliminary study, and $M_{max} = 100$ is chosen for further study and presented here.

Similar to MARS, ANNs require that the number of hidden layers and number of hidden nodes H in each hidden layer to be pre-seclected. This work uses one hidden layer for all implementations, and some or all of $H = 5, 10, 15, 20, 40, 60$ are test. Specifically, for inventory forcasting problem, it is found in Cervellera et al. (2005-2) that $H = 40$ and $H = 60$ give the best solutions for OA and OA-LH designs with $N = 1331$ and $N = 2197$, respectively, so only those results are presented here; while for NTMs, $H = 5, 10, 15, 20$ for both smaller size and larger size designs, in additional to $H = 40$ for smaller size designs, and $H = 60$ for larger size designs are presented. For the eight-dimensional water reservoir network management problem, it is found in the preliminary study that

$H = 10$ is sufficient, so only the results from $H = 10$ are presented here. For the thirty-dimensional water reservoir problem, $H = 10$ and $H = 15$ are tested for designs with $N = 961$ and $N = 1849$, respectively, after a preliminary study.

By choosing discretizations and statistical models as specified above, various discretization-approximation combinations are utilized to obtain the approximated future value functions for each stage, according to the SDP algorithm listed in Chapter 2. Each combination yields a SDP solution. For example, OA-LH with $N = 1331$ and MARS with $I = 3$, $Tknots = 35$, $M_{max} = 300$ yields a SDP solution; Faure point set with $N = 1313$ and ANN with $H = 15$ yields another SDP solution. For inventory forecasting problem, a total of 170 SDP solutions are obtained, and for eight-dimensional water reservoir network management problem, a total of 40 SDP solutions are obtained, while for the thirty-dimensional water reservoir network management problem, a total of 8 SDP solutions are obtained.

After the approximated future value functions for each stage are obtained, simulations are conducted to find the optimal control schemes for given initial state vectors and random sequences, $\boldsymbol{\epsilon}$, to compare the quality of the SDP solutions. The simulation represents the "on-line" forward phase of the optimization procedure: at the beginning of each stage, optimal control vectors are computed by equation (2.5) using the future value functions approximations obtained "off-line." Then, such optimal vectors are fed into the state equation, which provides the state vector for the next stage according to the actual random vector in the system, and the new control vector can be computed in the same way. Starting from each initial vector, a simulation of a certain number of "on-line" random vector sequences is conducted for each SDP solution. For the inventory forecasting problem, 100 initial vectors are randomly chosen, each vector is simulated according to 1000 random sequences; for the eight-diemsional water reservoir network management problem, the simulation is conducted for 50 initial vectors and each over

100 random sequences; for the thirty-dimensional water reservoir problem, one initial vector is chosen and the simulation is conducted over 100 random sequences.

For the inventory forecasting problem, the simulation output provides the 100 mean costs (corresponding to the 100 initial state vectors), each over 1000 sequences. For each initial state vector, the smallest mean cost achieved by the set of 170 SDP solutions is used as the "true" mean cost for computing the "errors" in the mean costs for all 170 solutions. The average over the 100 "true" mean costs is approximately 60.

Likewise, for the eight-diemsional water reservoir network management problem, the simulation output provided the 50 mean costs (corresponding to the 50 initial state vectors), each over 100 sequences. For each initial state vector, the smallest mean cost achieved by the set of 40 SDP solutions is used as the "true" mean cost for computing the "errors" in the mean costs for all solutions. The average over the 50 "true" mean costs is approximately -47.

The way of simulation for the thirty-dimensional water reservoir problem is slightly different from those for the inventory forecasting problem and the eight-dimensional water reservoir problem. Only one initial vector is chosen for the simulation, and the simulation is conducted over 100 random sequences. For a given SDP solution, the simulation output provides the mean costs over the 100 sequences. The smallest cost achieved among the set of 8 SDP solutions for each of the 100 random inflows sequences is used as the "true" cost for that particular sequence. The average "true" mean cost over the 100 sequences (i.e., the average obtained by taking the best cost for each sequence) is approximately -282. This is the mean cost to which the absolute error of the single SDP solution is computed.

Boxplots in the below subsections will display the distributions of these errors, and a discussion of these boxplots is presented.

4.3.1.1 SDP Solutions for Inventory Forecasting Problem

Figures (4.2) through (4.9) display the distribution of absolute error for all 170 SDP solutions.

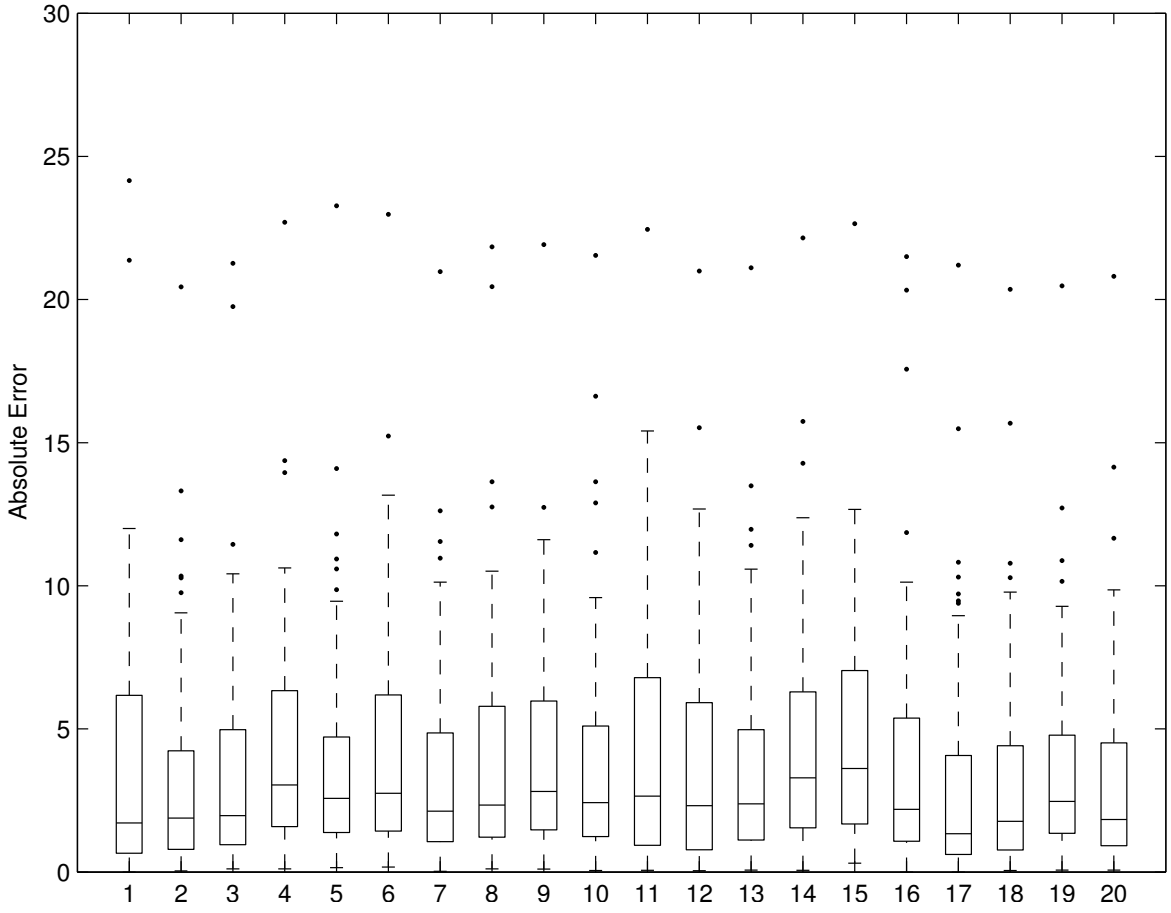

Figure 4.2. Boxplot for inventory forecasting(1).

$1 = MARSI3Tknot50M_{max}100 - Faure1313$; $2 = MARSI3Tknot50M_{max}200 - Faure1313$;

$3 = MARSI3Tknot50M_{max}300 - Faure1313$; $4 = MARSI3Tknot50M_{max}400 - Faure1313$;

$5 = MARSI3Tknot50M_{max}500 - Faure1313$; $6 = MARSI3Tknot50M_{max}100 - Faure2222$;

$7 = MARSI3Tknot50M_{max}200 - Faure2222$; $8 = MARSI3Tknot50M_{max}300 - Faure2222$;

$9 = MARSI3Tknot50M_{max}400 - Faure2222$; $10 = MARSI3Tknot50M_{max}500 - Faure2222$;

$11 = MARSI3Tknot50M_{max}100 - Hammersley1331$; $12 = MARSI3Tknot50M_{max}200 - Hammersley1331$;

$13 = MARSI3Tknot50M_{max}300 - Hammersley1331$; $14 = MARSI3Tknot50M_{max}400 - Hammersley1331$;

$15 = MARSI3Tknot50M_{max}500 - Hammersley1331$; $16 = MARSI3Tknot50M_{max}100 - Hammersley2197$;

$17 = MARSI3Tknot50M_{max}200 - Hammersley2197$; $18 = MARSI3Tknot50M_{max}300 - Hammersley2197$;

$19 = MARSI3Tknot50M_{max}400 - Hammersley2197$; $20 = MARSI3Tknot50M_{max}500 - Hammersley2197$;

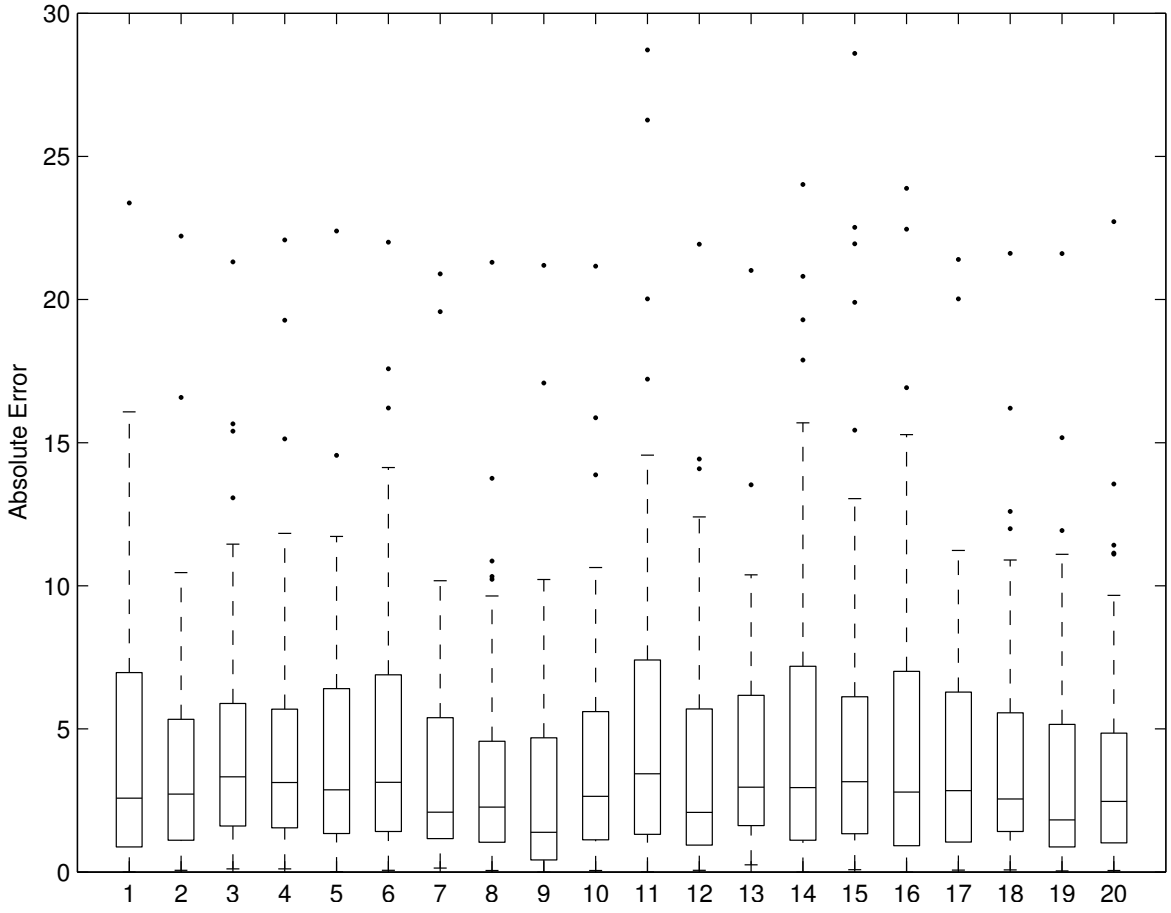

Figure 4.3. Boxplot for inventory forecasting(2).

$1 = MARSI3Tknot50M_{max}100 - NX1331$; $2 = MARSI3Tknot50M_{max}200 - NX1331$;

$3 = MARSI3Tknot50M_{max}300 - NX1331$; $4 = MARSI3Tknot50M_{max}400 - NX1331$;

$5 = MARSI3Tknot50M_{max}500 - NX1331$; $6 = MARSI3Tknot50M_{max}100 - NX2197$;

$7 = MARSI3Tknot50M_{max}200 - NX2197$; $8 = MARSI3Tknot50M_{max}300 - NX2197$;

$9 = MARSI3Tknot50M_{max}400 - NX2197$; $10 = MARSI3Tknot50M_{max}500 - NX2197$;

$11 = MARSI3Tknot50M_{max}100 - Sobol'1331$; $12 = MARSI3Tknot50M_{max}200 - Sobol'1331$;

$13 = MARSI3Tknot50M_{max}300 - Sobol'1331$; $14 = MARSI3Tknot50M_{max}400 - Sobol'1331$;

$15 = MARSI3Tknot50M_{max}500 - Sobol'1331$; $16 = MARSI3Tknot50M_{max}100 - Sobol'2197$;

$17 = MARSI3Tknot50M_{max}200 - Sobol'2197$; $18 = MARSI3Tknot50M_{max}300 - Sobol'2197$;

$19 = MARSI3Tknot50M_{max}400 - Sobol'2197$; $20 = MARSI3Tknot50M_{max}500 - Sobol'2197$;

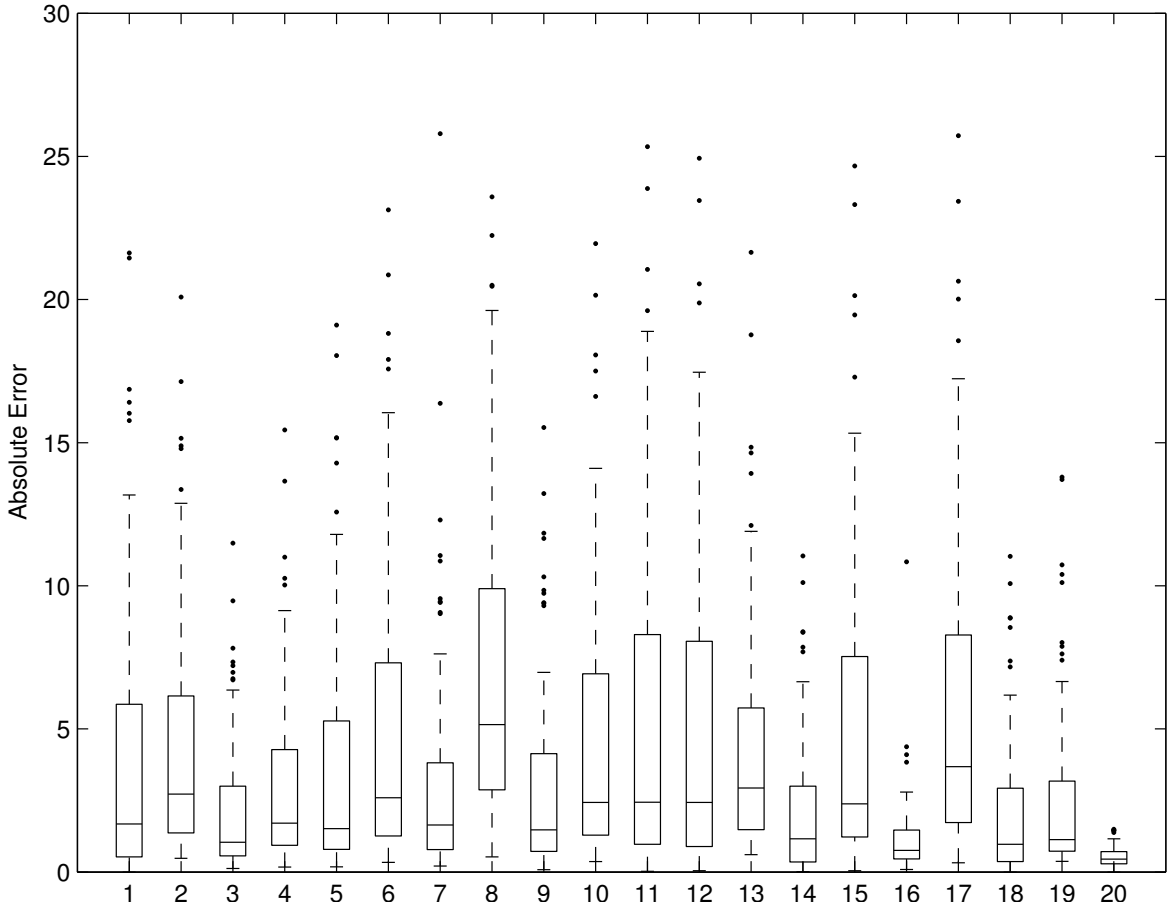

Figure 4.4. Boxplot for inventory forecasting(3).

