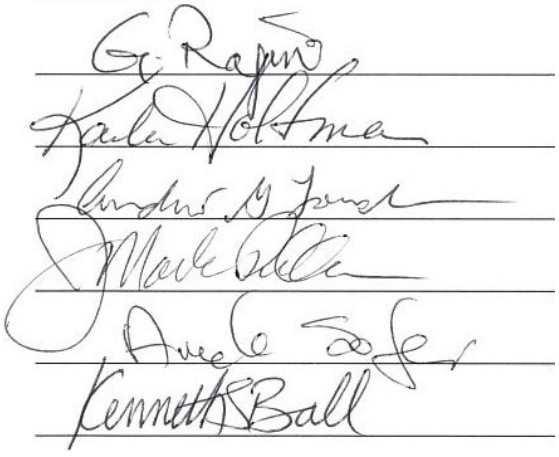
AN APPROXIMATE DYNAMIC PROGRAMMING APPROACH TO FINANCIAL EXECUTION FOR WEAPON SYSTEM PROGRAMS

by

Erich D. Morman

A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Systems Engineering and Operations Research

Committee:



Dr. Rajesh Ganesan, Dissertation Director

Dr. Karla Hoffman, Committee Member

Dr. Andrew Loerch, Committee Member

Dr. Mark Pullen, Committee Member

Dr. Ariela Sofer, Department Chair

Dr. Kenneth S. Ball, Dean
Volgenau School of Engineering

Date: 5 April 2013

Spring Semester 2013
George Mason University
Fairfax, VA

An Approximate Dynamic Programming Approach to Financial Execution for Weapon System Programs

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

by

Erich D. Morman
Master of Science
George Mason University, 2002
Bachelor of Arts
University of Dayton, 1996

Director: Rajesh Ganesan, Professor
Department of Systems Engineering and Operations Research

Spring Semester 2013
George Mason University
Fairfax, VA

DEDICATION

For my mom, dad, and sister.

ACKNOWLEDGEMENTS

I owe a tremendous amount of thanks and gratitude to all those who have assisted me over the many years it has taken to complete this research. The ability to obtain and complete a doctorate education was only possible due to the continual efforts, encouragement, and wisdom provided by the faculty and staff at George Mason University, my fellow colleagues at the Missile Defense Agency (MDA) as well as my family and friends.

I'd like to especially acknowledge Dr. Rajesh Ganesan for serving as my advisor. His accessibility and willingness to spend countless hours discussing mathematical theories, modeling strategies, and interpreting results were critical to maintaining momentum on this project. Additionally, his contagious enthusiasm for the field of Approximate Dynamic Programming (ADP) was instrumental in keeping me motivated and challenged. It was my pleasure to have worked with you on this endeavor.

I am also thankful to have had Dr. Karla Hoffman, Dr. Andrew Loerch, and Dr. Mark Pullen serve on my committee. I am grateful for Dr. Hoffman's insightful feedback and review of both my proposal and dissertation materials. Her knowledge on financial or resource allocation problems helped me appreciate the finer points of modeling structure and implementation. Dr. Loerch's perspective on weapon system modeling, military analysis, as well as client-customer relationships was extremely beneficial and very helpful. I'd also like to thank Dr. Pullen for his willingness to be on the committee and providing me perhaps the best advice which was that the secret to completing a Ph.D. is not necessarily intelligence but, persistence. Lastly, I'd like to thank Angel Manzo, Josefine Wiecks, Sally Evans, and Lisa Nolder for their professionalism and help in accomplishing all the necessary administrative tasks required to complete this degree.

I'd like to thank my many clients, colleagues, and friends associated with MDA who supported this project. I owe a special thanks to Ms. Jan Bond whose early on support and encouragement was absolutely critical to removing the roadblocks that helped initiate this research. Also, I'd like to acknowledge Mr. Terry Little for his willingness to review and discuss my draft materials. Lastly, I'd like to recognize Mr. Richard Ritter, Jim Watzin, Rob Cramer, Phil Gretzkowski, Scott Hillstead, David Lancaster, Sue Knapp, Alex Snead, Rob Cramer, Chris Szkrybalo, and Tammy Sledge for their support at all levels in helping me obtain the resources, perform the coordination, and navigate the administrative requirements necessary to accomplish this work at MDA.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF ABBREVIATIONS AND SYMBOLS

Thousands of Dollars ..... $K
Millions of Dollars ..... $M
Approximate/Adaptive Dynamic Programming ..... ADP
Artificial Intelligence ..... AI
Business Financial Manager ..... BFM
Continuing Resolution Authority ..... CRA
Department of Defense ..... DoD
Dynamic Programming ..... DP
General Accountability Office ..... GAO
Greatest Common Denominator ..... GCD
Inspector General ..... IG
Fiscal Year ..... FY
Missile Defense Agency ..... MDA
Markov Decision Process/Problem ..... MDP
Mean Square Error ..... MSE
Net-Present Value ..... NPV
Office of the Secretary of Defense ..... OSD
Operations & Maintenance ..... O&M
Program Director ..... PD
Planning, Programming, Budgeting & Execution ..... PPBE
Reinforcement Learning ..... RL
Research, Development, Test, and Evaluation ..... RDT&E
Transition Probability Matrix ..... TPM

ABSTRACT

AN APPROXIMATE DYNAMIC PROGRAMMING APPROACH TO FINANCIAL EXECUTION FOR WEAPON SYSTEM PROGRAMS

Erich D. Morman, Ph.D.

George Mason University, 2013

Dissertation Director: Dr. Rajesh Ganesan

During each twelve month fiscal year (FY) cycle weapon system programs across the Department of Defense (DoD) are expected to execute their allocated budgets in an expedient and timely manner. As the FY progresses, a weapon system's cash flow state at any given moment is primarily measured by the cumulative amounts of their budget that are either committed, obligated, accrued, or expended. Regulatory and oversight initiatives such as midyear financial execution reviews and published monthly execution goals serve as measures that are designed to ensure that there is in fact high utilization of a weapon system's allocated yearly budget. The challenge of finding an ideal monthly commitment cash flow policy that achieves a high level of utilization can be expressed as a sequential decision making problem over time. The mathematical area known as Markovian analysis is dedicated to modeling and finding solution methods that focus on such problems with emphasis on understanding how the system moves from state to state

throughout the decision process. The complexity of the problem examined in this research stems from the size of the multimillion dollar budgets in question and the numerous projects they fund. In turn, weapon system offices must make hundreds of commitment action determinations over any given fiscal year in an environment of uncertainty. This intricate decision system necessitates that decision makers have good mathematical tools that can assist them with determining an optimal commitment policy.

The research described in this thesis uses approximate dynamic programming (ADP) techniques as a solution method to the financial execution commitment problem for DoD weapon system programs. ADP ideas and concepts are extensions of Markovian analysis principles. The modeling effort uses a simulation based optimization method specifically geared towards solving sequential decision making problems. The more traditional dynamic programming (DP) approaches are variants on the implementation of Bellman's recursive optimality equation. Unfortunately, as a result of the "curse of dimensionality" and the "curse of modeling" these classical methods tend to breakdown when applied within the more complex problem structure scenarios. The ADP approach expands upon the original recursive idea embedded in Bellman's optimality equation and addresses the difficulties associated with the "curse of dimensionality" and the "curse of modeling".

As part of this research, two types of ADP models were built around the use of a post decision state (PDS) variable. The application of the models was tested against a collection of theoretical financial execution project scenarios. The initial model leveraged a Q-learning design, while the second model used value function learning. In

each approach, the formulation of an optimal policy was dependent upon three modeling phases. The three phases are referred to as exploration, exploitation (learning), and learnt. The exploration phase of the model relaxes the driving optimality conditions while simulating the execution decision system. The exploitation or learning phase incorporates the optimality conditions within the simulation environment. Lastly, the learnt phase leverages the outputs produced by exploration and exploitation to provide the recommended optimal policy. Additionally, the learnt phase of the models was designed to provide a means for conducting various sensitivity analysis and financial execution drill excursions.

The research resulted in a unique application of ADP as a simulation and problem solving method for generating financial execution commitment policies. The generated ADP polices or commitment plans were compared against an alternative myopic policy approach referred to as a stubby pencil policy. The learnt modeling phase examined and tested the reaction of both the ADP and stubby pencil policies under various expenditure conditions. The analysis showed that the ADP commitment strategy was often either equal or less than that of the myopic stubby pencil strategy. These results suggest that a decision maker following an ADP strategy would either reach full commitment of the budget at a later date or would not reach full commitment of the budget prior to the end of the FY. In the latter case, the remaining uncommitted dollars serve as an indication that improved cash utilization could be obtained by incorporating more work or projects into the budget.

CHAPTER ONE – THE PROBLEM

1.1 Problem Statement

Military acquisition programs within the Department of Defense (DoD) are currently confronting difficult challenges that are caused by a decreasing budget line that is receiving greater scrutiny from various oversight authorities. In turn, program directors (PD) that are responsible for these weapon system programs are under considerable pressure to ensure that their allocated budgets are executed in a timely and judicious manner. As such, PDs need good tools to assist them with executing and implementing good cash flow policies.

There are a number of nuances involved when attempting to define exactly what is considered effective cash flow management. DoD budgets are executed on a fiscal year bases that starts on 1 October and ends on 30 September. In general, weapon system programs are expected to cover the majority of costs that occur in a fiscal year with the funding appropriated for that year. However, given the complexities of weapon system acquisition, the Office of the Secretary of Defense (OSD) publishes different yearly obligation and expenditure benchmarks for various appropriation categories. Table 1 provides the OSD published benchmarks for Operations & Maintenance (O&M), Research, Development, Test & Evaluation (RDT&E), and Procurement funding (Defense Acquisition University (DAU)).

Table 1: OSD Benchmarks for Obligations and Expenditures

Appropriation Category	First Year Available		Cumulative for Second Year		Cumulative for Third Year	
	Obligation	Expenditure	Obligation	Expenditure	Obligation	Expenditure
O&M	100%	75%	100%	100%	100%	100%
RDT&E	90%	55%	100%	90%	100%	100%
Procurement	80%	N/A	90%	N/A	100%	N/A
Initial Spares	92%	N/A	96%	N/A	100%	N/A
Advance Proc	100%	N/A	100%	N/A	100%	N/A

The different goal benchmarks for the various funding appropriations do alleviate some of the pressure on PDs to fully expend their budgets within the first year of availability. However, unless spending is driven by a contractual structure that necessitates delayed multiyear payments, there is little incentive to deliberately not utilize or expend appropriated funding prior to year's end. The opportunity cost of doing so is the possibility of accomplishing more work in the current year. An additional nuance is that the use of end year spending goals as a performance metric may incentivize excessive and unnecessary late in the year spending. As such, effective cash management as defined for this research attempts to utilize as much of the fiscal year budget as possible within the year appropriated without engaging in year-end frivolous spending.

Counterintuitive organizational incentives and institutionalized factors often contribute to inefficient cash flow management. Several reasons that may cause wasteful accelerated spending include the threat of budget cuts due to low expenditures, the practice of setting a future year's funding level on the current end of year expenditure

position, and the pressure to meet established spending metrics. In contrast some of the pressures that lead to under executing are to avoid the danger of exhausting a budget too early, holding excessive management reserve to cover unplanned expenses, and the tendency of industry to over promise on the amount of cash that is necessary for the initial start of a project. All these factors contribute to creating an environment that necessitates the need for mathematical programming tools that can assist with making unbiased dynamic cash flow commitment policy determinations.

1.2 Execution Definitions

There are four critical definitions that help define the status of funds as it moves through the spending process. These four financial execution parameters are referred to as commitments, obligation, accruals, and expenditures. A commitment of dollars will reserve or set aside funding for a given project. While committed, funding cannot be used or aligned to any other project. After commitments, funding is obligated on a contract. At this point, the government has a contractual obligation to pay for work performed by industry. Once the work is performed the contractor accrues expenses that are billed to the government. After the government pays for the services or materials the contractor provided, the funding is said to be expended. The below formal definitions of these four terms are taken from the DoD financial management regulation glossary:

> Commitment - An administrative reservation of funds based on firm procurement requests, unaccepted customer orders, Directives, and equivalent instruments.

Obligation - Amount representing orders place, contracts awarded, services received, and similar transactions during an accounting period that will require payment during the same, or a future, period.

Accrual (Accrued Expenditure) - The term used for the credits entered into the budgetary accounts to recognize liabilities incurred for (1) services performed by employees, contractors, other Government accounting entities, vendors, carriers, grantees, lessors, etc.: (2) goods and other tangible property received; and (3) items such as annuities or insurance claims for which no current service is required.

Expenditures – An actual disbursement of funds in return for goods or services.

These parameters comprise the vernacular used to articulate the progress of cash flow as it may relate to programmed funding for an individual project, a collection of projects, or a total weapon system budget in its entirety.

1.3 State-Space Vector

The four definitions combined with the concept of programmed dollars define a multi-attribute vector of financial execution parameters. The vector whether applied to an individual project, a collection of projects, or an entire weapon system budget serves as a means for quantifying the existing state of a cash flow system. The vector syntax for programmed dollars, commitments, obligations, accruals, and expenditures is captured by [P C O A E]. At different points within the fiscal year, the status of spending is reflected in the values given to each of these attributes within the vector. A point of contention among the stakeholders vested in cash management relates to how future actions and uncertain events will impact the performance and predicted status of this vector. As such, the study and examination of the behavior of the [P C O A E] vector serves as a solid

basis for designing mathematical optimization tools to assist with the problem of efficient cash management.

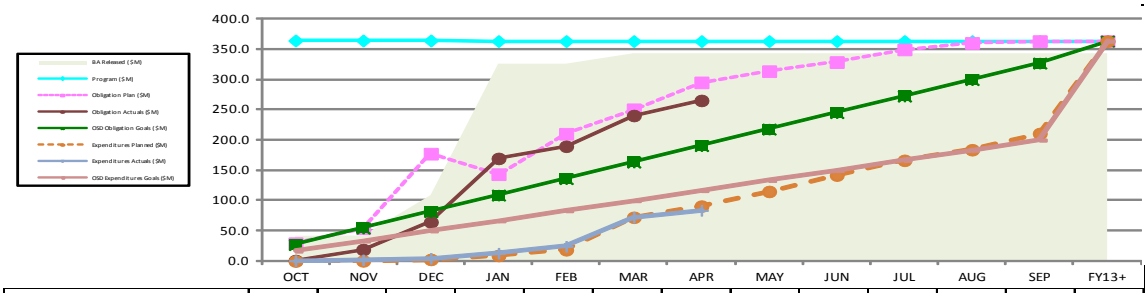
1.4 Problem Environment

The types of programs that require the most assistance are large scale complex weapon system acquisition efforts. These programs have yearly budgets that run in the hundreds of millions of dollars. Their budgets involve many funding projects that may easily number well above sixty. The execution of these projects requires hundreds of financial transactions that occur at various points throughout the fiscal year. Furthermore, the weapon system acquisition environment often contains many elements of uncertainty. The magnitude and complexity of these efforts makes it impossible to effectively manage without the incorporation of sophisticated computational tools.

There are many stakeholders that have a particular position and vested interest in cash flow decisions. However, it is primarily two conflicting points of view that dominate the discussions over the true nature of a program's financial execution position as it progresses throughout the year. The two perspectives are between the program office and the agency comptroller office. Three critical stakeholders for the weapon system program office include the Program Director (PD), Business Financial Manager (BFM), and the execution analysts. The PD oversees all operations necessary for effective development, acquisition, and fielding of the weapon system. The BFM is responsible for all financial and cost related activities associated with the weapon system. The execution analyst performs the day-to-day activities necessary to initiate and monitor the movement of cash as it pertains to projects related to the weapon system. In contrast,

the comptroller is concerned with the overall effectiveness of cash flow as it relates to the agency or military service as a whole. In addition to the program office and comptroller office other stakeholders that have an active interest in DoD cash flow management include Congressional oversight committees, the General Accountability Office (GAO), the Inspector General (IG), Office of the Secretary of Defense (OSD), as well as the weapon system's contracted industry teams. Regardless of the stakeholders, the fundamental positions and arguments of cash flow management are about the predictability of the [P C O A E] vector and the implications that has on the potential unutilized budget that will exist at the end of the year.

The debate is often centered on the program office's spend plan that is put forward at the start of the year. A spend plan usually consists of a month-by-month predictive cash flow status for the weapon system's yearly budget and the individual projects it comprises. An extensive spend plan will include the status of all five financial parameters; programmed amount, commitments, obligations, accruals, and expenditures. However, simplified versions are often used as part of formal end of month or midyear reviews conducted with the comptroller office. Figure 1 shows a consolidated example of a program office's month-end April financial execution status for a Missile Defense Agency (MDA) weapon system program. The charted lines and boxed color codes are indicators that highlight the statistics and trends on how the cash flow has progressed against both the OSD benchmarks as well as the program office's own initial spend plan from the start of the fiscal year (FY) up to the end of April.



Program ($M)	364.1	364.1	364.1	363.1	363.1	363.1	363.1	363.1	363.1	363.1	363.1	363.1	363.1
BA Released ($M)	40.9	41.4	108.2	325.9	325.9	343.1	343.1	343.1	343.1	343.1	343.1	343.1	343.1
Obligation Plan ($M)	29.4	54.2	177.2	143.4	210.2	250.5	294.9	313.4	328.4	349.3	360.1	363.1	363.1
Obligation Actuals ($M)	1.0	19.1	64.5	169.5	189.5	240.2	265.2						
OSD Obligation Goals ($M)	27.2	54.5	81.7	108.9	136.2	163.4	190.6	217.9	245.1	272.3	299.6	326.8	363.1
Obligation Actuals % Status	0.3%	5.3%	17.8%	46.7%	52.2%	66.2%	73.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Expenditures Planned ($M)	0.1	0.3	2.7	9.3	18.6	72.3	90.8	114.9	141.9	166.3	183.8	210.6	363.1
Expenditures Actuals ($M)	0.6	2.0	4.5	13.1	24.3	72.3	83.7						
OSD Expenditures Goals ($M)	16.6	33.3	49.9	66.6	83.2	99.9	116.5	133.1	149.8	166.4	183.1	199.7	363.1
Expenditures Actuals % Status	0.2%	0.6%	1.2%	3.6%	6.7%	19.9%	23.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%

Average Actual Burn Rate	(Current Month -November)/Number of months from November	**16.3**
Highest Month Burn Rate	Highest actual one month of expenditures for the year	48.0
Highest 4 Months Burn Rate	Average of highest four months of actual burn rates	19.8
Portfolio BFM Projected Burn Rate	Average of Portfolio BFM projected burn rates for remainder of FY	24.0
Most Favorable Actual	Projected Unexpended at EOM Oct 2012 based on highest of FY 2012 monthly burn rate	0.0
Average Highest Actuals	Projected Unexpended at EOM Dec 2012 based on average of highest 4 months FY 2012	121.0
Based On Portfolio BFM Projections	Projected Unexpended at EOM Dec 2012 based on Portfolio BFM projections	80.6

Figure 1: Example Obligation and Expenditure End-Month April Status

During execution review sessions, contentious debates and energy is often centered on the amount of unexpended budget that will likely exist on 30-September at the end of the FY and how far into the successive FY this money will last. The issue is whether or not the expenses covered by these carryover dollars represent an acceptable amount of forward funding or an excessive and unnecessary level of forward funding. As is shown on Figure 1, the bottom half of the chart reflects various comptroller projections of what could be considered as excessive budget. These predictions use different burn rate variables that were derived from the recorded actual burn rates accrued between the start of the FY to the current end of April. Program offices tend to counter this evidence with direct insight on the programmatic nuances and nature of how the program is progressing at the time. The responses and arguments put forward are late in the FY

initiatives that have yet to start or initial projects that are going through ramp-up and have yet to reach their maximum burn rate levels. Additional disconnects could also stem from invoices that have yet to post in the accounting system. The core basis of the counter argument is that the presented historical trend is not an accurate depiction of the future activity. However, if the assumptions of the program office predictions are not correct, the realization of this fact may occur too late in the year and the opportunity to have either spent the money on other projects related to the weapon system or within the agency as a whole will be lost.

1.5 Markov Decision Process

The DoD cash flow management problem is best described as a sequential decision making problem over time in an environment of uncertainty. These types of problems stem from a mathematical area known as Markovian analysis. A wide array of applied problems falls into this categorization including resource allocation, inventory control, and network flow. The field of dynamic programming (DP) includes a number of concepts and special features designed to address the specific complexities that arise from the Markovian problem structure. Puterman (1994) offers a good overview of Markov decision analysis and its relationship with stochastic dynamic programming methods.

The Markov decision problem or Markov decision process (MDP) used to model the sequential decision making problem contains a number of specific characteristics. For this problem structure framework, the decision maker attempts to take a series of ideal actions over a finite or infinite planning horizon. The planning horizon may be

discrete points of time or simply delineations between required successive actions. These separate decision points are referred to as stages or epochs. At each stage the decision maker exists in a state. The attributes of the various states are captured by the variables or vector of variables describing the state. The decision maker takes an action that moves them from state to state over successive stages. Each time an action is taken the decision maker receives either an immediate reward or penalty. The MDP captures the fundamental problem of having to balance between the immediate reward or penalty acquired in the current stage with rewards or penalties resulting from future actions taken during the successive stages.

A critical feature of the MDP is the memoryless property. There is a level of uncertainty that exists in states arrived at and visited as a result of the decision makers selected actions. However, the probability of arriving at any given state is dependent only upon the preceding state and not any of the earlier states that were already visited. For a set of finite states S = $\{s_0, s_1, s_2, \ldots, s_n\}$ this notion is expressed as shown in Equation 1.

Equation 1: Memoryless Property

$$Pr\{S_{t+1} = s_{t+1} | S_t = s_t, S_{t-1} = s_{t-1}, \ldots, S_0 = s_0\} = Pr\{S_{t+1} = s_{t+1} | S_t = s_t\}$$

Each time a decision is made in the Markov process that moves the system through consecutive states a cost or reward is incurred. The term $r_t(i,x,j)$ is defined as the reward for making a decision x at time period t that causes the system to move from state i to state j. Furthermore, the probability of reaching state j from state i given decision x is

$P_t(j|i,x)$. As shown in Equation 2, these terms can be combined together to express the expected reward for decision x while in state i given the set of all possible states S.

Equation 2: Expected Reward

$$\bar{r}_t(i,x) = \sum_{j \in S} r_t(i,x,j) P_t(j|i,x)$$

The objective of the MDP is to determine an optimal policy that maps the best decision $x \in X$ to take for each state $s \in S$ in the decision system. Any feasible mapping of actions to states is considered a policy. The best or optimal policy is the one that will either maximize or minimize the total overall long run reward or cost for the system. The expression $P_t(j|i,x)$ is taken from a one-step transition probability matrix (TPM) which dictates the behavior of a particular policy. The element in the i^{th} row and j^{th} column of the TPM, expressed as P_{ij}, is the probability of moving from state i to state j under a particular policy.

The overall expected reward obtained by implementing a policy can be determined through calculating the limiting probabilities of the TPM. This is achieved by solving the following system of equations for all π_j:

Equations 3: Limiting Probabilities

$$\pi_j = \sum_{i=0}^{S} \pi_i P_{ij} \qquad \forall\ j \in S$$
$$\sum_{j=0}^{S} \pi_j = 1$$

These limiting probabilities are then used to determine the total expected reward of the policies under consideration. For a given policy, this total expected reward is

$\sum_{i=0}^{S} \pi_i \bar{r}(i,x)$. Similar total expected rewards are easily calculated for other feasible policies that each have a unique TPM which governs the policy's behavior. An optimal solution is obtained by calculating the total expected rewards among all possible policies and selecting the maximum.

This approach to finding the ideal policy for a MDP is only practical for relatively simple problems. Consider a basic MDP that contains four states and at each state one of three possible decisions is allowed. As such, this means that for this MDP there exists 3^4 = 81 potential policies. In order to find the optimal solution the limiting probabilities for each policy is calculated from 81 different sets of systems of equations. Additionally, it is readily observed that the number of calculations required to solve the problem grows exponentially should either an additional state or decision variable be added.

1.6 Dynamic Programming (DP) Introduction

DP attempts to alleviate the computational burden imposed on a MDP that occurs as a result of the direct enumeration of all policy possibilities. The DP approach is attempting to find the ideal state and decision pairing that is described by theorem 1 (Gosavi 2009), (Bertsekas 1995).

> Theorem 1: *For a discounted reward MDP in which all Markov chains are regular, there exists a vector* $\vec{J}^* \equiv \{J^*(1), J^*(2), \dots, J^*(|\vartheta|)\}$ *such that the following system of linear equations is satisfied:*
>
> $$J^*(i) = max_{a \in A(i)} [\bar{r}(i,a) + \gamma \sum_{j=1}^{|\vartheta|} p(i,a,j) J^*(j)] \quad \text{for all } i \in \vartheta$$

DP examines the solution approach for finding this optimal vector through capitalizing on the inherent recursive structure embedded in the system of linear equations. In the simplest of forms the objective function value for any state i can be expressed by Equation 4.

Equation 4: DP Recursive Objective Function

$$f_t(i) = min_j/max_j\{c_{ij} + f_{t+1}(j)\}$$

Here, the objective function states that the best value of being in state i at time or stage t is obtained by determining the action that provides either the maximum or minimum of the cost c_{ij} incurred by moving from state i to state j plus the value of being in state j at the next unit of time or stage t+1. The objective function structure requires the consideration of all the successive decisions that occur beyond just the imminent timeframe t. The concern is that outside of very straight forward problems the use of recursion will present computational challenges. Fortunately, there are advanced concepts in the field of DP that are specifically designed to address these challenges. In an effort to provide context for the advanced DP concepts, it is worth taking a closer look at a specific solution approach which uses this basic recursive objective function formulation.

The recursive concept is easily demonstrated by examining a simple resource allocation problem (Winston 94). In this example, a company is considering how to allocate $6,000 in $1,000 increments across three potential investment opportunities. The individual net present value (NPV) returns for each investment along with a network flow diagram are provided below:

Equations 5: NPV for Resource Allocation Problem

$r_1(d_1) = 7d_1 + 2 \quad (d_1 > 0)$
$r_2(d_2) = 3d_2 + 7 \quad (d_2 > 0)$
$r_3(d_3) = 4d_3 + 5 \quad (d_3 > 0)$
$r_1(0) = r_2(0) = r_3(0) = 0$

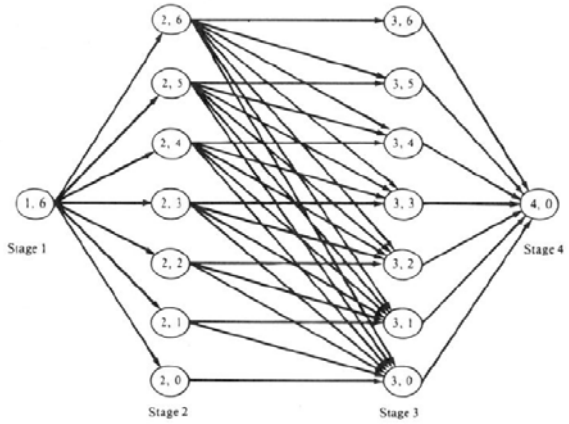


Figure 2: Backward Recursion Network

The nuances of the problem are best explained by examining the stage-space and state-space design. In this sequential decision making problem each stage represents one of the three individual investment choices. The nodes at each stage show the various states representing the balance on the $6,000. The final and fourth stage is added as a sink node that is arrived at once the funding is completely distributed across all three investments. At a given stage, the state arrived at is dependent on the choices made in the prior stages. If the choice in stage one is to allocate x_1 = $2,000, then the state arrived at in stage two is $4,000 or (2, 4). Although the problem is characterized by how the action choices move one forward through the network, the solution method actually evaluates the stage-spaces in reverse through the use of a technique called backward recursion.

Backward recursion systematically calculates the value of each stage-space in reverse order. Starting with investment three, one examines the NPV obtained for all of the investment allocation possibilities. These values along with the corresponding allocation choice are as follows:

Table 2: Investment Three NPV

$f_3(0) = 0$	$x_3(0) = 0$
$f_3(1) = 9$	$x_3(1) = 1$
$f_3(2) = 13$	$x_3(2) = 2$
$f_3(3) = 17$	$x_3(3) = 3$
$f_3(4) = 21$	$x_3(4) = 4$
$f_3(5) = 25$	$x_3(5) = 5$
$f_3(6) = 29$	$x_3(6) = 6$

Once known, the stage three values are then utilized to calculate the stage two values. If d_2 is the amount of dollars available for investment two, then the total NPV obtained by the investment action x_2 is the sum of both the immediate return of investment two $r_2(x_2)$ plus investment three $f_3(d_2-x_2)$. In short, at stage two the solution approach is now looking to maximize the following:

Equation 6: Stage 2 Maximization Formulation

$$f_2(d_2) = \max_{x_2} \{r_2(x_2) + f_3(d_2 - x_2)\}$$

In a similar manner, the backward recursion algorithm continues such that the various stage one values are calculated using the stage two values. At stage one, the net present value obtained from an investment action x_1 is the sum of the initial investment

return $r_1(x_1)$ plus the cumulated returns of investments two and three $f_2(6\text{-}x_1)$. The following table shows the NPV calculations and corresponding actions for both stage two and stage one.

Table 3: Investment One and Two NPV

d_2	x_2	$r_2(x_2)$	$f_3(d_2\text{-}x_2)$	$r_2(x_2) + f_3(d_2\text{-}x_2)$
0	0	0	0	0
1	0	0	9	9
1	1	10	0	10
2	0	0	13	13
2	1	10	9	19
2	2	13	0	13
3	0	0	17	17
3	1	10	13	23
3	2	13	9	22
3	3	16	0	16
4	0	0	21	21
4	1	10	17	27
4	2	13	13	26
4	3	16	9	25
4	4	19	0	19
5	0	0	25	25
5	1	10	21	31
5	2	13	17	30
5	3	16	13	29
5	4	19	9	28
5	5	22	0	22
6	0	0	29	29
6	1	10	25	35
6	2	13	21	34
6	3	16	17	33
6	4	19	13	32
6	5	22	9	31
6	6	25	0	25

NPV from Investments 2,3

d_1	x_1	$r_1(x_1)$	$f_2(6 - x1)$	$r_1(x_1) + f_2(6 - x_1)$
6	0	0	35	35
6	1	9	31	40
6	2	16	27	43
6	3	23	23	46
6	4	30	19	49*
6	5	37	10	47
6	6	44	0	44

NPV from Investments 1-3

The stage one calculations provide the optimal solution which is a total NPV of 49 for all three investments. Although this optimal value was obtained through backward recursion, the actual decision policy is now realized by stepping forward back through the decision problem. At stage one, the optimal value is obtained by taking action $x_1(6)$ = \$4,000. This leaves a balance available for investment two of d_2 = \$6,000 - \$4,000 = \$2,000. In turn, the optimal action for stage two is $x_2(2)$ = \$1,000. This leaves a balance of d_3 = \$1,000 for investment three along with the associated optimal action $x_3(1)$ = \$1,000. In summary, the optimal NPV value of 49 is obtained with a decision policy of [x_1=\$4,000 x_2=\$1,000 x_3=\$1,000] and resulting returns [$r_1(x_1)$=30 $r_2(x_2)$=10 $r_3(x_3)$=9].

This algorithmic approach is only viable for relatively simple well defined problems. In this example, the decision problem contains no stochastic or probabilistic components. Additionally, the state-space size was relatively small making it viable to calculate all possible action and outcome state pairings. Once the decision maker had access to all aggregate NPV outcome possibilities, the problem of determining an optimal policy was rather straight forward. Furthermore, the implementation of the DP approach ensures that the decision maker does not make a myopic allocation. For example, using a pure greedy heuristic an individual may place all $6,000 in investment one because as a stand-alone option this does in fact produce the highest NPV. Unfortunately, when applied to more complex and sophisticated problems backward recursion and other classical DP solution methods quickly breakdown. Solution approaches to more challenging problem structures incorporate advance DP techniques that are described within a simulation based area known as adaptive dynamic programming or approximate dynamic programming (ADP). This thesis examines the development of some of these advance DP tools and theoretically tests them within the context of determining optimal commitment decision policies for financial execution.

1.7 Allocation Parameter

The initial consideration in the ADP design was to determine the discrete funding increment of a project with which to track the execution status. Each project has an assigned [P C O A E] multiattribute vector that expresses the financial execution status of a project at any point in the ADP simulation. In theory, each of these five variables can take on any dollar amount between zero and the maximum allowable budget for the

project in question. However, for projects ranging in value from hundreds of thousands to multimillion dollar amounts there was little added value in tracking the [P C O A E] vector to the nearest single dollar amount. Therefore, restrictions were put on each project such that any of the P, C, O, A, or E values could only take on and be expressed in multiples of the assigned allocation parameter.

Consider a $5.0M project with an assigned $0.250M allocation parameter. This means that any of the five financial execution variables can take on only one of the 21 different multiple values of $0.250M that range between zero and $5.0M. Obviously, this assumes that through the course of the simulated fiscal year the project's programmed amount does not receive a plus-up or reduction. If the project does receive a programmed amount plus-up or reduction, the number of possible allowable values increases or decreases accordingly. Nonetheless, as a basis the 21 different allowable values means that the number of state-space possibilities that exist for the entire [P C O A E] vector is $21^4 = 194{,}481$.

The ADP model was designed with the flexibility to select different allocation parameters based on the size of the project. The various allocation parameter options were $0.100, $0.250M, $0.500M, $1.000M, $2.000M, and $5.000M. The actual allocation assignment to a given project was based on the magnitude of the programmed dollars for that project.

1.8 Why Use Approximate Dynamic Programming (ADP)

ADP has a number of features that make it an appealing solution method for this particular problem vice the available alternative existing mathematical programming

approaches. As pointed out by Das et al. (1999), "Well known algorithms, such as value iteration, policy iteration, and linear programming find optimal solutions (i.e., optimal policies) of MDPs. However, the main drawback of these classical algorithms is that they require, for every decision, computation of the corresponding one step transition probability matrix and the one step transition reward matrix using the distributions of the random variables that govern the stochastic processes underlying the system." The argument is that the alternative methods to ADP are dependent on transition probability matrices (TPM) to articulate the nature of the uncertainty and randomness that exists in the decision system. In many real world problems there is no easily available TPM for a given MDP. The computational complexities in generating a TPM along with the potential burden of storage may cause these alternative approaches to be intractable. The ADP solution approach includes a method for incorporating the stochastic component of the decision system without the need for an explicit TPM.

Additional arguments for using ADP are related to methodology differences. Many alternative methods to solving sequential decision making problems are burdened by the requirement to satisfy the modeling constraints from all time periods of the problem at the same time (Denardo 2003). Since the alternative solution methods need to consider many of the facets of the problem simultaneously, even the formulation of problems that are evaluated over a moderate number of time periods are large-scale. In reference to a transportation application, Powell (2007) states, "formulated as a single, large linear (or integer) program (over 50 time periods), we obtain mathematical programs with hundreds of thousands of rows and upwards of millions of columns". The

example is taken from a discussion that suggests the use of ADP as a decomposition technique for these types of large-scale math problems. The advantage of ADP under these large-scale problem conditions is that at any given moment in the solution process the methodology only evaluates data from two consecutive time periods vice having to evaluate data from all time periods simultaneously.

1.9 Contributions and Structure of Dissertation

The contributions of this thesis are threefold. The first research objective is to build an ADP model that can examine and mimic the sequential decision making problems associated with financial execution as it relates to weapon system acquisition within the MDA and possibly within the DoD as a whole. Second, the ADP will be used to generate and recommend commitment strategies. The commitment strategies will be unbiased to the organizational pressures to either over execute or under execute a budget. Ideally, they represent a commitment policy that can move the decision maker from one good state-space to the next good state-space as a project or group of projects moves through the execution process. Lastly, the commitment strategies generated by the ADP will be compared against an alternative commitment strategy referred to as stubby pencil. As the name suggests, the stubby pencil strategy attempts to mimic real world behaviors that allocate dollars based on standard linear projections. These approaches tend to exhibit a myopic decision strategy that provides positive short-run benefits but, do not take into account the downstream impacts of decisions on the system as a whole. A specific theoretical portfolio of projects is evaluated as a basis of comparison between the

ADP and stubby pencil approaches. Various sensitivity analysis drills are performed to further evaluate the responses of these two perspectives.

The structure of this thesis is as follows. Chapter one defines the problem and provides an overview of financial execution as well as basic Markovian and DP concepts. Chapter two expands on some of the critical mathematical tools and developments in the field of ADP. Additionally, this chapter provides an overview of the relevant authors and their associated publications. Chapter three provides a walk-through of the development hurdles and functionality of the financial execution ADP model. In chapter four, the learnt phase of the model is discussed and the results of sensitivity analysis drills are shown. Chapter five provides some summary remarks, observations, and opportunities for further research.

CHAPTER TWO – ADP CONCEPTS AND LITERATURE OVERVIEW

2.1 Dynamic Programming Background

During the 1950s and early 1960s, DP began developing prominence as a mathematical tool for modeling and solving MDP. There are a number of publications during this timeframe that significantly helped define and advance the field of DP. Some of these seminal works include Bellman (1954, 1957), Howard (1960), and Bellman & Dreyfus (1962). At that time, these publications helped articulate the core principles of DP and its utility as a tool for solving Markov decision and other stochastic control problems. The DP approach is designed to specifically address the inherent structure of sequential decision making problems which encompasses a vast diversity of problem classifications. A few of these areas include inventory management, resource allocation, job shop scheduling, shortest path problems, technology switching, and maintenance/ repair scheduling. Over the last few decades, the DP field has evolved into a rich array of techniques designed to respond to the inherent difficulties with using traditional DP solution approaches and the complexities of real world problem structures. The following provides a short overview of the different lexicon used to refer to advance dynamic programming concepts as well as some of the published authors in the field.