$1 = ANN5 - Faure1414$; $2 = ANN10 - Faure1414$;

$3 = ANN15 - Faure1414$; $4 = ANN20 - Faure1414$;

$5 = ANN40 - Faure1414$; $6 = ANN5 - Faure2222$;

$7 = ANN10 - Faure2222$; $8 = ANN15 - Faure2222$;

$9 = ANN20 - Faure2222$; $10 = ANN60 - Faure2222$;

$11 = ANN5 - Hammersley1331$; $12 = ANN10 - Hammersley1331$;

$13 = ANN15 - Hammersley1331$; $14 = ANN20 - Hammersley1331$;

$15 = ANN40 - Hammersley1331$; $16 = ANN5 - Hammersley2197$;

$17 = ANN10 - Hammersley2197$; $18 = ANN15 - Hammersley2197$;

$19 = ANN20 - Hammersley2197$; $20 = ANN60 - Hammersley2197$;

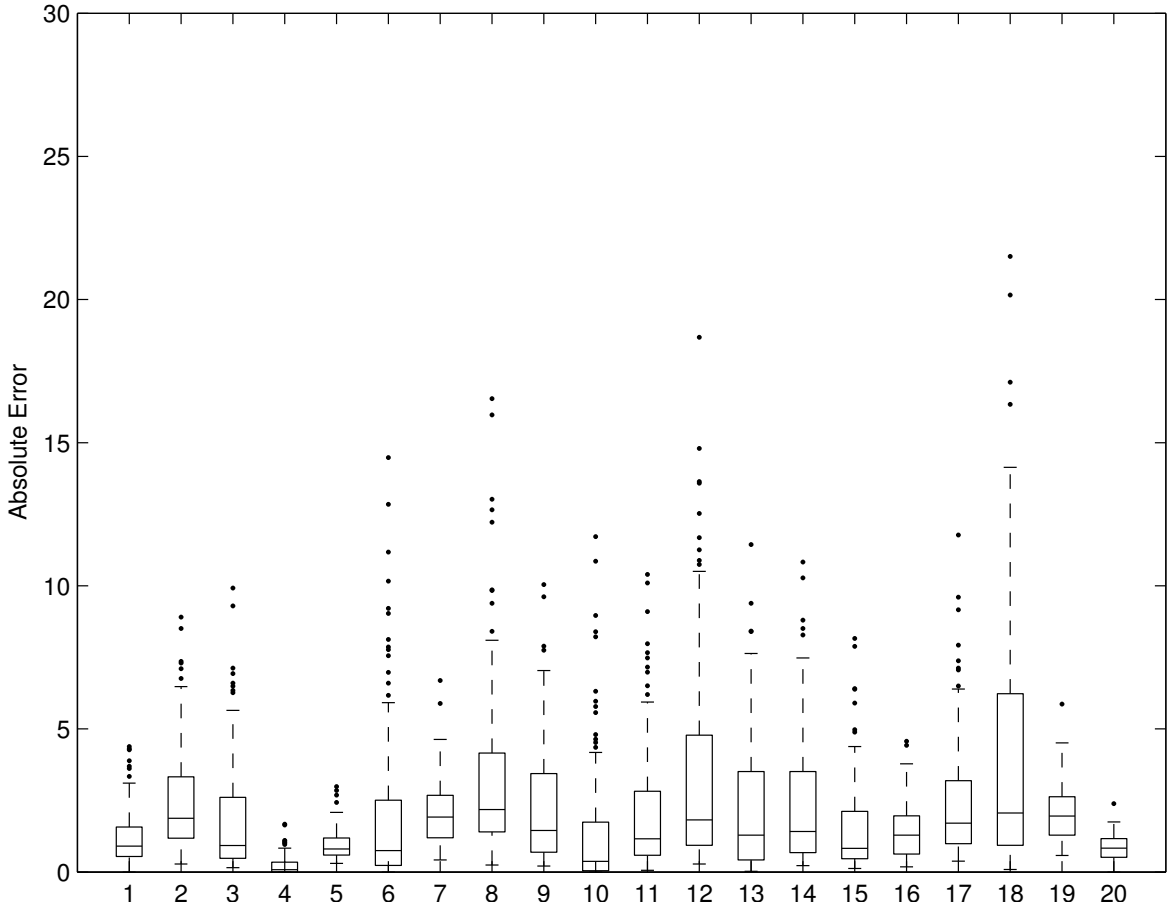

Figure 4.5. Boxplot for inventory forecasting(4).

$1 = ANN5 - NX1331$; $2 = ANN10 - NX1331$;

$3 = ANN15 - NX1331$; $4 = ANN20 - NX1331$;

$5 = ANN40 - NX1331$; $6 = ANN5 - NX2197$;

$7 = ANN10 - NX2197$; $8 = ANN15 - NX2197$;

$9 = ANN20 - NX2197$; $10 = ANN60 - NX2197$;

$11 = ANN5 - Sobol'1331$; $12 = ANN10 - Sobol'1331$;

$13 = ANN15 - Sobol'1331$; $14 = ANN20 - Sobol'1331$;

$15 = ANN40 - Sobol'1331$; $16 = ANN5 - Sobol'2197$;

$17 = ANN10 - Sobol'2197$; $18 = ANN15 - Sobol'2197$;

$19 = ANN20 - Sobol'2197$; $20 = ANN60 - Sobol'1331$;

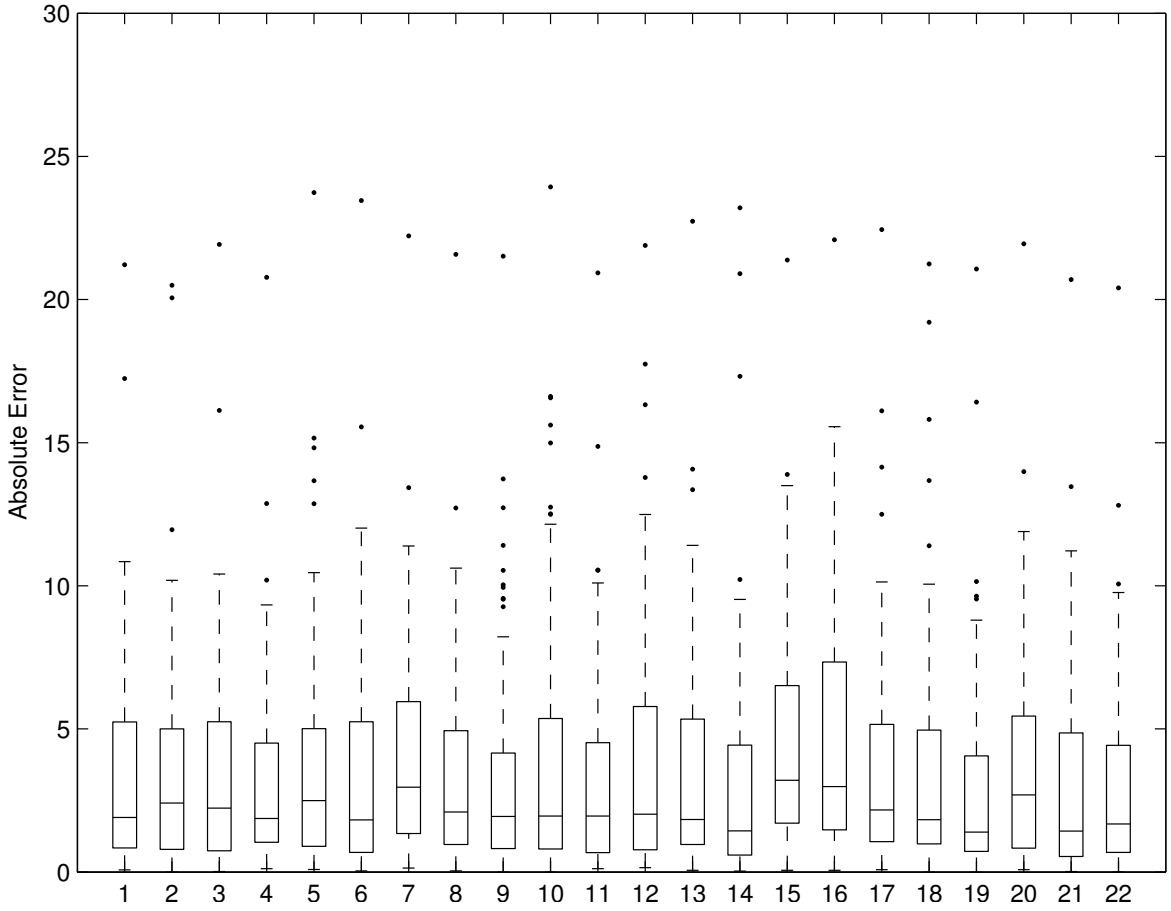

Figure 4.6. Boxplot for inventory forecasting(5).

1 = $MARSI3Tknot9M_{max}200 - OAgood1331$;
2 = $MARSI3Tknot9M_{max}300 - OAgood1331$;
3 = $MARSI3Tknot11M_{max}300 - OAgood2197$;
4 = $MARSI3Tknot11M_{max}400 - OAgood2197$;
5 = $MARSI3Tknot9M_{max}200 - OAneutral1331$;
6 = $MARSI3Tknot9M_{max}300 - OAneutral1331$;
7 = $MARSI3Tknot11M_{max}300 - OAneutral2197$;
8 = $MARSI3Tknot11M_{max}400 - OAneutral2197$;
9 = $MARSI3Tknot9M_{max}200 - OApoor1331$;
10 = $MARSI3Tknot9M_{max}300 - OApoor1331$;
11 = $MARSI3Tknot11M_{max}300 - OApoor2197$;
12 = $MARSI3Tknot11M_{max}400 - OApoor2197$;
13 = $MARSI3Tknot17M_{max}200 - OA - LH1good1331$;
14 = $MARSI3Tknot17M_{max}300 - OA - LH1good1331$;
15 = $MARSI3Tknot35M_{max}200 - OA - LH1good1331$;
16 = $MARSI3Tknot35M_{max}300 - OA - LH1good1331$;
17 = $MARSI3Tknot9M_{max}200 - OA - LH1good1331$;
18 = $MARSI3Tknot9M_{max}300 - OA - LH1good1331$;
19 = $MARSI3Tknot11M_{max}300 - OA - LH1good2197$;
20 = $MARSI3Tknot11M_{max}400 - OA - LH1good2197$;
21 = $MARSI3Tknot17M_{max}300 - OA - LH1good2197$;
22 = $MARSI3Tknot17M_{max}400 - OA - LH1good2197$;

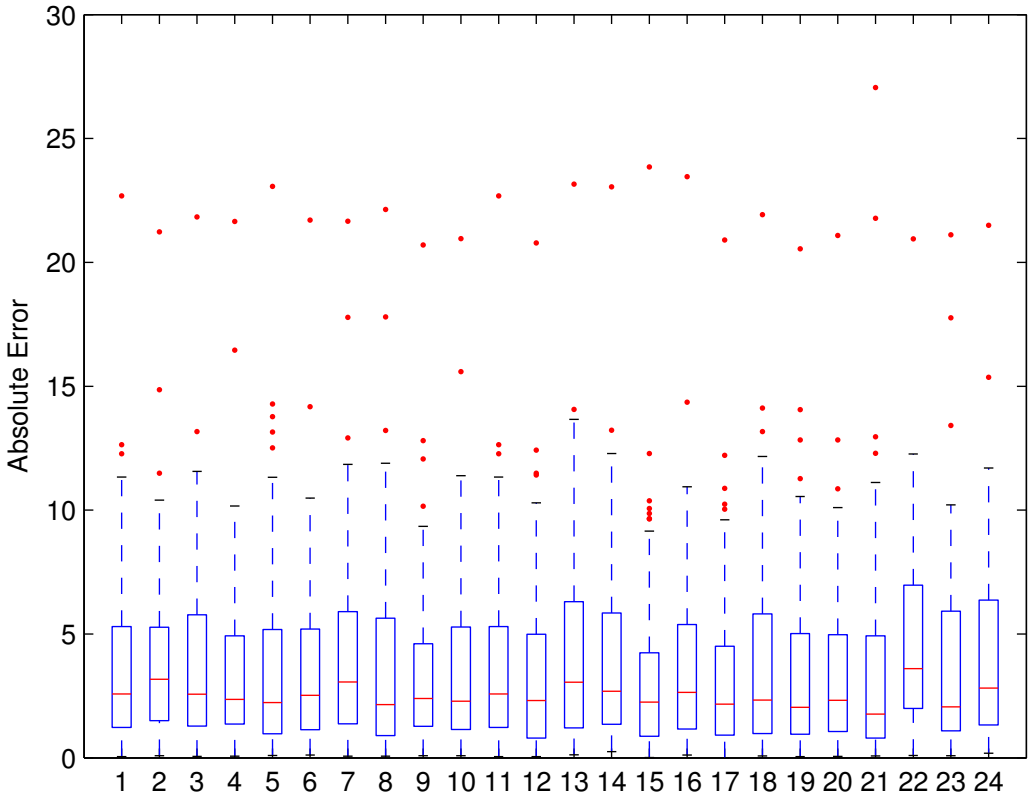

Figure 4.7. Boxplot for inventory forecasting(6).

$1 = MARSI3Tknot17M_{max}200 - OA - LH1neutral1331$;
$2 = MARSI3Tknot17M_{max}300 - OA - LH1neutral1331$;
$3 = MARSI3Tknot35M_{max}200 - OA - LH1neutral1331$;
$4 = MARSI3Tknot35M_{max}300 - OA - LH1neutral1331$;
$5 = MARSI3Tknot9M_{max}200 - OA - LH1neutral1331$;
$6 = MARSI3Tknot9M_{max}300 - OA - LH1neutral1331$;
$7 = MARSI3Tknot11M_{max}300 - OA - LH1neutral2197$;
$8 = MARSI3Tknot11M_{max}400 - OA - LH1neutral2197$;
$9 = MARSI3Tknot17M_{max}300 - OA - LH1neutral2197$;
$10 = MARSI3Tknot17M_{max}400 - OA - LH1neutral2197$;
$11 = MARSI3Tknot17M_{max}200 - OA - LH1neutral1331$;
$12 = MARSI3Tknot17M_{max}300 - OA - LH1poor1331$;
$13 = MARSI3Tknot35M_{max}200 - OA - LH1poor1331$;
$14 = MARSI3Tknot35M_{max}300 - OA - LH1poor1331$;
$15 = MARSI3Tknot9M_{max}200 - OA - LH1poor1331$;
$16 = MARSI3Tknot9M_{max}300 - OA - LH1poor1331$;
$17 = MARSI3Tknot11M_{max}300 - OA - LH1poor2197$;
$18 = MARSI3Tknot11M_{max}400 - OA - LH1poor2197$;
$19 = MARSI3Tknot17M_{max}300 - OA - LH1poor2197$;
$20 = MARSI3Tknot17M_{max}400 - OA - LH1poor2197$;
$21 = MARSI3Tknot17M_{max}200 - OA - LH2good1331$;
$22 = MARSI3Tknot17M_{max}300 - OA - LH2good1331$;
$23 = MARSI3Tknot35M_{max}200 - OA - LH2good1331$;
$24 = MARSI3Tknot35M_{max}300 - OA - LH2good1331$;

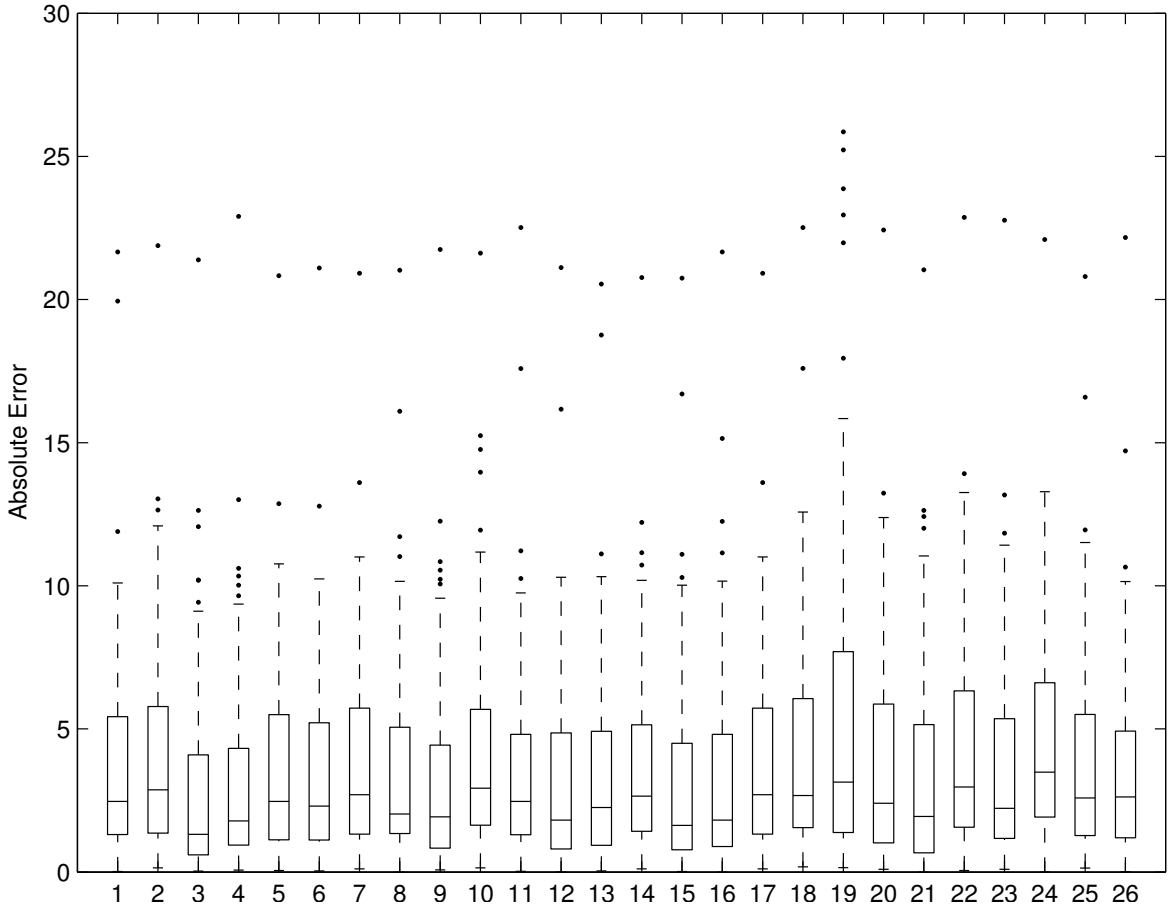

Figure 4.8. Boxplot for inventory forecasting(7).

$1 = MARSI3Tknot9M_{max}200 - OA - LH2good1331$;
$2 = MARSI3Tknot9M_{max}300 - OA - LH2good1331$;
$3 = MARSI3Tknot11M_{max}300 - OA - LH2good2197$;
$4 = MARSI3Tknot11M_{max}400 - OA - LH2good2197$;
$5 = MARSI3Tknot17M_{max}300 - OA - LH2good2197$;
$6 = MARSI3Tknot17M_{max}400 - OA - LH2good2197$;
$7 = MARSI3Tknot17M_{max}200 - OA - LH2neutral1331$;
$8 = MARSI3Tknot17M_{max}300 - OA - LH2neutral1331$;
$9 = MARSI3Tknot35M_{max}200 - OA - LH2neutral1331$;
$10 = MARSI3Tknot35M_{max}300 - OA - LH2neutral1331$;
$11 = MARSI3Tknot9M_{max}200 - OA - LH2neutral1331$;
$12 = MARSI3Tknot9M_{max}300 - OA - LH2neutral1331$;
$13 = MARSI3Tknot11M_{max}300 - OA - LH2neutral2197$;
$14 = MARSI3Tknot11M_{max}400 - OA - LH2neutral2197$;
$15 = MARSI3Tknot17M_{max}300 - OA - LH2neutral2197$;
$16 = MARSI3Tknot17M_{max}400 - OA - LH2neutral2197$;
$17 = MARSI3Tknot17M_{max}200 - OA - LH2neutral1331$;
$18 = MARSI3Tknot17M_{max}300 - OA - LH2poor1331$;
$19 = MARSI3Tknot35M_{max}200 - OA - LH2poor1331$;
$20 = MARSI3Tknot35M_{max}300 - OA - LH2poor1331$;
$21 = MARSI3Tknot9M_{max}200 - OA - LH2poor1331$;
$22 = MARSI3Tknot9M_{max}300 - OA - LH2poor1331$;
$23 = MARSI3Tknot11M_{max}300 - OA - LH2poor2197$;
$24 = MARSI3Tknot11M_{max}400 - OA - LH2poor2197$;
$25 = MARSI3Tknot17M_{max}300 - OA - LH2poor2197$;
$26 = MARSI3Tknot17M_{max}400 - OA - LH2poor2197$;

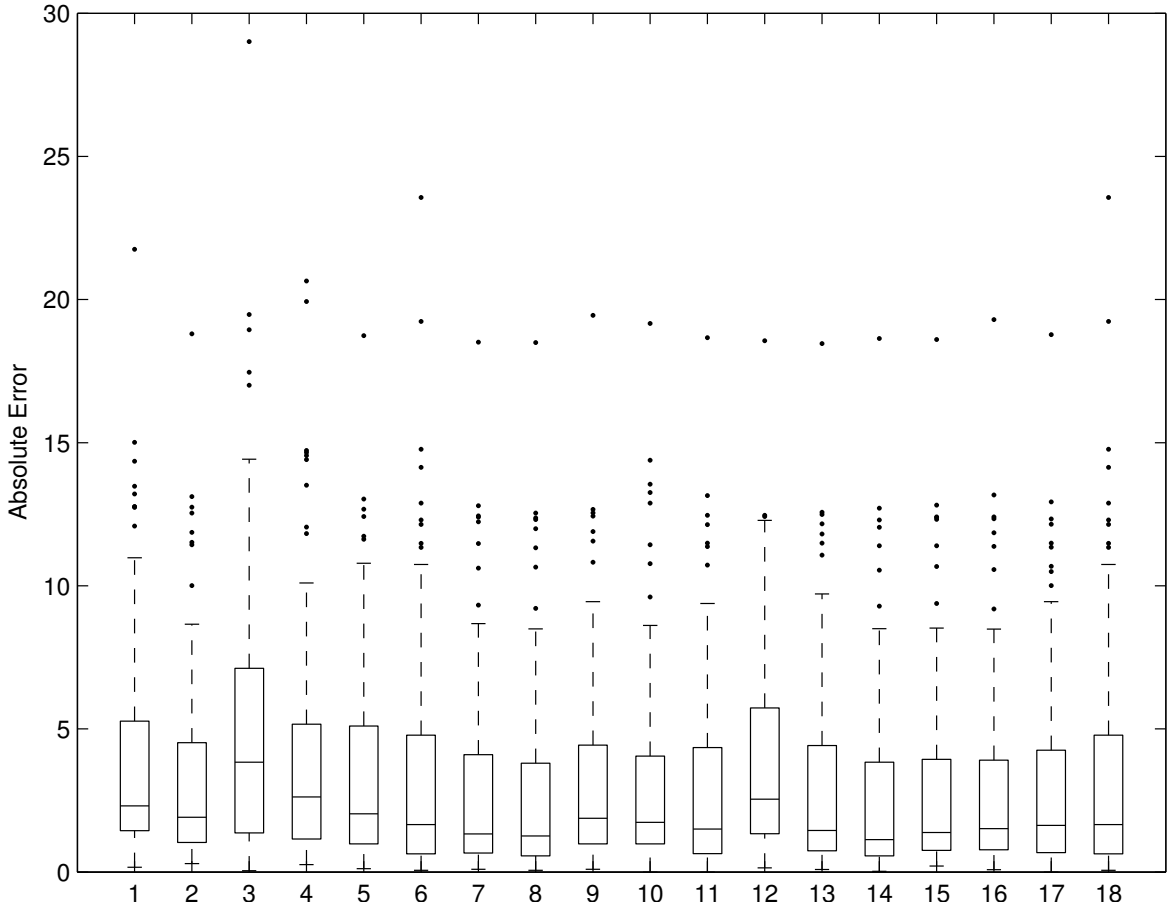

Figure 4.9. Boxplot for inventory forecasting(8).

$1 = ANN40 - OAgood1331$; $2 = ANN60 - OAgood2197$;

$3 = ANN40 - OAneutral1331$; $4 = ANN60 - OAneutral2197$;

$5 = ANN40 - OApoor1331$; $6 = ANN60 - OApoor2197$;

$7 = ANN40 - OA - LH1good1331$; $8 = ANN60 - OA - LH1good2197$;

$9 = ANN40 - OA - LH1neutral1331$; $10 = ANN60 - OA - LH1neutral2197$;

$11 = ANN40 - OA - LH1poor1331$; $12 = ANN60 - OA - LH1poor2197$;

$13 = ANN40 - OA - LH2good1331$; $14 = ANN60 - OA - LH2good2197$;

$15 = ANN40 - OA - LH2neutral1331$; $16 = ANN60 - OA - LH2neutral2197$;

$17 = ANN40 - OA - LH2poor1331$; $18 = ANN60 - OA - LH2poor2197$;

From Figures (4.2) through (4.9) one can see that all MARS solutions and (OA,OA-LH)/ANN solutions are relatively close to each other. Overall best solutions appear in Figure(4.4) and Figure(4.5), which are NTM/ANN combinations, and the best two are ANN($H = 20$)-NX1248 and ANN($H = 60$)-Hammersley2197. It is interesting that one of the overall best solutions is with the Hammersley point set of 2197 points, while the 1331-point Hammersley is one of the worst solutions. Sobol′ and NX achieved overall good solutions. However, the fourth NTM design, which are Faure point sets, does not provide good solutions in this application. For solutions using OAs, "good" OAs achieved better solutions then "neutral" and "poor" ones. All OA-LH solutions are fairly close. To some extent the goodness measures are no longer valid for the OA-LH designs, because the randomization used in generating them may have resulted in a poorly spaced realization. Intuitively, we assume larger size designs achieve better solutions than smaller size ones of the same type; however, sometimes it does not happen this way (in this case, "poor" OA-LH, Faure point sets and NX point sets), and this can be a topic to be investigated in the future.

The results for this application show that it is worth introducing ANN and NTM to SDP algorithm. Overall, ANN solutions provided comparable, if not better, quality than MARS ones; likewise, NTM solutions provided comparable, if not better, quality than OA/OA-LH ones; NTM/ANN combinations provided overall best solutions.

Regarding comparisons of computational requirement of utilizing MARS vs. FF1ANN for future value function approximation, the study of Cervellera et al. (2005-2) showed that they need a similar degree of computation time and memory requirement. Table(1) records the actual average CPU time for computing a value function approximation for a single time period t (which corresponds to (i) solving for future value function value, i.e., solving for minimization problem, equation (2.3) at N points and (ii) approximating the future value function) for $N = 1331$ and $N = 2197$ points. When C is used, the

optimization step utilizes NAG libraries and MARS fitting uses C code from Chen et al. (1999); When Matlab is used, the optimization step and the training of the FF1ANN approximations use "Optimization" and "Neural Network" toolbox respectly. Note that all (OA/OA-LH)/MARS runs were executed by Chen et al. (1999) and on a Pentium II (P2) machine, all (OA/OA-LH)/ANN runs were conducted by Cervellera et al. (2005-2) on 2 other different P2 machines, and all NTM/(MARS,ANN) runs are implemented on a Dual 3.06-GHz Intel Xeon Workstation. Overall, MARS and ANN codes require comparable computation time.

Table 1. Actual average run times (in minutes) for one SDP iteration of inventory forecasting problem

Model	**design and size**	**450MHz P2**	**550MHz P2**	**933MHz P2**	**3.06GHz IX**	**Programming Language**
ANN($H = 40$)	OA,LH 1331	65m		90m		Matlab
MARS($M_{\max} = 300$)	OA,LH 1331		61m			C
ANN ($H = 60$)	OA,LH 2197	185m		87m		Matlab
MARS ($M_{\max} = 400$)	OA,LH 2197		112m			C
ANN ($H = 40$)	NTM 1331				11m	Matlab
MARS ($M_{\max} = 300$)	NTM 1331				9m	C
ANN ($H = 60$)	NTM 2197				24m	Matlab
MARS ($M_{\max} = 400$)	NTM 2197				31m	C

4.3.1.2 SDP Solutions for Eight-dimensional Water Reservoir Network Management Problem

Figures (4.10) and (4.11) display the distribution of absolute error for all 40 SDP solutions with Figure(4.10) showing all the MARS solutions, and Figure(4.11) showing all the ANN solutions. Finally Figure(4.12) shows the overall 10 best solutions on the left and 10 worst solutions on the right.