Powell (2009) points out that different academic communities have a fundamental need to examine the solution capabilities and driving theories behind DP. A problem that developed was that as these sub-communities worked to further advance solution

techniques for MDPs each community simultaneously built their own vernacular and notional symbols to essentially express the same basic ideas. Powell (2007, 2009) references that these different communities include control theory (engineering/ economics), artificial intelligence (AI) (computer science), and operations research. As articulated below, there are significant drawbacks to using traditional DP approaches for solving a MDP. In response to these shortcomings, an iterative solution approach within the field of operations research was adopted know as adaptive/approximate dynamic programming (ADP). In control theory, Bertsekas and Tsitsiklis (1996) refer to this similar approach as neuro-dynamic programming. Furthermore, Sutton and Barto (1998) refer to this as reinforcement learning (RL) within the context of AI. Both Powell (2010) and Tsitsiklis (2010) offer perspectives and elaborate on the relationship that exists between the ADP and AI communities. Additional context behind the evolution of ADP is mentioned in Gosavi (2009) which states that "the modern science of RL has emerged from a synthesis of notions from four different fields: classical DP, AI (temporal differences), stochastic approximation (simulation), and function approximation (regression, Bellman error, and neural networks)."

In regards to this thesis, two algorithmic approaches in the form of Q-learning and value function learning are provided which served as the design framework for solving the financial execution commitment policy problem and providing good cash flow policies to the decision maker. The remainder of this chapter is dedicated to the rationality and pedagogical development of the ADP approach.

2.2 Problems with Dynamic Programming

As expressed in Ivengar (2005), "Dynamic programming (DP) is the mathematical framework that allows the decision maker to efficiently compute a good overall strategy by succinctly encoding the evolving information state." The DP approach is concerned with finding optimal policies to sequential decision making problems that are often expressed as a MDP. The previous chapter outlined two classical DP solution approaches related to finding optimal policies within these types of decision problem frameworks. Unfortunately, these two solution methods and other classical DP approaches often breakdown when applied beyond relatively simple problems.

The first solution approach attempted to find an optimal solution by computing the expected value of the objective function for all viable policy alternatives. This is considered a strict enumeration approach and involved finding limiting probabilities through solving a system of linear equations as expressed by each policy's TPM. The difficulty with this approach is that the calculations require a well defined TPM that reveals how one state moves to the next in the decision process. However, in practicality most real world problems do not have well defined probabilities that indicate how the decision process will transition from state to state. Instead, in a complex problem environment the movement from one state to the next is dictated by a combination of the decision maker's action and realized exogenous or random information that impacts the decision sequence at each stage. The dependency of traditional DP solution approaches to require a TPM is referred to in the literature as "the curse of modeling".

The second approach utilized a backward recursion method to calculate a system value for each state-space possibility in the sequential decision making system.

However, this approach becomes quickly problematic in that the computational burden grows exponentially for each variable element added to the problem. When implementing the backward recursion solution method, the expansion of the decision problem by adding just a single state-space or action possibility will easily result in the need for a sizable amount of additional calculation requirements. This exponential growth in computational demands for even relatively small problems is referred to as "the curse of dimensionality".

The curse of dimensionality and the curse of modeling are of critical importance to the limitations of traditional DP approaches and thus the need to develop ADP techniques that both issues are worth further examination.

2.3 "Curse of Modeling"

One of the major drawbacks of traditional approaches to solving a MDP is that they all assume access to the TPM. However, one positive aspect of having access to the TPM is that the value function expression used in many of these solution approaches can be calculated explicitly and as such obtain exact optimum solutions. Yet, most real world problems do not have a well defined TPM that can capture the probabilistic nuances of how a state action pairing will move the decision maker into the next state. Solution methods such as ADP that attempt to solve and mimic problems that do not have a readily available TPM structure are referred to as model-free. In these cases, the information necessary to move a decision maker from state to state will need to be generated artificially. The standard approach for doing this is to use a Monte Carlo simulation technique. As with the real world problem examined in this thesis the

sequential decision making process of determining what amount of money to commit at the start of each month in support of weapon system acquisition does not contain an easily definable TPM matrix.

2.4 "Curse of Dimensionality"

A second concern with the traditional methods of solving a sequential decision making problem is that the computational requirement grows exponentially as the dimensionality and state-space size of the problem increases. As a result of the curse of dimensionality the strict enumeration and backward recursion methods in chapter one would be intractable for any large scale problems. Also, other classical methods such as value iteration which requires performing a synchronous value update for all states in the decision system at each iteration in the solution process quickly becomes unmanageable as problem size increases.

Powell (2007) references the state-space, action space, and outcome space as the three main curses of dimensionality. As articulated, a decision problem with I state-space dimensions each of which can take on any of L possible values may have as many as L^I true state-space possibilities. This same exponential representation holds for an outcome space of size M^J that has M possibilities over J dimensions and an action space of size N^K that has N possibilities over K dimensions. Expressed in this fashion it is readily visible how any traditional DP approach which requires accounting of all possible state-space, action space, and outcome space possibilities is quickly overwhelmed by the curse of dimensionality.

An important concept to the ADP design is to delineate between a problem having a large state-space and a problem having large dimensionality. For the financial execution problem the size of the state-space is determined by the allocation factor assignment against a given project. A $5.0M project with an allocation parameter of $1.0M has in theory six possible programmed value states ranging from $0.0M through $5.0M. By decreasing the allocation parameter by one-half to $0.5M, this increases the number of possible programmed value state-spaces from six to eleven. Regardless of the allocation parameter the dimensionality of the problem does not change. The defining problem vector [P C O A E] has five dimensions and still only has five dimensions even if there is a change to the allocation parameter. The dimensionality of the problem is based on the number of attributes assigned to define a state-space. Nonetheless, the dimensionality of a problem and the state-space number of a problem are both factors that contribute directly to the curse of dimensionality.

Later in this thesis, the dimensionality of the problem vector is reduced from five to two. There are two reasons for this. The first was as a natural progression of the problem definition. The second was to reduce the state-space size as a means to mitigate the impacts from the curse of dimensionality and improve the algorithm performance time.

2.5 The Need for Approximate Dynamic Programming (ADP)

ADP is a direct response to the inherent problem nuances brought on by both the Curse of Modeling and the Curse of Dimensionality. Figure 3 shows the natural progression and rationality for examining sequential decision making solution methods

beyond what is provided by classical DP. The following sections of Chapter 2 are dedicated to the mathematical syntax and theories of the ADP concept. The equations and algorithms are expressed in terms of a finite horizon minimization problem to be consistent with the described research problem for this thesis. However, all the formulations are easily convertible to either a maximization or infinite horizon expression.

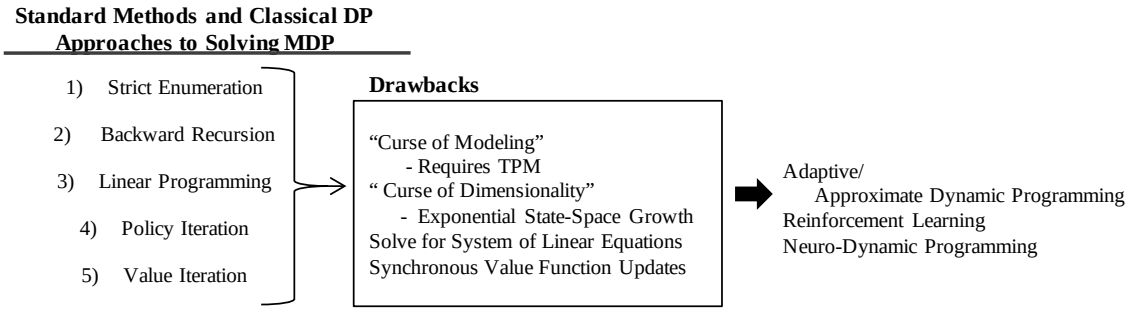


Figure 3: The Need for ADP

2.6 Transition Function

The ADP approach makes use of the idea of a transition function to model the behavior of how a decision making process moves from one state to the next. In light of no longer having a TPM, surrogate symbolism is instead used to describe the rules of how a decision system transitions between states. The expression $S^M(\cdot)$ taken from Powell (2007) is recognized as a generic modeling syntax in which the state-space is a model of the variables contained within the parenthetical $(\cdot)$. As described in the resource allocation problem from chapter one, the recursive expression $f_t(i) = \min_j\{c_{ij}+f_{t+1}(j)\}$

showed how the value of a given state i is dependent upon a cost or reward for an action plus the value of the state j arrived at by that action. In terms of the transition function notation, this same flow of events is expressed as $S_{t+1} = S^M(S_t,x_t)$. Here, the future state S_{t+1} is dependent upon the current state S_t and the action x_t taken at that time. However, an additional variable that impacts the future state-space and what was before expressed by the values contained in the TPM is the randomness that occurs between successive states. In order to capture the embedded uncertainty within the system, the variable W_{t+1} is added to the transition function which expands this modeling expression to what is shown in Equation 7. The variable W_{t+1} represent the randomness or exogenous information that occurs in the system at the start of every time period t greater than one.

Equation 7: Transition Function

$$S_{t+1} = S^M(S_t, x_t, W_{t+1})$$

Another concept that is often used to model the evolving information state of the system is the post decision state (PDS) variable which is expressed as S_t^x. The PDS captures the state of the system just after an action x_t is determined but, before randomness or exogenous information is introduced into the system. Through the use of the PDS variable the transition function is now broken into the two different parts captured by Equation 8 and Equation 9 below. The two equations capture the idea that the system decision process is moving successively from a pre-decision state into a PDS and then into a new pre-decision state. The first expression, Equation 8, takes the

decision system from the pre-decision state and into the PDS. The second expression, Equation 9, moves the system form the PDS into the next successive pre-decision state.

Equation 8: PDS Transition Function

$$S_t^x = S^{M,x}(S_t, x_t)$$

Equation 9: Pre-Decision State Transition Function

$$S_{t+1} = S^{M,W}(S_t^x, W_{t+1})$$

Assuming the possible actions x_t are all well defined variables, the selection of an action and the subsequent movement from a pre-decision state S_t to a PDS S_t^x as expressed in Equation 8 is a deterministic process. However, the expression in Equation 9 is considered a stochastic process since it represents when the system moves from the PDS S_t^x into the next pre-decision state S_{t+1} and randomness in the form of W_{t+1} is introduced into the system. Utilizing transition function notation, the sequential decision making system can be considered as a successive series of back-to-back deterministic and stochastic events. Figure 4 below captures this idea.

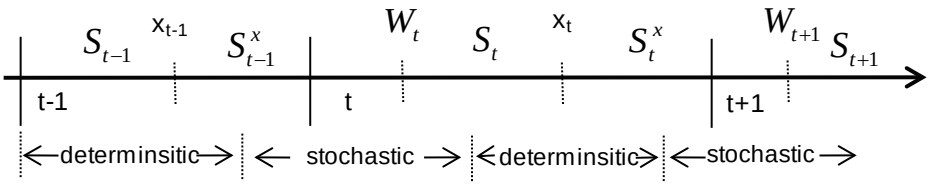


Figure 4: Transition Function Timeline

2.7 Bellman's Equation

The transition function syntax provides a compact method for expressing the objective of the sequential decision making problem. The goal is still to find the best policy that trades-off both immediate and long term costs to produce the minimum overall expense across the problem horizon. This idea is captured by the objective function expression in the following Equation 10. The variable $C(S_t, x_t)$ is the cost incurred for taking action x_t while in state S_t. The γ^t term is a discount factor for each time period t.

Equation 10: DP Objective Function Formulation

$$Min_{\pi \epsilon \Pi} \sum_{t=0}^{T} \gamma^t C(S_t, x_t)$$

The objective as expressed by Equation 10 is to find the policy $\pi \in \Pi$, such that the cost for taking all actions x_t across time period T is minimal. The nature of the problem can be further reduced to evaluate only the necessary requirements to select the optimal action x_t at any state in the system vice all states simultaneously. At any given state t in the sequential decision making process the optimal action for that state $x_t^*(S_t)$ will satisfy Equation 11. In this expression, the optimal action to take is dependent upon an initial immediate or myopic cost $C_t(S_t,x_t)$ plus the value of the next state in the system $V_{t+1}(S_{t+1})$. As it relates back to the transition function, the initial costs $C_t(S_t,x_t)$ represents the deterministic event in the decision process and the future value of the next state $V_{t+1}(S_{t+1})$ captures the stochastic component of the process. The new challenge that is presented by this formulation is to determine the value of the $V_{t+1}(S_{t+1})$ expression.

Equation 11: Optimal Action (Powell 2007)

$$x_t^*(S_t) = \arg min_{x_t \in X_t} (C_t(S_t, x_t) + \gamma V_{t+1}(S_{t+1}))$$

In most problems there is no readily available value for the $V_{t+1}(S_{t+1})$ term or a simple approach for calculating this value for a given time period t. The backward recursion algorithm presented in Chapter 1 showed a method for calculating state-space values $V_t(S_t)$ for a purely deterministic problem. However, determining this value becomes far more problematic once probabilistic or stochastic variables are introduced into the decision logic. Attempting to find this value is an exercise in solving for Equation 12 which is referred to as the standard form of Bellman's equation.

Equation 12: Standard Form of Bellman's Equation

$$V_t(S_t) = min_{x_t \in X_t}(C_t(S_t, x_t) + \gamma \sum_{s' \in S} P(S_{t+1} = s' | S_t, x_t) V_{t+1}(s')$$

Bellman and Dreyfus (1962) put forward that there is an optimal policy that satisfies the standard form of Bellman's equation. Furthermore, if one is able to model the state transitions with a well defined TPM, the value of each state-space $V_t(S_t)$ can be calculated explicitly. However, as a result of the curse of modeling most problems do not have a TPM. As such, Bellman's equation cannot be solved explicitly and the value function at each state in the decision space will need to be estimated. This idea of now having to find an estimated or expectation as a means to calculating the value function is more eloquently recognized by the expectation form of Bellman's equation shown below.

Equation 13: Expectation Form of Bellman's Equation

$$V_t(S_t) = min_{x_t \in X_t}(C_t(S_t, x_t) + \gamma E\{V_{t+1}(S_{t+1})|S_t\})$$

Equation 13 shows how the current state-space value is directly dependent upon the estimated expectation of the next state-space in the system. It is the methodologies and science behind ADP that attempts to tackle the problem of determining an intelligent approach to estimating this expectation.

2.8 Sampling the Value Function

The ADP algorithm works by collecting observed or sampled values $\hat{v}$ of the value function $V_t(S_t)$ through simulating the decision process as it steps forward in time. For each additional simulation run n, the ADP algorithm collects more and more $\hat{v}$ observations for a given state-space. Extrapolating from the expectation form of Bellman's equation, one could collect a sample value through Equations 14 from Powell (2007). Both equations are essentially equivalent and are expressing the same idea. The only difference between the two is the use of the transition function notation for the latter expression.

Equations 14: Sample Realization of Value Function (Powell 2007)

$$\hat{v}_t^n = min_{x_t \in X_t^n}[C_t(S_t^n, x_t) + \gamma \sum_{\hat{\omega} \in \hat{\Omega}_{t+1}} p_{t+1}(\hat{\omega})\, \bar{V}_{t+1}^{n-1}(S_{t+1})$$

$$\hat{v}_t^n = min_{x_t \in X_t^n}[C_t(S_t^n, x_t) + \gamma \sum_{\hat{\omega} \in \hat{\Omega}_{t+1}} p_{t+1}(\hat{\omega})\, \bar{V}_{t+1}^{n-1}(S^M(S_t^n, x_t, W_{t+1}(\hat{\omega})))]$$

During simulation n, a sample realization $\hat{v}_t^n$ at time period t is equivalent to the minimal sum of the immediate cost plus the discounted value of the pre-decision state at

time period t+1. The expression $\bar{V}_{t+1}^{n-1}(S_{t+1})$ is the recognized or estimated value of this pre-decision state that currently exists after n-1 simulation runs. Using this formulation, the approach is to randomly generate a set of outcomes $\widehat{\Omega}_{t+1}$ for the exogenous variable W_{t+1} and assign an associated probability for each outcome $\widehat{\omega}$. The summations are then across the probabilities of each $\widehat{\omega} \epsilon \widehat{\Omega}_{t+1}$.

2.9 Value Function Update

As the simulation progresses, each value $\hat{v}_t^n$ that is collected contributes to finalizing an expectation of the PDS variable $\overline{V}(S^x)$. Various stochastic approximation techniques can be used to smooth a collection of sampled values to calculate an expectation for a population. A common approach as suggested by Robbins & Monro (1951) and interpreted by Gosavi (2003) is to use the expression in Equation 15.

Equation 15: Smoothing Algorithm

$$\bar{V}_t^n(S_t^n) = (1 - \propto_{n-1})\bar{V}_t^{n-1}(S_t^n) + \propto_{n-1} (\hat{v}_t^n) \quad \text{where } \propto_{n-1} = \frac{1}{n}$$

Within the literature, the $\propto_{n-1}$term is referred to as either the step-size, alpha-decay parameter, or learning rate. There is extensive discussion in the ADP literature as to what step-size to use. Although, the ultimate step-size choice is often unique to the decision problem at hand and the particular ADP approach utilized to find a solution.

2.10 Q-Learning on the Pre-Decision State

The initial ADP approach examined in this thesis was a Q-learning algorithm. Instead of generating estimated values associated with each individual pre-decision state, the Q-learning algorithm generates values in the form of Q-factors for each augmented

state-space and action pairing. Equation 16 and Equation 17 from Gosavi (2003) describe the basic structure of the Q-learning approach. They present a method for calculating the estimate of a Q-factor around the pre-decision state variable and then using the Robbins-Monro methodology to update the expectation. Using this formulation, the expression Q(i,a) represents the Q-factor value for the augmented state i and action a pairing. As is consistent with Bellman's formulation the Q-factor is a combination of an immediate cost C(i,a,j) incurred from taking action a which moves the decision system from state i to state j plus the minimum viable Q-factor Q(j,b) value from the set of all actions $b \epsilon A(j)$.

Equation 16: Estimations of Q-Factors

$$\begin{aligned} Q(i,a) &= \sum_{j=1}^{|S|} p(i,a,j)\left[c(i,a,j) + \lambda min_{b\epsilon A(j)} Q(j,b)\right] \\ &= \mathrm{E}[\mathrm{c(i,a,j)} + \lambda min_{b\epsilon A(j)} Q(j,b)] \\ &= \mathrm{E[SAMPLE]} \end{aligned}$$

Equation 17: Robbins-Monro Q-Learning

$$Q^{n+1}(i,a) = (1-\alpha^{n+1})Q^n(i,a) + \alpha^{n+1}[r(i,a,j) + \lambda min_{b\epsilon A(j)} Q^n(j,b)]$$

The presumption of Gosavi (2003) is that a simulation process is used to generate the sample random variables of Equation 16. This explains the expectation that encompasses the entire right-hand side of the expression. The Robbins-Monro update then provides the latest evaluation of the Q-factor. For a step size term α^{n+1} that is set to $1/(n+1)$ the Q-factors become the averages of the observed random variables.

The following algorithm taken from Powell (2007) provides the method for implementing Q-learning around the pre-decision state variable.

Algorithm 1 (Pre-Decision State Q-Learning).

Step 0: Initialization

Step 0a: Initialize the approximation for the value function $\bar{Q}_t^0(S_t, x_t)$

for all $s\epsilon S, x\epsilon X, t\epsilon T$

Step 0b: Initialize S_0^1

Step 0c: Set n=1

Step 1: Choose a sample path ω^n

Step 2: For t = 1, 2, …., T,

Step 2a: Find the decision using the current Q-factors

$$x_t^n = \arg min_{x_t \in X_t^n} \bar{Q}_t^{n-1}(S_t^n, x_t)$$

Step 2b: Compute

$$\hat{q}_{t+1}^n = (C_t(S_t^n, x_t) + \gamma \bar{V}_{t+1}^{n-1}(S^M(S_t^n, x_t^n, W_{t+1}(\omega^n)))$$

Step2c: Update $\bar{Q}_t^{n-1}$ *and* $\bar{V}_t^{n-1}$

$$\bar{Q}_t^n(S_t^n, x_t^n) = (1 - \propto_{n-1})\bar{Q}_t^{n-1}(S_t^n, x_t^n) + \propto_{n-1} (\hat{q}_{t+1}^n)$$

$$\bar{V}_t^n(S_t^n) = min_{x_t} \bar{Q}_t^n(S_t^n, x_t^n)$$

Step 2c: *Find the next pre-decision state*

$$S_{t+1}^n = S^M(S_t^n, x_t^n, W_{t+1}(\omega^n))$$

Step 3: n = n+1. If n < N, go to Step 1.

Step 4: Return the Q-factors $(\bar{Q}_t^n)_{t=1}^T$

Credit is given to Watkins (1989) as the first recognized publication of the Q-learning algorithm. A formal narrative on the proof of convergence for the Q-learning design is given by Watkins & Dayan (1992). The proof of convergence assumes that actions are repeatedly sampled in all states where the action-values are discrete. Tsitsiklis (1994) provides an alternative perspective on the proof of convergence in Q-learning that parallels asynchronous stochastic approximation methods. In regards to Q-learning, Gosavi (2005) emphasizes that "an attractive feature of the algorithm is its stability which is partly due to the fact that the iterates remain bounded." Furthermore, Gosavi (2005) provides another alternative Q-learning proof of convergence from Watkins & Dayan (1992) using mathematical induction.

2.11 Arguments for Using the Post Decision State (PDS)

Powell (2007) argues that a better approach to implementing the algorithmic design to Bellman's equation is to use the post decision state (PDS) vice pre-decision state. Although the approach to finding Q-factors or value function estimates around the pre-decision state is fundamentally consistent with Bellman's optimality equation, the actual implementation of collecting observed $\hat{q}_t^n$ or $\hat{v}_t^n$ in this manner may be problematic. The issue is that at time t, the state S_{t+1} is a random variable. The simulation process that generates $W_{t+1}(\omega^n)$ is stochastic by nature. The selection of ω^n directly impacts which S_{t+1} is in fact the next state. Furthermore, as expressed in Equations 14 the current estimated value of this next state factors directly on the calculations for the observed sample $\hat{v}_t^n$. However, one does not know which

$\bar{V}_{t+1}^{n-1}(S_{t+1})$ value from all possible future S_{t+1} states that might occur will serve as the best estimate for the expectation $E\{V_{t+1}(S_{t+1})|S_t\}$ expressed in Bellman's formulation.

Given the stochastic complexities involved with estimating this embedded expectation, Powell (2007) argues the case for building the value update process around the PDS. His first step is to recognize the critical relationship that exists between the PDS and pre-decision states which are shown in Equations 18 below. The first and third equations express an intuitive relationship between the expectation of different state-spaces and their equivalency to post decision states (PDS) between separate time periods. The second equation expresses the relationship that exists between the pre-decision state and PDS within the same time period. It should be highlighted that the relationships captured in equations 1) and 3) are stochastic and that the expression in 2) is purely deterministic.

Equations 18: Pre-Decision State and PDS Values (Powell 2007)

1) $V_{t-1}^{x}(S_{t-1}^{x}) = E\{V_t(S_t)|S_{t-1}^{x}\}$
2) $V_t(S_t) = max_{x_t \epsilon X_t}\left(C_t(S_t, x_t) + \gamma V_t^{x}(S_t^{x})\right)$
3) $V_t^{x}(S_t^{x}) = E\{V_{t+1}(S_{t+1})|S_t^{x}\}$

The significance of these equations is recognizing the correlating relationship between using $\hat{v}_t^n$ as a sampling value for $V_t^n(S_t^n)$ and as a measure for the value of the earlier time period's PDS value $V_{t-1}^n(S_t^{x,n})$. As expressed in 1) from Equations 18 the expected value of the current pre-decision state value $V_t(S_t)$ is equivalent to the PDS of the earlier time period $V_{t-1}(S_t^x)$. As stated by Powell (2007), "while $\hat{v}_t^n$ is a sample of the value of being in state S_t^n, it is also a sample of the value of the decision that put us in

state $S_{t-1}^{x,n}$." Using this philosophy the sampling formulations for $\hat{v}_t^n$ presented in Equations 14 can be adjusted to what is reflected in Equation 19. Again, both equations are essentially equivalent with the transition function notion being used in the later equation.

Equation 19: Sample Realization of the Value Function using the PDS Variable

$$\hat{v}_t^n = min_{x_t \in X_t^n}[C_t(S_t^n, x_t) + \gamma \bar{V}_t^{n-1}(S_t^{x,n})]$$
$$\hat{v}_t^n = min_{x_t \in X_t^n}[C_t(S_t^n, x_t) + \gamma \bar{V}_t^{n-1}(S^{M,x}(S_t^n, x_t))]$$

This approach provides a sampling structure for $\hat{v}_t^n$ around the PDS variable. The realized sample values $\hat{v}_t^n$ are now a sum of the initial cost function $C_t(S_t, x_t)$ plus the discounted value of the estimated PDS position after n-1 iterations. The advantage of this formulation is that the stochastic component has been removed from the equation. The calculations involved in determining $\hat{v}_t^n$ are purely deterministic. However, the outstanding issue is the mechanics involved for actually determining the best expectation or estimated value of the PDS position $\bar{V}_t^{n-1}(S_t^{x,n})$.

In response to this dilemma, Powell (2007) modifies the implementation of the Robbins-Monro algorithm to reflect Equation 20. Unlike Equation 15, Equation 20 centers the smoothing process or value update function on the PDS variable. Using this formulation, Powell (2007) is taking advantage of the fact that at time t, the previous PDS S_{t-1}^x is already known and not a random variable. In this manner, emphasizing the PDS variable has reduced the stochastic complexities of the earlier pre-decision state approach. The difference between using the pre-decision state and PDS can best be

characterized by either using an update process that is looking forward or an update process that is looking backwards.

Equation 20: Smoothing Algorithm on PDS Variable

$$\bar{V}_{t-1}^{n}(S_{t-1}^{x,n}) = (1 - \propto_{n-1})\bar{V}_{t}^{n-1}(S_{t}^{x,n}) + \propto_{n-1} (\hat{v}_{t}^{n})$$

2.12 Q-Learning on the Post Decision State (PDS)

The initial model built to examine the financial commitment problem was a Q-learning algorithm that used a PDS value update process. A simulation process generated and collected Q-factor observations $\hat{q}_t^n$ based on Equation 21.

Equation 21: Sample Realization q-Values

$$\hat{q}_t^n = min_{x_t \in X_t^n}[C_t(S_t^n, x_t) + \gamma min_{x_t \in X_t^n}(\bar{Q}_t^{n-1}(S_t^{x,n}, x_t))]$$

The sample $\hat{q}_t^n$ values were calculated based on estimated PDS Q-factors $\bar{Q}_t^n(S_t^{x,n}, x_t)$ vice the pre-decision state Q-factors $\bar{Q}_t^n(S_t^n, x_t)$. As before, whether using a pre-decision state or PDS update, the syntax for the Q-factor always includes the action variable x_t. This highlights the special state-action relationship pairing captured by Q-factors. Another feature of Equation 21 is that the Q-factor used in the calculation of the sample realization $\hat{q}_t^n$ is the minimum Q-factor associated with that PDS regardless of the action taken to reach that PDS. This explains the minimization operand embedded within the overarching minimization problem. It is possible that the x_t value that solves the inner minimization expression is different from the one that solves the outer minimization

expression. A second and perhaps clearer expression that explains how $\hat{q}_t^n$ are now collected is to break Equation 21 apart into the two steps shown in Equation 22 and Equation 23.

Equation 22: Sample Realization q-Values

$$\hat{q}_t^n = min_{x_t \in X_t^n}[C_t(S_t^n, x_t) + \gamma \bar{V}_t^{n-1}(S_t^{x,n})]$$

Equation 23: Estimate of the PDS Value Function

$$\bar{V}_t^{n-1}(S_t^{x,n}) = min_{x_t \in X_t^n}[\bar{Q}_t^{n-1}(S_t^{x,n}, x_t)]$$

The expression $\bar{V}_t^{n-1}(S_t^{x,n})$ is the current estimate of the value function for PDS S_t^x. The estimated value function position is the minimum of the set of current estimated Q-factors values across all (S_t^x, x_t) pairings that exists after n-1 iterations. As before, the individually collected $\hat{q}_t^n$ observations serve as a basis for learning or approximating Q-factor values. The expression in Equation 24 adopts the Robins-Monro update for Q-learning on the PDS variable.

Equation 24: Q-Factor Learning

$$\bar{Q}_{t-1}^n(S_{t-1}^{x,n}, x_t) = (1 - \propto_{n-1})\bar{Q}_t^{n-1}(S_t^{x,n}, x_t) + \propto_{n-1} (\hat{q}_t^n)$$

The algorithm presented on the following page outlines the structure that was used to build the PDS Q-learning model.

Algorithm 2 (Post Decision State (PDS) Q-Learning). *Step 0: Initialization*

Step 0a: Initialize the approximation for the value function $\bar{Q}_t^0(S_t, x_t)$

for all $s \epsilon S, x \epsilon X$

Step 0b: Initialize S_0^1*, Set n=1*

Step 1: Choose a sample path ω^n

Step 2a: For t = 1, 2, …., T, Solve

$$\hat{q}_t^n = min_{x_t \in X_t^n}(C_t(S_t^n, x_t) + \gamma \bar{V}_t^{n-1}(S_{t-1}^{x,n}))$$

where $\bar{V}_t^{n-1}(S_t^{x,n}) = min_{x_t \in X_t^n}[\bar{Q}_t^{n-1}(S_t^{x,n}, x_t)]$

let x^n *be the value of x that solves the minimization problem*

Step2b: For t >1, update $\bar{Q}_{t-1}^n$ *and* $\bar{V}_{t-1}^n$

$$\bar{Q}_{t-1}^n(S_{t-1}^{x,n}, x_t) = (1 - \propto_{n-1})\bar{Q}_{t-1}^{n-1}(S_{t-1}^{x,n}, x_{t-1}) + \propto_{n-1} (\hat{q}_t^n)$$

$$\bar{V}_{t-1}^{n-1}(S_{t-1}^{x,n}) = min_{x_t \in X_t^n}[\bar{Q}_{t-1}^{n-1}(S_{t-1}^{x,n}, x_{t-1})]$$

Step 2c: *Find the next pre-decision state*

$$S_{t+1}^n = S^{M,W}(S_t^{x,n}, W_{t+1}(\omega^n))$$

Step 3: n = n+1. If n < N, go to Step 1.

Step 4: Return the Q-factors $\bar{Q}(S^x, x)$ *and value function approximations* $\bar{V}(S^x)$

2.13 Value Function Learning Algorithm

The Q-learning algorithm presented in the previous section produces both Q-factor values and estimates on the value function for the PDS variable. The Q-factors are stored in a 2-dimensional matrix with dimensions |S| · |X|. The horizontal rows of the matrix each represent a viable PDS S^x and the columns each represent an action

possibility x. The advantage of the Q-learning results is that the data provides a visual graphic pattern indicating which actions were selected or visited during the simulation that led to arriving at a particular PDS. In this manner, the Q-learning output serves as a logic test for the algorithm. If an action choice can not possibly lead to a PDS and the Q-factor value associated with this PDS-action pairing is positive and thus was visited during the simulation then it is clear that there is a problem with the implementation or code that is interpreting this algorithmic design.

Although the verification aspect of Q-learning makes it an ideal prototype approach for an ADP design, it is not practical for large scale sequential decision making problems. The matrix data requirement that is necessary to maintain Q-factor values tracking both the state and action pairing creates sizable storage and computational demands. Once again, as a result of the curse of dimensionality the Q-learning approach may become intractable for large scale complex problems.

An alternative approach is to compute the estimate of the value function directly without the use of Q-factors. The following value function learning algorithm adjusts the Q-learning algorithm in a way that allows for the value function updates to be calculated directly. The result of this design approach is to produce a single vector with dimension |S| that provides estimates for each PDS value function. Although this approach relieves some of the storage and computational demands of the Q-learning algorithm, it unfortunately provides no indication of exactly which actions were selected as part of learning or estimating the PDS value functions.

Algorithm 3 (Value Function Learning).

Step 0: Initialization

Step 0a: Initalize $\bar{V}^0(S^{x,0})$, S^0

Step 0b: Set n=1

Step 1: Choose a sample path ω^n

Step 2a: For t = 1, 2, …., T

Solve $\hat{v}_t^n = min_{x_t \in X_t^n}(C_t(S_t^n, x_t) + \gamma \bar{V}_t^{n-1}(S^{M,x}(S_t^n, x_t))$

And let x^n *be the value of x that solves the minimization problem*

Step2b: Update the value function for t >1

$$\bar{V}_{t-1}^n(S_{t-1}^{x,n}) = (1 - \propto_{n-1})\bar{V}_{t-1}^{n-1}(S_{t-1}^{x,n}) + \propto_{n-1} (\hat{v}_t^n)$$

Step 2c: *Find the next pre-decision state*

$$S_{t+1}^n = S^{M,W}(S_t^{x,n}, W_{t+1}(\omega^n))$$

Step 3: n = n+1. If n < N, go to Step 1.

Step 4: Return the PDS value function approximations $\bar{V}^x(S^x)$ for each state S

2.14 Q-Learning and Value Function Learning Design Summary

The implementation of these algorithms and the continual smoothing or feedback process used to obtain convergence of the state values is referred to as learning. Once it is determined that sufficient learning has occurred and that the estimates of the value function have converged, these values are then used to solve Bellman's optimality equation in what is known as the learnt phase of the model. This learnt phase is what ultimately produces a recommended optimal solution policy or state-action pairing.

CHAPTER THREE – THE MODEL DESIGN

3.1 Perspectives on Data Structures

The initial modeling design hurdle was determining an input data structure approach for the ADP. In order to make the model more user-friendly and easily explainable, it was important that the data inputs mimicked formats already utilized by the financial community and were easily understood by the involved decision makers. A quick study of the standard charts, tracking graphics, and outputs currently used as well as familiarization with the organization's financial database systems and tools all helped to define the ADP model's inputs. Much of the financial information examined tended to feature a month-to-month snapshot that captured the current status of the common five execution parameters: programmed amount, commitments, obligations, accruals, and expenditures. Figure 5 shows three common approaches consistently used to monitor and display financial tracking and planning data.

$M	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	July	Aug	Sept
Program	5.000											
Commitment		0.500	0.500	0.500	0.500	3.000						
Obligation			0.500	0.500	0.500	0.500	3.000					
Accruals			0.250	0.250	0.250	0.250	0.500	0.500	1.000	1.000	1.000	
Expenditures				0.250	0.250	0.250	0.250	0.500	0.500	1.000	1.000	1.000

A) Incremental Plan

$M	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	July	Aug	Sept
Program	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
Commitment		0.500	1.000	1.500	2.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
Obligation			0.500	1.000	1.500	2.000	5.000	5.000	5.000	5.000	5.000	5.000
Accruals			0.250	0.500	0.750	1.000	1.500	2.000	3.000	4.000	5.000	5.000
Expenditures				0.250	0.500	0.750	1.000	1.500	2.000	3.000	4.000	5.000

B) Cumulative Plan

$M	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	July	Aug	Sept
Program	5.000	4.500	4.000	3.500	3.000							
Commitment		0.500	0.500	0.500	0.500	3.000						
Obligation			0.250	0.500	0.750	1.000	3.500	3.000	2.000	1.000		
Accruals			0.250	0.250	0.250	0.250	0.500	0.500	1.000	1.000	1.000	
Expenditures				0.250	0.500	0.750	1.000	1.500	2.000	3.000	4.000	5.000

C) Audit Plan

Figure 5: Funding Planning Matrices

Each of the three planning pictorials provides a slightly different depiction of the same spend plan for a five million dollar project, $5.000M. Conducting a closer evaluation of the anticipated February position reveals nuanced differences emphasized in each plan. As indicated by the incremental plan A, it is expected that $0.500M is committed in February. In actuality, since individual commitment increments of $0.500M are planned each month from November through February, this brings the predicted month-end February total commitment amount to $2.0M. This total commitment amount is called out in the cumulative plan B. The audit plan C reveals that of the total $2.0M committed by February, only $0.500M will be remaining as unobligated. Similarly, for the $1.5M cumulative obligation amount called out in plan B, the audit plan C reveals that by February there will be a balance of $0.750M remaining to be accrued. The imposed naming conventions of incremental, cumulative, and audit

attempt to qualify the unique point of view that each of these three planning input structures provides.

These three approaches used to monitor execution planning each have unique appeal depending upon the daily concerns that arise due to one's particular job role. The incremental plan is ideal for the working level execution analyst who prepares funding documents and needs data points on a month-to-month basis regarding the incremental funding amount each document must incorporate. Additionally, this view is helpful for the working level analyst that wants to compare the predicted accruals and expenditures against those that are actually billed each month by a contractor. The cumulative plan has more appeal to those concerned with understanding the larger overarching strategic situation. For example, program directors and business financial managers often want to know if their programs and projects are meeting aggregate month-end expenditure goals. If there is evidence that a project is either over executing or under executing, these decision makers can take corrective actions in an attempt to better align cash allocations to actual needs. A critical aspect of this thesis is to compare the commitment decision policies produced by a myopic stubby pencil approach against that of an ADP model. The intent is to have the ADP model create good commitment decision policies that attempt to avoid over-execution or under-execution scenarios. Lastly, the audit plan provides a snapshot that tends to be preferred by comptrollers and auditors. The audit plan quickly recognizes unutilized funding balances and highlights those dollars which could be swept from the program and reallocated to other prioritize across the service or agency.