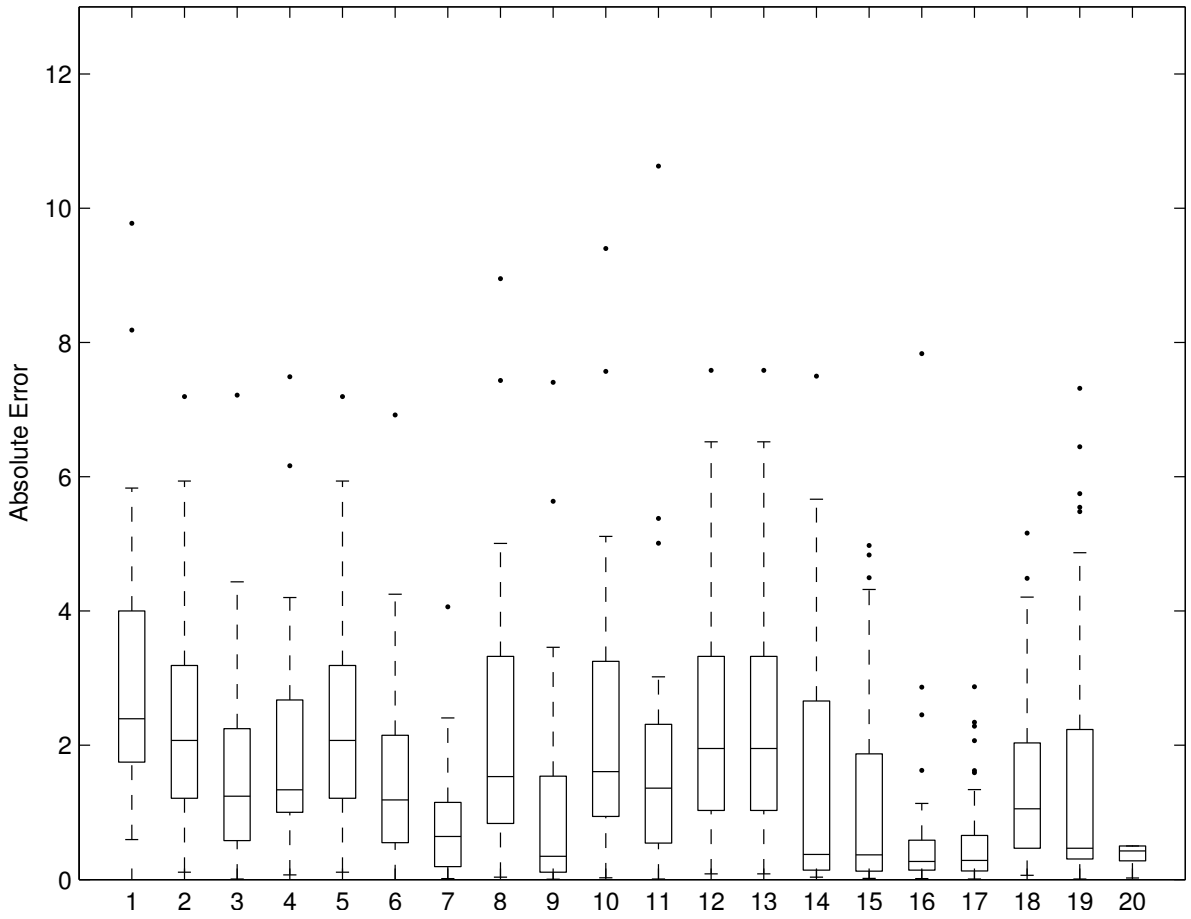

Figure 4.10. Boxplot for eight-dimensional water reservoir network management problem(1). All MARS are of $I = 3$, $Tknot = 50$ (for OAs $Tknot = 9$ when $N = 1331$ and 11 for $N = 2197$), and $M_{max} = 100$.

$1 = MARS - OA1331poor$; $2 = MARS - OA1331neutral$;

$3 = MARS - OA1331good$; $4 = MARS - OA2197poor$;

$5 = MARS - OA2197neutral$; $6 = MARS - OA2197good$;

$7 = MARS - OA - LH1331poor$; $8 = MARS - OA - LH1331neutral$;

$9 = MARS - OA - LH1331good$; $10 = MARS - OA - LH2197poor$;

$11 = MARS - OA - LH2197neutral$; $12 = MARS - OA - LH2197good$;

$13 = MARS - Faure1313$; $14 = MARS - Hammersley1331$;

$15 = MARS - NX1248$; $16 = MARS - Sobol'1248$;

$17 = MARS - Faure2222$; $18 = MARS - Hammersley2197$;

$19 = MARS - NX2197$; $20 = MARS - Sobol'2197$;

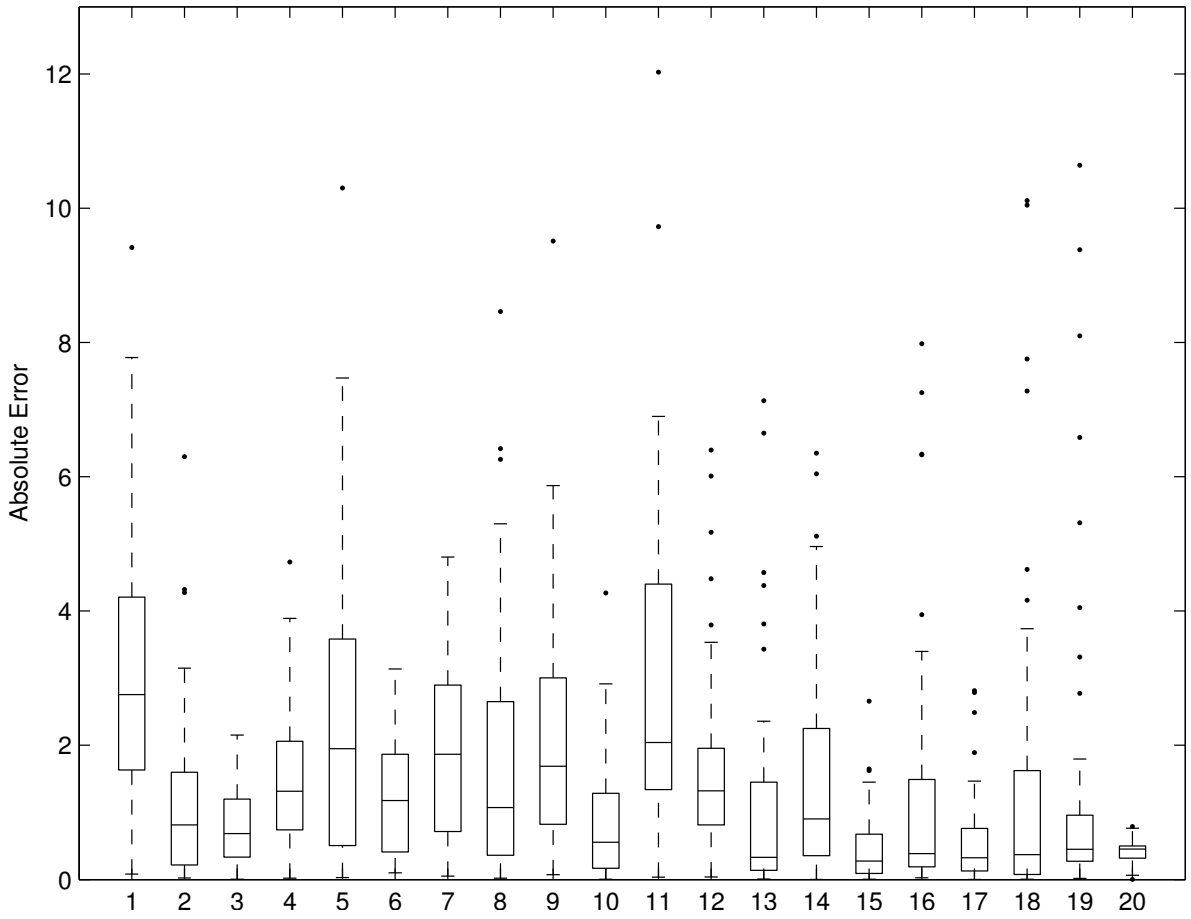

Figure 4.11. Boxplot for eight-dimensinal water reservoir network management problem(2). All ANN is of $H = 10$.

$1 = ANN - OA1331poor$; $2 = ANN - OA1331neutral$;

$3 = ANN - OA1331good$; $4 = ANN - OA2197poor$;

$5 = ANN - OA2197neutral$; $6 = ANN - OA2197good$;

$7 = ANN - OA - LH1331poor$; $8 = ANN - OA - LH1331neutral$;

$9 = ANN - OA - LH1331good$; $10 = ANN - OA - LH2197poor$;

$11 = ANN - OA - LH2197neutral$; $12 = ANN - OA - LH2197good$;

$13 = ANN - Faure1313$; $14 = ANN - Hammersley1331$;

$15 = ANN - NX1248$; $16 = ANN - Sobol'1248$;

$17 = ANN - Faure2222$; $18 = ANN - Hammersley2197$;

$19 = ANN - NX2197$; $20 = ANN - Sobol'2197$;

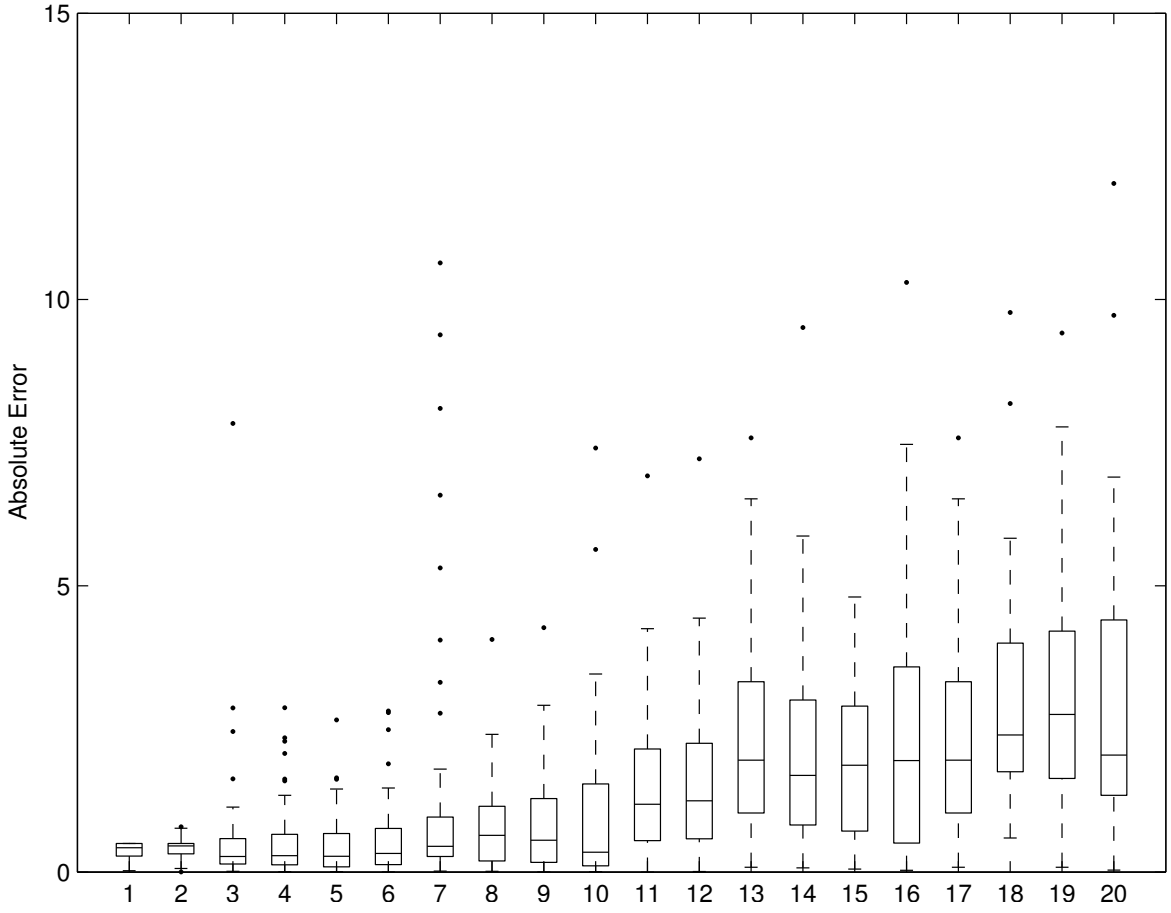

Figure 4.12. Boxplot for eight-dimensional water reservoir network management problem, 10 best solutions v.s. 10 worst solutions.

$1 = MARS - Sobol'2197$; $2 = ANN - Sobol'2197$;

$3 = MARS - Sobol'1248$; $4 = MARS - Faure2222$;

$5 = ANN - NX1248$; $6 = ANN - Faure2222$;

$7 = ANN - NX2197$; $8 = MARS - LH1331poor$;

$9 = MARS - OA - LH1331good$; $10 = ANN - OA - LH2197poor$;

$11 = ANN - OA - LH2197neutral$; $12 = ANN - OA1331poor$;

$13 = MARS - OA1331poor$; $14 = MARS - Faure1331$;

$15 = MARS - OA1331neutral$; $16 = MARS - OA2197neutral$;

$17 = ANN - OA2197neutral$; $18 = ANN - LH1331poor$;

$19 = ANN - LH1331good$; $20 = MARS - LH2197good$;

Within all the MARS solutions, Sobol′ point sets of both the smaller and the bigger size provided best solutions. OA-LH with $N = 1331$ ("poor" and "good" ones) gave pretty good solutions also. Faure point sets, which did not give good solutions in inventory forecasting problem, provided one of the best solution here with $N = 2222$ points. Within all the ANN solutions, Sobol′ of $N = 2197$ and NX of $N = 1248$ gave best solutions. It appears Hammersley point sets did not provide good results in this application. From Figure (4.12) one can see MARS and ANN are overall comparable in quality, this agrees with the findings in the previous application.

In order to conduct a fairer computational comparison on the same programming platform, considerable effort is expended to develop a Matlab code that could access MARS C code (a general description and instruction of calling C code from Matlab can be found in Appendix A) Unfortunately, the ability to call C code from Matlab is still flawed, and the MARS runs suffered significant slow down, especially when the approximated MARS future value function for one stage is used to provide estimation of future value function value for new points in one stage prior. For the water reservoir problem, all runs are conducted at the same place; however, the slow down experienced by calling the MARS C code from Matlab is obvious. Table(2) recordes the average CPU time for one iteration as explained in previous subsection. All runs are executed on Dual 3.06-GHz Intel Xeon Workstation. Unfortunately, at this time, a completely fair comparison that uses the same programming platform is not readily implementable. Overall, the experience for this application demonstrates that the ANN model is easier to be implemented in Matlab.

Table 2. Actual average run times (in minutes) for one SDP iteration of eight-dimensional water reservoir network managment problem

Model	**design size**	**Programming Language**	**time**
ANN($H = 10$)	1331	Matlab	13m
MARS($M_{\max} = 100$)	1331	Matlab Calling C	43m
ANN ($H = 10$)	2197	Matlab	18m
MARS ($M_{\max} = 100$)	2197	Matlab Calling C	63m

4.3.1.3 SDP Solutions for Thirty-dimensional Water Reservoir Network Management Problem

Boxplots in Figure 4.13 display the distributions of the absolute errors for the 8 SDP solutions with respect to the 100 random inflows sequences. The worst solutions are those of the 961-point OA and OA-LH, the former because of the higher absolute error and the latter because of the larger variance. The remaining designs are somewhat equivalent; however, the 1849-point Sobol′ sequence is seen as providing the best solution.

Both the minimization for the computation of the value function at a given point and the training of the FF1ANN approximations are implemented in MATLAB, using the "Optimization" and "Neural Network" toolboxes. Computing a value function approximation for a single time period t required approximately 2 hours and 45 minutes for $N = 961$ points and 10 neural units on a Pentium IV 1.60 GHz machine with 256Mb RAM. The successful sovling of the thirty-dimensional problem verified that SDP algorithm can be used to solve high-dimensional problems with moderate computational requirements, when the state space is efficiently discretized and appropriate statistical model is chosen for function approximation.

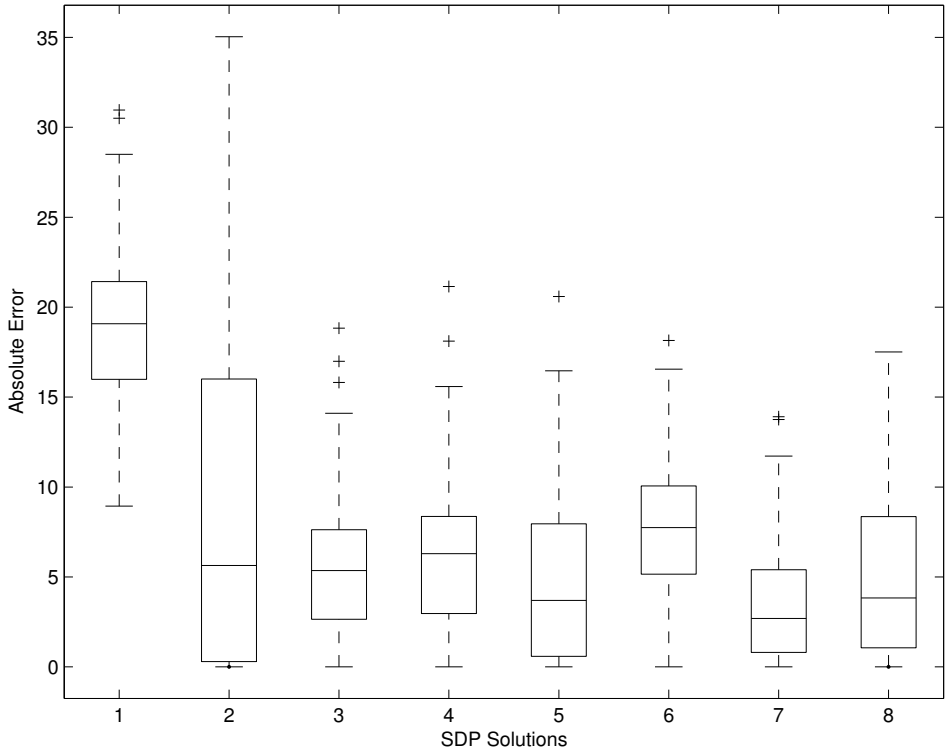

Figure 4.13. Boxplots for thirty-dimensional water reservoir network management problem.

$1 = ANN10 - OA961;$ $2 = ANN10 - OA - LH961;$

$3 = ANN10 - Sobol'961;$ $4 = ANN10 - NX961;$

$5 = ANN15 - OA1849;$ $6 = ANN15 - OA - LH1849;$

$7 = ANN15 - Sobol'1849;$ $8 = ANN15 - NX1849;$

4.3.2 SG Solutions

For the SG algorithms, only the 8-dimensional water reservoir network management problem is tested. To facility the gradient descent method, a penalty for the violation of constraints is added to the cost function so as to transform the original constrained optimization problem to an unconstrained one. The original constraints are of the form as in equation(4.4), i.e.,

$$0 \leq u_t^{(i)} \leq \min \left\{ L_t^{(i)} + \sum_{j \in U^i} u_t^{(j)} \ , \ R^{(i)} \right\}, \quad i = 1, 2, 3, 4, \quad t = 0, 1, 2,$$

Thus the penalty is added as:

$$B * \{ \sum_{t=0}^{2} \sum_{i=1}^{4} \{ [\max(0, -u_t^{(i)})]^2 + [\max(0, (u_t^{(i)} - L_t^{(i)} - \sum_{j \in U^i} u_t^{(j)}))]^2 + [\max(0, u_t^{(i)} - R^{(i)})]^2 \} \},$$

where B is a big enough number to ensure the unconstrained optimization problem converges to the original one, and each $u_t^{(i)}$ is parametrized by the components of $\boldsymbol{\omega}_\nu$. As is for the cost functions (4.1) and (4.5), the smoothed version is implemented for each component in the summation.

A fixed structure has to be preassigned to $\gamma_t^{(i)}(\boldsymbol{x}_t)$, for $i = 1, 2, 3, 4$, and time periods $t = 0, 1, 2$. This dissertation chooses to use the same structure of feedforward one-hidden-layer neural network (FF1ANN) with sigmoidal activation function as that used for SDP tests. Various numbers of hidden nodes are tested and $H = 40$ appeared to be the best option for this application. Thus, each $\gamma_t^{(i)}(\boldsymbol{x}_t)$, $i = 1, 2, 3, 4$, $t = 0, 1, 2$ has the form specified in equation(3.4) and equation(3.5) with $H = 40$. It is easy to calculate that each $\gamma_t^{(i)}(\boldsymbol{x}_t)$, $i = 1, 2, 3, 4$, $t = 0, 1, 2$ has 401 free parameters when $H = 40$ (H is set to be same for all stages and all control functions).

MARS is also planned to be implemented in SG. But the utilization of MARS is not as straightforward as that of ANN structure. The nature character of MARS is "adaptive," which means even the parameters I, $Tknot$ and M_{max} are pre-decided, it has

no fixed structure until seeing the "training" data, and the model is fitted in a manner that addes terms step by step. One can add all the possible terms in the model and use the resulted structure in SG algorithms, but this will cause exponential increasing of number of parameters when the dimension of the problem increases.

4.3.2.1 SG Algorithm-1 Solutions

For SG algorithm-1, the initial state vector is fixed, and every iteration begins system evolution from this fixed initial state vector. For water reservoir network management application, the random sequence involved in the system, $\boldsymbol{\epsilon}$, has a standard normal distribution. Traditional normal random sequences generated based on uniformly distributed points, is tested against the normal sequences generated based on experimental designs to see whether using Quasi-random point sets can provide more information about the stochastic space than the traditional random sequence, so as to speed up the SG algorithm and improve solution quality. Since Sobol′ point sets achieved overall good performance in almost all tests for the SDP algorithm, they are tested for the above task. The same 50 initial state vectors as those used for the SDP algorithm simulation are fed to the SG algorithm-1 (in 50 seperated runs). For each initial vector, 2000 iterations are conducted to update the free parameters, where each iteration involves a normal random sequence generated either from the traditional uniform random numbers, or from the Sobol′ point sets. Figures (4.14) and (4.15) display the trend of the cost during the iterations involving random sequences generated by the traditional way and by using Sobol′ point set, respectively. It can be seen that the cost converges pretty quickly in both cases, but oscillates in both cases even after convergence. It is hard to tell in this example whether the utilizing of the Sobol′ point set for random sequence generation is "better," but the one utilizing Sobol′ point set has a slightly lower mean cost over all the

iterations (the mean cost over the 2000 iterations is -21.6428 in Figure(4.14) and -22.0515 in Figure(4.15)).

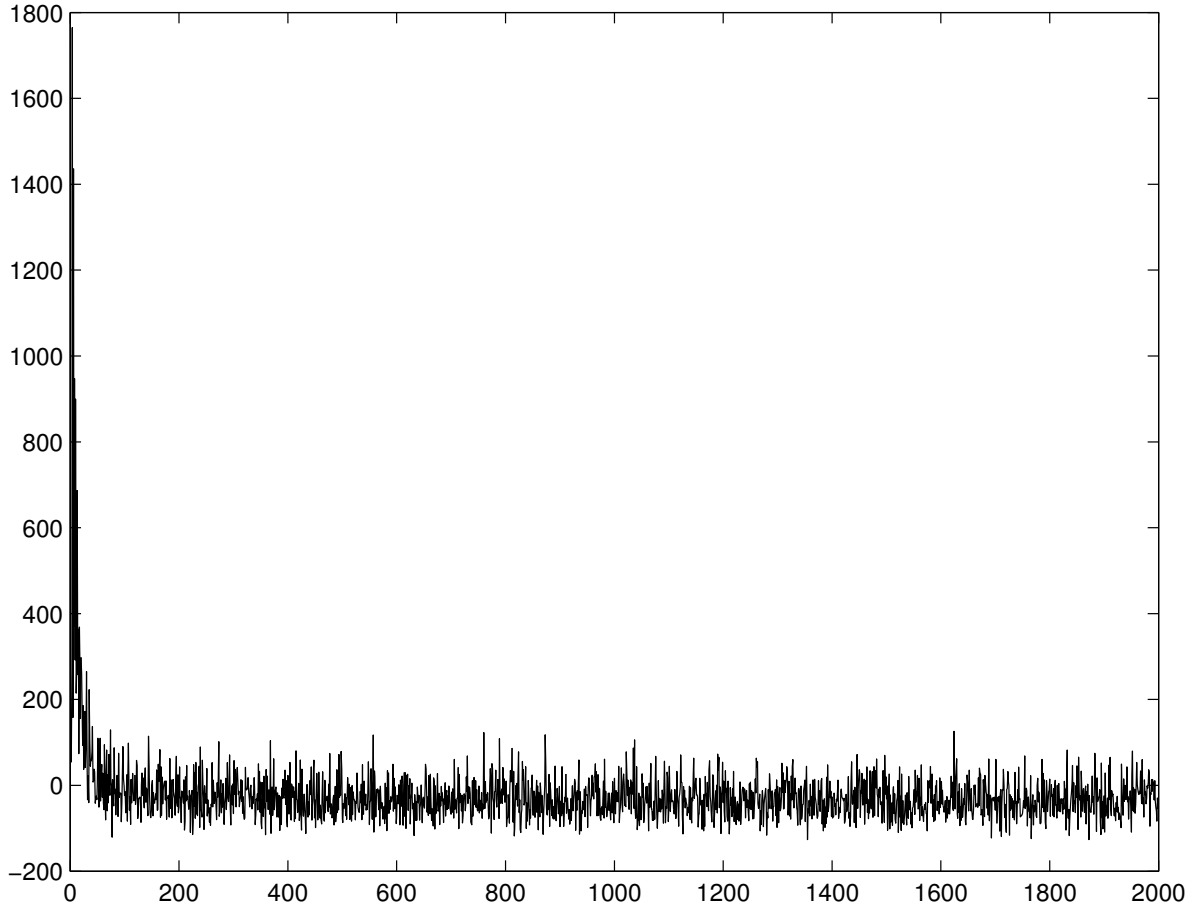

Figure 4.14. An example of cost trend of using SG algorithm-1 for eight-dimensional reservoir network management problem when traditionally generated uniform random numbers are used for normal random sequence generation.

For each initial vector, the final values of the free parameters are recorded, and simulation is conducted utilizing the resulted control functions and the 100 random se-

quences that are tested for SDP algorithm simulation. The simulation evolves the system from stage 0 to stage 3 according to the control functions, state transition functions and random sequences. The average cost over the 100 random sequences for each specific initial vector are compared, and these costs are also compared to the mean (over 40 SDP solutions) meancost(which is over 100 random sequence in the SDP simulation) achieved by the SDP algorithm. Figure(4.16) displays these meancosts, with + represents costs from SDP algorithm, o represents costs from SG algorithm-1 when the Sobol$'$ is used for normal sequence generation, and $*$ represents costs from SG algorithm-1 when the normal sequence is generated traditionally. It appears SG algorithm-1 performs slightly better when Sobol$'$ point set is used for normal sequence generation. In the 50 initial state vectors, 32 achieve lower cost with Sobol$'$. The mean of the cost over the 50 initial vectors is -41.6107 when Sobol$'$ is used, while the mean of these cost over the 50 initial vector is -39.7307 when the random sequence is generated traditionally.

For this application, the results from SG algorithm are overall worse than those achieved by the SDP algorithm: in the 50 initial state vectors, 38 achieve smallest cost when use SDP algorithm, and the mean of the cost over the 50 initial vector is -47 (as reported in section(4.3.1) when SDP algorithm is used.

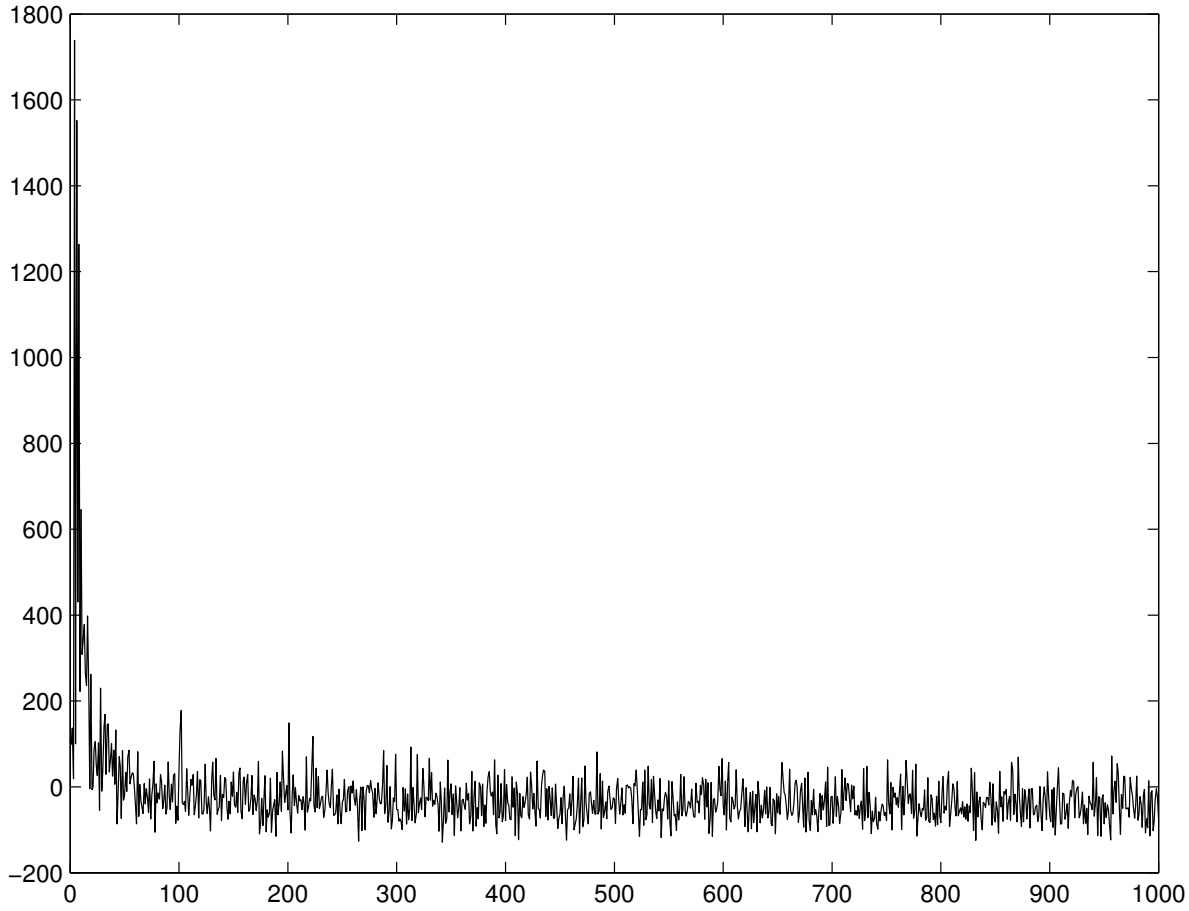

Figure 4.15. An example of the cost trend of using SG algorithm-1 for eight-dimensional reservoir network management problem when the Sobol′ point set is used for normal random sequence generation.