3.2 Immediate Cost Function and the ADP Network

The simulation aspect of the ADP design reiterates and calculates responses to the objective function hundreds of thousands of times until a convergence point is reached forming the ADP action policy vector. Through simulation, the ADP model is attempting to learn an action policy that can move the decision maker from one good state to another good state. Qualitatively the objective is to determine an efficient use of cash whereby the actual allocation of funding matches the true cost needs. The decision maker is essentially attempting to avoid the negative consequences that result from either over-committing or under-committing funding. The recommended commitment policy generated by ADP is a direct result from repeated exposure to Bellman's optimality equation and the embedded model's immediate or myopic cost function at each stage within the sequential decision making network.

The immediate cost function used for this thesis was constructed in an effort to determine the best commitment action to take at the start of each month that could pay the cash needs for three months. The model attempts to commit funding in a manner to ensure expenditure coverage for the current month as well as two more additional months or for what is required from time period t through the end of time period t+2. In this case, the decision maker is looking to take an action x_t that will minimize the absolute value between the cumulative commitment position at state S_{t-1} plus any new commitments or de-commitments from action x_t minus the cumulative actual expenditures at state $S_{t+2.}$ In short, at each stage in the simulation the model incurs the following immediate or myopic cost due to taking action x_t:

Equation 25: Immediate (Myopic) Cost Function

$$C_t(S_t^n, x_t) = \left\{ \text{ABS}\left| \begin{array}{c}\text{Cumulative}\\ \text{Commitments}\\ \text{from } S_{t-1}\end{array} + x_t - \begin{array}{c}\text{Predicted}\\ \text{Cumulative}\\ \text{Expenditures at } S_{t+2}\end{array} \right| \right\}$$

The rationale for using a three month time period was due to cash flow lag times. The three month window tends to provide a sufficient lag time to allow for a commitment action to become obligated on a contract vehicle where it is available for payout against invoices.

In this simplified form, the immediate cost penalty does not incorporate certain peculiarities. As stated, there is no preference between an action that causes over funding and an action that causes under funding. In reality, a decision maker is likely to possess a bias towards whether he or she wants to risk arriving at an under-commitment state that momentarily delays payment to a contractor or an over-commitment state that generates an opportunity cost trapping funds that could otherwise be used towards a different project. A second nuance of the stated cost function is that it assumes there is no cost associated with taking the actual commitment action. Given the labor hours involved, a commitment or de-commitment action does cost time and money. A decision maker may want to minimize the number of commitment actions taken over a year. As such, the cost or penalty function could be implemented in a manner where instead of making a commitment action every month, it is made once every other month or once a quarter. Nonetheless, the current from of the cost function does serve the immediate purpose of designing an ADP model that can emulate the DoD financial execution process.

Due to the construct of the immediate cost function the ADP model design will need to utilize, update, and maintain the data structures as depicted in the incremental plan A and cumulative plan B matrices. Since the cost function does not incorporate unutilized balances, the ADP model design will not require the data from the audit planning matrix C. Figure 6 shows the ADP model financial execution input data requirements of a $5.0M twelve month project for the five execution parameters of programmed, committed, obligated, accrued, and expended.

		P	C	O	A	E
t = 1	Oct	5.000				
t = 2	Nov		0.250			
t = 3	Dec		0.250	0.500	0.250	
t = 4	Jan		0.250	0.250	0.250	0.250
t = 5	Feb		0.250		0.250	0.250
t = 6	Mar		0.500	0.500	0.250	0.250
t = 7	Apr		0.500		0.500	0.250
t= 8	May		1.000	1.000	0.500	0.500
t = 9	Jun		1.000	1.500	1.000	0.500
t = 10	Jul		1.000	1.250	1.000	1.000
t = 11	Aug				1.000	1.000
t = 12	Sept					1.000

Incremental Planning Matrix A

	P	C	O	A	E
Oct	5.000				
Nov	5.000	0.250			
Dec	5.000	0.500	0.500	0.250	
Jan	5.000	0.750	0.750	0.500	0.250
Feb	5.000	1.000	0.750	0.750	0.500
Mar	5.000	1.500	1.250	1.000	0.750
Apr	5.000	2.000	1.250	1.500	1.000
May	5.000	3.000	2.250	2.000	1.500
Jun	5.000	4.000	3.750	3.000	2.000
Jul	5.000	5.000	5.000	4.000	3.000
Aug	5.000	5.000	5.000	5.000	4.000
Sept	5.000	5.000	5.000	5.000	5.000

Cumulative Planning Matrix B

Figure 6: Initial Planning $5.0M Project

The mechanics of the ADP model are easily represented pictorially by a sequential decision making network. Figure 7 shows the step-by-step flow of the ADP algorithm. A single iteration of the model will simulate month-to-month execution activity for time periods t = 1 through t = T, where T is the total number of months. The first step is to recognize the execution parameters of the current state-space or pre-

decision state S_{t-1} in a given t-1 time period. Next, an action x_{t-1} is taken based on the implementation of Bellman's optimality equation. The action x_{t-1} then moves the model into a post decision state (PDS) S_{t-1}^{x}. At this point the model enters into the next time period t where realized exogenous information W_t is incorporated into the system. The execution parameters are updated based on the revealed exogenous information and a new current pre-decision state S_t is defined. Once again, an action x_t is taken and moves the model into an updated PDS S_t^x. The model continues to step forward through the decision network for T months and then repeats itself for a total of N simulation iterations.

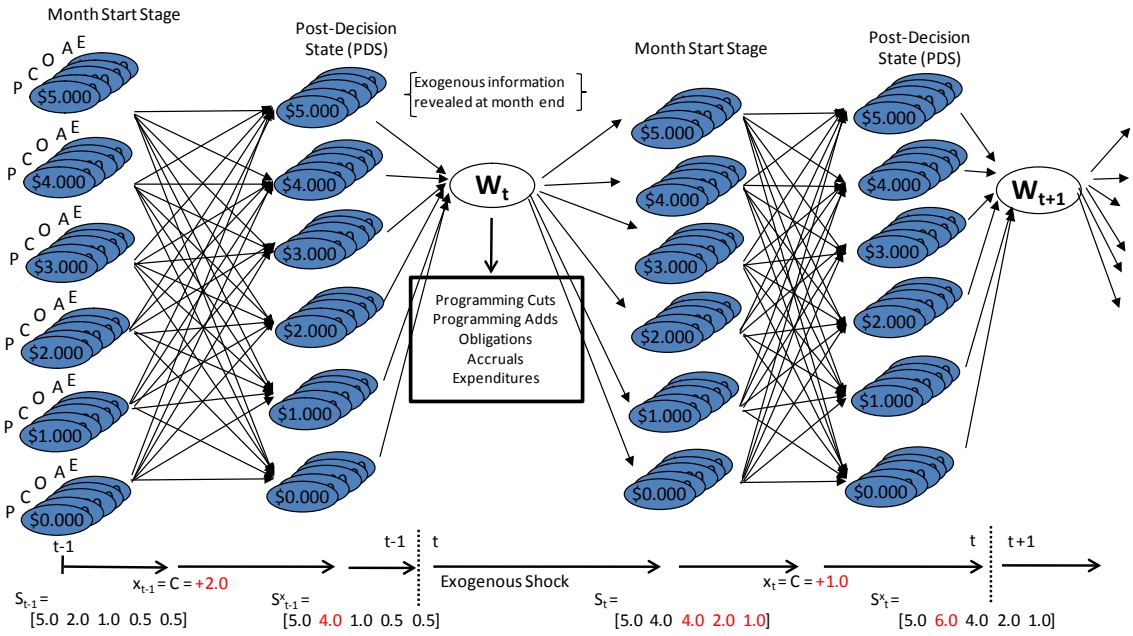


Figure 7: ADP Network Model of Commitment Cash Flow Problem

Figure 7 also shows how the values of the $5M project's state-space problem vector changes as the ADP model moves through the decision network. The state-space vector is defined by the current values of programmed amount, commitments, obligations, accruals, and expenditures [P C O A E]. In the initial pre-decision state, the values of these state-space execution parameters are shown as S_{t-1} = [5.0 2.0 1.0 0.5 0.5]. During month t-1, an action is taken to commit an additional x_{t-1} = $2.0M dollars. The state-space vector is then updated to reflect the $2.0M action as shown in the PDS S_t^x = [5.0 4.0 1.0 0.5 0.5]. In the next successive time period t, exogenous information reveals that during the prior month $3.0M was obligated, $1.5M was accrued, and $0.5M was expended. This realized information is incorporated into the new pre-decision state-space vector which now takes on values of S_t = [5.0 4.0 4.0 2.0 1.0]. Lastly, during time period t the action taken is to commit an additional x_t = $1.0M. As such, the new PDS vector is updated accordingly to S_t^x = [5.0 4.0 1.0 0.5 0.5]. This action selection and state-space vector update process will repeat itself hundreds of thousands of times during the ADP model's simulation process.

3.3 Subroutines of the ADP Model

The ADP model uses four Matlab subroutines to simulate the process of moving from one pre-decision state to the next. These four subroutines are best described as 1) selecting a commitment action, 2) updating the Q-factors or value function estimate, 3) incorporating the exogenous information, and 4) updating the incremental planning matrix A and cumulative planning matrix B data. These subroutines are the basis of the ADP approach and are designed to incorporate the flow of the sequential decision making

network, the application of Bellman's optimality equation, and the behavior of a DoD financial execution process.

The first subroutine selects a commitment action. At the start of each month, the set of all viable commitment actions is initially restricted by two factors. First, a commitment action is not allowed if it results in a total commitment level that is beyond the project's current programmed amount. Second, all commitment action possibilities are in multiples of a project's assigned allocation parameter. Figure 8 shows the model making a commitment action at the start of January or time period t = 4.

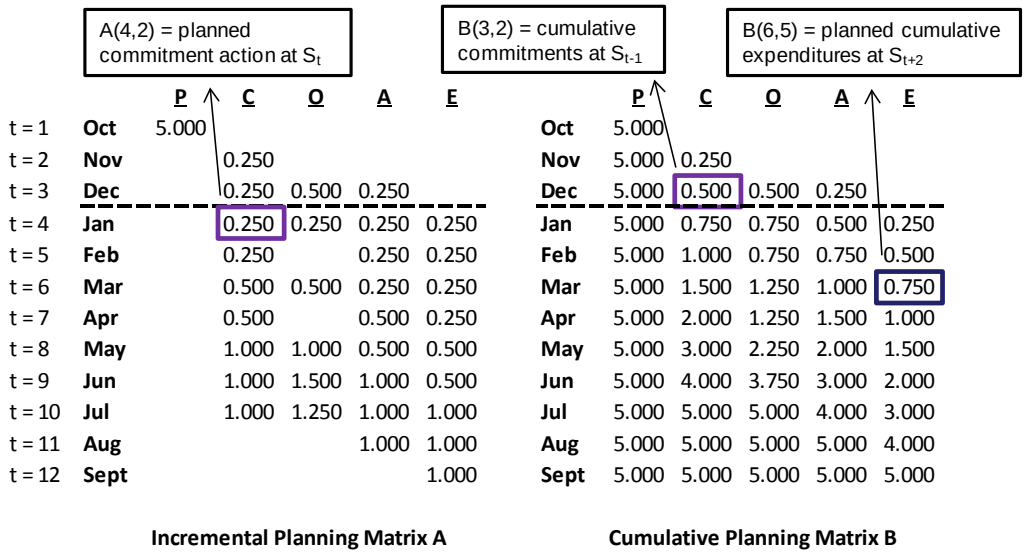


Figure 8: Matrix A & B Commitment Action Selection

The months October through December represent the simulated historical or actual data. The months from January forward represent the latest predicted values of the system. At this point, the total programmed funding level is $5.000M and the month-end

December t-1 commitment total is \$0.500M. Also at this point, there are no realized expenditures. This means that the set of all possible actions x_t ranges from a de-commitment of \$0.500M to a positive commitment of at most \$4.500M. Given an allocation parameter of \$0.250M for the \$5.000M project, the set of all viable commitment actions is as follows:

$$X_t = \left\{ \begin{array}{lll} (\$0.500M) & \$1.250M & \$3.000M \\ (\$0.250M) & \$1.500M & \$3.250M \\ \$0.000M & \$1.750M & \$3.500M \\ \$0.250M & \$2.000M & \$3.750M \\ \$0.500M & \$2.250M & \$4.000M \\ \$0.750M & \$2.500M & \$4.250M \\ \$1.000M & \$2.750M & \$4.500M \end{array} \right\}$$

After the subroutine makes an action determination, the planned incremental A commitment amount highlighted in Figure 8 is updated accordingly. This simulated actual commitment action may or may not be the same as the planned amount and is highly dependent upon the decision system's current value function estimate.

The Q-factor or value function update section integrates the use of Bellman's optimality equation and serves as the critical aspect of the ADP design. Depending upon the approach, realized $\hat{q}$ or $\hat{v}$ observations are calculated from the minimum commitment action resulting from the Equation 26 and Equation 27 expressions.

Equation 26: Sample Realization q-Values

$$\hat{q}_t^n = min_{x_t \in X_t^n}[C_t(S_t^n, x_t) + \gamma min_{x_t \in X_t^n}[\bar{Q}_t^{n-1}(S_t^{x,n}, x_t)]$$

Equation 27: Sample Realization v-Values

$$\hat{v}_t^n = min_{x_t \in X_t^n}[C_t(S_t^n, x_t) + \gamma \bar{V}_t^{n-1}(S_t^{x,n})]$$

The expression $C_t(S_t^n, x_t)$ represents the immediate or myopic cost from committing x_t dollars. As mentioned earlier, this cost is the absolute value between the resulting cumulative commitment position by taking action x_t and the predicted expenditure position at state S_{t+2}. These values are highlighted in the cumulative planning matrix B shown in Figure 8. In this example, the action x_t that minimizes the expression $C_t(S_t^n, x_t)$ is x_t = \$0.250M. Here, the immediate cost from taking a commitment action of \$0.250M is $C_t(S_t^n, x_t)$ = | \$0.500M + \$0.250M - \$0.750M | = 0. However, this commitment action may or may not be the same action that minimizes the objective function expressions for Q-learning and value function learning. These realized sample $\hat{q}$ and $\hat{v}$ values are calculated as the minimum of the sum of this immediate cost $C_t(S_t^n, x_t)$ plus the current value function estimate associated with the PDS arrived at after taking action x_t. Sections 3.6 and 3.7 discus in depth the different ADP approaches of Q-learning and value function learning as applied to the financial execution commitment problem. These sections will elaborate on the specific details for how the Q-factors and value function estimates are 'learnt' and actually updated as the simulation model progresses.

The subroutine that generates exogenous information serves as a mechanism to incorporate randomness and uncertainty into the model. Once the ADP algorithm selects a commitment action x_t, the incremental commitment parameter in matrix A and the cumulative commitment parameter in matrix B are updated accordingly. However, it is

not until the start of the following month or stage that exogenous information is realized regarding what occurred to the other execution parameters. At the start of each month, an update process reveals what happened to the programmed amount, obligations, accruals, and expenditures during the prior month.

In the ADP model, rules were developed to generate random data dictating the possible ranges of values for each parameter. A unique set of rules was designed for the initial $5.0M twelve-month scenario. At each stage or month in the simulation the programmed amount had a 2% chance of receiving a plus-up equal to $0.250M and a 5% chance it would incur a cut of $0.500M. If there was no plus-up or reduction the programmed amount would remain the same. The model assigned a 20% chance that the obligation associated with a commitment action would be delayed into the next month. A rule was created that only allowed obligations to be delayed for at most two months.

A plus-minus factor was created to generate variability around the planned accruals and expenditure figures. The plus-minus factor sets upper and lower bounds on the month-to-month simulated accruals and expenditures. Given a plus-minus factor of $0.500M, if the predicted accrual amount for the month was $1.000M then the range for the simulated accrual would be from a low of $0.500M to a high of $1.500M.

Figure 9 shows how realized exogenous information impacts the status of the incremental matrix A and cumulative matrix B data. Here, the program amount was cut by $0.500M, the anticipated obligation of $0.250M was delayed, and the accrual and expenditure positions came in at $0.500M vice the anticipated $0.250M. During

successive model runs, the exogenous subroutine's rules were adjusted and tested as part of conducting various modeling sensitivity analysis and data drills.

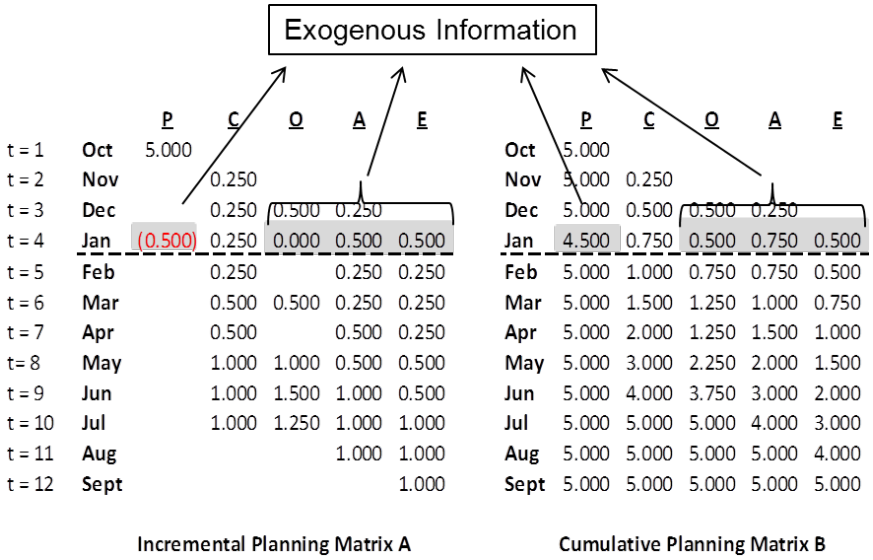


Figure 9: Matrix A & B Exogenous Updates

At the start of the new time period t+1 and before a new action decision is made, the incremental planning matrix A and B data is updated. The results of the commitment action and the exogenous information subroutines have a cascading impact on the prediction of future time period's financial execution. This impact needs to be properly captured so that the next commitment action determination is based on the best available prediction of the future state-space positions of the system.

Figure 10 shows changes made to the matrix A and B planning data due to the results produced by the prior subroutines. The reduction of $0.500M, resulted in a new

predicted cumulative planning amount of only $4.500M for the months of February through September. Also, given that the January accruals and expenditures were higher than anticipated, the future year planning for accruals and expenditures now exhibits an accelerated spending trend. The updated accrual and expenditure planning is governed by the assumption that accelerated spending will continue but, eventually slow to fit within the constraints of the new programmed amount.

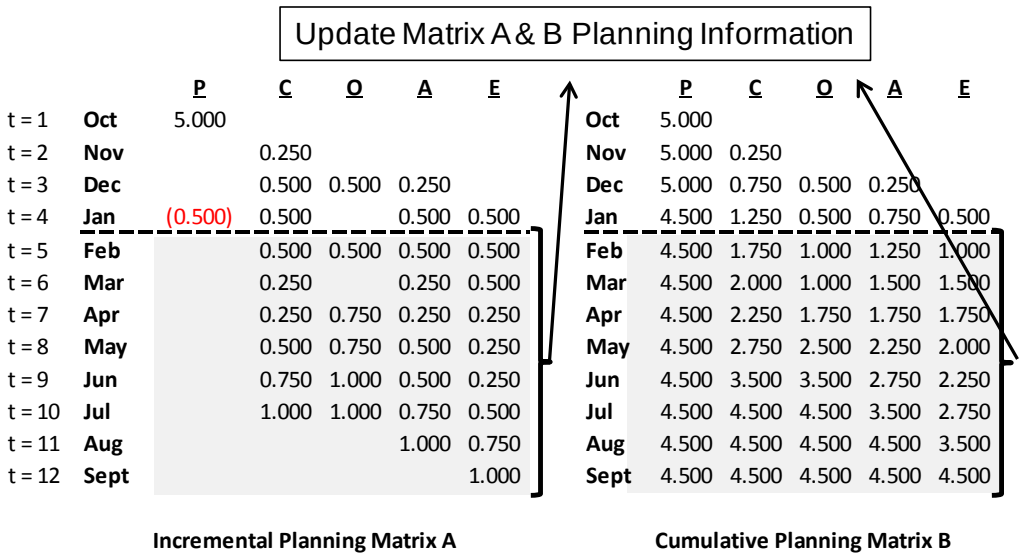


Figure 10: Matrix A & B Planning Updates

The matrix A and B update subroutine adjusts accruals and expenditures based on how the previous time period's actuals compare to expectations. If the actual was larger than the expectation, accrual and expenditure predictions are pulled forward in anticipation that the trend will continue. In a similar manner, if the actual was less than

the expectation, accruals and expenditure predictions are pushed into future time periods. In both cases, the assumption is that the full programmed amount is eventually exhausted and as such the updated cumulative amount will at some point always reach the programmed amount. The matrix A and B update subroutine ensures that the ADP algorithm design of the model is consistent with a Markov process in that the next decision is only based on the current state of the system and not the earlier prior states.

3.4 Complexities Due to Adding Multiple Projects

A number of complex issues arose once the ADP modeling process was expanded from modeling a single project scenario to modeling multiple projects simultaneously. As a result of adding projects, the size of the state-space, outcome space, and action space grew exponentially. This is consistent with the well know curse of dimensionality. In an effort to reduce the memory and run-time requirements of the model the state-space definition was limited to only those critical attributes absolutely necessary for determining a commitment action. Given that the cost of taking an action was defined as a function of just commitments and expenditures, it was no longer helpful to continue to track either the obligation or accrual status of a project. Additionally, the state-space definition was further constrained by removing the programmed plus-ups and reductions variability from the exogenous subroutine. This had the effect of fixing a project's funding amount at a static level throughout the decision making process. Once the project's programmed amount was defined as a static variable, it was no longer necessary to include it as part of the state-space vector. Therefore, the state-space definition of a project was simplified to just commitments and expenditures [C E].

These adjustments significantly reduced the size of the state-space and respective outcome space vectors. Prior to any adjustments a $5.0M project with an allocation parameter of $0.250M had 3,200,000 state-space and outcome space possibilities. Additionally, if at any time throughout the simulation process the $5.0M project received a programming plus-up the number of state-space and outcome space possibilities would grow considerably. After restricting the state-space definition to just commitments and expenditures, the number of state-space and outcome space possibilities for the $5.0M project dropped to only 400.

In this form, the model was now structured around examining the behavior between a single system predictive parameter and an associated action parameter. Here, the expenditure level serves as the predictive measure and the commitment choice serves as the action parameter for the system. In order to incorporate the multiple projects, the incremental and cumulative planning information is now loaded as an array. Figure 11 shows the updated multiple project state-space array input structure required to run the model.

In order to further improve run-time and memory requirements, additional adjustments were made to the viable action space possibilities. The range of all actions during each time period t was limited to reflect a more realistic commitment decision policy. If the current total expenditure amount was above the current total commitment amount, the commitment action must at a minimum bring the total commitment amount up to a level that is equal to the current total expenditure amount. Similarly, a de-commitment action was not allowed if it dropped the total commitment level below the

current total expenditure amount. These limitations not only reduced the size of the action state but, had the added affect of further reducing the size of the outcome space. Under these circumstances, the model could never arrive at a PDS in which expenditures were larger than commitments. In the case of the $5.0M project, the outcome space possibilities were further reduced from 400 to 231.

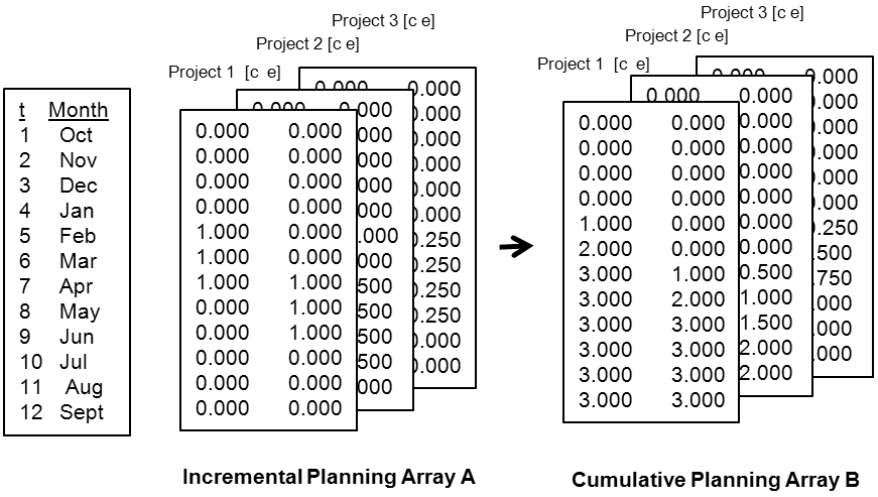


Figure 11: Updated Matrix A & B Planning Data for Multiple Projects

The inclusion of multiple projects into the ADP also added new challenges to the design of the commitment action subroutine. At each time period t, it was now not only necessary to select a commitment action for the aggregate system, it was also necessary to understand how that commitment action was allocated across each project. As such, the commitment action subroutine was modified to include a number of features that are best described with an example. Consider a three-project scenario in which the

individual budgetary programmed amounts and allocation parameters are as shown in Table 4.

Table 4: Multiple Projects Allocation Parameter

	Programmed Amount	Allocation Parameter
Project 1	$5.000M	$0.500M
Project 2	$2.000M	$0.250M
Project 3	$1.500M	$0.100M

Once multiple projects are incorporated, the model needs to calculate a programmed amount and allocation parameter for the decision system as a whole. The programmed amount for the system is merely the sum of the individual project's programmed amounts and in this case is equal to $8.500M. The allocation parameter for the system is the greatest common denominator (GCD) among the set of individual project's allocation parameters. Since the model design restricts the individual allocation parameters for each project to the following set [$5.0M $2.0M $1.0M $0.500M $0.250M $0.100M], the system allocation parameter is necessarily restricted to the same set with the addition of a $0.050M allocation parameter possibility [$5.0M $2.0M $1.0M $0.500M $0.250M $0.100M $0.050M]. In the above example, the system allocation parameter for the three given project's is in fact $0.050M, the GCD for the three projects.

As the model moves through successive stages, the incremental planning matrix A and cumulative planning matrix B information is continually maintained and updated for all three projects as well as for the total system. For the provided example, Figure 12

shows the ADP reaching time period t = 7, the start of April. The figure shows the current status of the cumulative planning B data for each of the three projects at this point. The information above the dashed line represents actual commitment and expenditures produced by the ADP while the information below the dashed line represents the planning or predictive commitment and expenditure position for the remaining time periods.

		C	E	C	E	C	E
t = 1	Oct	0.500	0.000	0.750	0.250	0.200	0.000
t = 2	Nov	1.000	0.000	1.000	0.500	0.300	0.100
t = 3	Dec	1.500	0.500	1.250	0.750	0.400	0.200
t = 4	Jan	2.000	1.000	1.500	1.000	0.500	0.300
t = 5	Feb	3.000	1.500	1.500	1.250	0.500	0.400
t = 6	Mar	3.500	2.000	1.750	1.500	0.500	0.500
t = 7	Apr	4.000	3.000	2.000	1.750	1.200	0.800
t = 8	May	4.500	3.500	2.000	2.000	1.300	1.000
t = 9	Jun	5.000	4.000	2.000	2.000	1.400	1.200
t = 10	Jul	5.000	4.500	2.000	2.000	1.500	1.300
t = 11	Aug	5.000	5.000	2.000	2.000	1.500	1.400
t = 12	Sep	5.000	5.000	2.000	2.000	1.500	1.500

Figure 12: Multiple Projects Matrix B Data

The current aggregate commitment and expenditure state-space vector for the system as a whole is [$5.750M $4.000M]. This is the sum of the individual project's state-space vectors shown at time period t = 6. Given a total programmed amount of $8.500M, this means that the range of commitment possibilities for the system is from a de-commitment action of $1.750M to a total positive commitment action of $2.750M. Since the system allocation parameter is $0.050M, the set of all viable system commitment actions includes all the amounts between negative $1.750M and positive $2.750M that are evenly divisible by $0.050M. This means that at time period t = 7 there

are 91 possible commitment options. However, what still remains unknown is how any of the viable 91 system commitment actions is allocated across each of the three projects.

At this point, it was necessary to create an internal decision algorithm that indicated how all 91 commitment options are allocated across the three projects. The choice was to design a demand rules algorithm that systematically apportioned each of the system commitment choices to the projects with the greatest demand or immediate need for funding. However, any funding allotment scheme still needed to fit the allocation parameters of the individual projects. A possible system commitment action was removed from consideration if it could not be evenly distributed within the allocation parameters of the individual projects while at the same time satisfying the priority order of meeting projects with the highest immediate demand first.

The logic here is twofold. First, at any time period t, a decision maker would not deliberately choose a commitment action that left an undistributed balance. Second, there always exists a viable allocation option that evenly distributes across the individual projects in priority order which will take precedent over any other evenly distributed allocation option that does not distribute according to a priority order. In this manner, common sense criterion was incorporated as part of determining which commitment action to ultimately pick.

The demand rules algorithm separately evaluated each of the 91 commitment options and either produced an allocation profile for that option or eliminated it from consideration. For example, given a possible commitment action of $1.900M, the demand rules algorithm assessed each project's month-to-month expenditure demand in

order to ultimately produce an apportionment plan. Using Figure 12, it is determined that at t = 7 project three has the greatest immediate forecasted expenditure demand. At this point, the current month's expenditures are forecasted to be $0.800M and the actual commitment level at this point is only $0.500M. Project three's immediate one month need for funding is $0.300M which is higher than the immediate one month demand for either project one and project two. As such, the demand rules algorithm allocates $0.300M of the $1.900M to project three, leaving a balance of $1.600M remaining to be allocated.

For the next month t = 8, there is a zero sum predicted demand for project one funding given the current project one commitment level, a predicted demand of $0.250M for project two, and an additional predicted demand of $0.200M for project three. Meeting the forecasted demand requirements for project two and three means that project two receives $0.250M of the remaining balance and project three receives an additional $0.200M of the balance, bringing the total allocation of funding for project three to $0.500M. The updated remaining allocation balance drops from $1.600M to $1.150M.

At t = 9, the forecasted expenditure demand for project one is $0.500M, for project two it is zero, and for project three it is $0.200M. Making these additional allocations means that to date project one receives $0.500M, project two $0.250M, and project three $0.700M. The updated remaining balance allocation has now dropped even further from $1.150M to $0.450M.

At this point when the demand rules algorithm is evaluating funding requirements beyond a month t+2 time period, it starts to compare the aggregate funding needs for all

the remaining time periods in the model vice for each successive month. In this example, for t = 10 and beyond the remaining demand for project one is $0.500M, for project two it is zero, and for project three it is $0.300M. Since there is insufficient funding remaining to meet neither the demand nor the minimum allocation parameter for project one, the demand rules algorithm defaults to allocating an additional $0.300M to project three. This brings the total allocation for project three up to $1.0M and further reduces the remaining allocation balance to just $0.150M.

Now, the only remaining forecasted project demand is $0.500M for project one. However, since the project one allocation parameter is also $0.500M, the remaining $0.150M balance cannot be evenly allocated. As such, the demand rules algorithm now eliminates the original $1.900M as a viable choice and moves in increments of $0.050M to the next viable commitment action of $1.950M.

Repeating the same the process but, this time with $1.950M produces an even allocation of $1.00M, $0.250, and $0.700M across the three projects respectively. Table 5 shows the results of the demand rules algorithm given the commitment options existing at time period t = 7. Of the original 91 commitment possibilities, 63 were eliminated leaving 28 options that fit evenly with the individual allocation parameters while meeting immediate demand in priority order. Table 5 provides the final individual commitment allocations across the three projects for each of the 28 acceptable system level commitment actions.

Table 5: Example Viable Commitment Allocations for Multiple Projects

Comm. Action #	Total Comm.	Proj. One Comm.	Proj. Two Comm.	Proj. Three Comm.	Proj. One Myopic Cost	Proj. Two Myopic Cost	Proj. Three Myopic Cost	Total Myopic Cos
	Commitment Action xt	Individual Project Commitment Actions			Individual Project Immediate Costs			$C_t(S_t^n, x_t)$
1	(1.750)	(1.500)	(0.250)	0.000	2.000	0.500	0.700	3.200
2	(1.500)	(1.500)	0.000	0.000	2.000	0.250	0.700	2.950
3	(1.000)	(1.000)	0.000	0.000	1.500	0.250	0.700	2.450
4	(0.500)	(0.500)	0.000	0.000	1.000	0.250	0.700	1.950
5	0.000	0.000	0.000	0.000	0.500	0.250	0.700	1.450
6	0.100	0.000	0.000	0.100	0.500	0.250	0.600	1.350
7	0.200	0.000	0.000	0.200	0.500	0.250	0.500	1.250
8	0.250	0.000	0.250	0.000	0.500	0.000	0.700	1.200
9	0.350	0.000	0.250	0.100	0.500	0.000	0.600	1.100
10	0.450	0.000	0.250	0.200	0.500	0.000	0.500	1.000
11	0.550	0.000	0.250	0.300	0.500	0.000	0.400	0.900
12	0.650	0.000	0.250	0.400	0.500	0.000	0.300	0.800
13	0.750	0.000	0.250	0.500	0.500	0.000	0.200	0.700
14	0.850	0.000	0.250	0.600	0.500	0.000	0.100	0.600
15	0.950	0.000	0.250	0.700	0.500	0.000	0.000	0.500
16	1.250	0.500	0.250	0.500	0.000	0.000	0.200	0.200
17	1.350	0.500	0.250	0.600	0.000	0.000	0.100	0.100
18	1.450	0.500	0.250	0.700	0.000	0.000	0.000	0.000
19	1.550	0.500	0.250	0.800	0.000	0.000	0.100	0.100
20	1.650	0.500	0.250	0.900	0.000	0.000	0.200	0.200
21	1.750	0.500	0.250	1.000	0.000	0.000	0.300	0.300
22	1.950	1.000	0.250	0.700	0.500	0.000	0.000	0.500
23	2.050	1.000	0.250	0.800	0.500	0.000	0.100	0.600
24	2.150	1.000	0.250	0.900	0.500	0.000	0.200	0.700
25	2.250	1.000	0.250	1.000	0.500	0.000	0.300	0.800
26	2.550	1.500	0.250	0.800	1.000	0.000	0.100	1.100
27	2.650	1.500	0.250	0.900	1.000	0.000	0.200	1.200
28	2.750	1.500	0.250	1.000	1.000	0.000	0.300	1.300

The next step after the demand rules algorithm determines the final allocation allotments is for the ADP to calculate a respective system level immediate or myopic cost $C_t(S_t^n, x_t)$ for each of the 28 commitment actions. The overall myopic system cost is the sum of the individual project myopic costs given the separate allocation allotments. The individual project's $C_t(S_t^n, x_t)$ costs remains as the absolute value difference between the current aggregate commitment level as a result of the action taken minus the next three month anticipated expenditure requirement. Table 5 also shows these respective

individual myopic costs attributed to the individual projects. The last column sums the individual project's myopic costs and provides the total myopic cost for the 28 commitment possibilities.

3.5 Q-Learning and Value Function Learning Designs

The initial ADP model used a Q-learning design. Through the use of simulation, the Q-modeling approach continually refines and smoothes Q-factors until they have reached a point of reasonable convergence. The Q-learning algorithm maintains and updates the Q-factors for all commitment action and PDS pairings in a Q-matrix. Figure 13 diagrams the simulation of the sequential decision process and steps involved in updating the Q-matrix and calculating the Q-factor information.

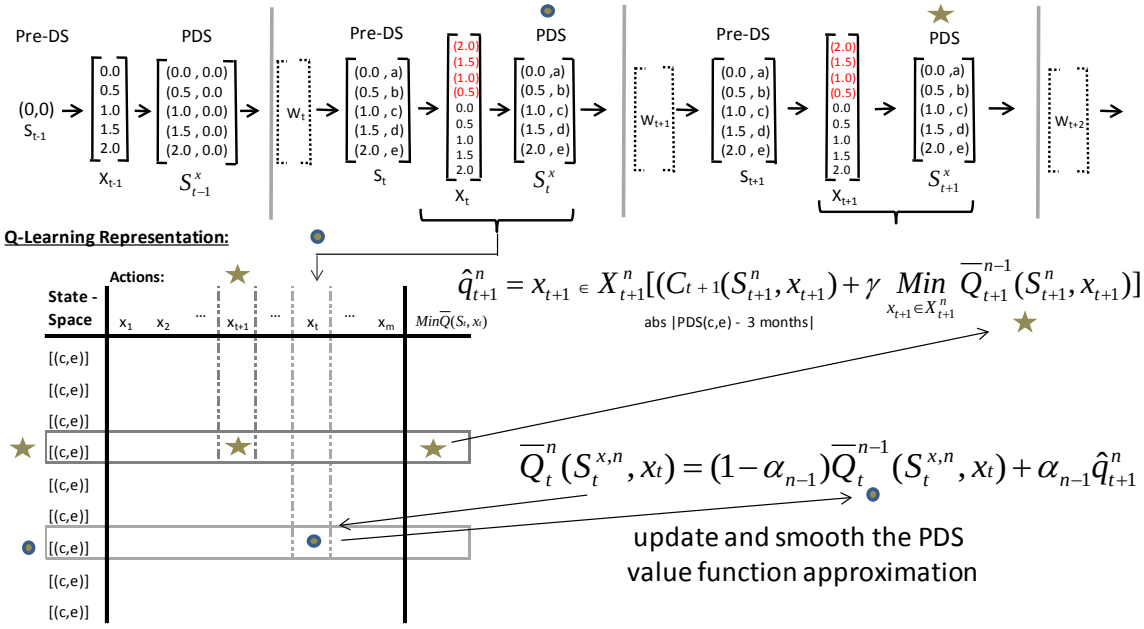


Figure 13: Q-Learning Diagram

The matrix is populated through an iterative process of observing a realized $\hat{q}$ value for a given state and using it to smooth the previous time periods PDS Q-factor. As depicted in Figure 13, an important aspect of the ADP design is that the value update process only requires data from the current stage and the previous stage of the sequential decision making problem. Since information from any prior stages is not needed, the ADP approach shares some commonality with the memoryless property used in a Markov decision system. All Q-factors are initialized at zero to start the algorithm. A sample $\hat{q}_{t+1}$ value is taken by combining the myopic cost of taking an action x_{t+1} plus the discounted value function estimate associated with the PDS arrived at from taking that action. The estimated value function is the minimum of the set of Q-factors associated with the PDS at time t+1. Once a $\hat{q}_{t+1}$ is obtained, it is then used to smooth the stored Q-factors associated with the previous time period t. An alpha-decay parameter $\propto_{n-1}$ is used to smooth the Q-factor.