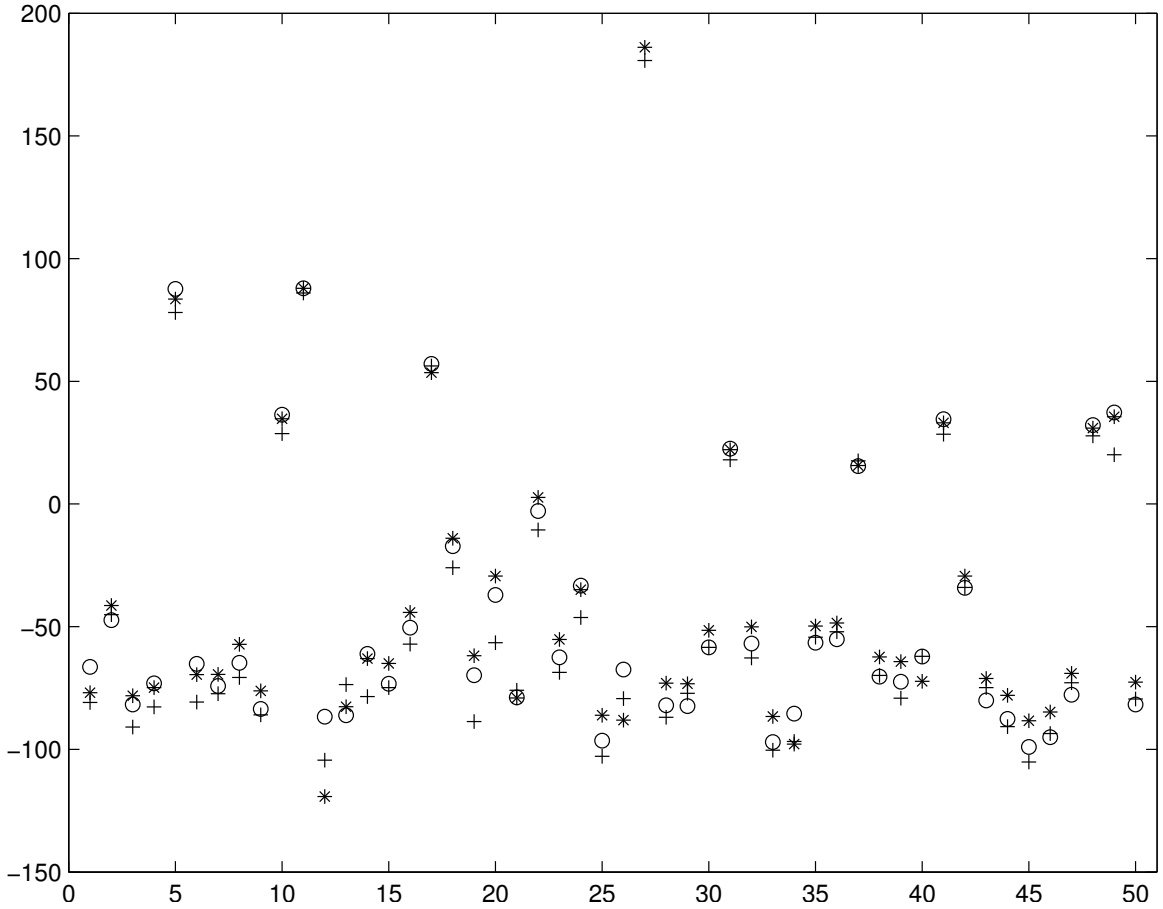

Figure 4.16. Plot of 50 mean costs for the eight-dimensional reservoir network management problem.

+: cost from SDP algorithm,

o: cost from SG algorithm-1 when Sobol′ is used for random sequence generation,

$*$: cost from SG algorithm-1 when the random sequence is generated traditionally.

4.3.2.2 SG Algorithm-2 Solutions

The SG algorithm-2 iterates through both the samples drawn from $\boldsymbol{X}_0$ and the random sequences $\boldsymbol{\epsilon}$. The Sobol′ point set is utilized in two ways here. First an eight-dimensional Sobol′ point set of $N = 1248$ is used to represent the initial state space. Second, a four-dimensional Sobol′ point set of a huge number of points is used to generate the normal sequences as those in SG aLgorithm-1. The Sobol′-based initial vector samples and Sobol′-based normal random sequences combination is tested against random number-based initial vector samples and Puesdo-random number-based normal random sequences combination to see if the utilization of Quasi-random numbers increase solution quality and speed up convergence. In the tests conducted, the initial vector sample set is iterated 100 times, thus a total of $1248 * 100 = 124800$ iterations are run, and the samples are utilized in the manner

$$\underbrace{sample1, sample2, \ldots, sample1248, \ldots, sample1, sample2, \ldots, sample1248}_{100\ times}.$$

The final values of the free parameters are recorded, and simulation is conducted in the same way as that for SG algorithm-1, but this time all 50 initial vectors evolve to stage 3 state vectors according to the same control functions. Figure(4.17) displays the comparison of the mean costs resulted from Sobol′ vs. Psuedo-random numbers. It shows clearly that in SG algorithm-2, Sobol′ point sets work better then traditional uniform random numbers. For all 50 initial vectors, the cost is equal or lower when Sobol′ is used. the mean cost over the 50 initial vectors is -40.7606 when Sobol′ point sets are used, while is -34.3610 when Sobol′ point sets are not used. It is not a surprise that the average cost achieved from SG algorithm-2 is higher then those from SG algorithm-1, since SG algorithm-1 solves the problem specifically for one initial vector at a time. However the disadvantage to the SG algorithm-1 is that the resulted control functions can not be applied to any other initial state vector, and this is not practical in real life.

Notice that again one can see the results obtained form SG algorithm-2 is overall worse than those from SDP algorithm.

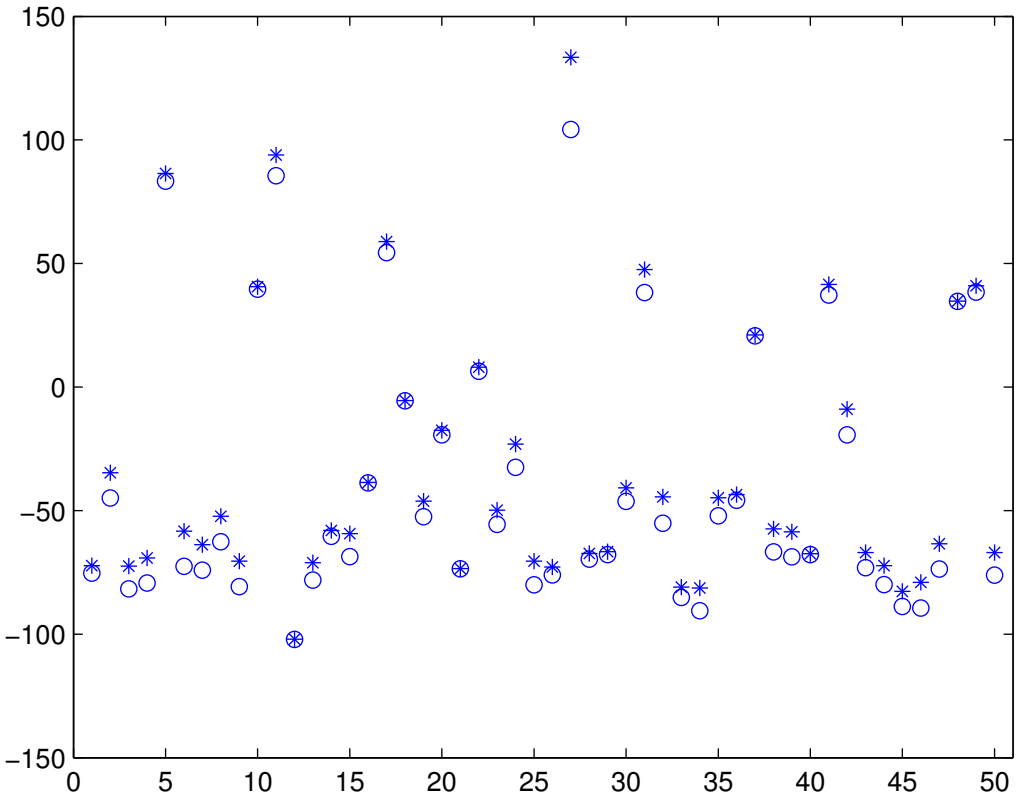

Figure 4.17. Plot of 50 mean costs for the eight-dimensional reservoir network management problem.

o: cost from SG algorithm-2 when Sobol′ is used for random sequences generation and initial state space representation,

$*$: cost from SG algorithm-2 when Psuedo-random numbers are used for random sequences generation and initial state space representation.

4.3.2.3 More Discussions on SG Algorithms

Some aspects related to SG algorithms found in the water reservoir network management application are:

1. About generating random sequences of nonuniform distributions. All the nonuniform random number generators transform the uniform distributed random numbers to the desired nonuniform distributed random numbers in some specific way (Bratley et al. 1987). The most broadly used is the rejection method. In this method, uniform points are generated in a superregion that encloses the region. Then points that do not lie within the region are rejected. More points are generated until enough have been accepted to satisfy the needs. For example, the rejection steps to generate the standard normal distribution are:
   (a) Generate U_1, U_2 uniform on $(0, 1)$.
   (b) Set $X^2 = d^2 - 2log(U_1)$.
   (c) If $U_2X \leq d$, then output X; otherwise go to step (a).

   If $d = 1$, then the procedure is about 66 percent efficient; that is, X is accepted with probability 0.66. If $d = 3$, the efficiency increases to 88 percent. As $d \rightarrow$, the efficiency $\rightarrow$ 100 percent. From this procedure one can see that some U_1 and U_2 are elimimated for any d. When Pusedo-random uniformly distributed variables are replaced with Quasi-random numbers in the procedure, some Quasi-random numbers are eliminated during the procedure, too. However, for NTMs, the low discrepancy is achieved by the whole point set, i.e., if some points are eliminated, the low discrepancy property of the point set will be flawed. Thus, more efficient ways to generate the desired distributed random sequences utilizing Quasi-random numbers needs to be studied.

2. Oscillation of the cost in SG algorithm-1. From equation (2.10) one can see a trend of oscillation is actually inherent in the SG algorithm, since every $\boldsymbol{\epsilon}(k)$ is a (arbitrary) random realization of $\boldsymbol{\epsilon}$, and the difference between $\boldsymbol{\epsilon}(k)$ and $\boldsymbol{\epsilon}(k-1)$ does not decrease when the number of iterations increases. The oscillation is constrained in a certain range, as long as $\boldsymbol{\epsilon}(k)$ is generated according to the distribution of $\boldsymbol{\epsilon}$.
3. Adding a suitable penalty function to transform a constrained optimization problem to an unconstrained one. Ideally, such a function should be differentiable and ensure convergence to the original solution for a finite large value of B. Finding a "good" penalty function is an on-going research topic in the literature (for instance, see Facchinei and Lucidi 1998). Theoretically, B should be a large value to force the constraints to be satisfied, but in the tests conducted in this work, the experience is that B actually has to be assigned to small value (as small as 0.001) to keep the algorithms numerically stable.
4. Solving the non-linear optimization problem. Rather than using a gradient descent method, which solves the constrainted problem approximatedly by adding penalty functions, are there any non-linear, constrained optimization techniques suitable for solving the problem, involving random variables of some continuous stochastic space?
5. Compared to the SDP algorithm, the SG algorithms are faster. Using $H = 40$, SG algorithm-1 required approximately 1.78 minutes for 2000 iterations on a Dual 3.06-GHz Intel Xeon Workstation (but the algorithm actually converges in less than 100 iterations). For SG algorithm-2, using $H = 40$, $N = 1248$ and 100 iterations required approximately 51 minutes on the same machine.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

In this dissertation, two approaches are presented to numerically solve continuous-state SOC problems in the absence of all or some LQG hypotheses. Both approaches integrate the experimental design and statistical modeling techniques from the area of DACE. The first approach is SDP, based on the high dimensional continuous-state SDP work of Chen et al. (1999) and Chen (1999). Their work utilized OA experimental designs to *discretize* the continuous state space, and MARS statistical models to *approximate* the future value function over the continuous state space. The work of Chen et al. (1999) and Chen (1999) is expanded here by considering (1) innovative model-indepedent experimental design schemes, known as NTMs, as alternatives to the OA discretization, and (2) the broadly accessible modeling method, known as ANNs, as alternatives to MARS modeling. The second approach is SG, based on the Ritz method (Ritz 1909) and recent development of Parisini and Zoppoli (1994) and Cervellera (2001). Experimental design and statistical modeling are introduced into the Ritz method to speed up the algorithm and increase solution quality.

For the SDP approach, comprehensive experimental design-statistical modeling combinations are tested for three applications: a nine-dimensional inventory forecasting problem, an eight-dimensional water reservoir network management problem, and a thirty-dimensional water reservoir network management problem. In the nine-dimensional inventory forecasting problem and eight-dimensional water reservoir network management problem, MARS performs similarly using all kinds of experimental designs, and in contrast, ANN solutions noticeably improved with NTM designs. The NTM/ANN combinations provided the overall best solutions, which indicates it is useful to introduce both NTMs and ANNs to the approximate SDP approach. To the best of the author's knowledge, the thirty-dimensional problem is the largest one ever addressed in the reser-

voir management literature without introducing restrictive hypotheses on the cost and/or the model. The employment of experimental designs and ANNs has enabled numeriacal solution to a thirty-dimensional problem taht was unsolvable by classic SDP techniques.

With regard to both computational requirements and quality of solutions in SDP, ANN and MARS are judged to be very competitive with each other. A small advantage of MARS is the information on the relationships between the future value function and the state variables that may be gleaned from the fitted MARS model. The advantage of ANNs is that they are easier to implement with the advanced development of the "Neural Network" toolbox in Matlab.

For the SG approach, only limited experimental tests are conducted on the eight-dimensional water reservoir network management problem. Two versions of SG algorithms are developed for two different situations. In the first situation, the control scheme is found for *one* specific initial state vector. In the second situation, the control scheme is found for *any* initial state vectors. In the first situation, random disturbance sequences have to be generated for each iteration of the algorithm. The results show that SG algorithm performs slightly better when experimental design is used for random disturbance sequences generation. In the second situation, samples from the initial state space as well as the random distubance sequences have to be generated for each iteration. It is shown that the results are clearly better when experimental design generates both of them. The structure of ANNs is used in both situations to approximate the control functions. Because utilizing MARS in SG approach is not as straightforward as that of utilizing ANN, only the structure of ANN is tested at this time.

For the eight-dimensional water reservoir problem tested in this work, the solution accuracy of SDP is generally higher then that of the SG approach. This can be understood since SDP approach approximates a $R^n \rightarrow R^1$ future value function for each stage, while the SG approach approximates a $R^n \rightarrow R^m$ control function for each stage (equivalent

to approximating m $R^n \rightarrow R^1$ functions). However, the SG approach is faster then SDP. One can choose a particular approach depending on the resource available.

The great interest in model-independent experimental designs and statistical modeling by the scientific community indicates that a further understanding and development of these techniques, together with their application in the SOC framework, can provide new interesting opportunities for solving even larger and more difficult problems.

Future research in the area of this dissertation includes the following:

1. Explore possible new statistical modeling method for both SDP and SG approaches, such as Kriging (Sacks et al. 1989) and Support Vector Machines (Cristianini and Shawe-Taylor 2000).
2. Develop a method to automatically decide parameters in statistical modeling, i.e., I, $Tknot$ and M_{max} for MARS, as well as H for ANN. This will permit flexibility in the approximation across different stages.
3. As stated previously, MARS is not suitable for SG algorithms directly. However, it is possible to pre-select some terms in the model, and then use this pre-processed structure in SG algorithms.
4. Compare the SDP and SG algorithms more generally and comprehensively.
5. Develop more effective approaches to implementing experimental designs to generate desired random sequences in SG algorithms.

APPENDIX A

CALLING C FROM MATLAB

Matlab provides interfaces to external routines written in other programming languages, data that needs to be shared with external routines, clients or servers communicating via Component Object Model (COM) or Dynamic Data Exchange (DDE), and peripheral devices that communicate directly with Matlab. Much of this interface capability is formally referred to under the title of the Matlab Application Program Interface, or API. Specifically, large pre-existing *C* and Fortran programs can be called from Matlab without having to be rewritten as M-files, and bottleneck computations (usually for-loops) that do not run fast enough in Matlab can be recoded in *C* or Fortran for efficiency. For this dissertation, an existing *C* code for MARS model fitting and evaluation (written by Dr. Victoria Chen) is adopted, and the SDP procedure for solving reservior management problem is coded in Matlab to make use of the Matlab "Neural Network Toolbox". This require the ability to call the MARS *C* code from Matlab when the future value function is approximated.

Matlab callable *C* and Fortran programs are referred to as *MEX*-files and they are dynamically linked so that the Matlab interpreter can automatically load and execute them. One can call his/her own *C* or Fortran subroutines from Matlab as if they were built-in functions (M-functions). For the Linux platform, the extension for *MEX*-files is *mexglx*. This appendix will focus on how to build *MEX*-files from *C* codes.

C *MEX*-files are built by using the *MEX* script to compile the *C* source code with additional calls to API routines. The Components of a *C MEX*-file consists of two distinct parts: A *computational routine* that contains the code for performing the computations that will be implemented in the *MEX*-file. Computations can be numerical computations as well as input and output data; A *gateway* routine that interfaces the computational routine with Matlab by the entry point *MexFunction* and its parameters *prhs, nrhs, plhs, nlhs*, where *prhs* is an array of right-hand input arguments, *nrhs* is the number of right-hand input arguments, *plhs* is an array of left-hand output arguments, and *nlhs* is the number of left-hand output arguments. The gateway calls the computational routine as a subroutine.

In the gateway routine, one can access the data in the *mxArray* structure and then manipulate this data in the *C* computational subroutine. For example, the expression *mxGetPr(prhs[0])* returns a pointer of type *double ** where *prhs[0]* points to the real data in the *mxArray*. One can then use this pointer like any other pointer of type *double ** in *C*. After calling the *C* computational routine from the gateway, one can set a pointer of type *mxArray* to the data it returns. Matlab is then able to recognize the output from the computational routine as the output from the *MEX*-file.

The two components of the *MEX*-file may be separate or combined. In either case, the files must contain the *#include "mex.h"* header so that the entry point and interface routines are declared properly. The name of the gateway routine **must** always be *mexFunction* and **must** contain the parameters *prhs, nrhs, plhs and nlhs.* i.e., the gateway routine begins like this:

```
void mexFunction(
    int nlhs, mxArray *plhs[],
    int nrhs, const mxArray *prhs[])
{
   /* more C code ... */}
```

The parameters *nlhs* and *nrhs* contain the number of left- and right-hand arguments with which the *MEX*-file is invoked. (In the syntax of the Matlab language, functions have the general form

```
[a,b,c,...] = fun(d,e,f,...)
```

where the ellipsis (...) denotes additional terms of the same format. The a,b,c,... are left-hand arguments and the d,e,f,... are right-hand arguments.)

The parameters *plhs* and *prhs* are vectors that contain pointers to the left- and right-hand arguments of the *MEX*-file. Note that both are declared as containing type *mxArray **, which means that they are pointing to Matlab arrays. Parameter *prhs* is a

length *nrhs* array of pointers to the right-hand side inputs to the *MEX*-file, and *plhs* is a length *nlhs* array that will contain pointers to the left-hand side outputs that the *C* function generates.

Matlab supports many compilers and provides preconfigured files, called options files, designed specifically for these compilers to control which compiler to use and link command options, and the runtime libraries to link against. One can choose his/her own *C* compiler anytime by using the command *MEX -setup* and select corresponding options files. In many cases, one simply can use the default provided *LCC* compiler.

The following is a simple example of *C* code and its *MEX*-file equivalent. The *C* code is a computational function that takes a scalar and doubles it.

```
#include <math.h>
double timestwo(double x)
{double y;
  y= 2.0*x;
return y;
 }
main()
{ double x; double y;
x=5;
y=timestwo(x);
return;}
```

Below is the same function written in the *MEX*-file format.

```
#include <math.h>
double timestwo(double x)
```

```
{double y;
  y= 2.0*x;
}
#include "mex.h"
void mexFunction(int nlhs, mxArray *plhs[], int nrhs,
                 const mxArray *prhs[])
{
  double x, y;
  int mrows, ncols;

  /* Check for proper number of arguments. */
  if (nrhs != 1) {
    mexErrMsgTxt("One input required.");
  } else if (nlhs > 1) {
    mexErrMsgTxt("Too many output arguments");
  }

  /* The input must be a noncomplex scalar double.*/
  mrows = mxGetM(prhs[0]);
  ncols = mxGetN(prhs[0]);
  if (!mxIsDouble(prhs[0]) || mxIsComplex(prhs[0]) ||
      !(mrows == 1 && ncols == 1)) {
    mexErrMsgTxt("Input must be a noncomplex scalar double.");
  }

  /* Create matrix for the return argument. */
```

```
  plhs[0] = mxCreateDoubleMatrix(mrows,ncols, mxREAL);

  /* Assign pointers to each input and output. */
  x = mxGetPr(prhs[0]);
  y = mxGetPr(plhs[0]);

  /* Call the timestwo subroutine. */
  timestwo(x,y);
}
```

Note that in Matlab, one can pass any number or type of arguments to the M-function, which is responsible for argument checking. In C, function argument checking is done at compile time. This is also true for *MEX*-files. The *MEX* program must safely handle any number of input or output arguments of any supported type.

Typically, to create a *MEX*-file from *C* code, one has to change the *main()* function in *C* code to *mexFunction()* as above example. Suppose the above *MEX*-file is saved as timestwo.c, to compile and link this example source file at the Matlab prompt, type

```
>> mex timestwo.c
```

This carries out the necessary steps to create the *MEX*-file called *timestwo.mexglx* in Linux platform. One can now call *timestwo* as if it were an M-function, for example, at the Matlab prompt,

```
>> x = 2;
>> y = timestwo(x)
>> y =
     4
```

The original *C* codes for generating MARS approximation include files *init.c*, *linls.c*, *quintic.c*, *newB.c*, *mars.c*, *marsinput.c* and *mars.h*, with *mars.h* defines all the functions involved in the *main* function written in *marsinput.c*. To call these *C* codes from Matlab, first seperate the function definitions in *mars.h* to their correspoding files, then remove the *main* function from *marsinput.c*, and write the following file named *functionmainmars.c*:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "matrix.h"
void            data_parm       ();
double **       xddata          ();
double **       xdNdataT         ();
double **       scaleN          ();
void            free_dmatrix    ();
unsigned **     xdoedata        ();
double **       scale           ();
void            free_umatrix    ();
double *        dvector         ();
struct Bfcn *   Bfcnvec         ();
double **       stdx            ();
double *        ydata           ();
unsigned        runmars         ();
void            free_dvector    ();
void            free_Bfcnvec    ();
```

```
functionmainmars(argc,iinput1,iinput2,iinput3,iinput4)
int argc;
char *iinput1,*iinput2,*iinput3,*iinput4;
{
  FILE *fpp;
  unsigned Mmax,maxIA,alg3,n,p,N,T,circle,**knot;
  double *SPLN_a,**xd,*xbar,*range,*begin,*end;
  double **xdN,level1,levelN; /* for design with p==N */
  double *smallest,*largest; /* scaled values in MARS*/
  struct Bfcn *MARS_Bf;
  unsigned MARS_M; /* MARS_M is the number of basis functions */
  unsigned t,v;
  char *fpparm,*fpxname,*fpyname,*fpqmars;
  /* int ieee_flags(); */
  double *y,**x;
  unsigned **xdoe;
  /**************/
  /* Initialize */
  /**************/
  if(argc < 2) {
    fpparm="data/marsparm.dat";
    fpxname="data/x.dat";
    fpyname="data/y.dat";
    fpqmars="data/qmars.dat";
  } else if(argc==2) {
    fpparm=iinput1;
```

```
    fpxname="data/x.dat";
    fpyname="data/y.dat";
    fpqmars="data/qmars.dat";
  } else if(argc==3) {
    fpparm=iinput1;
    fpxname=iinput2;
    fpyname="data/y.dat";
    fpqmars="data/qmars.dat";
  } else if(argc==4) {
    fpparm=iinput1;
    fpxname=iinput2;
    fpyname=iinput3;
    fpqmars="data/qmars.dat";
  } else {
    fpparm=iinput1;
    fpxname=iinput2;
    fpyname=iinput3;
    fpqmars=iinput4;
  }
  printf("Parameter file is %s.\n", fpparm);
  printf("X data file is %s.\n", fpxname);
  printf("Y data file is %s.\n", fpyname);
  printf("Output file is %s.\n", fpqmars);
  MARS_M=0;
 data_parm(&circle,&n,&p,&T,&N,&Mmax,&maxIA,&alg3,&begin,&end,
            &level1,&levelN,fpparm,&fpp);
```

```
       if(maxIA>9) {
printf("maxIA is currenlty limited to 9.\n");
printf("To change this, edit struct Bfcnsplit in file mars.h\n");
printf("  and this printout statement in main() in file marsinput.c .\n");
   exit(1);
 } else {
printf("circle,n,p,T,N,Mmax,maxIA,alg3\n");
printf("%u, %u, %u, %u, %u, %u, %u, %u\n",circle,n,p,T,N,Mmax,maxIA,alg3);
 if(!p) /* do not scale x data (x data are NOT a design matrix) */
     xd=xddata(fpxname,n,p,N,&knot,&T,begin,end,fpp);
 else if(p==N) { /* x data are design with sampling levels */
     xdN=xdNdataT(fpxname,n,p,N,&knot,&T,level1,levelN,fpp);
   printf("Knots based on design matrix:\n");
     for(v=1;v<=n;v++) {
       for(t=1;t<=T;t++)
         printf("v %u t %u knot[v][t] %u value %f\n",
v,t,knot[v][t],xdN[v][knot[v][t]]);
     printf("\n");
     }
     xd=scaleN(xdN,level1,levelN,n,p,N,begin,end);
     free_dmatrix(xdN,n,1,1);
   }
   else { /* x data are input as design matrix on grid */
     xdoe=xdoedata(fpxname,n,p,N,&knot,&T,fpp);
   printf("Last 10 rows of design matrix:\n");
     for(v=1;v<=n;v++) {
```

```
        for(t=N-10;t<=N;t++)
          printf("xdoe[%u][%u]=%4u\n",v,t,xdoe[v][t]);
        printf("\n");
      }
      printf("Knots based on design matrix:\n");
      for(v=1;v<=n;v++) {
        for(t=1;t<=T;t++)
          printf("v %u t %u knot[v][t] %u value %u\n",
                  v,t,knot[v][t],xdoe[v][knot[v][t]]);
        printf("\n");
      }
       xd=scale(&circle,xdoe,n,p,N,begin,end,fpp);
      free_umatrix(xdoe,n,1,1);
    }
    fclose(fpp);
    printf("Knots based on scaled/actual x-values:\n");
    for(v=1;v<=n;v++) {
      for(t=1;t<=T;t++)
        printf("v %u t %u knot[v][t] %u value %f\n",
                v,t,knot[v][t],xd[v][knot[v][t]]);
  printf("\n");
}
    printf("Min/Max x-values:\n");
for(v=1;v<=n;v++)
  printf("v %u min %f max %f\n",v,begin[v],end[v]);
  fflush(stdout);
```

```
    /* allocate SPLN_a, MARS_Bf;  reuse space */
    smallest=dvector(n);
    largest=dvector(n);
    MARS_Bf=Bfcnvec(Mmax);
    xbar=dvector(n);
    range=dvector(n);
     SPLN_a=(double *)tmalloc((Mmax+1)*sizeof(double));
    if (!SPLN_a)  nrerror("allocation failure in main()");
      /* x <- centered and scaled xd, save xd,
       smallest and largest become -1 and +1. */
    x=stdx(xd,xbar,range,n,N,begin,end,smallest,largest);
    free_dmatrix(xd,n,1,1);
    y=ydata(fpyname,n,N);
    MARS_M=runmars(x,y,n,fpqmars,MARS_Bf,SPLN_a,smallest,largest,xbar,range,
   N,Mmax,T,maxIA,alg3,knot);
    free_dmatrix(x,n,1,1);
    free_dvector(smallest);
    free_dvector(largest);
    free_umatrix(knot,n,1,1);
    free_Bfcnvec(MARS_Bf);
} /* end else maxIA is okay */
} /* end main */

#include "mex.h"
void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
```

```
{
  int x;
  char *y1,*y2,*y3,*y4;
  int status,mrows,ncols, buf_len;
  char **fnames;       /* pointers to field names */
  FILE f,*fid;
  void *buf;
  /*  Check for proper number of arguments. */
    if (nrhs != 5)
    mexErrMsgTxt("five inputs required.");
  if (nlhs != 0     )
    mexErrMsgTxt("no output required.");
   /* Check to make sure the first input argument is a scalar. */
  if (!mxIsDouble(prhs[0]) || mxIsComplex(prhs[0]) ||
      mxGetN(prhs[0])*mxGetM(prhs[0]) != 1) {
    mexErrMsgTxt("Input x must be a scalar.");
  }
   /* Get the scalar input x. */
  x = mxGetScalar(prhs[0]);
  /* Get the inputs */
 mrows = mxGetM(prhs[1]);
 ncols = mxGetN(prhs[1]);
 buf_len = (mrows * ncols) + 100;
 /*+100 to ensure long file names are accepted*/
     y1 = mxCalloc(buf_len, sizeof(char));
     y2 = mxCalloc(buf_len, sizeof(char));
```

```
    y3 = mxCalloc(buf_len, sizeof(char));
    y4 = mxCalloc(buf_len, sizeof(char));
    mxGetString(prhs[1], y1, buf_len);
  mxGetString(prhs[2], y2, buf_len);
  mxGetString(prhs[3], y3, buf_len);
  mxGetString(prhs[4], y4, buf_len);
  /* Call the C subroutine. */
  functionmainmars(x,y1,y2,y3,y4);
}
```

To compile and link the *C* code in Matlab, at the Matlab prompt type (note that *mars.h* is not used) :

```
>> mex functionmainmars.c init.c lin_ls.c quintic.c newB.c mars.c marsinput.c
```

Matlab compiles the *C* codes and creates the *MEX*-file called *functionmainmars.mexglx* in Linux platform. One can now call *functionmainmars* as if it were an M-function, for example, at the Matlab prompt, type