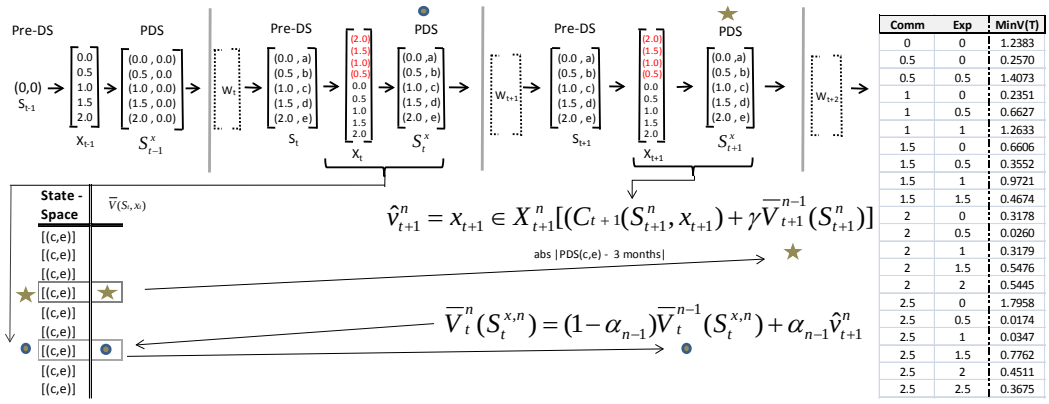
The ultimate Q-matrix result is a data set similar to the one depicted in Figure 14. The top row of the Q-matrix represents the commitment action choice used to reach the PDS commitment and expenditure position listed in the first two columns on the left. The data pattern assures us that the simulation model is in fact visiting the anticipated action and PDS pairings. A concern however is that the complexities of Q-factor calculations and storage requirements for the Q-matrix make Q-learning an inefficient approach for large scale problems.

Comm	Exp	(5.0)	(4.5)	(4.0)	(3.5)	(3.0)	(2.5)	(2.0)	(1.5)	(1.0)	(0.5)	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
0	0	4.593105	3.240517	3.942909	2.506222	2.713528	2.8977	4.342192	3.155769	2.039053	1.238259	4.399158	0	0	0	0	0	0	0	0	0	0
0.5	0	0	4.182829	1.820157	4.442513	2.603491	1.876836	2.584721	3.301983	2.200308	0.316237	3.121326	0.256951	0	0	0	0	0	0	0	0	0
0.5	0.5	0	2.978315	2.19727	2.535189	1.667643	2.360228	1.903759	2.404394	2.90757	4.813437	1.407338	2.783722	0	0	0	0	0	0	0	0	0
1	0	0	0	2.446311	3.014853	1.720142	3.218167	3.87171	4.355842	3.404336	0.235118	2.769361	0.522621	1.420244	0	0	0	0	0	0	0	0
1	0.5	0	0	1.746253	2.231379	4.760294	2.149056	3.146523	2.44335	1.897579	2.022749	0.662734	2.109432	2.264098	0	0	0	0	0	0	0	0
1	1	0	0	1.263345	3.454174	2.079029	1.840341	1.588968	3.092034	2.992489	2.265497	2.444747	1.879282	2.800189	0	0	0	0	0	0	0	0
1.5	0	0	0	0	1.620696	2.055022	4.351505	2.022041	2.411808	2.532145	3.135927	2.239712	0.660561	2.799926	1.726016	0	0	0	0	0	0	0
1.5	0.5	0	0	0	4.314988	1.663595	1.682072	2.363636	1.569646	1.992463	1.826735	0.355203	1.853347	0.830416	2.059368	0	0	0	0	0	0	0
1.5	1	0	0	0	1.492676	1.368906	2.319119	1.669449	3.498979	1.876675	0.972091	1.470205	2.032333	1.870047	2.128677	0	0	0	0	0	0	0
1.5	1.5	0	0	0	2.116797	1.630707	1.308034	1.334396	3.208789	0.467402	2.443047	1.217063	2.504269	3.272062	0.965751	0	0	0	0	0	0	0
2	0	0	0	0	0	2.285307	2.338855	3.35987	4.615637	3.258509	2.994544	3.635513	1.407253	0.317822	3.976046	3.059694	0	0	0	0	0	0
2	0.5	0	0	0	0	2.320516	1.778747	1.80124	1.284532	1.38232	0.34586	0.260081	2.673415	0.02597	0.027527	1.06126	0	0	0	0	0	0
2	1	0	0	0	0	1.343074	2.162401	2.946712	2.586787	1.576827	2.800488	2.608973	0.317857	3.497271	1.518783	2.934368	0	0	0	0	0	0
2	1.5	0	0	0	0	1.331283	1.200556	1.450437	2.303498	2.066899	1.782557	0.547623	3.179604	1.394261	2.071729	1.492124	0	0	0	0	0	0
2	2	0	0	0	0	1.818528	1.049782	1.10884	1.945241	0.544543	0.716931	0.850127	2.117909	1.849248	0.741475	0	0	0	0	0	0	0
2.5	0	0	0	0	0	0	2.920025	3.07924	2.728074	4.397483	3.35777	2.153177	1.795837	3.181114	2.628892	3.102666	2.240804	0	0	0	0	0
2.5	0.5	0	0	0	0	0	3.368114	3.300087	1.701658	2.987257	3.075026	1.794605	2.934653	0.017414	0.017554	3.038411	1.724059	0	0	0	0	0
2.5	1	0	0	0	0	0	2.509793	1.967639	1.288188	1.194157	1.914613	0.038399	0.034709	1.442583	0.853858	1.274971	1.429999	0	0	0	0	0
2.5	1.5	0	0	0	0	0	2.228385	0.776239	1.314933	1.530243	0.930553	1.428295	1.249322	1.895467	2.802327	2.034466	1.826942	0	0	0	0	0
2.5	2	0	0	0	0	0	2.25336	0.933223	1.432997	1.885266	2.243655	1.393963	1.517107	0.692764	2.048173	0.451052	0	0	0	0	0	0
2.5	2.5	0	0	0	0	0	0.949605	0.470348	0.379902	1.831057	1.172398	0.367475	1.54609	1.716537	0.482119	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	3.149052	2.967441	1.644242	4.577967	1.967207	2.197378	1.863188	2.801777	3.490615	2.583073	2.654369	0	0	0	0
3	0.5	0	0	0	0	0	0	3.069655	2.63204	2.5366	1.945799	2.63465	1.841423	1.652242	0.076903	0.072731	1.875528	2.884137	0	0	0	0
3	1	0	0	0	0	0	0	2.677918	3.173489	2.768688	2.934666	2.730004	0.080055	2.317524	0.248332	0.146261	1.108611	1.328518	0	0	0	0
3	1.5	0	0	0	0	0	0	1.191406	2.527674	1.63037	1.942176	1.199113	0.016674	0.016728	0.016793	1.988132	3.116973	1.738701	0	0	0	0
3	2	0	0	0	0	0	0	1.027168	2.841133	2.366059	2.037856	1.154233	2.926934	1.844707	0.016295	1.291345	1.52436	0	0	0	0	0
3	2.5	0	0	0	0	0	0	1.56042	0.750052	1.127166	2.003189	0.445673	0.627824	1.129074	1.501873	1.204346	0	0	0	0	0	0
3	3	0	0	0	0	0	0	0.604309	0.86695	0.701165	0.271282	0.567113	0.411723	0.108271	1.559428	0	0	0	0	0	0	0
3.5	0	0	0	0	0	0	0	0	1.863006	2.355526	2.318502	2.957	2.765366	2.893568	2.559316	3.294894	3.112401	4.398045	4.893306	0	0	0
3.5	0.5	0	0	0	0	0	0	0	3.524906	2.360812	2.028675	1.643429	1.867633	3.698398	1.383996	2.191923	1.053271	2.765881	3.7494	0	0	0
3.5	1	0	0	0	0	0	0	0	0.749138	1.302747	1.521823	1.718323	2.648074	2.367437	1.33171	0.017008	0.282158	1.358296	2.592928	0	0	0
3.5	1.5	0	0	0	0	0	0	0	2.283783	1.330461	1.418729	2.445431	0.016433	0.016516	0.016541	0.016344	3.117331	1.059731	0.969224	0	0	0
3.5	2	0	0	0	0	0	0	0	1.592784	0.599008	0.803637	2.038322	0.865072	0.015969	0.016122	0.958269	2.768699	2.271091	0	0	0	0
3.5	2.5	0	0	0	0	0	0	0	0.328137	1.017807	0.572215	2.166552	1.044566	1.01797	1.261174	1.107751	2.191004	0	0	0	0	0
3.5	3	0	0	0	0	0	0	0	0.491123	0.290359	0.499027	0.541447	0.440648	1.118863	0.351439	0.063733	0	0	0	0	0	0
3.5	3.5	0	0	0	0	0	0	0	0.337134	0.754151	0.042837	0.704714	0.085515	0.208794	0.019131	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	4.315004	2.322459	2.196213	3.244404	2.808249	2.298483	0.109498	4.146659	2.577183	3.213183	1.718718	0	0
4	0.5	0	0	0	0	0	0	0	0	2.427396	2.56899	2.902206	1.783043	3.643825	2.078255	1.555369	2.670643	3.330069	2.133228	1.499013	0	0
4	1	0	0	0	0	0	0	0	0	3.034836	1.8732	1.964737	1.494175	2.183134	1.409588	1.429607	2.53048	1.682277	2.391575	1.520064	0	0
4	1.5	0	0	0	0	0	0	0	0	1.778178	2.923229	2.478399	0.482843	1.401054	2.080638	1.335151	1.320966	0.016362	1.697827	1.606884	0	0
4	2	0	0	0	0	0	0	0	0	1.210115	2.876774	0.847329	0.016167	0.016097	0.016007	0.780615	0.016123	3.12254	0.830024	0	0	0
4	2.5	0	0	0	0	0	0	0	0	0.663556	0.511806	0.015442	1.664434	1.464394	0.015452	0.015715	1.289731	0.25377	0	0	0	0
4	3	0	0	0	0	0	0	0	0	0.237587	0.750948	0.445712	0.491764	0.926025	0.825486	0.711645	1.426235	0	0	0	0	0
4	3.5	0	0	0	0	0	0	0	0	0.021583	0.151247	0.588819	0.08161	0.088372	0.722349	0.077202	0	0	0	0	0	0
4	4	0	0	0	0	0	0	0	0	0.719324	0.146092	0.016896	0.042631	0.020721	0.160871	0	0	0	0	0	0	0
4.5	0	0	0	0	0	0	0	0	0	0	0.167616	1.898067	2.961028	2.699125	2.851881	3.434008	2.926889	2.258716	3.239837	4.05265	2.835013	0
4.5	0.5	0	0	0	0	0	0	0	0	0	2.32826	0.171509	3.386052	1.180675	2.30247	2.261945	0.175006	1.441662	2.72776	2.376759	2.467306	0
4.5	1	0	0	0	0	0	0	0	0	0	2.840816	3.869975	0.016559	3.482842	2.006814	2.225709	1.641043	4.203259	0.771404	1.47875	3.820565	0
4.5	1.5	0	0	0	0	0	0	0	0	0	1.319669	0.6982	1.614665	1.324325	2.743748	1.246729	2.425029	1.259691	1.61527	1.457016	1.611032	0
4.5	2	0	0	0	0	0	0	0	0	0	1.883712	2.323542	1.81459	0.234413	0.016097	0.490425	1.352257	0.772907	1.362282	0.993081	0	0
4.5	2.5	0	0	0	0	0	0	0	0	0	0.707289	1.241748	2.539489	0.01539	0.01562	0.254177	0.661947	0.891696	0.878488	0	0	0
4.5	3	0	0	0	0	0	0	0	0	0	0.452429	0.621188	0.014732	1.046127	0.01408	0.546166	0.376599	0.907052	0	0	0	0
4.5	3.5	0	0	0	0	0	0	0	0	0	0.461139	0.175942	1.041782	0.357745	0.397091	0.253937	1.247019	0	0	0	0	0
4.5	4	0	0	0	0	0	0	0	0	0	0.424944	0.463545	0.063618	0.021353	0.172568	0.010034	0	0	0	0	0	0
4.5	4.5	0	0	0	0	0	0	0	0	0	0.011252	0.011487	0.346337	0.009857	0.47281	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	1.59874	2.904082	0.082805	2.203494	2.0931	4.496563	0.281944	4.344734	2.222045	2.389115	3.126225
5	0.5	0	0	0	0	0	0	0	0	0	0	0.01714	1.780656	0.017094	2.609909	0.017195	0.017475	0.017542	2.429628	2.767272	1.711176	1.647942
5	1	0	0	0	0	0	0	0	0	0	0	0.016583	0.284796	0.680927	0.016962	0.017074	0.017077	0.017022	0.637242	1.471282	0.535322	3.196594
5	1.5	0	0	0	0	0	0	0	0	0	0	0.016126	0.209833	0.016364	0.01643	0.016468	0.016354	0.016545	1.460797	1.95933	1.938136	1.047965
5	2	0	0	0	0	0	0	0	0	0	0	0.015708	0.01559	0.01622	0.015984	0.312213	0.016171	0.015883	0.746813	0.698041	2.356466	0
5	2.5	0	0	0	0	0	0	0	0	0	0	0.015147	0.015725	0.015421	0.01556	0.01557	0.015684	1.150902	0.754049	1.446082	0	0
5	3	0	0	0	0	0	0	0	0	0	0	0.014912	0.015014	0.015199	0.015274	0.015094	0.613646	0.554658	0.43728	0	0	0
5	3.5	0	0	0	0	0	0	0	0	0	0	0.014474	0.014442	0.014611	0.014643	0.01447	0.0461	0.117523	0	0	0	0
5	4	0	0	0	0	0	0	0	0	0	0	0.013917	0.013766	0.013797	0.455701	0.68726	0.329732	0	0	0	0	0
5	4.5	0	0	0	0	0	0	0	0	0	0	0.012735	0.051754	0.022943	0.018172	0.387121	0	0	0	0	0	0
5	5	0	0	0	0	0	0	0	0	0	0	0.013346	0.013346	0.012346	0.012903	0	0	0	0	0	0	0

Figure 14: Example Q-Matrix

Value function learning is an extension of the Q-learning approach which alleviates the need to store and maintain the Q-matrix. Instead of producing a matrix of Q-factors, the value function learning approach stores and maintains a vector of PDS value function estimates. Each value is only associated with a particular PDS position vice a given PDS and action pairing. The algorithm does remain consistent with the Q-

learning approach in that it still only requires data from two successive stages to perform the value update process. For each time period t+1, a sample $\hat{v}$ observation is obtained in the same fashion as $\hat{q}$. However, in value learning the $\hat{v}$ observation is used with the alpha-decay parameter to directly smooth the estimated value function vice Q-factors. Figure 15 diagrams the more simplified value learning approach and provides a partial snapshot of the single vector output of PDS value function estimates. The problem with the value learning approach is that the final vector does not provide a visual data pattern which can confirm that the model is in fact visiting a wide array of possible commitment actions. It is only through visiting a wide array of commitment actions and PDS possibilities that the model is able to properly learn. The visual confirmation provided by the Q-learning is the primary reason why that approach is used to initially test the effectiveness of the ADP model prior to implementing the value function learning method.



Comm	Exp	MinV(T)
0	0	1.2383
0.5	0	0.2570
0.5	0.5	1.4073
1	0	0.2351
1	0.5	0.6627
1	1	1.2633
1.5	0	0.6606
1.5	0.5	0.3552
1.5	1	0.9721
1.5	1.5	0.4674
2	0	0.3178
2	0.5	0.0260
2	1	0.3179
2	1.5	0.5476
2	2	0.5445
2.5	0	1.7958
2.5	0.5	0.0174
2.5	1	0.0347
2.5	1.5	0.7762
2.5	2	0.4511
2.5	2.5	0.3675

Figure 15: Value Function Learning Diagram and Output Vector

3.6 Convergence: Alpha-Decay

The ultimate choice of the alpha-decay or step size parameter is critical to the model's success of reaching a convergence point after tens of thousands of simulation iterations. Since the value update calculation requires data from two successive stages or months, the number of Q-factor or value function updates that occur in a single iteration is one less than the total number of months in the sequential decision system being modeled. For example, a one year twelve-month system will have eleven updates for one simulation iteration. Regardless of whether the ADP model is using Q-learning or value function learning, the stochastic approximation update process takes the form shown in Equation 28

Equation 28: Value Update Process (Powell 2007)

$$\bar{\theta}^n = (1 - \alpha_{n-1})\bar{\theta}^{n-1} + \alpha_{n-1}\hat{\theta}^n$$

In this form, $\hat{\theta}^n$ represents sample observations that are similar to the $\hat{q}^n$ and $\hat{v}^n$ sample values that are taken during the Q-learning and value function learning ADP algorithms. Additionally, the $\bar{\theta}^n$ term which is similar to the $\bar{Q}^n(S^{x,n}, x)$ and $\bar{V}^n(S^{x,n})$ terms, represent what is considered either the signal, mean, or expectation value for the dataset or population from which the $\hat{\theta}^n$ observations are taken. As the simulation progresses the alpha-decay value systematically changes with the objective of ultimately obtaining a converged value for the $\bar{\theta}^n$ term and thus providing a reasonable estimated expectation. The following three alpha-decay properties are the necessary conditions for convergence.

$$\alpha_{n-1} \geq 0, \;\; n = 1,2, \ldots$$

$$\sum_{n=1}^{\infty} \alpha_{n-1} = \infty$$

$$\sum_{n=1}^{\infty} (\alpha_{n-1})^2 < \infty$$

These conditions ensure that the alpha parameter will always decline or decay during each successive simulation iteration. Since the calculation of the observed sample $\hat{\theta}^n$ is dependent upon the very parameter $\bar{\theta}^n$ that is being estimated, $\hat{\theta}^n$ is considered a biased proxy or estimate for the true $\bar{\theta}^n$ value. The nature of this bias and the relationship that exists between the observation term $\hat{\theta}^n$ and the signal term $\bar{\theta}^n$ as they evolve over time will have significant impact on the ideal step-size to use and the number of iterations that are required to obtain convergence. In the ADP literature, there is an emphasis placed on the alpha-decay decision. However, the alpha-decay choice is problem dependent and often discovered through a matter of trial and error. Powell (2007), provides an extensive discussion on various alpha-decay possibilities.

Some of the characteristics that impact the performance of the alpha-decay parameter include issues on whether the true data is stationary or non-stationary, the amount of "noise" that may exist in the sampling process, and whether or not the true data exhibits a consistent trend or projection over time. Larger or slower decaying step-sizes tend to perform better with non-stationary data since the weight of the smoothing

process will emphasize the latest observations $\hat{\theta}^n$ vice the historical or earlier obtained $\hat{\theta}^n$ observations.

An error term can be used as a measure on the amount of noise that exists in the system as well as a gauge on whether or not the data is exhibiting a trend or projection. Equation 29, shows the expression for the amount of error that exists between the observation term $\hat{\theta}^n$ and current estimate $\bar{\theta}^n$:

Equation 29: Estimate Error

$$\varepsilon^n = \bar{\theta}^{n-1} - \hat{\theta}^n$$

The behavior of the error term ε^n can provide indications on a better performing alpha-decay option. If the true data is either monotonically increasing or decreasing such that the error term ε^n is always the same sign, then once again a larger step-size or one that favors the latest observation terms $\hat{\theta}^n$ will perform better. However, if the data is relatively stationary but, continues to exhibit a lot of noise or variability between each $\hat{\theta}^n$observation without trending in a specific direction then a step-size that decreases quickly will likely perform better. Unfortunately, the inherent problem with using a step-size that decreases too quickly is the possibility of observing data that has only appeared to converge, when in reality the ADP algorithm has not achieved the best possible estimate of the signal term $\bar{\theta}^n$. This argument is the primary reason why the sample average alpha-decay parameter of $\alpha_n = 1/n$ is normally not a good step-size choice in spite of the fact that it does satisfy the conditions for convergence.

Alpha-decay parameters can either be deterministic or stochastic. Deterministic alpha-decay parameters are usually correlated to the iteration number n and tend to be easier to program and maintain throughout the simulation processes. A stochastic step-size is correlated to the sample observation $\hat{\theta}^n$. Since the observations $\hat{\theta}^n$ are random variables, a stochastic step-size is essentially also a random variable. Stochastic step-sizes tend to be more complicated to program and incorporate into the ADP design. However, they may have some advantages when working with data-sets that are either monotonically increasing or decreasing. Furthermore, a stochastic step-size has appeal if the various states of the decision system each converge at different rates.

The ADP algorithms presented in this thesis used an adapted deterministic harmonic alpha-decay parameter as suggested by both Darken et al. (1992) and Gosavi (2003). Equation 30 captures this particular alpha-decay parameter update process.

Equation 30: ADP Alpha-Decay Parameter

$$\alpha_n = \frac{\alpha_{n-1}}{1+\frac{n^2}{\beta+n}}$$

The attractive feature of this alpha-decay parameter is its ability to decay slowly during the early iterations and then accelerate before ultimately slowing again and tapering during the final iterations. This structure helps the ADP algorithm avoid two critical concerns. The first concern is that the alpha-decay parameter will decrease too slowly causing the algorithm to stall-out before reaching convergence. The second is that

the alpha-decay parameter will decrease too quickly and give the false impression of convergence. By having an alpha-decay parameter that remains relatively high early on, the value update process will initially favor the sample $\hat{q}^n$ and $\hat{v}^n$ values. This emphasis helps ensure that sufficient learning is occurring as part of the algorithmic process. Additionally, the deterministic properties as well as the simple structure of this alpha-decay parameter make it relatively easy to code and incorporate into either the Q-learning or value function learning ADP structures. Only the constant term β required tuning when experimenting with the number of iterations to perform as part of each simulation run.

3.7 Convergence: Mean Square Error (MSE)

The initial method used to confirm convergence was to evaluate the progression of a select sample of PDS values. Figure 16 provides an example of the value changes in two PDS positions for a twelve-month value function learning model across 80,000 simulation iterations. There are a number of problems with this depiction. First, although it appears that the variability in the estimated PDS values is decreasing towards the end of the simulation, there is no definitive reassurance that the algorithm is in fact reaching a point of convergence. Furthermore, the two sample data patterns selected may not necessarily serve as accurate surrogates for all the PDS possibilities encompassed in the decision system. Since a typical financial execution commitment problem will have hundreds of thousands of viable PDS, the memory requirement necessary to produce the Figure 16 graphic for all the PDS values makes this an inefficient approach for verifying convergence.

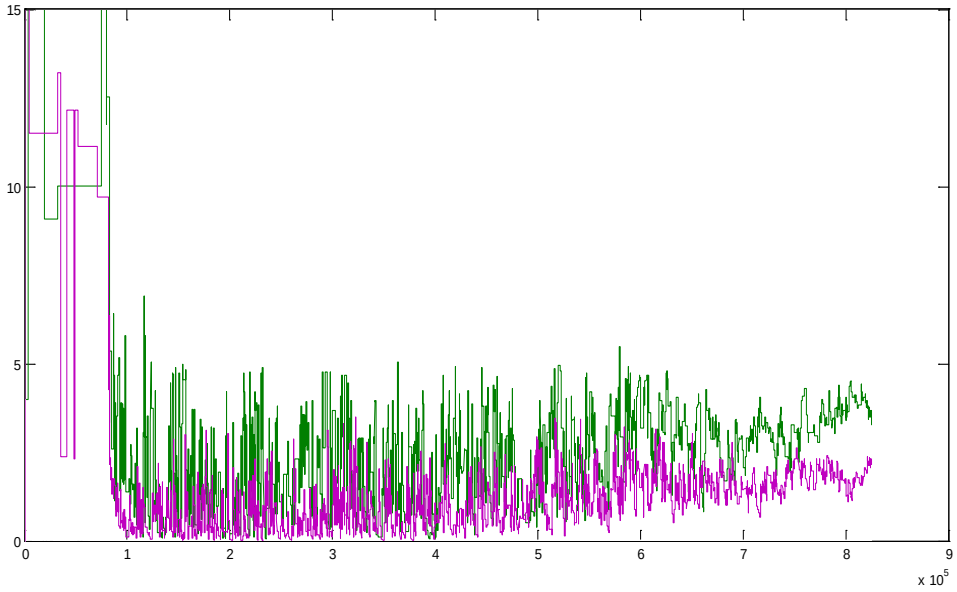


Figure 16: PDS Value Function Estimates Convergence Patterns

A better approach is to track the Mean Square Error (MSE) of the decision system. Instead of checking for convergence by maintaining the changing expectations for each PDS value, the MSE serves as single reference point for how quickly or slowly the expected PDS values are converging for the whole system. The MSE calculation is provided in Equation 31:

Equation 31: Mean Square Error (MSE) Calculation

$$MSE = (1 - \alpha_{n-1})MSE^{n-1} + \alpha_{n-1}(\bar{V}_t^{n-1}(S_t^{x,n}) - \hat{v}_{t+1}^n)^2$$

Figure 17 shows an example MSE convergence plot for a twelve-month 50,000 iteration decision model. If at the end of all simulation iterations the MSE term still remains relatively high, this may serve as an indication that further learning is required and that the model will need to run additional iterations. If that is the case, the rate of alpha-decay will likely need to be slower so that it does not time out or reach zero prior to the model's final iteration.

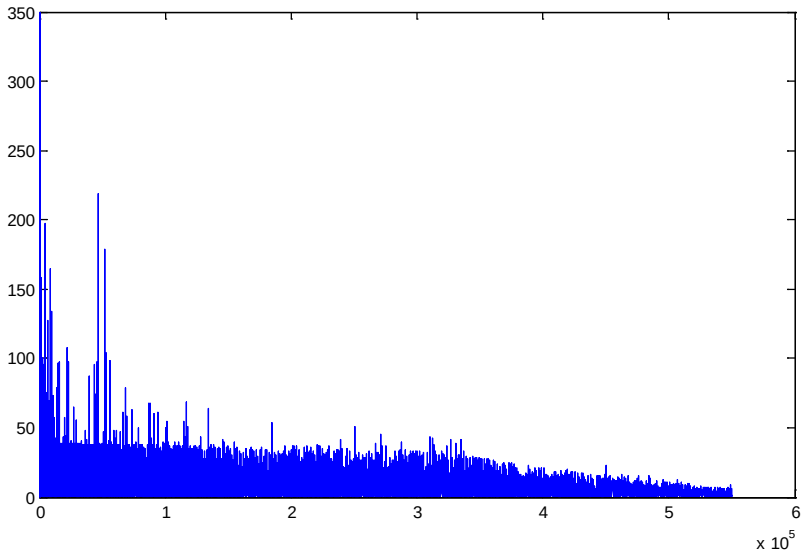


Figure 17: Mean Square Error (MSE) Plot

3.8 Exploration Vs. Exploitation (Learning)

Another sensitivity variable to consider in an ADP model design is the balance between the amount of exploration and exploitation iterations to conduct in a simulation run. The exploration iterations relax the minimization requirement when determining which action to take at a particular stage in the model. The exploration phase changes the

formulation for obtaining sample $\hat{q}$ and $\hat{v}$ observations to the following approaches shown in Equation 32 and Equation 33.

Equation 32: q-value Exploration

$$\hat{q}_t^n = Random_{x_t \in X_t^n}[C_t(S_t^n, x_t) + \gamma min_{x_t \in X_t^n}[\bar{Q}_t^{n-1}(S_t^{x,n}, x_t)]$$

Equation 33: v-value Exploration

$$\hat{v}_t^n = Random_{x_t \in X_t^n}[C_t(S_t^n, x_t) + \gamma \bar{V}_t^{n-1}(S_t^{x,n})]$$

Using exploration, the actual action taken is allowed to float and incorporates potential commitment choices that are 'good' and might deserve attention but, would otherwise not be visited using an absolute minimization policy. Exploration gives the model a chance to visit and learn the values for a broader range of action and PDS possibilities. The exploitation or learning phase of the model re-introduces the minimization operand as part of the commitment action criterion. Figure 18 shows an alpha-decay pattern where the first 5,000 iterations of the simulation used exploration while the successive 75,000 iterations used exploitation or learning. The benefit of using exploration is that it may incorporate potentially good solutions into the simulation process which deserve further investigation. However, a potential drawback is that it may take longer for the learning phase of the model to reach a point of sufficient convergence. Any ADP solution approach to a particular decision model design will need to strike a balance between the amount of exploration to conduct and the demands it will put on the run-time costs of the model.

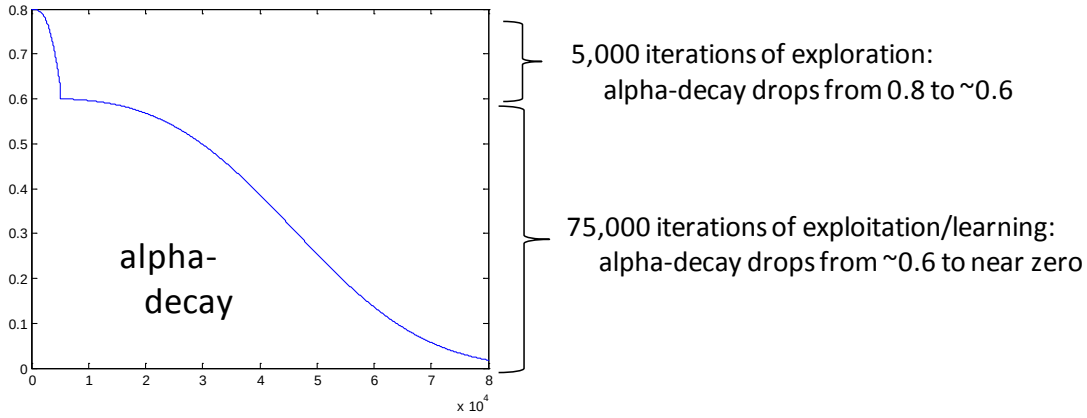


Figure 18: Alpha-Decay for Exploration and Exploitation

CHAPTER FOUR – RESULTS AND ANALYSIS

4.1 Learnt Phase

The ADP model is built around the three separate phases of exploration, exploitation (learning), and learnt. During the exploration phase, the minimization operand in Bellman's optimality equation is relaxed allowing x_t to be randomly chosen from the set of viable actions at time t. The algorithm is allowed to explore and gather information on good action and PDS combinations that would otherwise not be considered during learning. Once the exploration phase is complete, the learning phase is used to find the actual minimum x_t that satisfies Bellman's optimality equation at each time period t. The exploitation or learning portion of the ADP continues until a point of sufficient convergence is reached on the value function estimates for each of the PDS. The result of the learning phase is a stored single vector 'look-up' table containing the model's estimated values on each PDS. It is at last, in the learnt phase of the ADP approach where these value function estimates are used to create a decision policy. During this final learnt phase, the decision maker is now provided a recommended commitment policy of what action to take for a given pre-decision state position.

Unlike the exploration and learning phases, the learnt phase of the ADP algorithm is a stand-alone simulation process. The exploration and learning phases of the model work jointly to generate the value function estimates. If $n < N$ simulation runs were designated for exploration, then N-n were dedicated to learning. However, during the

learnt phase a separate set of N simulations are performed. The results of the N learnt simulation runs are used not only to provide a policy recommendation but, they are also used to conduct various types of sensitivity analysis and financial execution excursions. The subroutines for the learnt phase of the model are similar to those used for the exploration and learning phase with the exception that it is no longer necessary to have either a Q-factor or value function update subroutine. At each time period t, the subroutines for the learnt portion of the model now only include 1) selecting a commitment action, 2) incorporating exogenous information, and 3) updating the incremental planning matrix A and cumulative planning matrix B information.

The learnt phase of the model has three critical objectives. The first objective is to provide an optimal commitment strategy policy for the execution of a weapon system's monthly fiscal year budget or the budget of a collection of individual projects given a baseline forecasted position. The second objective is to provide a comparison between the recommended commitment policies provided by the ADP and that of the stubby pencil or myopic policy approach. The third objective of the learnt phase is to conduct sensitivity analysis and financial execution drills. This allows one to compare and contrast how the ADP and stubby pencil approaches each react to different modeling assumptions and various financial execution environments.

The input data structures used for the exploration and learning phases will need to be slightly modified in order to accomplish the stated objectives of the learnt phase. As before, a data set array is used to load each of the individual project's initial forecasted commitment and expenditure information in an incremental planning A and cumulative

planning B matrix format. However, this time a duplicate set of the same A and B planning data is loaded. As the simulation progresses, the first set of A and B matrices will be dedicated to tracking how the ADP responds within the decision system and the second set of A and B matrices will be dedicated to tracking how the stubby pencil policy responds. Figure 19 provides an example snapshot of the updated input structure used for the learnt phase of the ADP model.

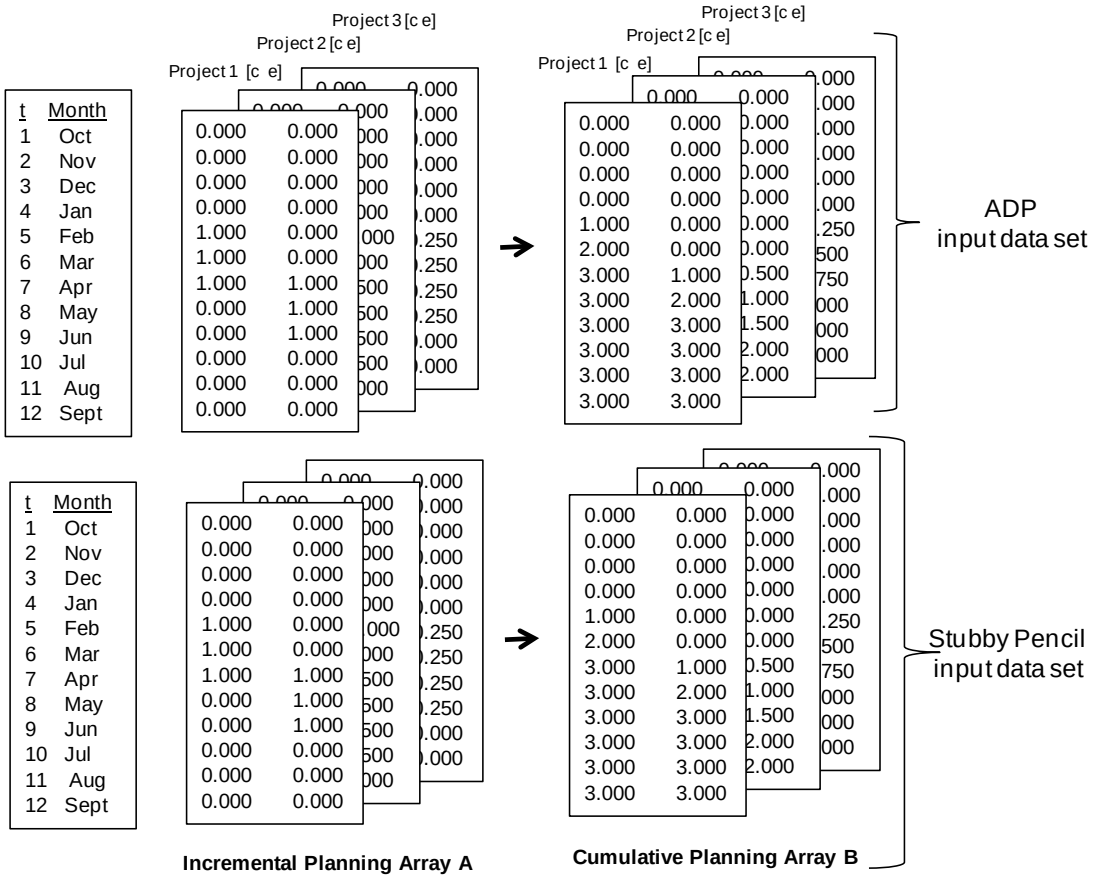


Figure 19: Learnt Phase Inputs

4.2 Comparative Results

The learnt phase utilizes N = 100 simulation iterations to collect the data necessary for conducting the comparative analysis between the ADP and stubby pencil approaches. Each iteration $n \epsilon N$ simulates a financial execution experience from t = 1 up to t = T months. Each simulation iteration n cycles through the three learnt phase subroutines T times. At each stage t, the learnt phase performs a unique commitment action selection x_t for both the ADP and stubby pencil policies, generates simulated exogenous information, and updates the financial execution incremental A and cumulative B planning arrays for both the ADP and stubby pencil. The exogenous impacts for each time period t still represent the simulated realizations of actual expenditures for that time period. This realized actual expenditure amount will be the exact same for both the ADP and stubby pencil data set arrays. As such, once there is a realized expenditure actual for time period t, the respective updated predicted expenditures for time period t+1 and beyond will also be the exact same for the stubby pencil data set arrays. As the 100 simulated iterations of the learnt phase progress, only the commitment attribute of the state-space variable will be different between the ADP and stubby pencil approaches. In this manner, the model can observe the different responses of ADP and stubby pencil to the same simulated expenditure event.