```
>> functionmainmars(5,'marspar.dat','xinput.dat','youtput.dat','marsresult.dat')
```

will generate the MARS approximation file and save the output in *marsresult.dat*, utilizing *marspar.dat* as the parameter file, and the data in *xinput.dat* and *youtput.dat* as input and output, while "5" indicates there are totally 5 arguments (include "5" itself) as input to *functionmainmars*.

REFERENCES

Abdelaziz, A. Y. , M. R. Irving, M. M. Mansour, A. M. El-Arabaty, A. I. Nosseir (1997). "Neural network-based adaptive out-of-step protection strategy for electrical power systems." *International Journal of Engineering Intelligent Systems for Electrical Engineering and Communications*, **5(1)**, pp. 35–42.

Archibald, T. W., K. I. M. McKinnon and L. C., Thomas (1997). "An aggregate stochastic dynamic programming model of multireservoir Systems. " *Water Resources Research*, **33**, pp. 333–340.

Baglietto, M., C. Cervellera, T. Parisini, M. Sanguineti, and R. Zoppoli (2001). "Approximating Networks for the Solution of T-stage Stochastic Optimal Control Problems." *Proceedings of the IFAC Workshop on Adaptation and Learning in Control and Signal Processing.*

Barron, A. R. (1993). "Universal Approximation Bounds for Superpositions of a Sigmoidal Function." *IEEE Trans. on Inf. Theory*, **39**, pp. 930–945

Barron, A. R., R. L. Barron, and E. J. Wegman (1992). "Statistical Learning Networks: a Unifying View." *Computer Science and Statistics: Proceedings of the 20th Symposium on the Interface*, pp. 192–203.

Bellman, R. E. (1957). *Dynamic Programming.* Princeton: Princeton University Press.

Benveniste, A. , M. Métivier, and P. Priouret (1990). *Adaptive Algorithms and Stochastic Approximation.* Heideberg: Springer-Verlag.

Bertsekas, D. P. and J. N. Tsitsiklis (1996). *Neuro-Dynamic Programming.* Belmont, Massachusetts: Athena Scientific.

Bertsekas, D. P. and J. N. Tsitsiklis (2000). *Dynamic Programming and Optimal Control.* Belmont, Massachusetts: Athena Scientific.

Bratley, P. , L. F. Bennett and E. S. Linus (1987). *A Guide to Simulation.* Berlin: Springer-Verlag.

Bose, R. C. and K. A. Bush (1952). "Orthogonal Arrays of Strength Two and Three." *Annals of Mathematical Statistics*, **23**, pp. 508–524.

Box, G. E. P. and N. R. Draper (1987) *Empirical Model Building and Response Surfaces.* New York, NY: John Wiley & Sons.

Brogan, W. L. (1991) *Modern Control Theory.* Upper Saddle River, NJ: Prentice Hall.

Bryson, A. E. and Y. C. Ho (1975). *Applied Optimal Control: Optimization, Estimation, and Control.* New York, NY: John Wiley & Sons.

Carey, W. P. and S. S. Yee (1992). "Calibration of nonlinear solid-state sensor arrays using multivariate regression techniques." *Sensors and Actuators, B: Chemical*, **B9(2)**, pp. 113–122.

Cervellera, C. (2001). *Approximating Networks for the Solution of T-stage Stochastic Optimal Control Problems.* Ph.D. Dissertation, University of Genoa, Italy.

Cervellera, C., V. Chen, and A. Wen (2005). "Optimization of a Large-scale Water Reservoir Network by Stochastic Dynamic Programming with Efficient State Space Discretization." *Eutopean Journal of Opeational Research*,

Cervellera, C., A. Wen, and V. Chen (2005). "Neural Network and Regression Spline Value Function Approximations for Stochastic Dynamic Programming." *Computers and Operations Research*, to appear.

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

Chen, V. C. P. (2001). "Measuring the Goodness of Orthogonal Array Discretizations for Stochastic Programming and Stochastic Dynamic Programming." *SIAM Journal of Optimization*, **12**, pp. 322–344.

Chen, V. C. P., J. Chen, and M. B. Beck (2000). "Statistical Learning within a Decision-Making Framework for More Sustainable Urban Environments." In *Proceedings of the*

Joint Research Conference on Statistics in Quality, Industry, and Technology, Seattle, Washington, June 2000.

Chen, V. C. P., D. Günther and E. L. Johnson (2003). "Solving for an Optimal Airline Yield Management Policy via Statistical Learning." *Journal the Royal Statistical Society, Series C*, **52 Part 1**, pp. 1–12.

Chen, V. C. P. and D. K. Rollins (2000). "Issues Regarding Artificial Neural Network Modeling for Reactors and Fermenters." *Bioprocess Engineering*, **22**, pp. 85–93.

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

Chen, V. C. P., K.-L. Tsui, R. R. Barton, and J. K. Allen (2003). "A Review of Design and Modeling in Computer Experiments." In *Handbook of Statistics: Statistics in Industry* (R. Khattree and C. R. Rao, eds.), **22**, Amsterdam: Elsevier Science, pp. 231–261.

Chen, V. C. P. , K.-L. Tsui, R. R. Barton, and M. Meckesheimer (2005). "Design, Modeling, and Applications of Computer Experiments." *IIE Transactions*, accepted.

Cheng, B. and D. M. Titterington (1994). "Neural Networks: a Review from a Statistical Perspective (with discussion)." *Statistical Science*, **9**, pp. 2–54.

Cherkassky, V. and F. Mulier (1998). *Learning from Data: Concepts, Theory and Methods*. New York: Wiley.

Cristianini, N. and J. Shawe-Taylor (2000). *An Introduction to Support Vector Machines*. Cambridge, UK: Cambridge University Press.

Culver, T. B. and C. A. Shoemaker (1997). "Dynamic Optimal Ground-Water Reclamation with Treatment Capital Costs." *Journal of Water Resources Planning and Management*, **123**, pp. 23–29.

De Veaux, R. D. , D. C. Psichogios, and L. H. Ungar (1993). "Comparison of two nonparametric estimation schemes: MARS and neural networks." *Computers & Chemical Engineering*, **17(8)**, pp. 819–837.

Fang, K.-T. and Y. Wang (1994). *Number-theoretic Methods in Statistics.* London: Chapman & Hall.

Facchinei, F. and S.Lucidi (1998). "Convergence to second order stationary points in inequality constrainted optimization." *Mathematics of Operation Research*, **23(3)**,pp. 746–765.

Faure, H. (1982). "Discrepance de suites associees a un systeme de numeration (en dimension s)." *Acta Arithmetica*, **XLI**, pp. 337-351.

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

Friedman, J. H. (1991). "Multivariate Adaptive Regression Splines (with discussion)." *Annals of Statistics*, **19**, pp. 1–141.

Gal, S. (1979). "Optimal Management of a Multireservoir Water Supply System." *Water Resources Research*, **15**, pp. 737–748.

Griffin, W. L. , N. I. Fisher, J. H. Friedman, C. G. Ryan (1997). "Statistical techniques for the classification of chromites in diamond exploration samples." *Journal of Geochemical Exploration*, **59(3)**, pp. 233–249.

Hadley, G. and T. M. Whitin (1963). *Analysis of Inventory Systems.* Englewood Cliffs, NJ: Prentice-Hall.

Hagan, M. T. and M. Menhaj (1994). "Training Feedforward Networks with the Marquardt Algorithm." *IEEE Trans. on Neur. Networks*, **5**, pp. 989–993.

Halton, J. H. (1960). "On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals". *Journal of Numerical Mathematics*, **2**, pp. 84–90.

Hammersley, J. M. (1960). "Monte Carlo Methods for Solving Multivariable Problems." *Annals of New York Academic Science*, **86**, pp. 844-874.

Haykin, S. S. (1999). *Neural Networks: A Comprehensive Foundation*, second edition. Upper Saddle River, NJ: Prentice Hall.

Heath, D. C. and P. L. Jackson (1991). "Modelling the Evolution of Demand Forecasts with Application to Safety Stock Analysis in Production/Distribution Systems." Technical Report #989, School of Operations Research & Industrial Engineering, Cornell University, Ithaca, NY.

Hedayat, A. S. and W. D. Wallis (1978). "Hadamard Matrics and their Applications." *Annnals of Statistics*, **6**, pp. 1184–1238.

Hua, L. K., and Y. Wang (1981). *Applications of Number Theory to Numerical Analysis.* Berlin: Springer-Verlag.

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

Kottapalli, S. (2002). "Neural network based representation of UH-60A pilot and hub accelerations." *Journal of the American Helicopter Society*, **47(1)**, pp. 33–41.

Kuhnert, P. M. , K.-A. Do, and R. McClure (2000). "Combining non-parametric models with logistic regression: An application to motor vehicle injury data." *Computational Statistics and Data Analysis*, **34(3)**, pp. 371–386.

Kutner, M. H. , C. J. Nachtsheim, J. Neter, and W. Li (2005). *Applied Linear Statistical Models*, Fifth Edition. New York: McGraw-Hill/Irwin.

Kushner, H. J. and G. Yin (1997). *Stochastic Approximation Algorithms and Applications.* New York, NY: Springer-Verlag.

Labossiere, P. and N. Turkkan (1993). "Failure prediction of fibre-reinforced materials with neural networks." *Journal of Reinforced Plastics and Composites*, **12(12)**, pp. 1270–1280.

Lamond, B. F., and M. J. Sobel, (1995). "Exact and approximate solutions of affine reservoirs models." *Operations Research*, **43**, pp. 771–780.

Li, Q. S., D. K. Liu, J. Q. Fang, A. P. Jeary, and C. K. Wong (2000). "Damping in buildings: Its neural network model and AR model." *Engineering Structures*, **22(9)**, pp. 1216–1223.

Lippmann, R. P. (1987). "An Introduction to Computing with Neural Nets." *IEEE ASSP Magazine*, pp. 4–22.

Liu, J. (2001). "Prediction of the molecular weight for a vinylidene chloride/vinyl chloride resin during shear-related thermal degradation in air by using a back-propagation artificial neural network." *Industrial and Engineering Chemistry Research*, **40(24)**, pp. 5719–5723.

Lewis, F. L. and V. L. Syrmos (1995). *Optimal control.* New York, NY: John Wiley & Sons.

Myers, R. H. and D. C. Montgomery (1995). *Response Surface Methodology: Process and Product Optimization Using Designed Experiments.* New York, NY: John Wiley & Sons.

Niederreiter, H. (1992). *Random Number Generation and Quasi-Monte Carlo Methods.* Philadelphia: SIAM.

Niederreiter, H. , and C. P. Xing (1995). "Low-discrepancy sequences obtained from algebraic function fields over finite fields." *Acta Arithmetica* **72** pp. 281–298.

Niederreiter, H. , and C. P. Xing (1996). "Low-discrepancy sequences and global function fields with many rational places." *Finite Fields and Their Application* **2** pp. 241–273.

Owen, A. B. (1998). "Scrambling Sobol$'$*rime* and Niederreiter-Xing Points." *Journal of Complexity*, **14**, pp. 466–489.

Paoageorgiou, A. and J. F. Traub (1997) "Faster evaluation of multidimensional integrals." *Computational Physics*, **11**, pp. 574–578.

Pham, D. T. and B. J. Peat (1999). "Automatic learning using neural networks and adaptive regression." *Measurement and Control*, **32(9)**, pp. 270–274.

Pigram, G. M. and T. R. Macdonald (2001). "Use of neural network models to predict industrial bioreactor effluent quality." *Environmental Science and Technology*, **35(1)**, pp. 157–162.

Puterman, M. L. (1994). *Markov Decision Processes.* New York, NY: John Wiley & Sons, Inc.

Rao, C. R. (1946). "Hypercubes of Strength '*d*' Leading to Confounded Designs in Factorial Experiments." *Bulletin of the Calcutta Mathematical Society*, **38**, pp. 67–78.

Ripley, B. D. (1993). "Statistical Aspects of Neural Networks." In *Networks and Chaos - Statistical and Probabilistic Aspects* (O. E. Barndorff-Nielsen, J. L. Jensen, and W. S. Kendall, eds.). New York: Chapman & Hall, pp. 40–123.

Ritz, W. (1909). "Uber eine neue methode zur losung gewisser variationsprobleme der mathematischen physik." *J.fur die reine und angewandte Math*, **135**, pp. 1–61.

Robbins, H. and S. Monro (1951). " A Stochastic Approximation Method." *The Annals of Mathematical Statistics*, **22**, pp. 400–407.

Parisini, T. and R. Zoppoli (1994). "Neural Networks for Feedback Feedforward Nonlinear Control Systems." *IEEE Transaction on Neural Networks*, **5**, pp. 436–449.

Parisini, T. and R. Zoppoli (1998). "Neural Approximations for Infinite-horizon Optimal Control of Nonlinear Stochastic Systems." *IEEE Transaction on Neural Networks*, **9**, pp. 1388–1408.

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

Rumelhart, D. E. , G. E. Hinton and R. J. Williams (1986). "Learning Internal Representations by Error Propagation." In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (D. E. Rumelhart and J. L. McClelland, eds.). Cambridge, MA: MIT Press, pp. 318–362.

Sacks, J. S. B. Schiller and W. J. Welch (1989). "Designs for Computer Experiments," *Technometrics*, **31(1)**, pp. 41–47.

Shoemaker, C. A. (1982). "Optimal Integrated Control of Univoltine Pest Populations with Age Structure," *Operations Research*, **30**, pp. 40–61.

Smets, H. M. G. and W. F. L. Bogaerts (1992). "Deriving corrosion knowledge from case histories: The neural network approach." *Materials & Design*, **13(3)**, pp. 149–153.

Tang, B. (1993). "Orthogonal Array-Based Latin Hypercubes." *Journal of the American Statistical Association*, **88**, pp. 1392–1397.

Tejada-Guibert, J. A., S. A. Johnson, and J. R. Stedinger (1993). "Comparison of Two Approaches for Implementing Multireservoir Operating Policies Derived Using Stochastic Dynamic Programming." *Water Resources Research*, **29(12)**, pp. 3969–3980.

Tezuka, S. (1995). *Unifoam Randon Unmbers: Theory and Practice.* Dordrecht, The Netherlands: Kluwer Academic Publishers.

Tsai, J. C. C., V. C. P. Chen, J. Chen, and M. B. Beck (2004). "Stochastic Dynamic Programming Formulation for a Wastewater Treatment Decision-Making Framework." *Annals of Operations Research*, Special Issue on Applied Optimization Under Uncertainty, **132**, pp. 207–221.

Turgeon, A. (1981). "A decomposition method for the long-term scheduling of reservoirs in series." *Water Resources Research*, **17**, pp. 1565–1570.

Wang, Y. and K.-T. Fang (1994). "Number theoretic Methods in Applied Statistics." *Chinese Annals of Mathematics Series.* B **11**, pp. 41–55.

Werbos, P. K. (1994). *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting.* New York: Wiley.

Weyl, H. (1916). "Uber die Gleichverteilung von Zahlen mod." *Mathematische Annalen* **77** pp. 313–352.

Weyl, H. (1956). *Selecta Hermann Weyl.* Basel, Switzerland: Birkhäuser.

White, D. J. (1985). "Real Applications of Markov Decision Processes." *Interfaces*, **15(6)**, pp. 73–83.

White, D. J. (1988). "Further Real Applications of Markov Decision Processes." *Interfaces*, **18(5)**, pp. 55–61.

White, H. (1989). "Learning in Neural Networks: a Statistical Perspective." *Neural Computation*, **1**, pp. 425–464.

Yakowitz, S. (1982). "Dynamic programming applications in water resources." *Water Resources Research*, **18**, pp. 673–696.

BIOGRAPHICAL STATEMENT

Aihong Wen was born in Jiangxi, China in 1971. She received her B.S. degree in Mathematics from Gannan Normal University, China, in 1993, her M.S. degree in Operations Research and Control Theory from Shanghai University, China in 1996, and Ph.D. in Industrial Engineering from University of Texas in Arlington, USA, in 2005, respectively. From 1996 to 2001, she was with the Economic Commission of Shanghai Municipal Government, Shanghai, China working as a data analyst and operations research analyst. Her current research interest is in the areas of Dynamic Programming, Optimal Control and Data Mining.