The critical data points collected during the 100 simulation iterations of the learnt phase are the different commitment actions taken each month by the two different ADP and stubby pencil approaches. During the exploration and learning phases of the model, the simulated action choice was a direct result of the application of Bellman's equation as part of the selecting a commitment action subroutine. The learnt phase of the model

determines an action choice in a similar manner except that two separate optimality equations are utilized during the selecting a commitment action subroutine. One optimality equation generates the ADP commitment actions and the other optimality equation generates the stubby pencil commitment actions.

The two optimality equations used in the learnt phase of the model are shown by Equation 34 and Equation 35. Equation 34 shows the optimality equation for the ADP approach. In this case, the optimal action $x_t^*(S_t)$ is driven by the familiar structure of Bellman's equation. The choice of commitment action at each time period t is the minimal sum of the immediate cost incurred by that action $C_t(S_t, x_t)$ plus the discounted estimate of the value function $\bar{V}_t(S_t^x)$. In contrast, the optimality equation for the stubby pencil or myopic approach is provided by Equation 35. What is readily visible is that for the stubby pencil approach the optimal action $x_t^*(S_t)$ is driven exclusively by the immediate cost function $C_t(S_t, x_t)$ without any consideration of the impact from the values of the successive state-spaces.

Equation 34: Learnt Phase ADP Optimality Equation

$$x_t^*(S_t) = \arg min_{x_t \in X_t} \left(C_t(S_t, x_t) + \gamma \bar{V}_t(S_t^x)\right)$$

Equation 35: Learnt Phase Stubby Pencil (Myopic) Optimality Equation

$$x_t^*(S_t) = \arg \min_{x_t \in X_t} \left(C_t(S_t, x_t)\right)$$

The fact that the stubby pencil approach only takes into consideration the immediate cost function $C_t(S_t, x_t)$ without regard to how this decision may impact future

decisions is why this is considered a myopic decision policy. As with the learning and exploration phases of the ADP model, the cost function $C_t(S_t, x_t)$ still remains the absolute value difference between the cumulative commitment position obtained after the incremental commitment action x_t and the predicted month-end expenditure position at time period t+2. As such, under the stubby pencil approach the optimal choice $x_t^*(S_t)$ will always be selected such that the associated immediate cost function value $C_t(S_t,x_t)$ is equal to zero. The Equation 35 formulation reflects the stubby pencil's myopic preference to take the greedy commitment action. This commitment selection will always provide the lowest immediate cost without regard to the impact current decisions may have on the future costs in the system.

4.3 Collected Data

The output generated by the learnt phase of the ADP model is a series of graphic pictorials. These pictorials contain various trend lines depicting the expenditure and commitment activities from time t = 1 to t = T. In order to create these graphics, the learnt phase needed to generate and store a number of data sets containing the results of the 100 learnt phase simulation iterations. The following is a list of the seperate data sets produced by the learnt phase and leveraged to build the graphics provided in this chapter.

1) ADP Commitments – A 100 x T matrix that maintains the month-end cumulative ADP totals for the aggregate sum of all projects
2) Stubby Pencil Commitments – A 100 x T matrix that maintains the month-end cumulative stubby pencil commitment totals for the aggregate sum of all projects

3) Expenditures – A 100 x T matrix that maintains the month-end cumulative actual expenditure positions aggregated across all projects
4) Three-Month Actual Expenditures – A 100 x T matrix that provides the cumulative actual expenditure position at the end of month t+2 for time period t. Recall that the immediate or myopic cost function $C_t(S_t, x_t)$ is the absolute value delta difference between the cumulative commitment position after action x_t and the predicted expenditure amount for month-end t+2. This three-month actual expenditure matrix provides the actual month-end t+2 cumulative expenditure amount at time period t. Obviously, this data point can not be calculated at the point t in the simulation iteration. Rather, the model must go back and provide this expenditure position once the simulation iteration completes time t+2.
5) Three-Month Predicted Expenditures – A 100 x T matrix that provides at time t the predicted cumulative expenditure totals for month-end t+2.
6) One-Month Predicted Expenditures – A 100 x T matrix that provides the predicted month-end cumulative expenditures for time period t.
7) ADP and Stubby Pencil Commitment Decision – A 100 x T x 2 array that maintains the individual commitment action/choice made each month by the ADP and stubby pencil policies. The commitment choice is the sum of the individual commitment actions made for each project.
8) ADP and Stubby Pencil Value Array – A 100 x T x 2 array that reports for each simulation run n and month t the value function estimates $\bar{V}_t^n(S_t^x)$ used

in the optimality equation for the ADP approach and the myopic cost value $C_t(S_t, x_t)$ for the stubby pencil approach. As mentioned earlier the optimality function $C_t(S_t,x_t)$ for the stubby pencil approach will always equal zero. As such, the stubby pencil portion of this array will merely be a 100 x T matrix of zero value entries.

9) Project Expenditures – A 100 x T x number of projects array that maintains each project's individual realized expenditure for each month.

4.4 Model Input Examples

Figure 20 provides the input parameters for a typical modeling scenario, referred to here as Test Case #2 - Trial #1. As described in Chapter 3, the ADP modeling inputs required for each project include the planned funding level, the allocation parameter, and the assigned plus-minus factor. Also, the ADP simulation requires the number of T = months that are modeled in the decision system. Lastly, as shown Figure 20, the ADP model requires the initial incremental A forecasted commitment and expenditure matrix data for each month t ϵ T in the planning horizon. Once this A matrix data is loaded, the ADP model code will generate and store the respective cumulative B forecasted commitment and expenditure matrix information for each project. As the model progresses through each time period t, the information contained in both the incremental A and cumulative B matrices will be updated and replaced.

The initial Test Case #2 - Trial #1 scenario examined the behavior of a three-project system over twelve months. The three projects each had individual budgets of \$5.0M, \$2.0M, and \$15.0M giving the system a total budget amount of \$22.0M.

Additionally, the individual allocation parameters were \$0.500M, \$0.250M, and \$1.000M. The overarching allocation parameter for the decision system was \$0.250M. This value is the GCD across the allocation parameters of each of the three projects.

Modeling Input Parameters	Funding Level	Allocation Parameter	PlusMinus	Months
Project #1	5.000	0.500	1.000	12
Project #2	2.000	0.250	0.750	12
Project #3	15.000	1.000	2.000	12

	Project #1		Project #2		Project #3	
month	Comm	Exp	Comm	Exp	Comm	Exp
Oct	0.000	0.000	1.500	0.000	0.000	0.000
Nov	0.000	0.000	0.250	1.000	0.000	0.000
Dec	0.000	0.000	0.250	0.500	1.000	0.000
Jan	0.000	0.000	0.000	0.250	1.000	0.000
Feb	0.500	0.000	0.000	0.250	2.000	1.000
Mar	1.000	0.000	0.000	0.000	4.000	1.000
Apr	1.500	0.500	0.000	0.000	4.000	2.000
May	1.000	1.000	0.000	0.000	2.000	4.000
Jun	0.500	1.500	0.000	0.000	1.000	4.000
Jul	0.500	1.000	0.000	0.000	0.000	2.000
Aug	0.000	0.500	0.000	0.000	0.000	1.000
Sept	0.000	0.500	0.000	0.000	0.000	0.000
	5.000	**5.000**	**2.000**	**2.000**	**15.000**	**15.000**

Figure 20: Test Case #2 – Trial #1 Input Parameters

Besides the initial input parameters and incremental A planning information, the modeling input information also requires assignments for the number of exploration iterations and number of learning iterations the model will perform. For Test Case #2 - Trial #1, the initial model run used 5,000 iterations of exploration and 50,000 iterations

of learning. The actual Matlab structure used to load all the relevant input parameters is provided below in Figure 21.

```
%% Transaction Parameters %%
proj = 3;                              % Number of projects in model
budget = [ 5.000 2.000 15.000 ];       % Maximum allowable budget for each individual project
incr = [ 0.500 0.250 1.000 ];          % The incremental allocation parameter for each project
months = 12;                           % Number of planning months
PlusMinus = [ 1.000 0.750  2.000 ];    % Plus-Minus factor to be used in the exogenous information sub-routine

expite = 5000;       % The number of iterations conducted during the exploration phase of the model
learningite = 50000; % The number of iterations conducted during the learning phase of the model

dfactor = 0.9;       % Discount Factor

%% Load Incremental Planning Data - Matrix A
% Load planning data once here, store as Z Matrix
% Recall during each simulation as Matrix A

Z = zeros(months,2,proj);
Z(:,:,1) = xlsread('InputFileMultiProjects.xlsx','MatrixA','AC67:AD78');
Z(:,:,2) = xlsread('InputFileMultiProjects.xlsx','MatrixA','AF67:AG78');
Z(:,:,3) = xlsread('InputFileMultiProjects.xlsx','MatrixA','AI67:AJ78');
```

Figure 21: Test Case #2 – Trial #1 Matlab Structure

4.5 Model Output Examples

In order to produce the graphic pictorials of the learnt phase of the model, the exploration and learning phases must first generate the value function estimates. One of the challenges with using the ADP approach is determining the number of iterations necessary to obtain sufficient convergence on the value function estimates. The MSE chart was the standard graphic used to judge the progress of overall convergence. Figure 22 provides the original input parameters for Test Case #2 – Trial #1 along with the MSE chart generated by the 50,000 iterations of learning. Recall that each simulation n will produce T-1 MSE data points. Therefore, the Test Case #2 – Trial #1 MSE graphic was generated using 50,000 X 11 = 550,000 data points. The blowup portion of the graph shows that for the last 10,000 observations the MSE was consistently beneath twenty.

Modeling Input Parameters	Funding Level	Allocation Parameter	PlusMinus	Months
Project #1	5.000	0.500	1.000	12
Project #2	2.000	0.250	0.750	12
Project #3	15.000	1.000	2.000	12

Load Incremental Planning Data - Array A

t	month	Project #1 Comm	Project #1 Exp	Project #2 Comm	Project #2 Exp	Project #3 Comm	Project #3 Exp
1	Oct	0.00	0.00	1.50	0.00	0.00	0.00
2	Nov	0.00	0.00	0.25	1.00	0.00	0.00
3	Dec	0.00	0.00	0.25	0.50	1.00	0.00
4	Jan	0.00	0.00	0.00	0.25	1.00	0.00
5	Feb	0.50	0.00	0.00	0.25	2.00	1.00
6	Mar	1.00	0.00	0.00	0.00	4.00	1.00
7	Apr	1.50	0.50	0.00	0.00	4.00	2.00
8	May	1.00	1.00	0.00	0.00	2.00	4.00
9	Jun	0.50	1.50	0.00	0.00	1.00	4.00
10	July	0.50	1.00	0.00	0.00	0.00	2.00
11	Aug	0.00	0.50	0.00	0.00	0.00	1.00
12	Sept	0.00	0.50	0.00	0.00	0.00	0.00
		5.00	5.00	2.00	2.00	15.00	15.00

Number of state-spaces: 4,005
Number of Months: 12
Total Budget = $22.000M
5,000 iterations Exploration
50,000 iterations Learning

RunTime: 11hrs 18Minutes
2.33 GHz Intel® Xeon
3.00 GB RAM

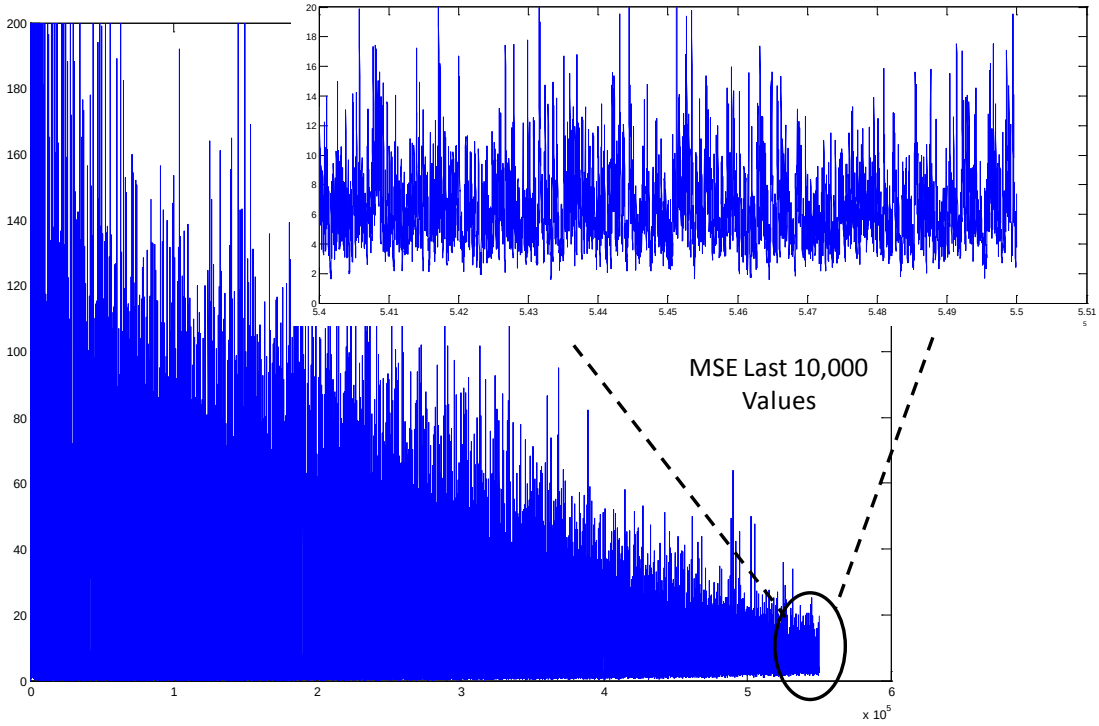


Figure 22: Test Case #2 – Trial #1 Mean Square Error (MSE)

After the exploration and learning phases of the ADP algorithm produces value function estimates for each PDS, N = 100 simulations of learnt phase analysis are then used to produce the graphics shown in Figure 23 and Figure 24. These graphics are the default commitment and expenditure outputs of the learnt phase. The top box in Figure 23 shows three different expenditure trend lines. The black line shows the original planned cumulative expenditures. The red line provides the average predicted expenditures. Unlike the initial black line that provides the predicted expenditures for all months at the start of October and remains static throughout the course of the simulation, the red line provides the predicted expenditure amount for time t given what is known at t-1. As such, the red line represents the continual updates to the forecasts on expenditures at a point in time just one month before the actual is realized. And lastly, the green line represents the average expenditures that actually occurred that month.

Additional trend patterns shown in Figure 23 are captured by the maximum and minimum asterisks. During the exogenous information subroutine, a strand of information ω^n is generated to simulate the actual expenditures that occur each month. This is the manner in which uncertainty or randomness is injected into the model. The maximum and minimum asterisks represent the upper and lower limits that are possible for the individual ω^n values. Therefore, the blue asterisks that are inside the top box in Figure 23 represent the month-to-month expenditure pattern if the maximum allowable actual expenditure occurred each month. Likewise, the purple asterisks represent the month-to-month expenditure pattern if the minimum allowable expenditure occurred each month.

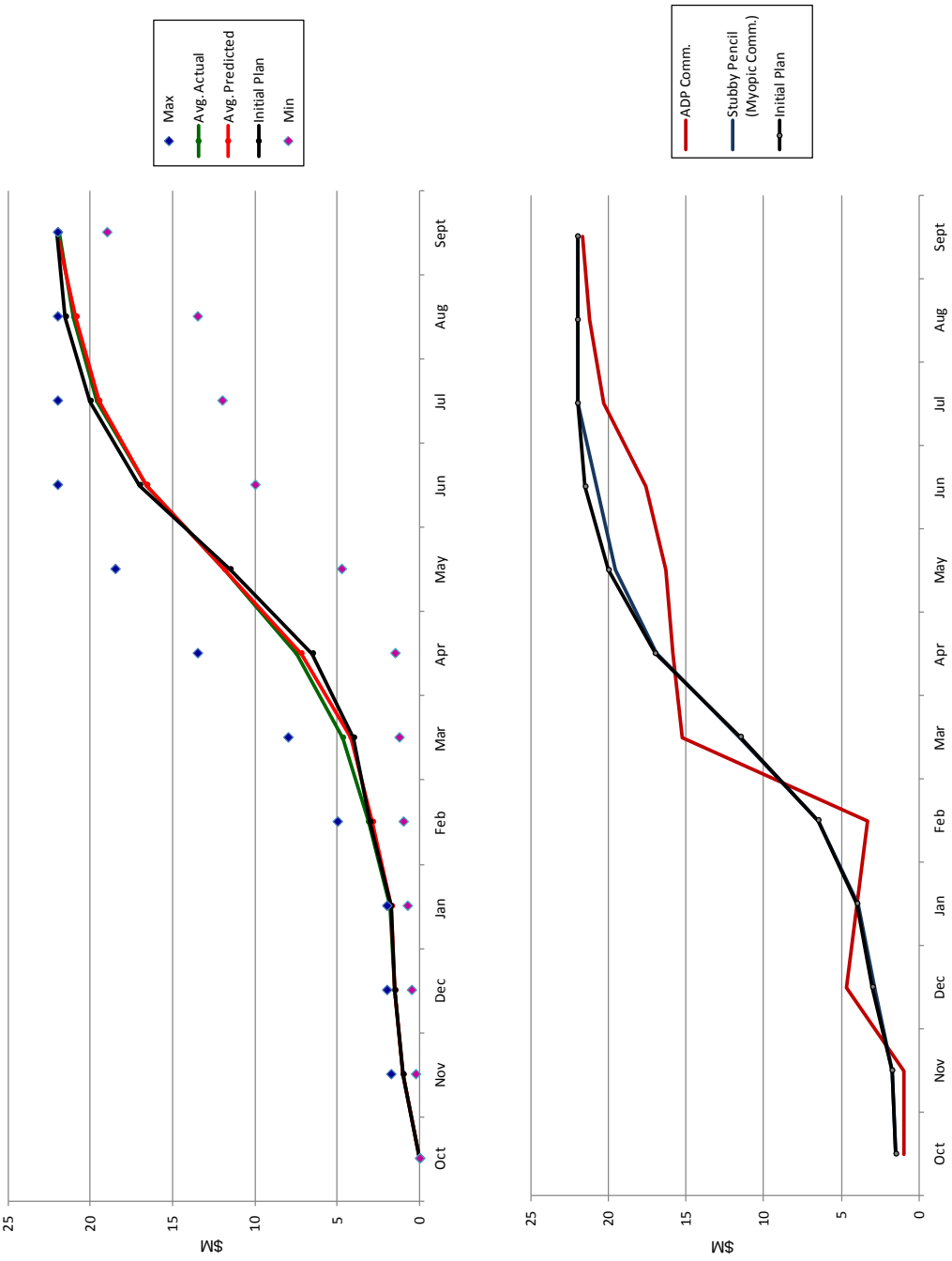


Figure 23: Test Case #2 – Trial #1 Output Graphic 1

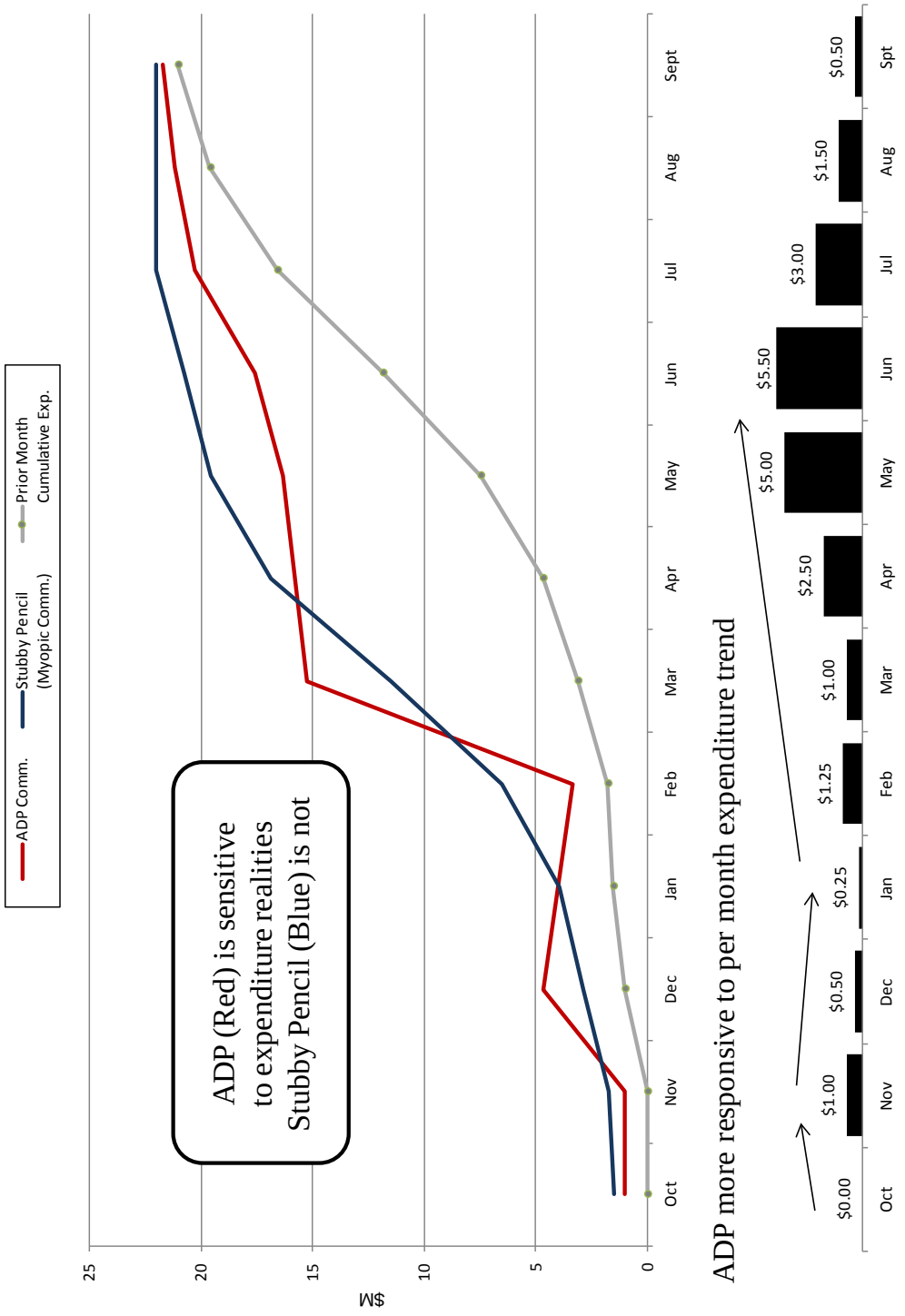


Figure 24: Test Case #2 – Trial #1 Output Graphic 2

The bottom box in Figure 23 provides trend lines that represent the commitment patterns associated with the expenditure results for Test Case #2 – Trial #1. This bottom graphic shows three different commitment trend line outputs. The black line depicts the baseline cumulative plan outlined at the start of October. The red line and the blue line respectively represent the results of the ADP cumulative commitment choices and the stubby pencil cumulative commitment choices.

The picture in Figure 24 is the second standard graphic that is produced by the learnt phase of the ADP model. This graphic combines both the commitment and expenditure information onto the same grid. The ADP and stubby pencil commitment trend lines used in Figure 23 are replicated here. However, they are now superimposed with a gray prior month expenditure trend line.

4.6 Learnt Phase Observations from Test Case #2 – Trial #1

There are a number of observations one can make from these output graphics. The first observation taken from the top box in Figure 23, is that there is little variability between the three expenditure trend lines. This low variability may be due to a number of factors which could include the small total budget size used in this scenario of only $22M or it may be due to the small plus-minus assignments for each of the projects. As such, the use of this particular scenario may make it difficult to understand the impacts of randomness on the different ADP and stubby pencil approaches.

There are two additional observations associated with the commitment trends in the bottom graph on Figure 23 and in the graphic shown in Figure 24. The first issue is the sharp up and down movements of the ADP commitment actions. Although this

commitment strategy at first may seem erratic, upon further investigation this pattern appears to be a direct response to the individual planned monthly expenditures. In November, total planned expenditures jump to $1.000M and then tapers to $0.500M and further to $0.250M for December and January. Furthermore, once the model moves beyond January, the individual expenditure totals increase substantially from month-to-month. The bar chart at the bottom of Figure 24 plots the original by month incremental expenditure amounts. When compared against the two different commitment trend lines, the dramatic up and down movements seem to suggest that ADP is more sensitive to these same up and down expenditure patterns that existed in the initial plans.

The second observation is that as a whole the ADP policy for the most part was consistently beneath the stubby pencil policy. Except for the months of December and March the red ADP commitment strategy remained under that of the stubby pencil commitment strategy. This may suggest that the ADP strategy is recommending a more conservative approach when making commitment decisions. Even after the model reaches the end of the time period in September, ADP has not committed the full $22.0M budget. Lastly, it should be noted that neither the ADP nor the stubby pencil commitment polices ever move into the undesirable state where the cumulative commitment amount has dropped below last month's expenditure amount. The gray prior month expenditure trend line provided in Figure 24 serves as a cross-check that neither policy allowed projects to expend more funding that what has been received to date.

4.7 Exploration Vs. Exploitation (Learning) Revisited

One aspect of the ADP approach is examining the impacts that occur to a modeling scenario as a result of altering the number of iteration runs dedicated to either exploration or exploitation (learning). The following Test Case #2 -Trial #3 scenario examines what happens to the Test Case #2 - Trial #1 baseline case when the exploration iterations are increased from 5,000 to 10,000 and the learning iterations are increased from 50,000 to 100,000. The results of the Test Case #2 - Trial #3 scenario are shown in Figure 25, Figure 26, and Figure 27. The immediate observation is that the MSE statistic starts at an initial higher level and as such takes longer to reach the same convergence point that was obtained in Test Case #2 – Trial #1. As mentioned before, since there are twelve months in the modeling scenario each iteration n produces eleven MSE data points. For the Test Case #2 – Trial #3 scenario, the 100,000 iterations of learning will provide 1,100,000 MSE measurements.

Figure 25 highlights the last 10,000 MSE data points produced by this excursion. This graphic shows how it took nearly twice as many learning iterations for the model to reach roughly the same MSE values that existed in Test Case #2 –Trial #1. It appears that the doubling of the exploration iterations required the number of learning iterations also be doubled in order to obtain a MSE point that was consistently under twenty for the final iterations of the model. Although the exploration phase provides an opportunity for the ADP simulation to learn information regarding the values of potentially 'good' action and PDS combinations it has the undesirable consequence of also incorporating 'bad' action and PDS combinations. A possible conjecture is that it was necessary to perform more learning iterations to reach an acceptable convergence point. The ADP literature

discusses the pros and cons of incorporating exploration into the model design along with alternative implementation schemes that limit exploration from being completely random and to focus more on just an exclusive subset of 'good' states. An exploration scheme that can focus the ADP model to visit a better subset of only 'good' state space positions will likely obtain improved convergence results during the learning phase of the simulation.

4.8 Learnt Phase Observations from Test Case #2 – Trial #3

The Test Case #2 – Trial #3 scenario was exclusively a modeling excursion that examined the impact of incorporating more exploration iterations into the model design. The graphics in Figure 26 and Figure 27 provide the expenditure and commitment trend line results for this particular modeling excursion. When comparing the results to the initial Test Case #2 – Trial #1 case, it is observed that the overall results are similar. The one nuance is that the ADP commitment choice does not exhibit the same dramatic up and down commitment pattern. However, although this initial commitment action is slightly smoother than in the initial case, the overall ADP commitment choice has still remained beneath that of the stubby pencil. Again, it appears that ADP still prefers a more conservative allocation approach over the simulated duration from time period t = 1 to time period t = T when compared against the stubby pencil approach.

Modeling Input Parameters	Funding Level	Allocation Parameter	PlusMinus
Project #1	5.000	0.500	1.000
Project #2	2.000	0.250	0.750
Project #3	15.000	1.000	2.000

Number of state-spaces: 4,005
Number of Months: 12
Total Budget = $22.000M
10,000 iterations Exploration
100,000 iterations Learning

RunTime: 31hrs 10Minutes
2.33 GHz Intel® Xeon
3.00 GB RAM

Load Incremental Planning Data - Array A

		Project #1		Project #2		Project #3	
t	month	Comm	Exp	Comm	Exp	Comm	Exp
1	Oct	0.00	0.00	1.50	0.00	0.00	0.00
2	Nov	0.00	0.00	0.25	1.00	0.00	0.00
3	Dec	0.00	0.00	0.25	0.50	1.00	0.00
4	Jan	0.00	0.00	0.00	0.25	1.00	0.00
5	Feb	0.50	0.00	0.00	0.25	2.00	1.00
6	Mar	1.00	0.00	0.00	0.00	4.00	1.00
7	Apr	1.50	0.50	0.00	0.00	4.00	2.00
8	May	1.00	1.00	0.00	0.00	2.00	4.00
9	Jun	0.50	1.50	0.00	0.00	1.00	4.00
10	July	0.50	1.00	0.00	0.00	0.00	2.00
11	Aug	0.00	0.50	0.00	0.00	0.00	1.00
12	Sept	0.00	0.50	0.00	0.00	0.00	0.00
		5.00	5.00	2.00	2.00	15.00	15.00

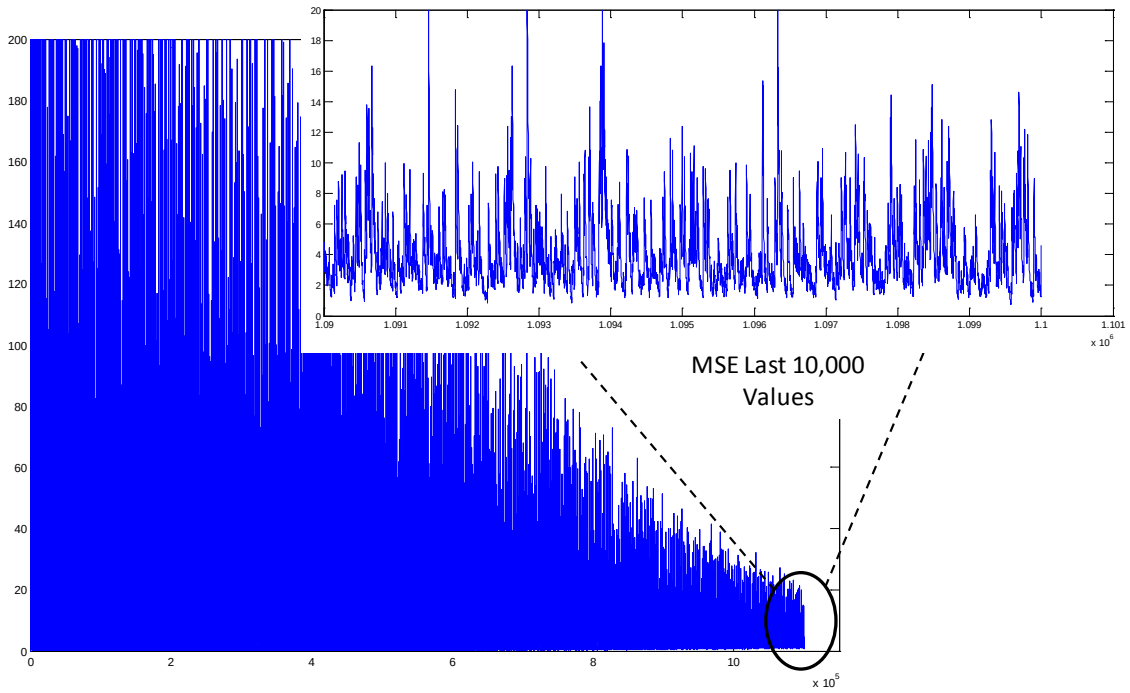


Figure 25: Test Case #2 – Trial #3 Input Parameters and MSE

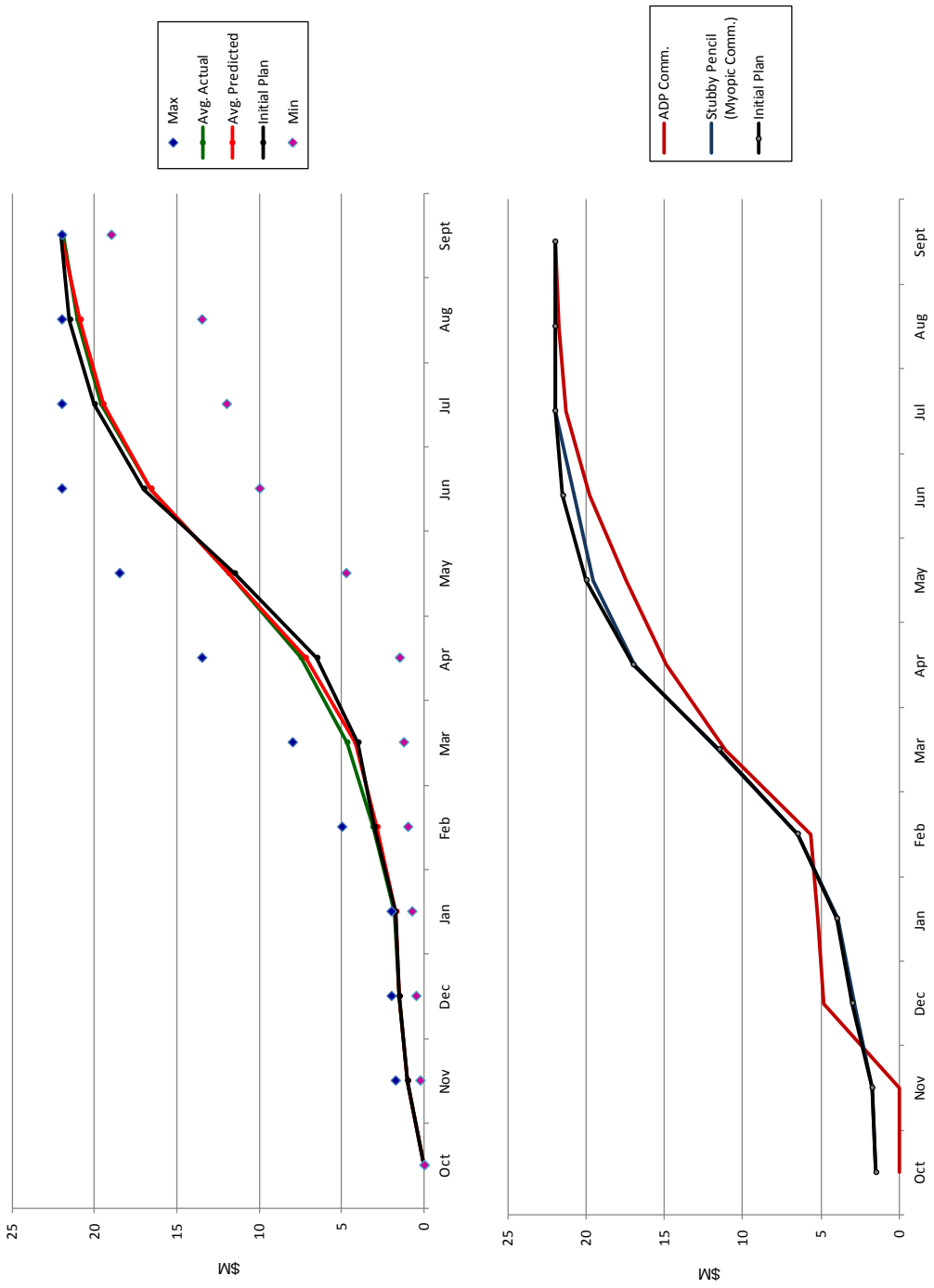


Figure 26: Test Case #2 – Trial #3 Output Graphic 1

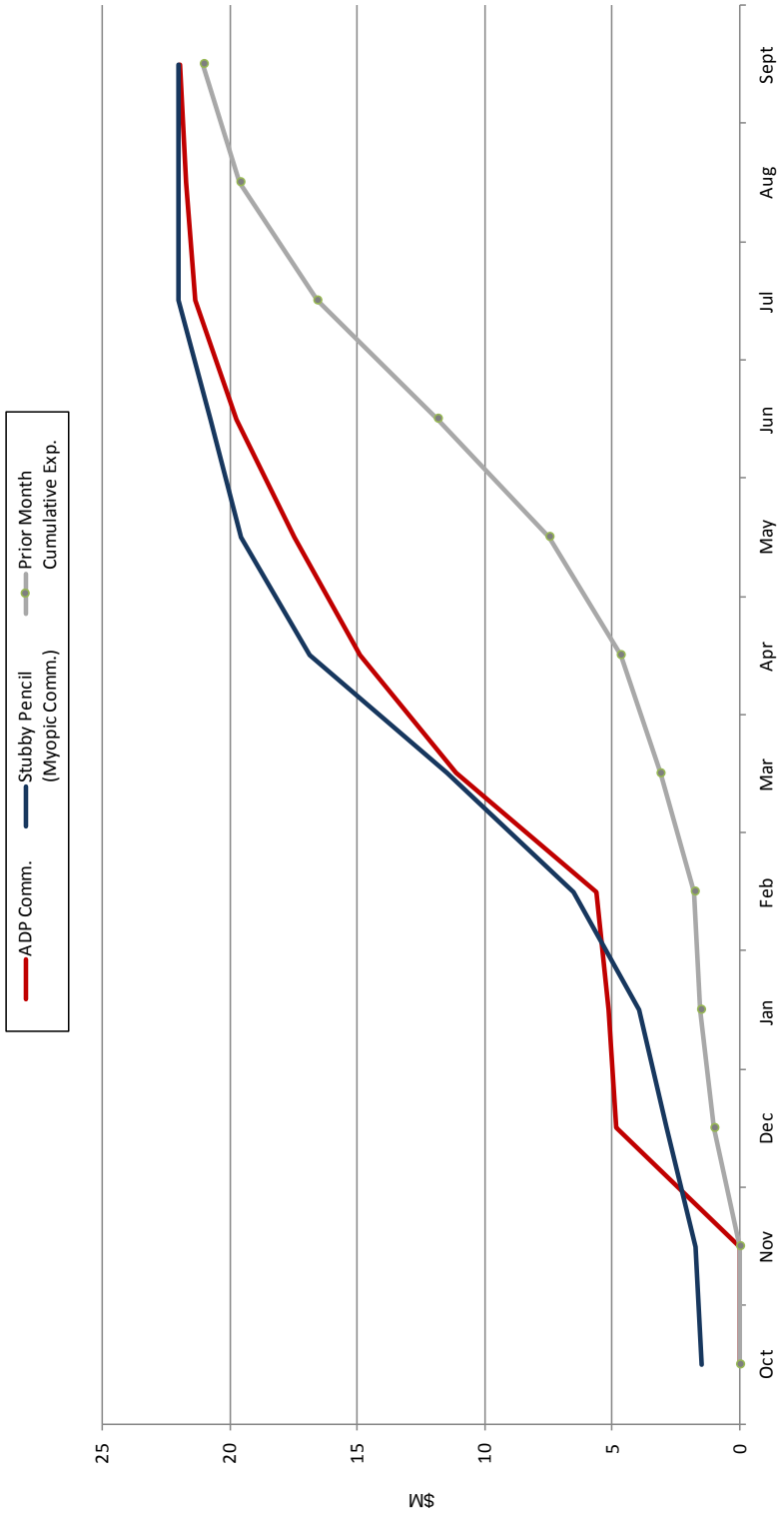


Figure 27: Test Case #2 – Trial #3 Output Graphic 2

4.9 Learnt Phase Sensitivity Analysis

The learnt phase of the model contains unique sensitivity toggles that allow an analyst to consider how the ADP and stubby pencil commitment choices react under various execution modeling excursions. The sensitivity toggles are designed to alter the manner in which expenditures occur each month throughout the modeling horizon. The variable 'exog', short for exogenous information, was used in the Matlab code to establish the type of expenditure randomness used during that particular learnt phase excursion. The various settings for the 'exog' variable are 'all', 'high', 'low', or 'mix'. The setting for the 'exog' variable establishes the nature of how the plus-minus factor for each project is used to simulate the monthly actual expenditures.

The 'all' setting is the default setting. In this case, each month's actual expenditures occur as a random uniform variable in which the predicted amount for a given month is the mean. While using the 'all' setting, the minimum and maximum values for the distribution are a positive or negative plus-minus value either above or below the anticipated mean. The 'all' setting establishes that all values within this range are equally-likely to occur as the expended amount for the month. The 'high' setting throws out the bottom half of the uniform distribution such that the new minimum value of the distribution is the original mean while the maximum possible value remains the same. Under the 'high' setting only the high values of the original distribution are equally-likely to occur as the actual expenditure for the month. The 'low' setting flips the distribution such that the maximum possible value is now the original mean, while the minimum value is the same as it was under the 'all' setting. In contrast, under the 'low'

setting only the low values of the original distribution are equally-likely to occur as the actual expenditure for the month.

The last possible sensitivity setting for the 'exog' parameter is the 'mix' setting. The 'mix' setting switches the exogenous uncertainty or randomness of the actual expenditure variable at a point halfway through the simulated time period. For each month from t = 1 to t = T/2, the 'exog' assignment will either be 'high' or 'low', then for each month after t = T/2 up to t = T the 'exog' assignment will switch over to the opposite setting either 'low' or 'high'. When the 'mix' setting is in use, another Matlab variable called 'fiftyfifty' is set to either 'LowToHigh' or 'HighToLow'. The 'fiftyfifty' designation determines whether the 'exog' variable shifts from 'low' to 'high' or from 'high' to 'low' during each simulation run n.

The different 'exog' settings provide the ADP model an opportunity to emulate the various real world expenditure behaviors that may occur over an observed time period or spending horizon. Essentially, the model now mimics four possible expenditure scenarios that could occur. One scenario is when contractors are consistently overrunning and expending funds at a pace faster than anticipated. The second scenario is when the contractor is consistently underrunning and expending funding at a pace slower than anticipated. The third scenario is when the contractor starts slow and is initially underrunning then accelerates spending to a point where they are later overrunning and expending dollars at a much faster rate. The fourth and final alternative is when the contractor starts out hot and is initially overrunning and then must pull back later to expend dollars at a much slower rate.

The following Figure 28 provides a snapshot of the Matlab input screen for the learnt phase of the ADP model. The inputs include the same original project parameter requirements used for the exploration and learning phases of the model. Additionally, the example snapshot shows the assignments of the 'exog' and 'fiftyfifty' variable. Lastly, the final read command loads the 'look-up' table data containing the value function estimates that were produced during exploration and learning.

```
%% Projects Parameters %%
proj = 3;                              % Number of projects in model
budget = [ 5.000   2.000   15.000 ];   % Maximum allowable budget for each individual project
incr = [ 0.500   0.250   1.000 ];      % The incremental allocation parameter for each project
months = 12;                           % Number of Planning Months
PlusMinus = [ 1.000   0.750   2.000 ]; % Plus-Minus factor to be used in the exogenous information sub-routine

%% Simulation Parameters %%
exog = 'all';   % Inputs are 'mix','high', 'low', or 'all' -->
                % Allows the expenditures to be cover either 'high', 'low', or cover 'all' possibilities

fiftyfifty = 'Off'; % Inputs are 'LowToHigh', HighToLow', 'Off' -->
                    % Switches the exog shocks half-way through cycle from either
                    % low-to-high or from high-to-low
learntite = 100;
ReadWriteTab = 'Learning-MinV-TC2Trial4';

%% Chart Parameters
TestCase = 1;   %TestCase Number and Trial Number for chart Titles
Trial = 2;
startDate = datenum('10-01-2011');
endDate = datenum('09-30-2012');
xData = linspace(startDate,endDate,months);

%% Load Incremental Planning Data - Matrix A
Z = zeros(months,2,proj);
Z(:,:,1) = xlsread('InputFileMultiProjects.xlsx','MatrixA','AC67:AD78');
Z(:,:,2) = xlsread('InputFileMultiProjects.xlsx','MatrixA','AF67:AG78');
Z(:,:,3) = xlsread('InputFileMultiProjects.xlsx','MatrixA','AI67:AJ78');

% Load the PDS states and associated Q-Matrix/value function approximate learned values
Learnt = xlsread('V-LearningOutputMultiProjects.xlsx', ReadWriteTab ,'A2:C4006');
RowLearnt = length(Learnt);
```

Figure 28: Learnt Phase Matlab Inputs

4.10 Learnt Phase Observations from Test Case #3 – Trial #2

An alternative approach to conducting sensitivity analysis during the learnt phase is to consider the impact from adjusting the initial phase of the predicted expenditures for each of the projects. This type of sensitivity drill keeps static the number of projects, the

budget size of each project, and the number of months simulated. The changes are to the phasing of each project's initial planned expenditures and consequently planned monthly commitment actions with the intent to observe how these changes impact the separate ADP and stubby pencil policy recommendations. The following Test Case #3 – Trial #2 results consider this type of sensitivity drill. This exercise takes the same three projects from the earlier Test Case #2 – Trial #1 and Test Case #2 – Trial #3 scenarios and alters the original expenditure phasing. The updated expenditure planning figures are deliberately set to avoid the situation where the aggregate expenditures for the three projects have an initial spike in the early months, then taper off only to increase quickly in the later months. Figure 29 provides the input parameters for this modeling scenario along with the MSE statistics that resulted from using a 5,000 exploration iteration setting and a 75,000 exploitation iteration setting.

In addition to examining the impact from re-phasing the original expenditures, the Test Case #3 – Trial #2 scenario was used as a basis to test the reaction of the model to the various built in sensitivity toggles that are part of the learnt phase of the model. The sequence of charts shown from Figure 30 through Figure 39 provides the standard two pictorial graphics for each sensitivity drill considered. In all, fives separate excursions were performed. These included 1) the default case, 2) exogenous information 'exog' set to high, 3) exogenous information 'exog' set to low, 4) exogenous information 'exog' set to 'mix' with the 'fiftyfifty' variable set to 'LowToHigh', and 5) exogenous information 'exog' set to 'mix' with the 'fiftyfifty' variable set to 'HighTo Low'.

Modeling Input Parameters	Funding Level	Allocation Parameter	PlusMinus
Project #1	5.000	0.500	1.000
Project #2	2.000	0.250	0.750
Project #3	15.000	1.000	2.000

Number of state-spaces: 4,005
Number of months: 12
Total Budget = $22.000M
5,000 iterations Exploration
75,000 iterations Learning

RunTime: 17hrs 31Minutes
2.33 GHz Intel® Xeon
3.00 GB RAM

Load Incremental Planning Data - Array A

t	month	Project #1 Comm	Project #1 Exp	Project #2 Comm	Project #2 Exp	Project #3 Comm	Project #3 Exp
1	Oct	0.000	0.000	1.000	0.000	1.000	0.000
2	Nov	0.500	0.000	0.500	0.500	1.000	0.000
3	Dec	0.500	0.000	0.250	0.500	1.000	1.000
4	Jan	0.500	0.500	0.250	0.500	2.000	1.000
5	Feb	1.000	0.500	0.000	0.250	2.000	1.000
6	Mar	1.000	0.500	0.000	0.250	3.000	2.000
7	Apr	1.000	1.000	0.000	0.000	3.000	2.000
8	May	0.500	1.000	0.000	0.000	2.000	3.000
9	Jun	0.000	1.000	0.000	0.000	0.000	3.000
10	July	0.000	0.500	0.000	0.000	0.000	2.000
11	Aug	0.000	0.000	0.000	0.000	0.000	0.000
12	Sept	0.000	0.000	0.000	0.000	0.000	0.000
		5.000	5.000	2.000	2.000	15.000	15.000

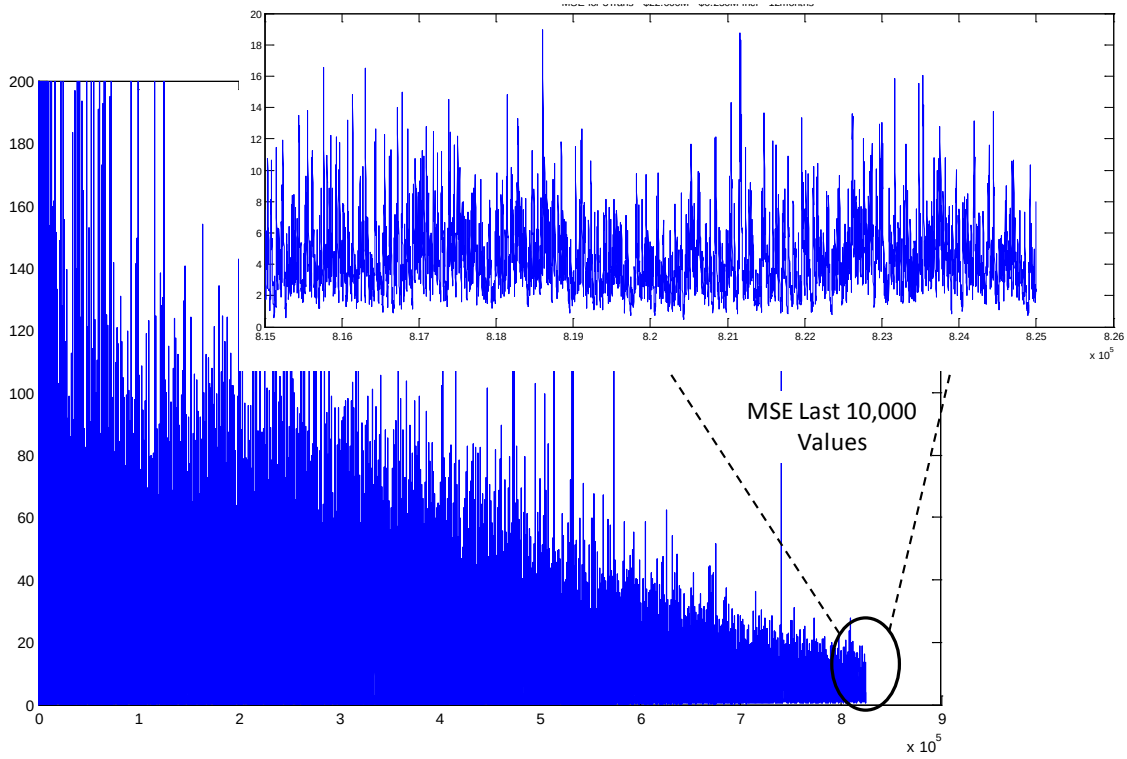


Figure 29: Test Case #3 – Trial #2 Input Parameters and MSE

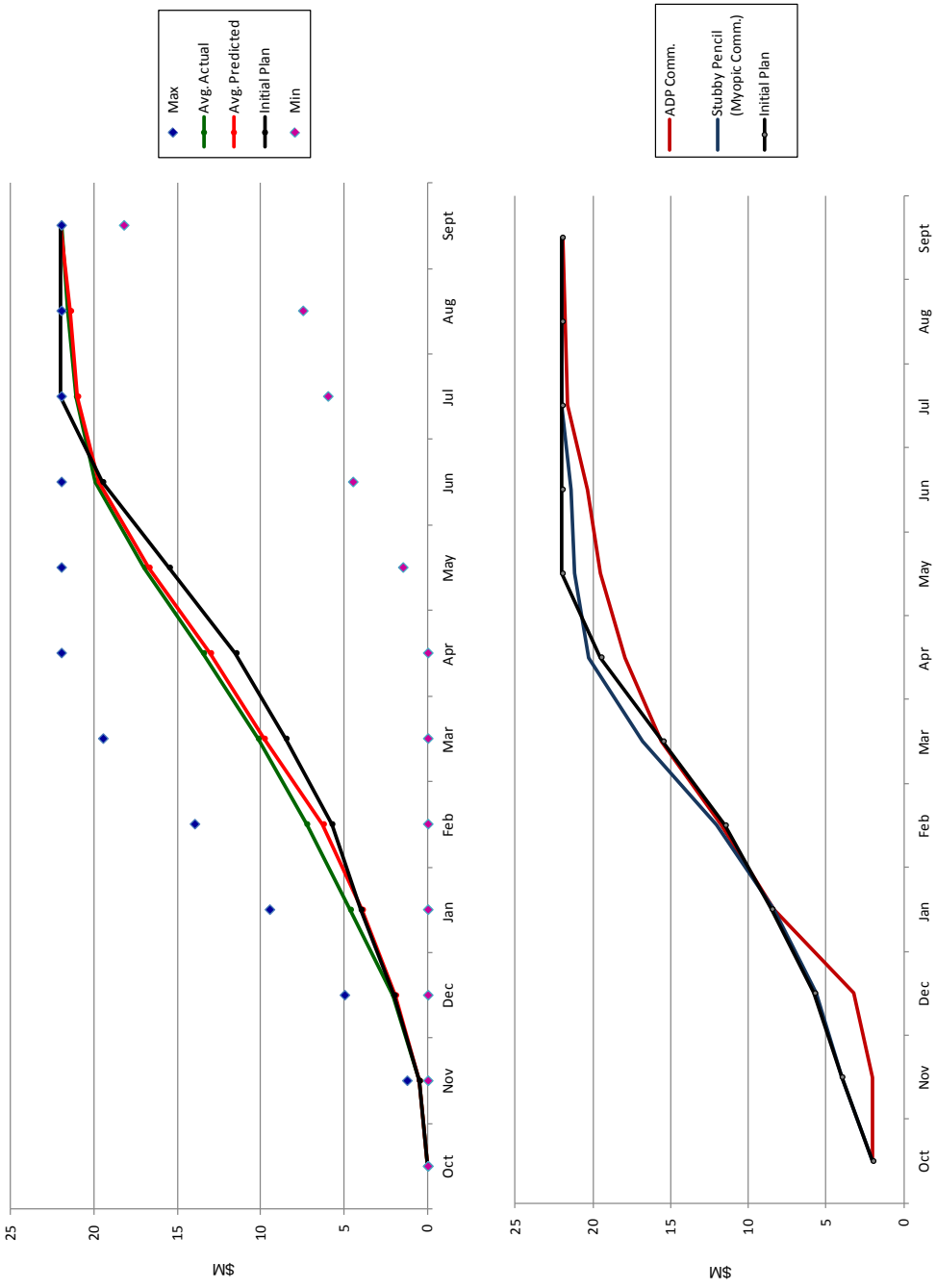


Figure 30: Test Case #3 – Trial #2 Output Graphic 1 / 'exog' = All

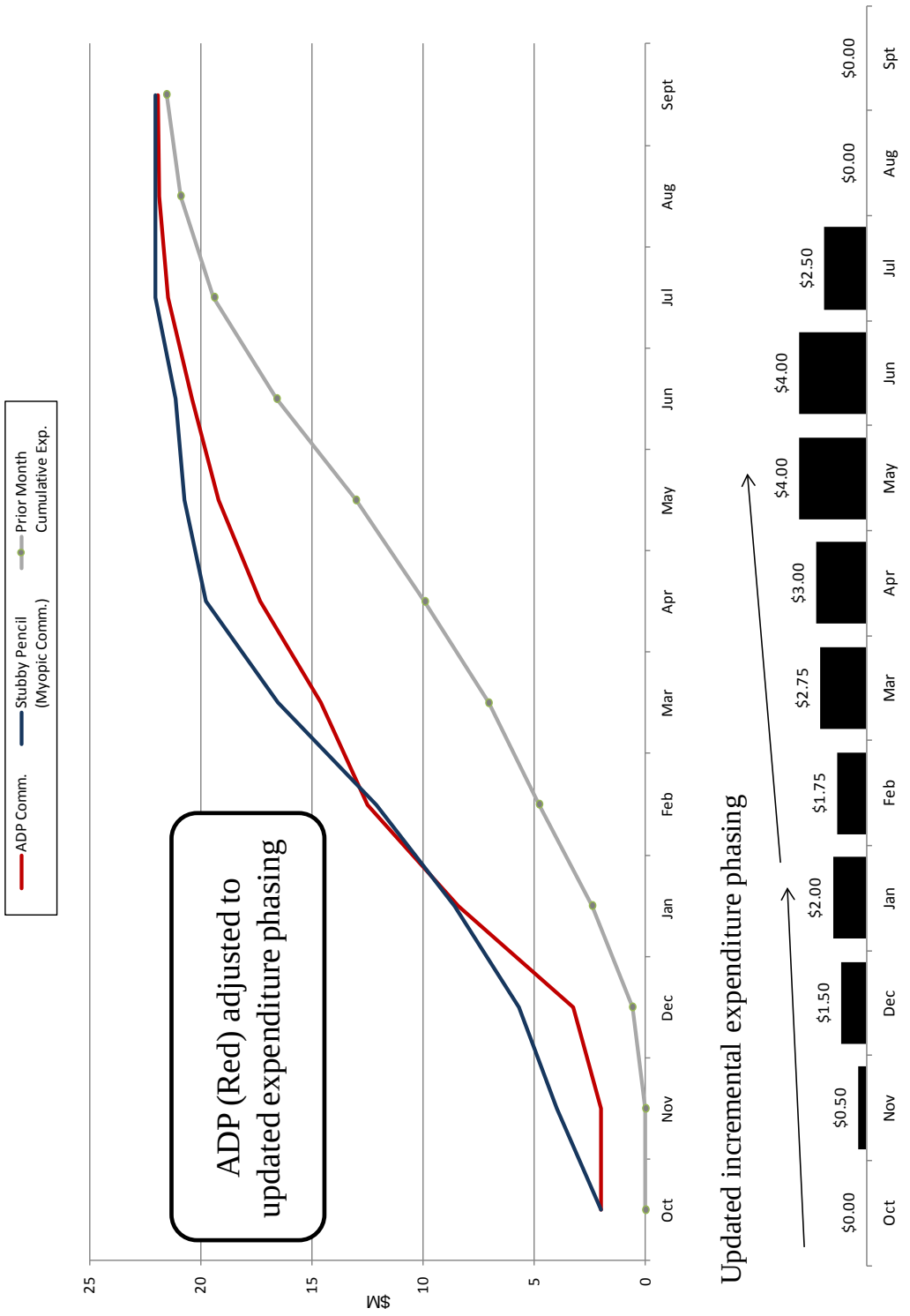


Figure 31: Test Case #3 – Trial #2 Output Graphic 2 / 'exog' = All

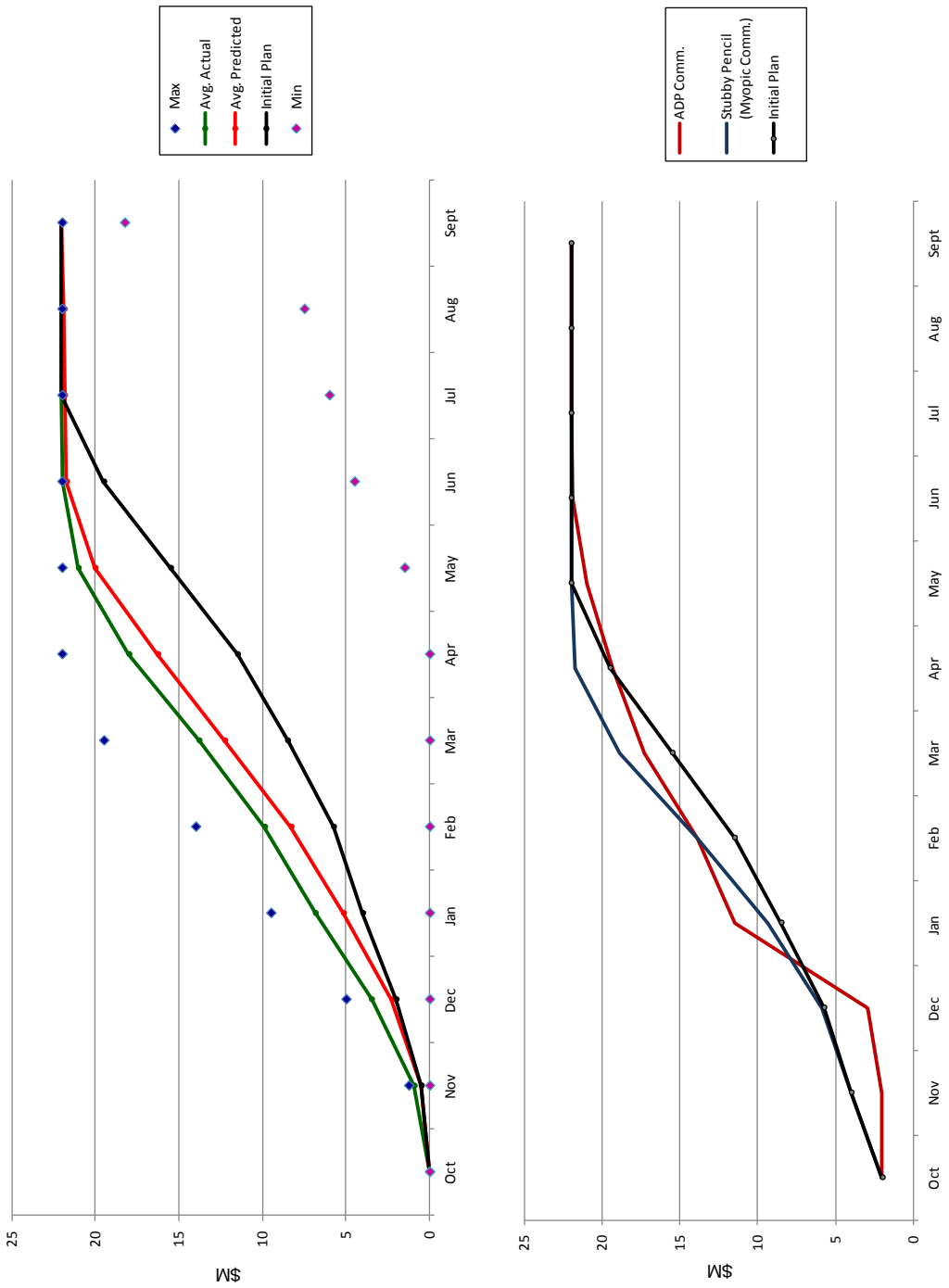


Figure 32: Test Case #3 – Trial #2 Output Graphic 1 / 'exog' = High

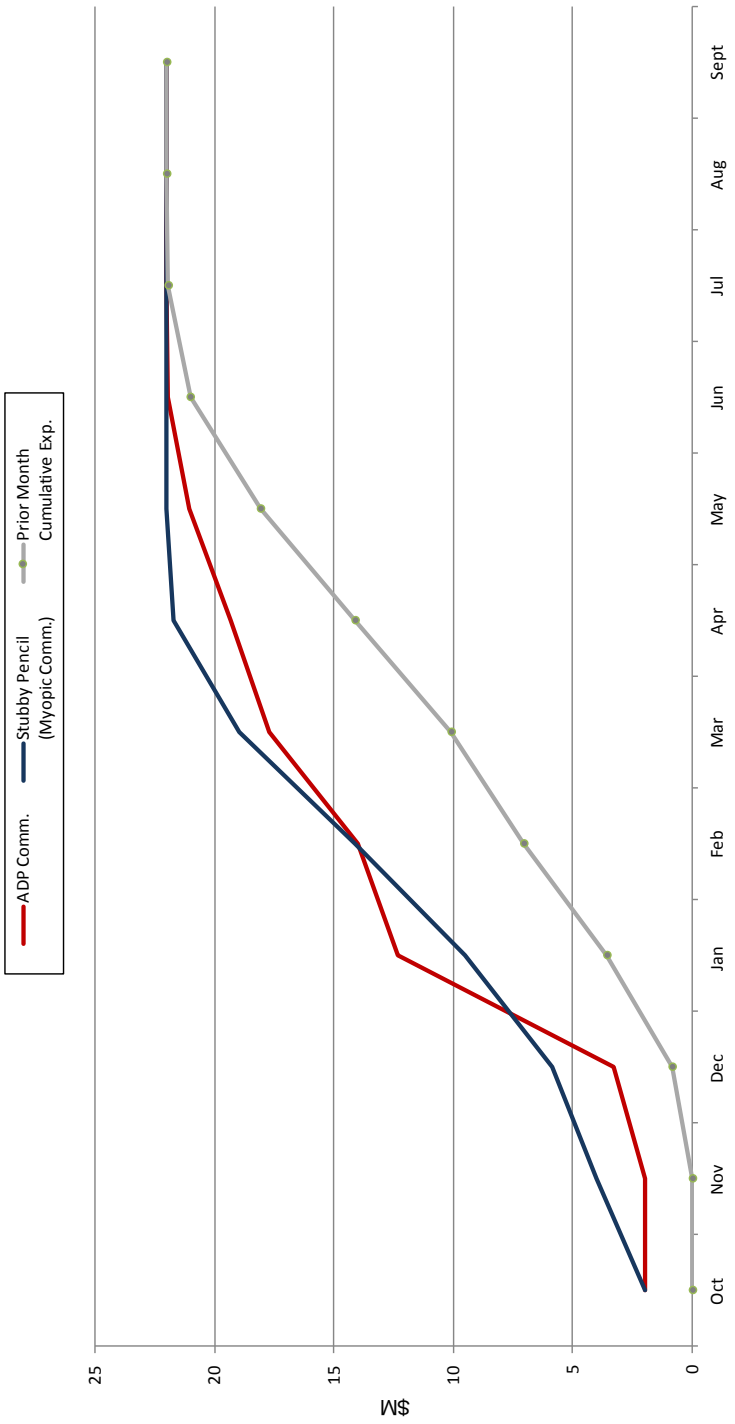


Figure 33: Test Case #3 – Trial #2 Output Graphic 2 / 'exog' = High

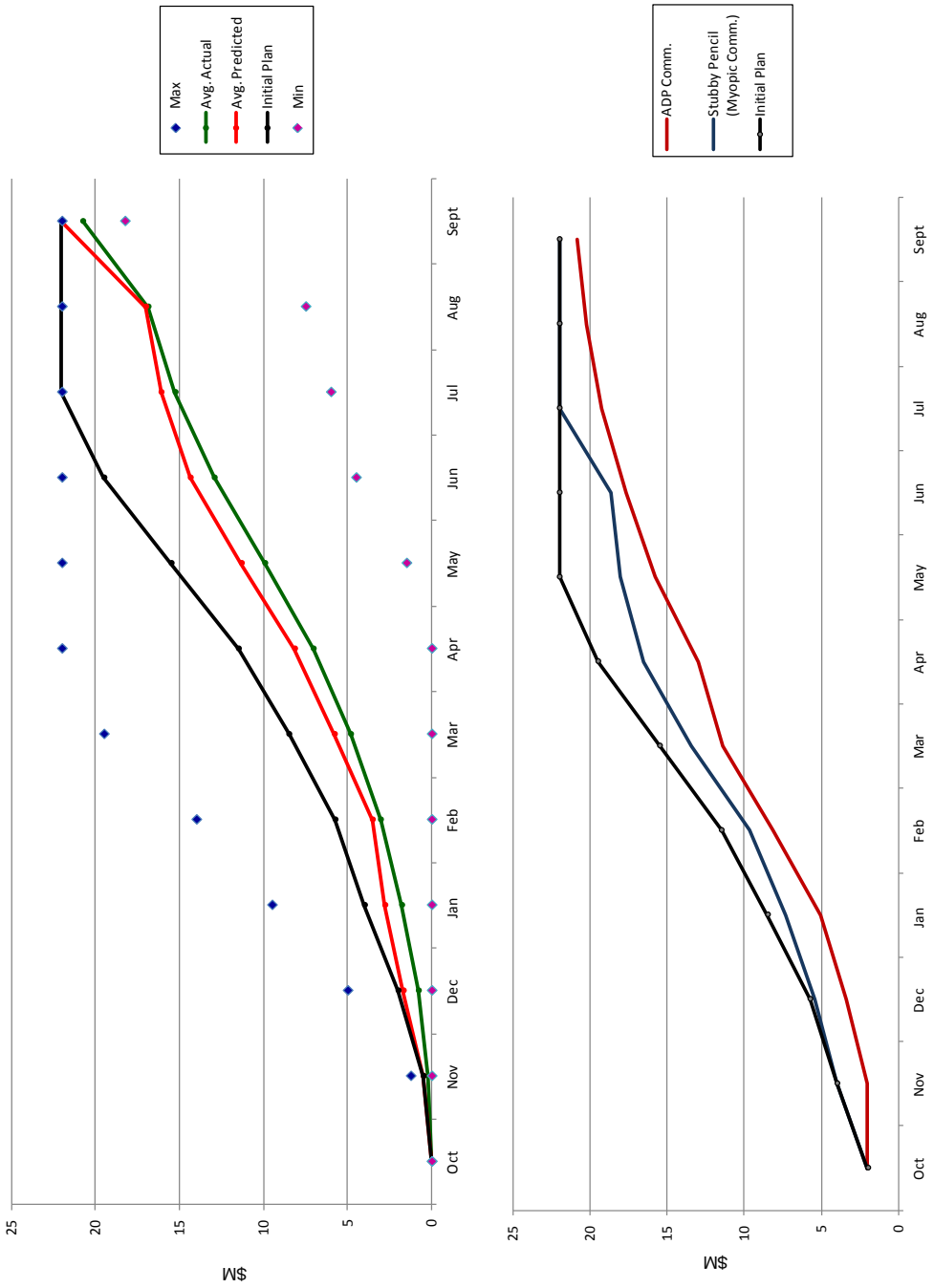


Figure 34: Test Case #3 – Trial #2 Output Graphic 1 / 'exog' = Low

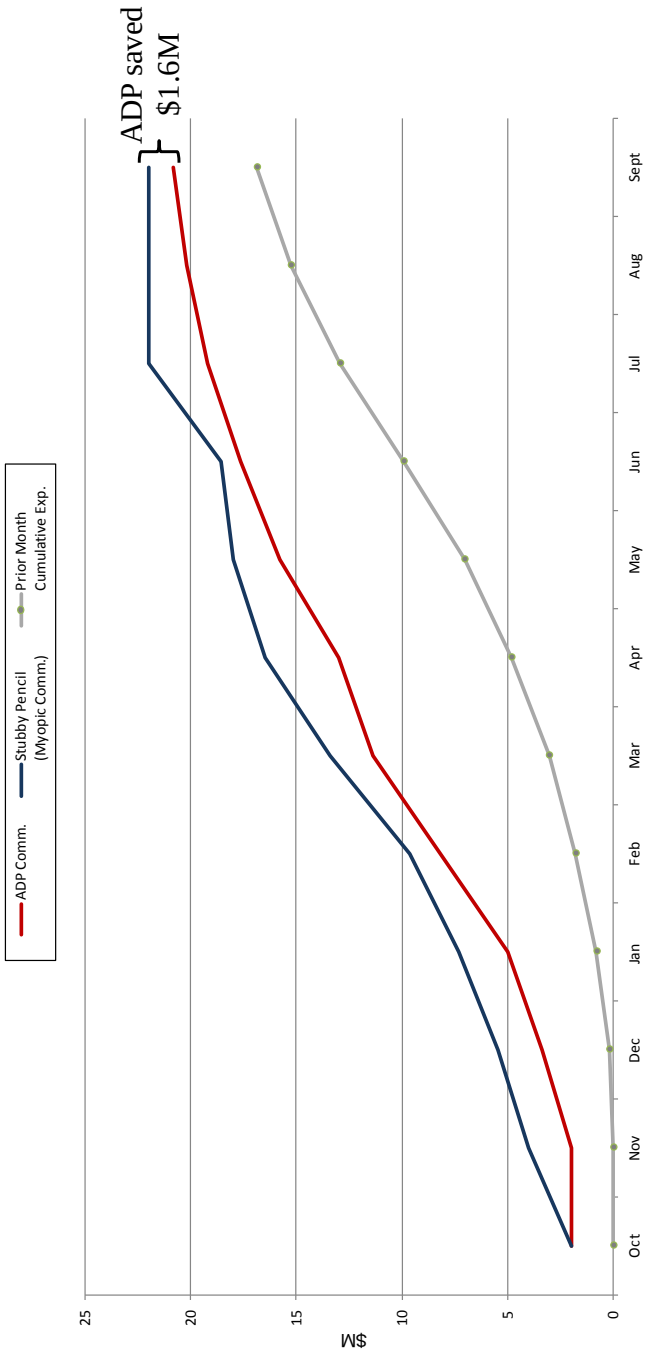


Figure 35: Test Case #3 – Trial #2 Output Graphic 2 / 'exog' = Low

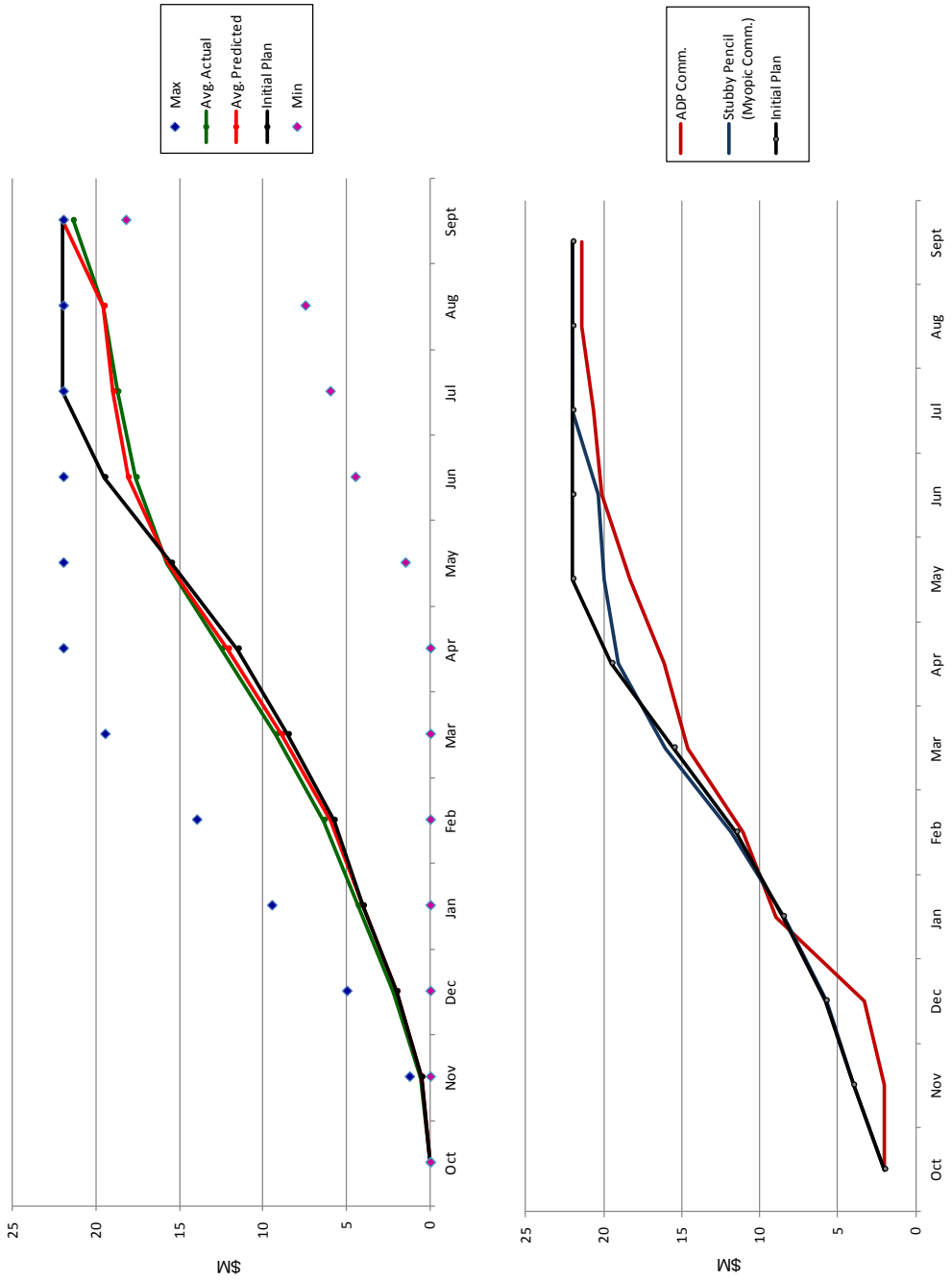


Figure 36: Test Case #3 – Trial #2 Output Graphic 1 / 'exog' = Low to High

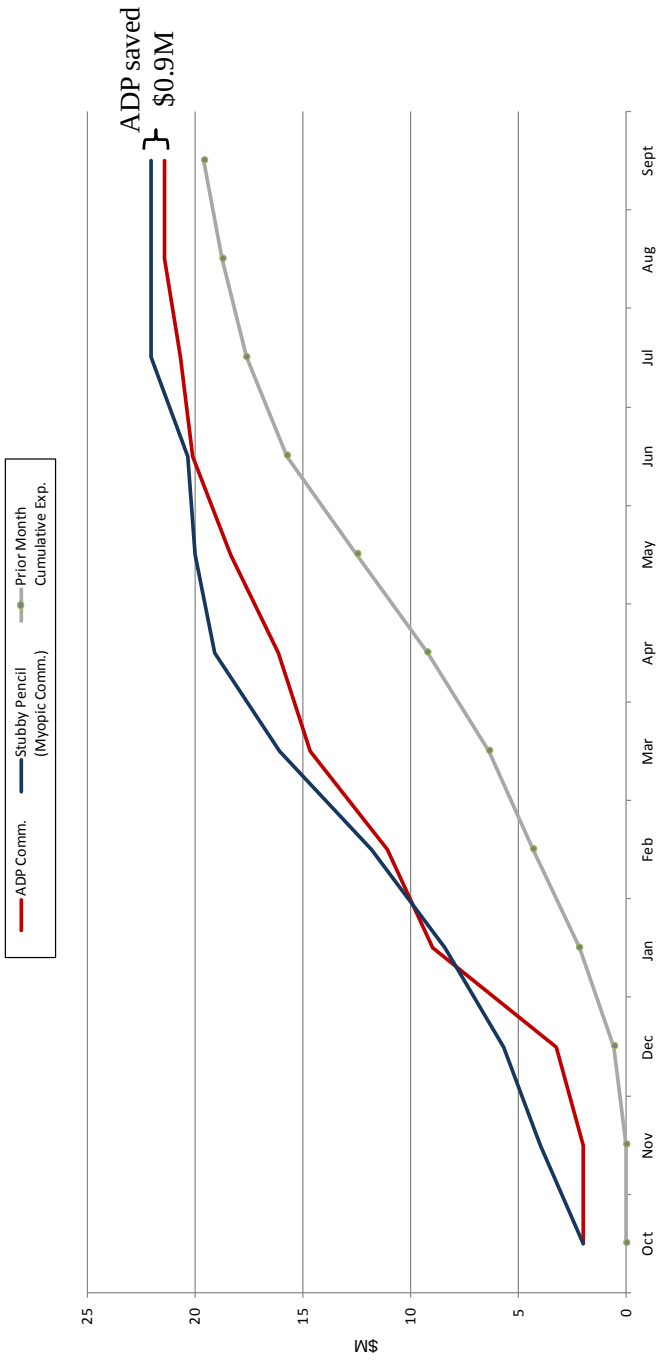


Figure 37: Test Case #3 – Trial #2 Output Graphic 2 / 'exog' = Low to High

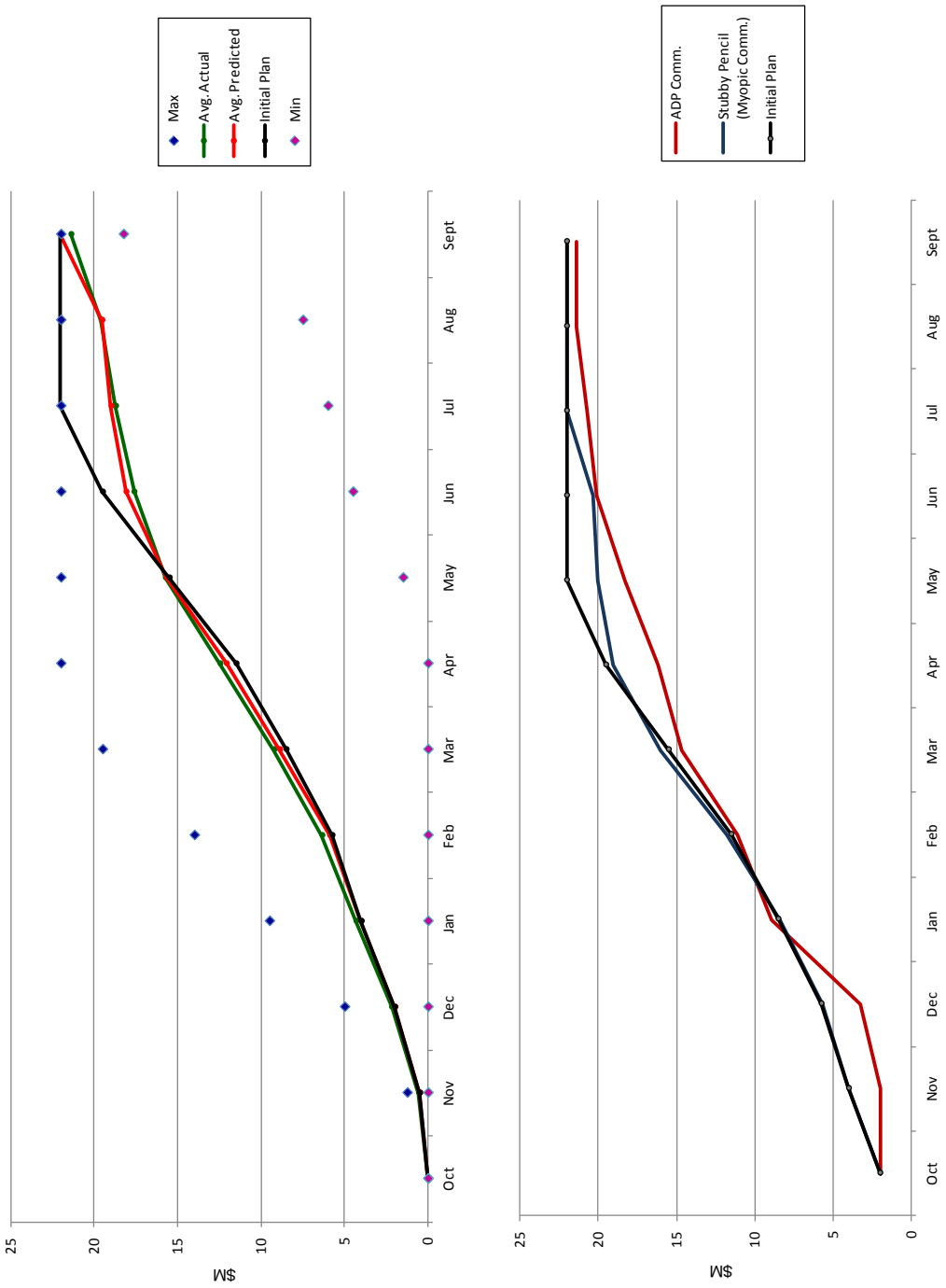


Figure 38: Test Case #3 – Trial #2 Output Graphic 1 / 'exog' = High to Low

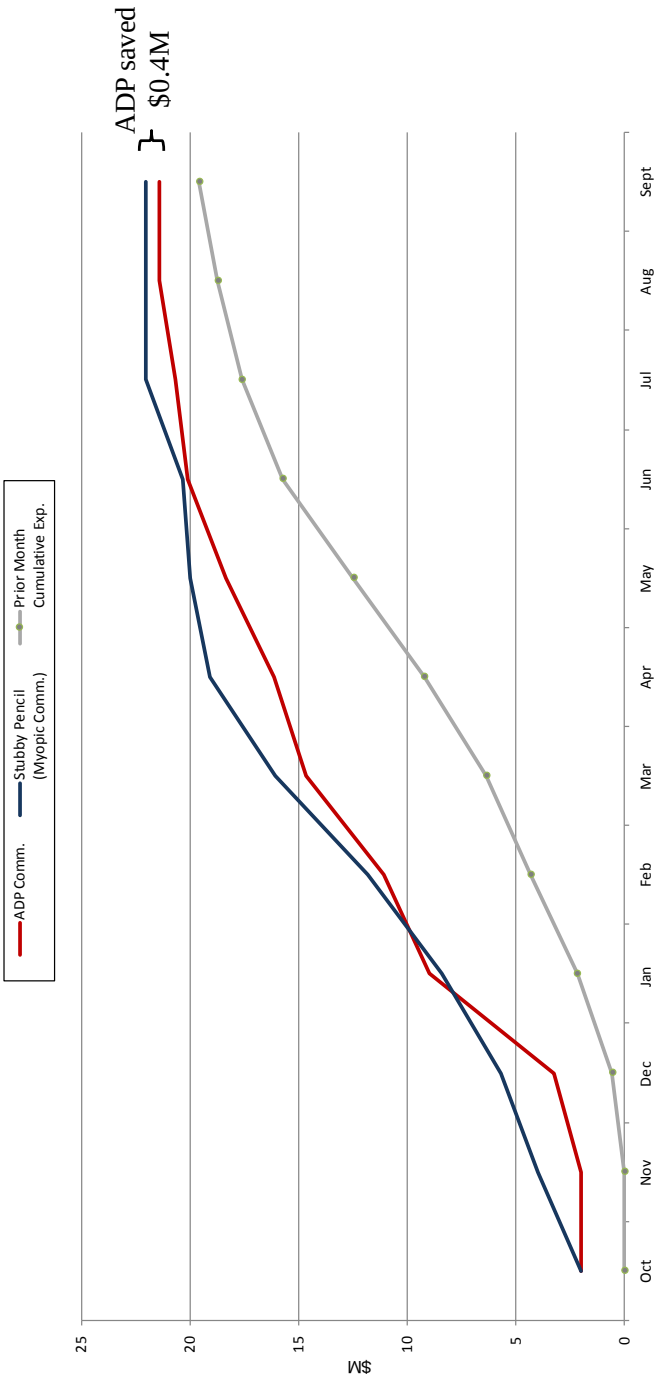


Figure 39: Test Case #3 – Trial #2 Output Graphic 2 / 'exog' = High to Low

The various sensitivity analysis drills were conducted to help isolate a narrative on either the potential advantage offered by the ADP approach or what might be missed by using a purely stubby pencil approach for making commitment action determinations. The sensitivity analysis drills were centered on the value function estimates produced using the updated expenditure and commitment planning profiles in the Test Case #3 – Trial #2 scenario. Each drill examined the average results from 100 iterations of learnt phase simulations. A total of five possible sensitivity analysis scenarios were considered.

The first execution sensitivity analysis drill used the default setting to calculate and determine the exogenous information or actual expenditures for each month or simulated time period t. Figure 30 and Figure 31 provide the graphic pictorials associated with this excursion. The results show both similarities and differences to those provided in Test Case #2 – Trial #1 and Test Case #2 – Trial #3. As shown in the top box in Figure 30, there is once again little observed variability between the average actual expenditures, average predicted expenditures, and the initial expenditure plan. Also, as shown in the bottom box in Figure 30 and in the graphic shown in Figure 31, the ADP commitment strategy is consistently below that of the recommended stubby pencil strategy. Furthermore, one observes that the ADP commitment policy no longer contains those initial up and down spikes that occurred during the Test Case #2 – Trial #1 scenario. Again, for comparison purposes a bar chart of the original incremental planned monthly expenditures is provided at the bottom of Figure 31. Upon closer examination, it appears that the recommended ADP commitment policy has in fact adjusted accordingly to a smoother per month expenditure profile. This observation is consistent with the

theory that the ADP commitment perturbations associated with Test Case #2 – Trial #1were correlated to the initial expenditure planning profile and the fact that there was noticeable up and down incremental phasing of the planned expenditures during the early months of the modeling scenario. The final observation, shown in Figure 31, is that at no point did either the ADP or stubby pencil commitment policy move below the actual expenditures that had occurred up to one month prior to the current commitment decision point.

The second sensitivity analysis drill examined the results of setting the randomness of the actual expenditures or the exogenous variable to 'high'. The graphic pictorials for this scenario are provided in Figure 32 and Figure 33. As expected, the top graph of Figure 32 now shows a wider discrepancy between the average actual expenditures, average predicted expenditures, and initial plan. With an actual expenditure setting of 'high', both the average predicted expenditures and the average actual expenditures are above the initial expenditure plan. Once again, with the exception of January, the ADP commitment policy was consistently below that of the stubby pencil commitment policy. Additionally, as to be expected with 'high' expenditures both the ADP and stubby pencil approaches committed the full $22M at a point much earlier in time as compared with the default 'all' sensitivity analysis excursion.

The third drill examined the behavior of the model when the 'exog' variable was set to 'low'. This time as shown in the top box in Figure 34, the average actual expenditures and average predicted expenditures have shifted below the initial planning

line. The ADP-recommended commitment policy still remained below that of the stubby pencil policy. Furthermore, a significant observation is that in this case the stubby pencil commitment policy approach reached the full $22M commitment policy during the month of July while the ADP policy never committed the full $22M. Figure 35 highlights the potential savings obtained by following an ADP commitment strategy while operating in a 'low' expenditure environment. These ADP savings represent additional work opportunities or projects that a decision maker could incorporate into the weapon system program.

The fourth excursion evaluated impacts to the ADP and stubby pencil policies based on using a 'mix' of simulated actual expenditures that initially started low and moved to high. Figure 36 and Figure 37 show the two pictorial graphics associated with this learnt phase execution drill. In this case, the ADP commitment policy still remained beneath that of the stubby pencil commitment policy. Additionally, the ADP approach never committed the full $22M whereas the stubby pencil approach was completely committed by July. Figure 37 highlights the savings obtained by following an ADP commitment strategy while operating in an expenditure environment that moved from 'low' to 'high'.

The fifth and final excursion evaluated the impacts of switching the mix simulation of actual expenditures to 'high' to 'low'. The final two graphics, Figure 38 and Figure 39 provide the results for this last excursion scenario. As is consistent with the other drills the ADP approach committed funding at a slower pace as compared with

the stubby pencil approach. Once again, the ADP approach did not commit the full $22M by year's end and the realized savings are captured in Figure 39.

After evaluating the five sensitivity analysis cases, there are some critical observations that seem to suggest general ideas or inferences on the tendencies of commitment actions. In all five cases, the ADP model consistently suggested a commitment allocation plan that was more conservative than the stubby pencil recommended commitment plan. Figure 40 shows the end of year commitment levels of both the stubby pencil and ADP policies for all of the five test case scenarios. In each of the stubby pencil cases the recommended strategy was to commitment the full $22M prior to the last month T. However, in three of the five cases using the ADP approach the model recommended not committing the full $22M prior to the end of the FY. In regards to these three cases, noted in Figure 40 are the ADP-reported savings.

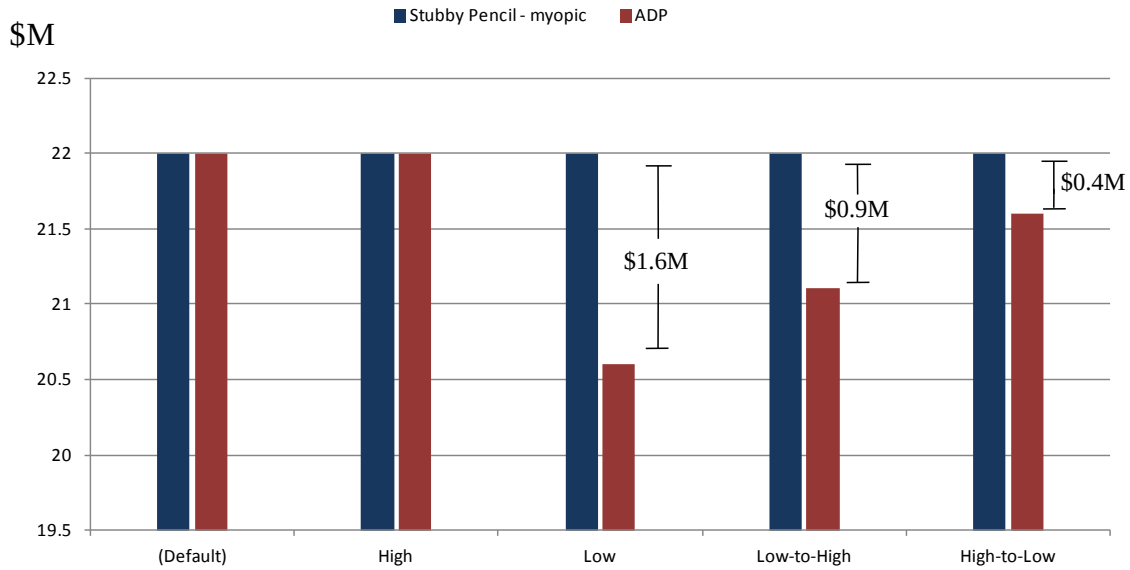


Figure 40: End of Fiscal Year Commitment Amounts

An issue for the decision maker is to determine a point in the fiscal year in which the anticipated ADP savings can be allocated to new work initiatives. Figure 41 provides a snapshot status of the end of month May commitment levels for all five Test Case #3 – Trial #2 sensitivity analysis scenarios. Although the myopic stubby pencil policy was able to adjust its' commitment strategy from the initial plan during the various sensitivity scenarios, Figure 41 however shows that for the three cases in which ADP saved dollars the stubby pencil policy already committed more funding in May than what the projects will expend by the end of the fiscal year. In contrast, for these same scenarios the ADP commitment levels in May are beneath the end of fiscal year expenditure levels indicating that there is budget available to incorporate additional work.

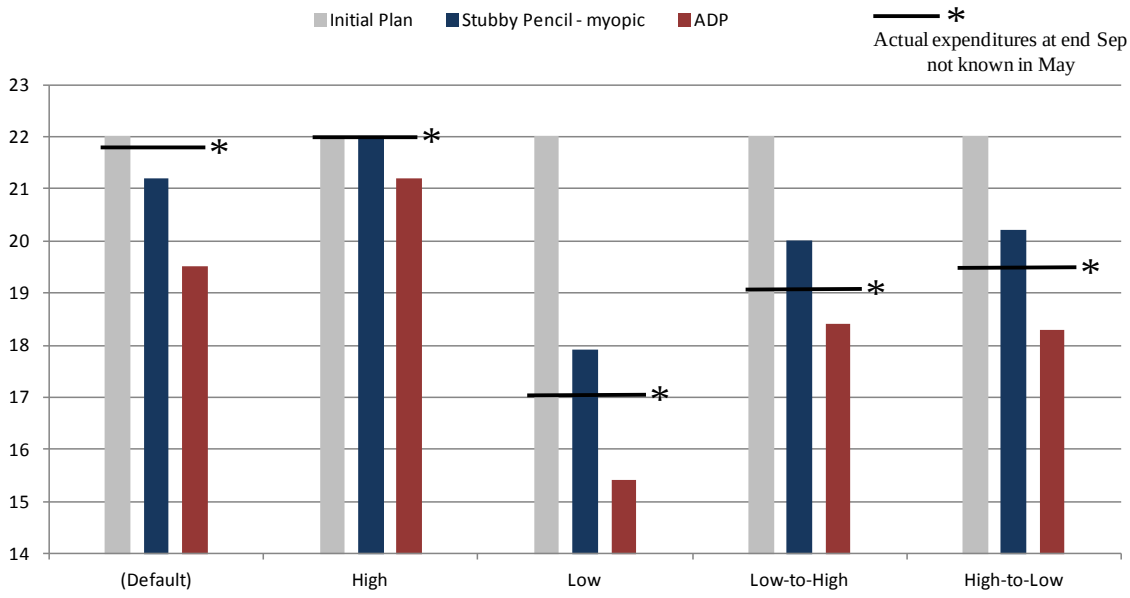


Figure 41: End of Month May Commitment Levels

4.11 Additional Test Cases and Analysis

The development of the two ADP models involved the analysis of numerous test case scenarios. Table 6 and Table 7 provide a summary list of various test cases examined. The summary indicates whether a test case used Q-learning or value function learning. Also, the tables provide additional test case information including number of projects, total budget, number of iterations, MSE observations, size of state-space, model run-times, and the properties for each of the individual projects.

The Q-learning test cases provided confirmation that the execution simulation design was in fact visiting only viable action and state-space combinations. For these five test cases, the Q-matrix output resulted in a data pattern showing that the model was not selecting any erroneous action possibilities. However, one observation was the increased sparsity in the Q-matrix given the larger state-space test case scenarios. The addition of exploration iterations will likely reduce some of the sparsity. However, given the current ADP design this may cause the ADP model to visit some bad action and state-space combinations. As a result, the model will likely require more learning iterations before converging. A second observation from the Q-learning test cases was the increase in run-time between the Q-learning Test Case #2 - Trial #1 and Test Case #3 - Trial #1. Although Test Case #3 - Trial #1 contained fewer state-spaces, this scenario did involve twice the number of months as well as one additional project. This seems to indicate that the number of months in a test case scenario is a significant driver of the modeling run-times.

Table 6: Test Case Summaries

#	Name:	Model Type	# of Projects:	# of Mths:	$M Total Budget:	Exp Ite:	MSE:	Alpha-Decay	# of States	Run Time	Project #1			Project #2			Project #3			Project #4		
											Budget	Alloc. Param.	+/- Factor	Budget	Alloc. Param.	+/- Factor	Budget	Alloc. Param.	+/- Factor	Budget	Alloc. Param.	+/- Factor
1	**TestCase#1** **Trial#1** Notes: Intel® Xeon® 2.33GHz, 3.00 GB RAM Q-Matrix: visited acceptable state-spaces	Q-Learning	2	12	7.000	Exp: 5,000 Lrn: 50,000	Last 10,000 observations all under 0.6	Exp: 0.8 to 0.6 Lrn: 0.6 to 0.1	435	4Hrs 34Min	5.000	0.500	1.000	2.000	0.250	0.750						
2	**TestCase#1** **Trial#2** Notes: Intel® Xeon® 2.33GHz, 3.00 GB RAM Q-Matrix: visited acceptable state-spaces; noticeable decrease in MSE Statistic from TC#1Trail #1	Q-Learning	2	12	7.000	Exp: 5,000 Lrn: 75,000	Last 10,000 observations all under 0.15	Exp: 0.8 to 0.6 Lrn: 0.6 to 0.1	435	Did Not Record	5.000	0.500	1.000	2.000	0.250	0.750						
3	**TestCase#1** **Trial#3** Notes: Intel® Xeon® 2.33GHz, 3.00 GB RAM Q-Matrix: visited acceptable state-spaces; slight increases in MSE from TC#1Trial#2	Q-Learning	2	12	7.000	Exp: 5,000 Lrn: 100,000	Last 10,000 observations all under 0.3	Exp: 0.9 to 0.6 Lrn: 0.6 to 0.1	435	Did Not Record	5.000	0.500	1.000	2.000	0.250	0.750	15.000	1.000	2.000			
4	**TestCase#2** **Trial#1** Notes: Intel® Xeon® 2.33GHz, 3.00 GB RAM Q-Matrix: visited acceptable state-spaces; Q-Matrix: some sparsity: a number viable state-spaces not visited	Q-Learning	3	12	22.000	Exp: 5,000 Lrn: 50,000	Last 10,000 observations between 2 and 25	Exp: 0.8 to 0.6 Lrn: 0.6 to 0.1	4,005	23 hrs 11Min	5.000	0.500	1.000	2.000	0.250	0.750						
5	**TestCase#3** **Trial#1** Notes: Intel® Xeon® 2.33GHz, 3.00 GB RAM Q-Matrix: visited acceptable state-spaces; Q-Matrix: some sparsity: a number of viable state-spaces not visited compared to TC#2Trial#1: fewer state-spaces, doubled number of months - - run-time nearly doubled	Q-Learning	4	24	68.000	Exp: 5,000 Lrn: 50,000	Last 10,000 observations most under 5.0 some between 5.0 & 10	Exp: 0.8 to 0.6 Lrn: 0.6 to 0.1	2,415	1 day 21hrs 6Min	40.000	2.000	6.000	10.000	1.000	1.000	6.000	1.000	2.000	12.000	2.000	4.000
6	**TestCase#1** **Trial#1** Notes: Intel® Core® 2.40GHz, 6.00 GB RAM different platfrom and V-Learning appear to improve run-time	Value Learning	2	12	7.000	Exp: 5,000 Lrn: 50,000	Last 10,000 observations all under 0.6	Exp: 0.8 to 0.6 Lrn: 0.6 to 0.1	435	2hrs 15Min	5.000	0.500	1.000	2.000	0.250	0.750						
7	**TestCase#1** **Trial#2** Notes: Intel® Core® 2.70GHz, 8.00 GB RAM number of months appears to impact run-times heavily	Value Learning	2	16	9.500	Exp: 5,000 Lrn: 50,000	750,000 MSE data points	Exp: 0.8 to 0.6 Lrn: 0.6 to 0.1	780	3hrs 53Min	7.000	0.500	0.500	2.500	0.250	0.250						
8	**TestCase#2** **Trial#1** Notes: Intel® Core® 2.70GHz, 8.00 GB RAM V-Learning appear to improve run-time	Value Learning	3	12	22.000	Exp: 5,000 Lrn: 50,000	Last 10,000 observations between 2 and 25	Exp: 0.8 to 0.6 Lrn: 0.6 to 0.1	4,005	11hrs 18Min	5.000	0.500	1.000	2.000	0.250	0.750	15.000	1.000	2.000			
9	**TestCase#2** **Trial#2** Notes: Intel® Xeon® 2.33GHz, 3.00 GB RAM slight improvement on MSE from TestCase#2Trial#1	Value Learning	3	12	22.000	Exp: 5,000 Lrn: 75,000	Last 10,000 observations between 1 and 20	Exp: 0.8 to 0.6 Lrn: 0.6 to 0.1	4,005	18hrs 10Min	5.000	0.500	1.000	2.000	0.250	0.750	15.000	1.000	2.000			
10	**TestCase#2** **Trial#3** Notes: Intel® Xeon® 2.33GHz, 3.00 GB RAM at start of learning, MSE statistics higher than TestCase#2Trial#2	Value Learning	3	12	22.000	Exp: 10,000 Lrn: 100,000	Last 10,000 observations between 1 and 20	Exp: 0.9 to 0.6 Lrn: 0.6 to 0.1	4,005	1 day 7hrs 10Min	5.000	0.500	1.000	2.000	0.250	0.750	15.000	1.000	2.000			

Table 7: Test Case Summaries (Cont.)

#	Name:	Model Type	# of Projects:	# of Mths:	$M Total Budget:	Exp 1e:	MSE:	Alpha-Decay	# of States	Run Time	Project #1			Project #2			Project #3			Project #4		
											Budget	Alloc. Param.	+/- Factor	Budget	Alloc. Param.	+/- Factor	Budget	Alloc. Param.	+/- Factor	Budget	Alloc. Param.	+/- Factor
	TestCase#3	Value					Last															
11	**Trial#1**	Learning	3	12	22.000	Exp: 5,000 Lrn: 50,000	10,000 observations between 2 and 25	Exp: 0.8 to 0.6 Lrn: 0.6 to 0.1	4,005	11hrs 57Min	5.000	0.500	1.000	2.000	0.250	0.750	15.000	1.000	2.000			
	Notes: Intel® Xeon® 2.33GHz, 3.00 GB RAM slight improvement on MSE from TestCase#2Trial#1 re-phasing initial commitment/expenditure planning appears to smooth ADP strategy compated with TestCase#2																					
	TestCase#3	Value					Last															
12	**Trial#2**	Learning	3	12	22.000	Exp: 5,000 Lrn: 75,000	10,000 observations between 1 and 20	Exp: 0.8 to 0.6 Lrn: 0.6 to 0.1	4,005	17hrs 31Min	5.000	0.500	1.000	2.000	0.250	0.750	15.000	1.000	2.000			
	Notes: Intel® Xeon® 2.33GHz, 3.00 GB RAM slight improvement on MSE from TestCase#3Trial#1																					
	TestCase#3	Value					Last															
13	**Trial#3**	Learning	3	12	22.000	Exp: 5,000 Lrn: 100,000	10,000 observations between 1 and 20	Exp: 0.8 to 0.6 Lrn: 0.6 to 0.1	4,005	23hrs 00Min	5.000	0.500	1.000	2.000	0.250	0.750	15.000	1.000	2.000			
	Notes: Intel® Xeon® 2.33GHz, 3.00 GB RAM no noticeable improvement on MSE from TestCase#3Trial#2																					
	TestCase#3	Value					Last															
14	**Trial#4**	Learning	3	12	22.000	Exp: 5,000 Lrn: 150,000	10,000 observations between 1 and 15	Exp: 0.8 to 0.6 Lrn: 0.6 to 0.1	4,005	1 day 10hrs 06Min	5.000	0.500	1.000	2.000	0.250	0.750	15.000	1.000	2.000			
	Notes: Intel® Xeon® 2.33GHz, 3.00 GB RAM slight improvement on MSE from TestCase#3Trial#3																					
	TestCase#3	Value					Last															
15	**Trial#5**	Learning	3	12	22.000	Exp: 5,000 Lrn: 175,000	10,000 observations between 1 and 17	Exp: 1.0 to 0.8 Lrn: 0.8 to 0.1	4,005	1 day 16hrs 27Min	5.000	0.500	1.000	2.000	0.250	0.750	15.000	1.000	2.000			
	Notes: Intel® Xeon® 2.33GHz, 3.00 GB RAM Used an alternative deterministic alpha-decay parameter Powell (2007) no noticeable improvement on MSE data compared to Trials#1-#4																					
	TestCase#3	Value					Last															
16	**Trial#6**	Learning	3	12	22.000	Exp: 5,000 Lrn: 100,000	10,000 observations between 2 and 6	Exp: 0.8 to 0.6 Lrn: 0.6 to ~0.0	4,005	15hrs 41Min	5.000	0.500	1.000	2.000	0.250	0.750	15.000	1.000	2.000			
	Notes: Intel® Core® 2.70GHz, 8.00 GB RAM deliberately timed-out alpha-decay; alpha-decay reached 0.1 at the 75,000 iteration of learning variance much tighter during final iterations - stabilized MSE statistic																					
	TestCase#4	Value					Last															
17	**Trial#1**	Learning	4	12	44.000	Exp: 5,000 Lrn: 50,000	10,000 observations between 3 and 40	Exp: 0.8 to 0.6 Lrn: 0.6 to 0.1	4,005	16hrs 30Mins	12.000	1.000	2.000	7.000	0.500	1.500	15.000	1.000	2.000	10.000	0.500	2.000
	Notes: Intel® Core® 2.70GHz, 8.00 GB RAM learnt phase results consistent w/TestCase#2 & #3																					

Once the Q-matrix results provided reassurance that the simulation code was implementing the algorithm as expected, value function learning became the primary analysis tool. The studies numbered 6 through 17 from Table 6 and Table 7 were dedicated to value function learning. An initial interesting observation was that the value function learning Test Case #1 - Trial #1 provided similar MSE results as the Q-learning Test Case #1 - Trial #1 in nearly half the time. This result is not too surprising since some algorithmic efficiency was expected due to the fact that value function learning no longer requires the update and storage of a Q-matrix. As mentioned earlier, the Test Case #2 - Trial #3 scenario doubled the number of exploration iterations used from the Test Case #2 - Trial #1 scenario. The impact appeared to be higher initial MSE statistics for Test Case #2 - Trial #3 at the start of the learning phase. One possibility for this anomaly is the inclusion of some additional bad state-spaces along with potentially good ones as a result of performing more exploration iterations. The various test case 2 trials and test case 3 trials examined the same $22M three-project scenario. However, the test case 3 trials re-phased the original commitment and expenditure planning. As highlighted earlier the re-phasing of the initial commitment and expenditure planning figures produced a smoother ADP recommended commitment policy.

One difficult challenge with using the ADP approach is determining a stopping criterion for the algorithm. The Test Case #3 - Trials #1 through #4 scenarios were attempts at determining if the modeling MSE statistics could be improved by performing more iterations of learning. The trials were conducted in hopes that the MSE statistic could reach a small and stable value indicating that the model was no longer learning and

had reached a convergence point. During these trials, the MSE statistics showed marginal improvement when learning iterations were increase from 50,000 to 75,000 iterations. However, further increases in learning iterations did not produce sizable increases in the final MSE statistics. The 100,000 and 150,000 iterations of learning conducted in Test Case #3 - Trials #3 and #4 appeared to end with nearly the same MSE values.

In an effort to assist with the MSE analysis, a new chart was constructed that tracked MSE variance. This new graphic evaluated the progression of the MSE statistics in blocks of 500 data points. For each successive block of 500, a sample average MSE and sample variance MSE statistic was plotted. Figure 42 and Figure 43 show the standard MSE and new MSE variance charts for the Test Case #3 - Trial #2 and Test Case #3 -Trial #3 scenarios.

The middle graphic shows the relationship between the sample average MSE and a sample variance for the MSE. The blue line in the middle graph provides the average MSE statistic in blocks of 500. The green and red lines are plotted at three standard deviations above and below the MSE. The last graphic adjusts the y-axis scale to provide a blowup of the blue line in the middle chart. The downward trend depicted in the bottom graphic seems to suggest that the MSE statistics are continually improving and thus the model is still continuing to learn even at the point when the simulation was stopped. However, the extra learning iterations conducted in Test Case #3- trial #3, did not appear to improve the final MSE values. The Test Case #3 - Trial #4 scenario which involved 150,000 learning iterations produced similar results.

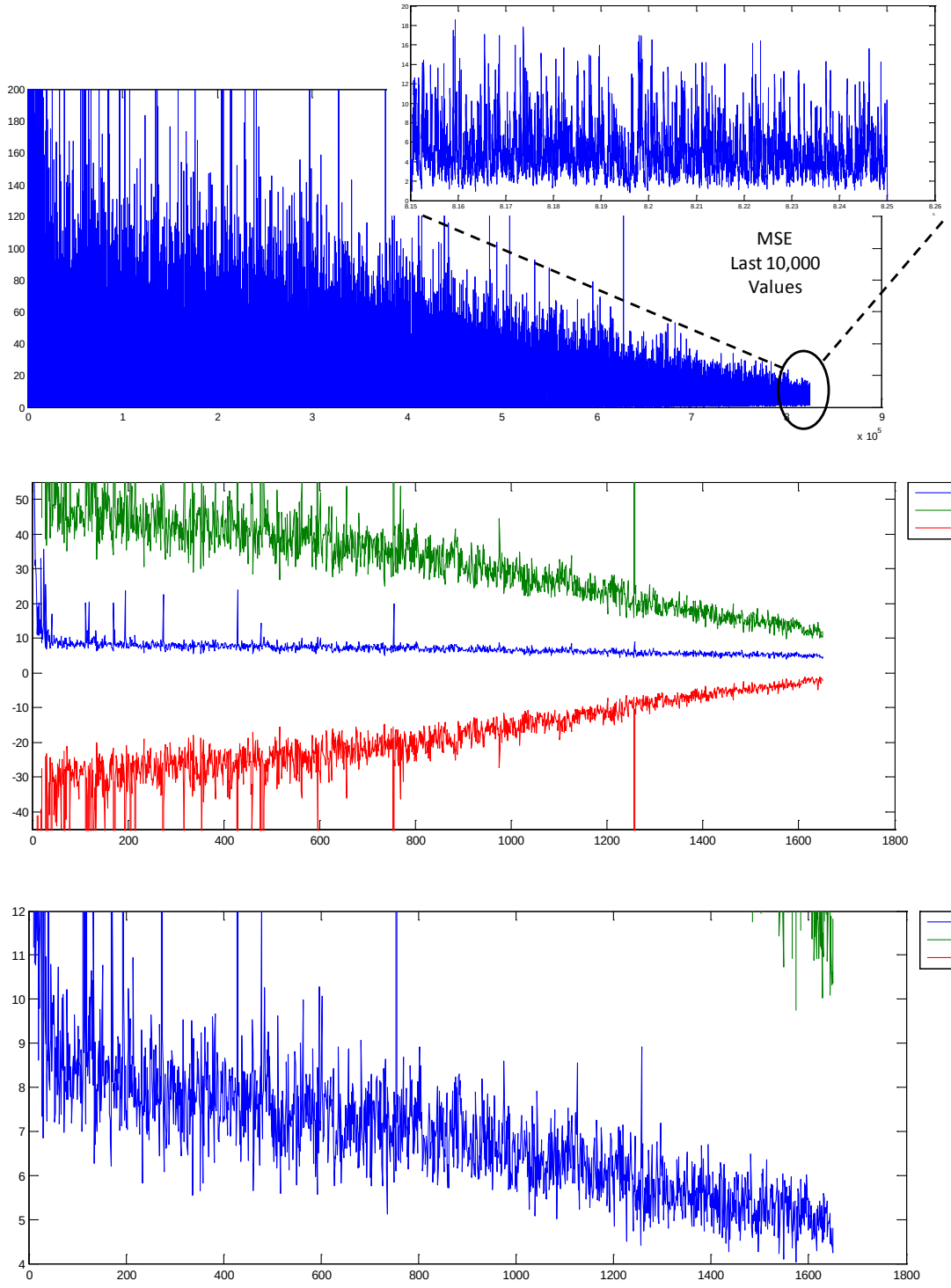


Figure 42: Test Case #3 – Trial #2 MSE and Variance Graphics

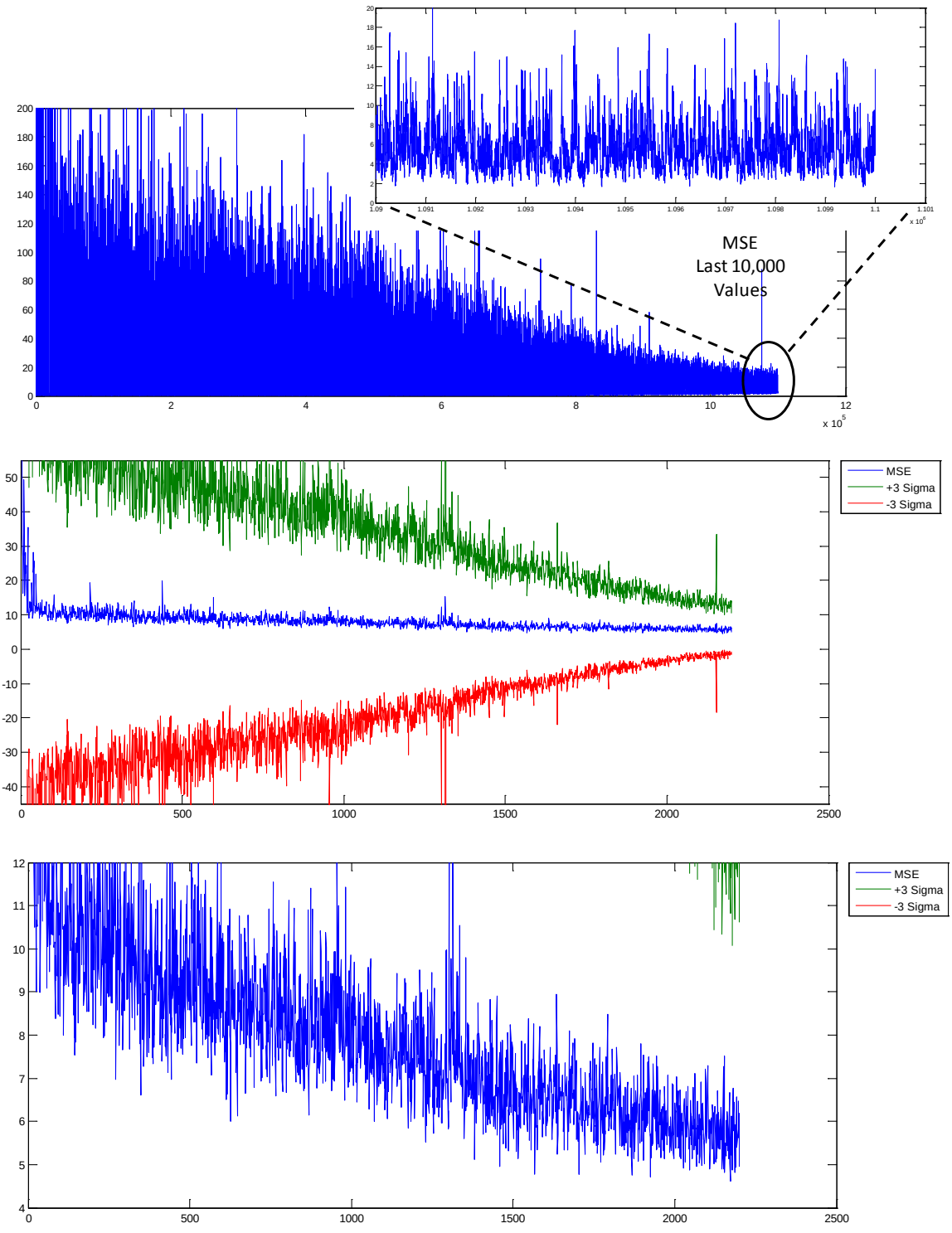


Figure 43: Test Case #3 – Trial #3 MSE and Variance Graphics

Test Case #3 - Trial #5 examined an alternative strategy which was to use a different alpha-decay design. This test case used all the same input information as Test Cases #3 – Trials #1 through #4 except for the new alpha-decay rule and for an increase in the number of learning iterations to 175,000. Figure 44 provides summary graphic information related to this test case scenario. The Test Case #3 - Trial #5 simulation used a deterministic alpha-decay process referenced in Powell (2007) and expressed by Equation 36. The top chart in Figure 44 provides the resulting alpha-decay pattern for the 5,000 iterations of exploration and the 175,000 iterations of learning. The middle graphic shows MSE statistics with the last 10,000 values highlighted. The bottom chart provides the sample MSE variance in blocks of 500 values.

Equation 36: Alternative Alpha-Decay (Powell 2007)

$$\propto_{n-1} = \propto_0 \frac{(b/n + a)}{(b/n + a + n^{\beta})}$$

Initially, the results from Test Case #3 – Trial #5 seem to exhibit promising behavior. The middle chart in Figure 44 shows a MSE that appears relatively stable for an extended amount of time. However, the final MSE values were not much improved over those obtained in trials #1 through #4. In this instance, since the step-size decreases rapidly during the early iterations, there is a danger of apparent convergence. Therefore, the concern with using this approach is that the model was unable to learn sufficiently from the collected $\hat{v}$ sample observations.

A final test case examined in this series was Test Case #3 - Trial #6. In order to avoid the dangers of apparent convergence, this scenario returned to utilizing the original alpha-decay process. However, for this trial alpha-decay was allowed to drop to a near zero value. In the earlier trials, the simulations were designed to stop once the alpha-decay process reached 0.1. When conducting test cases that involved more learning iterations, the alpha-decay process was slowed so that it would still equal approximately 0.1 during the final iteration of the simulation. Allowing alpha-decay to reach a near zero value resulted in the output graphics provided by Figure 45. The final MSE statistics during the last 10,000 iterations were bounded between two and six. Furthermore, the bottom graphic in Figure 45 shows that the average MSE for each block of 500 observations is flattening out. The combination of a steadily decreasing MSE trend that eventually flattens out is a positive sign that the simulation had ample number of iterations too learn and has reached a point where further iterations will not provide any new information.

The final scenario examined was to consider if a larger budget size would still produce a conservative ADP policy when compared to the stubby pencil policy. Test Case #4 – Trial #1 simulated a four-project, $44M dollar decision system. The problem inputs are provided in Figure 46. Output Graphics of the learnt phase under various expenditure sensitivity settings are provided in Figure 47, Figure 48, and Figure 49. In all cases the ADP commitment policy strategy was still more conservative than that of the stubby pencil policy.

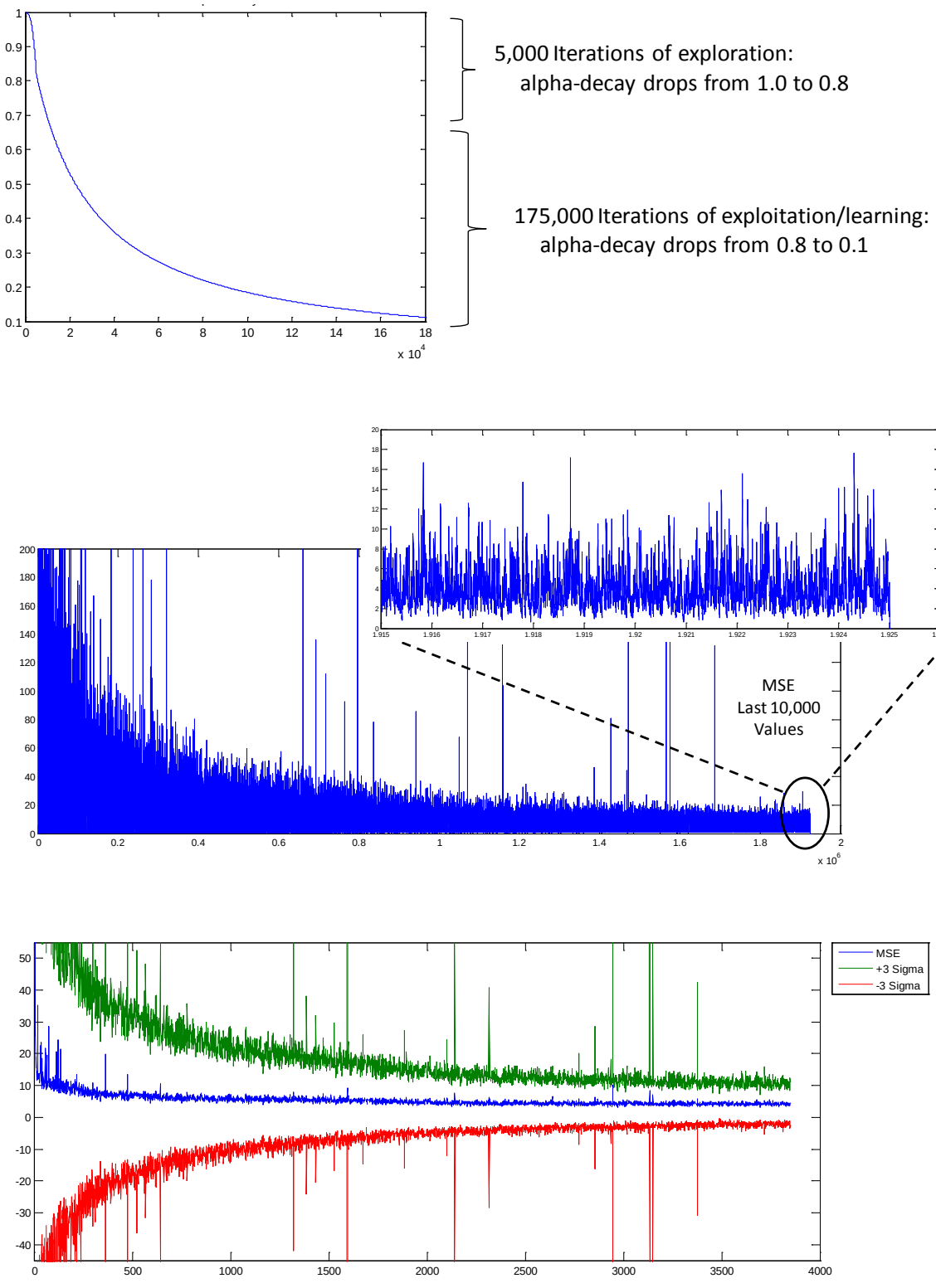


Figure 44: Test Case #3 – Trial #5 Alpha-Decay, MSE, and Variance Graphics

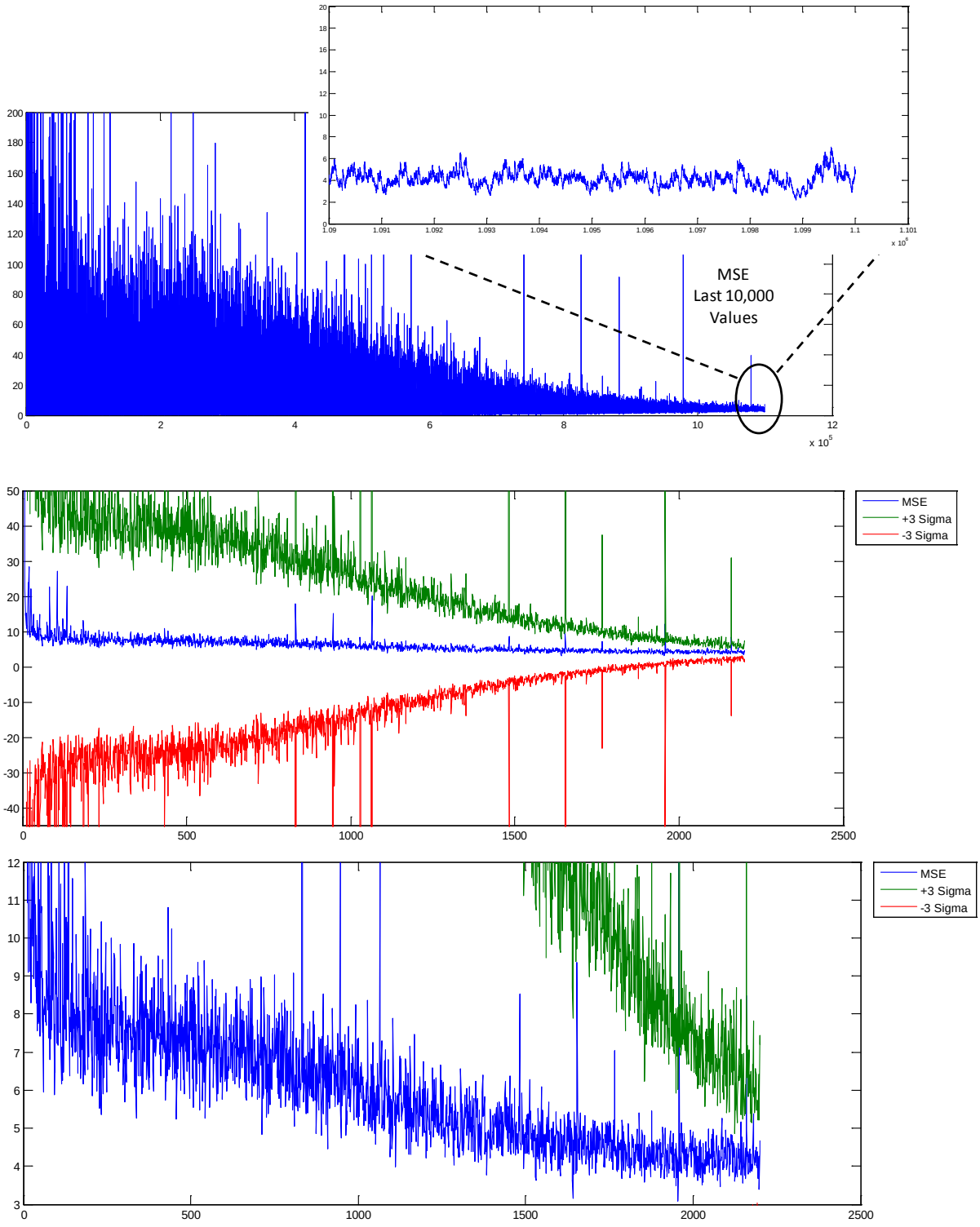


Figure 45: Test Case #3 – Trial #6 MSE and Variance Graphics

Modeling Input Parameters	Funding Level	Allocation Parameter	PlusMinus
Project #1	12.000	1.000	2.000
Project #2	7.000	0.500	1.500
Project #3	15.000	1.000	2.000
Project #4	10.000	0.500	2.000

Number of state-spaces: 4,005
Number of Months: 12
Total Budget = $44.000M
5,000 iterations Exploration
50,000 iterations Learning
RunTime: 16hrs 30Minutes
2.70 GHz Intel® Core®
8.00 GB RAM

		Project #1		Project #2		Project #3		Project #4	
t	month	Comm	Exp	Comm	Exp	Comm	Exp	Comm	Exp
1	Oct	0.000	0.000	0.000	0.000	1.000	0.000	3.500	1.000
2	Nov	0.000	0.000	0.000	0.000	1.000	0.000	2.000	1.000
3	Dec	0.000	0.000	0.000	0.000	1.000	1.000	2.000	1.500
4	Jan	1.000	0.000	0.500	0.000	2.000	1.000	1.500	2.000
5	Feb	1.000	0.000	0.500	0.000	2.000	1.000	1.000	2.000
6	Mar	3.000	1.000	1.000	0.500	3.000	2.000	0.000	1.500
7	Apr	3.000	1.000	1.000	0.500	3.000	2.000	0.000	1.000
8	May	2.000	3.000	2.000	1.000	2.000	3.000	0.000	0.000
9	Jun	2.000	3.000	2.000	1.000	0.000	3.000	0.000	0.000
10	July	0.000	2.000	0.000	2.000	0.000	2.000	0.000	0.000
11	Aug	0.000	2.000	0.000	2.000	0.000	0.000	0.000	0.000
12	Sept	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
		12.000	12.000	7.000	7.000	15.000	15.000	10.000	10.000

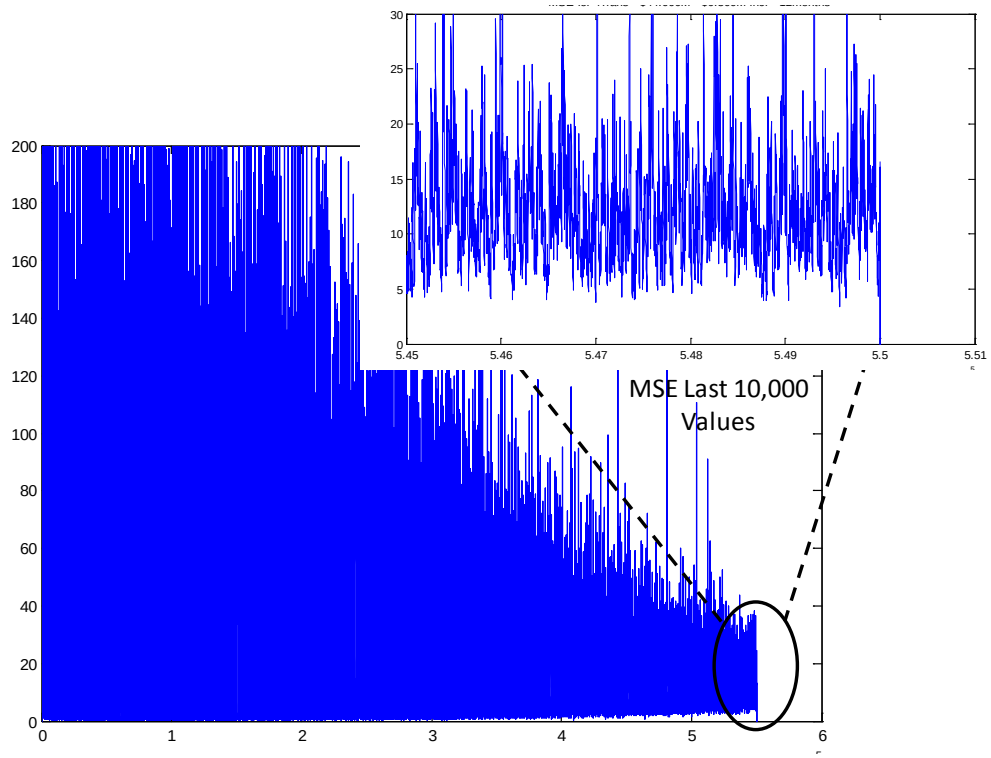


Figure 46: Test Case #4 – Trial #1 Input Parameters and MSE

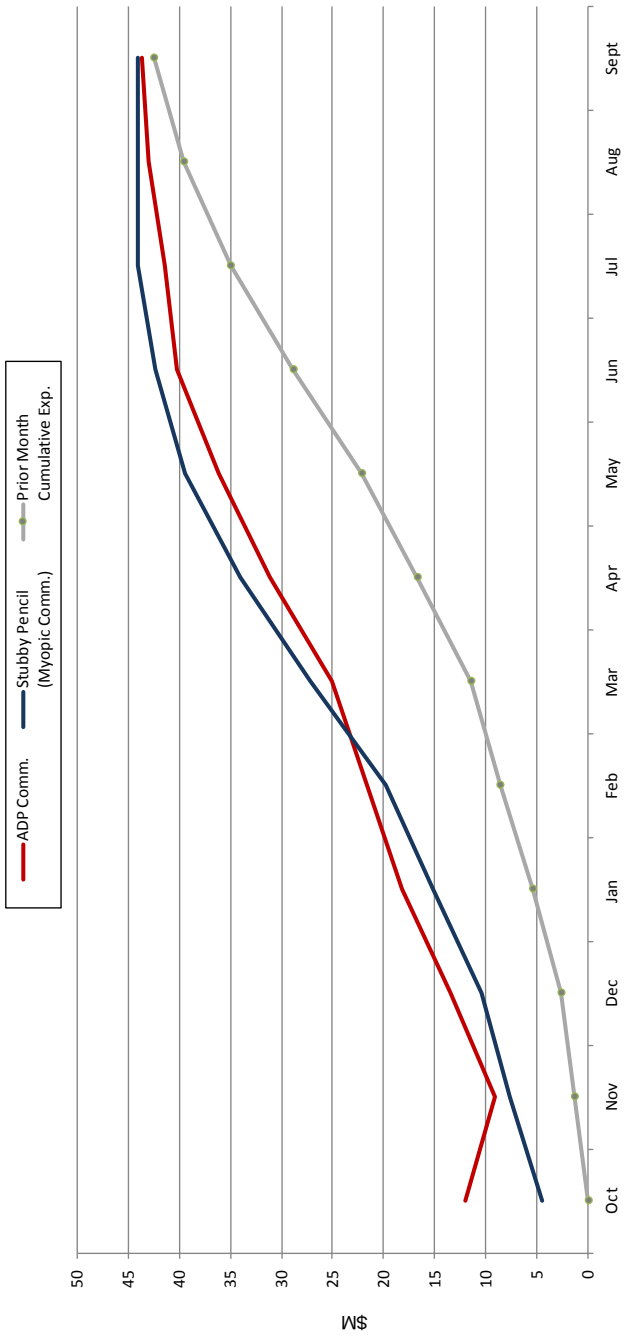


Figure 47: Test Case #4 – Trial #1 Output Graphic 2 / 'exog' = All

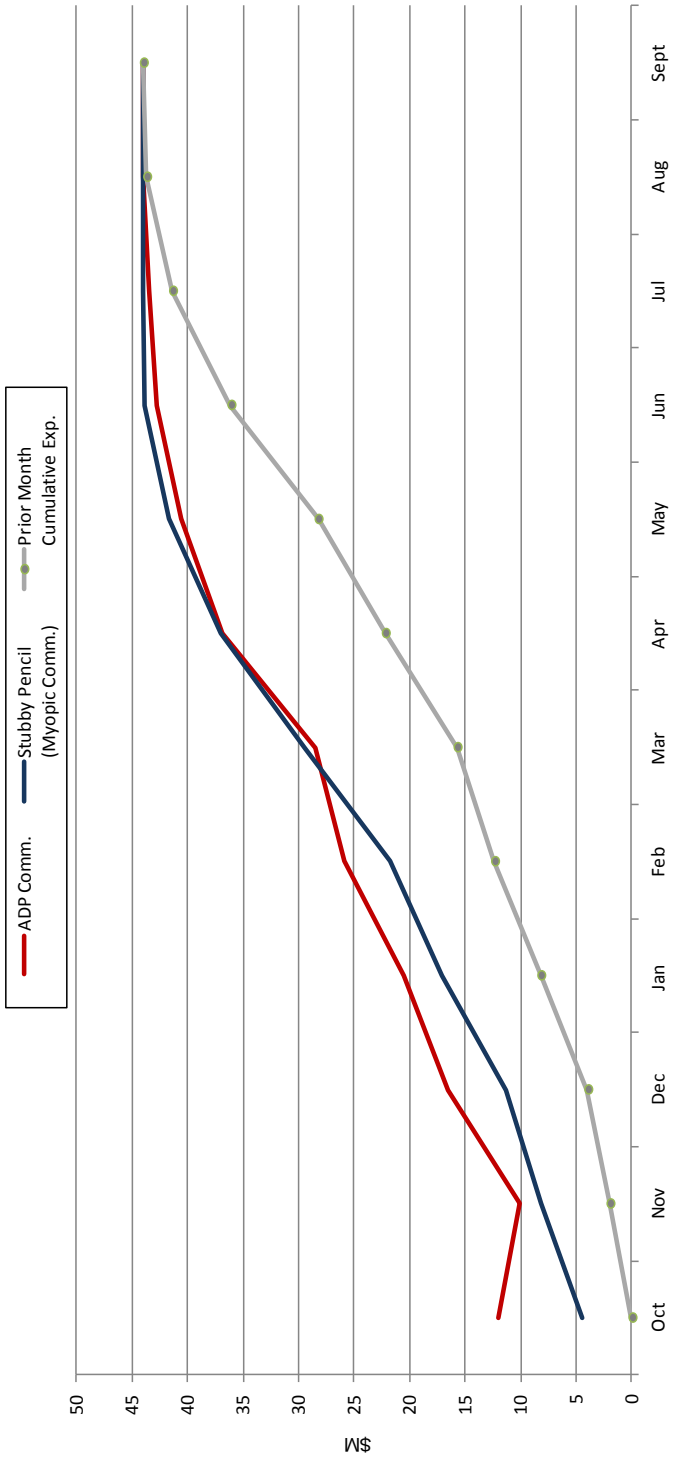


Figure 48: Test Case #4 – Trial #1 Output Graphic 2 / 'exog' = High

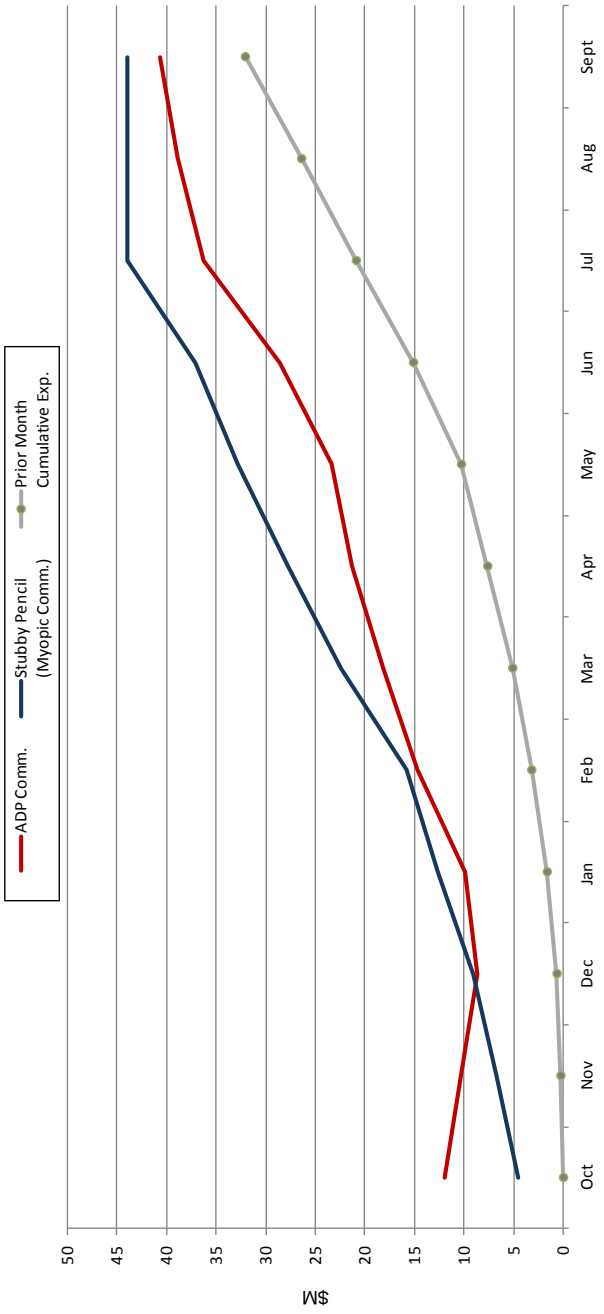


Figure 49: Test Case #4 – Trial #1 Output Graphic 2 / 'exog' = Low

CHAPTER FIVE – CONCLUSIONS, CONTRIBUTIONS, FUTURE RESEARCH, AND NEXT STEPS

5.1 Conclusions

During each fiscal year, weapon system programs across the DoD are tasked with the challenge of how to efficiently allocate their appropriated budgets. Due to the realities that many weapon system programs spend millions of dollars each year on numerous projects in an environment of uncertainty makes this a complex problem. However, improved commitment polices to this allocation problem may help alleviate the considerable amount of manpower dedicated to tracking the progress of weapon system cash flow as well as the energy exhausted on year-end financial close-out activities. Furthermore, commitment policy improvements may also help mitigate some of the contentious debates that exist between a weapon system program office and an agency comptroller office regarding cash flow. These potential advantages serve as incentives for continued research on models and tools that can assist with finding good policies for the financial execution commitment problem.

The challenge of finding the ideal or optimal commitment cash flow policy can be described as a sequential decision making problem over time. Although the use of DP as a tool for solving sequential decision problems that are structured as Markov decision processes has existed for many years, the more advanced methodologies that are found in the field referred to as approximate dynamic programming are relatively new. This

community provides a new approach to the application of Bellman's equation in an effort to mitigate the problems caused by "the curse of modeling" and "the curse of dimensionality" that plague the traditional DP solution methods. This thesis specifically examined the nature and complications that are involved in applying the ADP approach as a unique alternative to the problem of finding the best financial execution commitment strategy in an acquisition environment for weapon system programs.

Initial findings of this research suggest that ADP is an approach that can be used to simulate and mimic the financial execution environment of weapons system programs. Furthermore, the simulation results and graphical outputs provide a degree of verification that traditional stubby pencil and myopic commitment behaviors are more aggressive than those using an ADP approach. Recommended ADP commitment policies were consistently lower than the recommended stubby pencil plan regardless of the sensitivity settings used on the various expenditure excursions. In several of the scenarios, the ADP model did not fully commit the entire budget prior to the end of the FY. Thus, this indirectly implies that under certain expenditure conditions there is room within appropriated budgets for program managers and decision makers to initiate more work or projects at the onset of the FY.

5.2 Contributions

The initial objectives of this research were captured by three primary goals. The first was to build an ADP simulation model that could mimic the financial execution environment of a weapon system program. The second was to utilize the model to provide decision makers with recommended commitment policy strategies. Part of

developing these commitment policy strategies would entail conducting comparative analysis between the policy actions resulting from using an ADP approach and those that a stubby pencil methodology would provide given the same modeling conditions and assumptions. Lastly, the ADP execution modeling tool would be built to allow for various types of sensitivity analysis and execution excursions. This last feature would provide decision makers with a sense of how the recommended ADP commitment polices would be altered under presumed execution environments.

In an effort to accomplish the established objectives as related to the financial execution commitment problem, this thesis specifically examined two types of ADP model designs. The first model used a Q-learning approach. The Q-learning model creates and learns the values of Q-factors which represent a viable state-action pairing within the decision system. The main advantage of Q-learning is that the output consists of a visual $|S| \cdot |X|$ Q-matrix. The data pattern of the Q-matrix serves as an indication as to whether or not the logic of the model is structured properly in that the simulation is in fact visiting state-action combinations that are feasible. However, given the storage and computational demands of maintaining the Q-matrix throughout the simulation process makes using this approach for larger problems intractable. Nonetheless, Q-learning served as a solid foundation with which to build the second ADP model which is referred to as value function learning. The value function learning model was able to produce expectations of the value function in the form of Bellman's equation that were easier to manage and took the form of a single vector of dimension $|S|$.

Both models incorporated the use of several ADP techniques. One feature was to integrate the use of exploration and exploitation (learning). In both models, a certain number of simulation iterations were dedicated exclusively to exploration. While conducting exploration, the model is allowed to visit states and state-action pairings that might be good but, would otherwise not be visited if Bellman's equation were implemented using a strict minimum operand. As such, the model is allowed to learn information regarding the values of states or state-action pairings that a decision maker may wish to visit. Additionally, both the Q-learning and value function learning models were built around the post decision state variable as described in Powell (2007). This approach theoretically simplifies the value update process by separating the deterministic and stochastic components of the simulation as described by the transition function. As part of determining a stopping criterion for the simulation, mean square error (MSE) statistics and graphics were used as a method for gauging the convergence of Q-factors and expectations on the value function.

During the learnt phase, both the Q-model and value function model were able to provide ADP-recommended commitment policy strategies. These strategies are able to serve as cross-checks and alternatives to the stubby pencil approach. The comparative analysis between ADP and stubby pencil showed that the more conservative ADP policy could potentially improve cash flow efficiency. When compared against stubby pencil, ADP committed fewer dollars while still meeting the expenditure demands that occurred during the FY. Furthermore, the ADP policy continued to remain relatively conservative to the stubby pencil policy regardless of the simulated FY expenditure conditions.

The test cases simulated in this thesis involved scenarios that included only a small collection of projects. The protracted run times incurred precluded any extensive analysis of test case scenarios involving a sizable collection of projects simultaneously. However, based on the initial findings one can infer that there exists the possibility of an ADP policy recognizing significant cash flow efficiencies once the algorithm is scaled to simultaneously analyze the large numbers of projects that are often contained in a standard weapon system budget. Under these conditions, the ADP approach possesses the potential to provide decision makers with a rationale for starting additional projects at the onset of the FY. Additionally, the decision to fund more projects can be made without having to take on an unacceptable level of risk that the program will exhaust its budget prior to the end of the FY.

5.3 Future Research

One of the biggest concerns with the ADP model for financial execution is the extensive run times required to obtain a reasonable convergence on the Q-factors and estimates on the expectation of the value function. One method to improve run times that was used in this research was to reduce the number of attributes within the problem vector that defined a state-space from [P C O A E] to just [C E]. Nonetheless, this simplification of the state-space definition still noticeable suffered from excessively long run-times and the curse of dimensionality. Model run-times increased exponentially for each additive project that was incorporated into the simulation scenario. Although the ADP approach does offer improvements on the curse of dimensionality compared to traditional DP solution methods, the scalability of ADP models for real world problems

still remains as a recognized area of ongoing research. ADP techniques referred to as value function approximations encompass current methodologies that are examining approaches to address scalability and run-time issues.

Aggregation and function fitting are two types of value function approximation techniques that are receiving attention as opportunities for further research in the field of ADP. Aggregation works by lumping together a collection of similar outcome state-space possibilities into a single state-space result. George et al. (2008) provides a good overview of the use of aggregation for multi-attribute resource management problems. Through the use of an allocation parameter to define and narrow the outcome states, the ADP models presented in this thesis already included a type of aggregation method. The aggregation method utilized for the ADP financial execution models operates by effectively rounding the dollar values within the problem vector to the nearest factor of the assigned allocation parameter.

Function fitting or basis functions are ADP value function approximation approaches that remove the need to produce "look-up" table results for the state-space position that was presented in the Q-learning and value function learning models. Using this method the retained estimated value of the PDS $\bar{V}^n(S^x)$ is expressed as a basis function such as the ones shown by Equation 37. Instead of learning estimated values of the PDS position, through function fitting the algorithm will learn the estimated values of the coefficient variables captured by the vector $\bar{\theta}^n$. The algorithm proceeds by first obtaining an initial estimate of all viable state-spaces possibilities. Then, during the successive iterations the algorithm learns the values of $\bar{\theta}^n$ by obtaining a least squares

solution from $\bar{\theta}^n = [(X)^T X]^{-1}[X]^T \bar{V}^n(S^x)$. Here, the variable X represents the matrix form of the basis function.

Equation 37: Basis Function Examples

$$\bar{V}(S^x) = \theta_0 + \theta_1 S^x + \theta_2 S^{x2}$$
$$\bar{V}(S^x) = \theta_0 + \theta_1 e^{-S^x} + \theta_2 S^x e^{-S^x}$$
$$\bar{V}(S^x) = \theta_0 + \theta_1 S^x$$
$$\bar{V}(S^x) = \theta_0 + \theta_1 e^{-S^x}$$

In addition to scalability, there are several assumptions regarding the ADP model design that are worth re-examining and offer areas for further research. The first is to consider the implications of having a myopic cost function $C(S_t, x_t)$ definition that is dependent upon the current simulated month in question. Regardless of the current value t, the myopic or immediate cost is defined as the delta difference between the cumulative commitment allocation and the predicted cumulative expenditure needs over a three month window from time period t until time period t+2. However, due to the common occurrence of a continuing resolution authority (CRA) most programs are not in a position to fund three months worth of effort at the initial start of the fiscal year.

A CRA restricts the amount of funding available to weapon system programs during the earlier months of a fiscal year. These conditions may necessitate the need to redefine the myopic cost function $C(S_t, x_t)$ such that it measures the delta difference between cumulative commitments and cumulative predicted expenditures over just a single month t vice a three month window from t to t+2. This reflects the reality that in the earlier part of the fiscal year a weapon system program is likely to only have

sufficient funding available to cover one month worth of expenses at a time. However, at some point during the fiscal year the CRA conditions will no longer be in place. At this point, the definition of the myopic cost function $C(S_t, x_t)$ should switch back to its original three month designation or for a time period that is consistent with the decision maker's priorities. Although an evolving myopic cost function design may not alter the relationship between the ADP and stubby pencil recommended commitment policies, it is likely to alter the specific ADP commitment policy produced by the model.

Another opportunity for further research is the manner in which the exploration phase of the simulation is conducted. As currently designed, the exploration iterations randomly pick a viable commitment action during each time period t on an equally-likely basis. The intent of the exploration phase is to allow the model to learn and explore reasonably good state-space possibilities that otherwise would not be visited using a strict minimization operand as part of the value update process. The trade-off is that the model may require longer run times for convergence since the randomness of the exploration process results in visiting both reasonable good state-spaces as well as some bad state-spaces. The exploration phase of the ADP model can be more efficient by investigating alternative exploration designs that use a level of selective randomness so that the simulation process is only visiting reasonably good state-space possibilities and avoiding the bad state-space options all together. The study of effective exploration designs offers a rich opportunity for further research within the field of ADP.

Lastly, finding good solution methods to determining effective commitment policies does extend beyond the realm of purely mathematical modeling. In order to

understand this problem holistically, it would be imperative to further examine the organizational behavior, agency processes, and the various incentive structures that are all related to making financial commitment determinations. Some of the unresolved issues in this area include insight on how long it takes a committed amount of funding to move through the process and translate into expended dollars for any given project. The ADP model currently assumes that dollars committed in a given month are in fact readily available for billing and theoretically could be expended that same month. In reality, there is lag time involved as committed dollars become obligated, accrued, and finally expended. The amount of lag time is often associated with the specific project in question or to the unique circumstances of that particular transaction. Additionally, as described earlier, there are a number of incentive structures at work within the organization that are antithesis to the objective of overall efficient cash flow for the agency. The successful implementation of the ADP financial execution model within a weapon system program office is highly dependent on a thorough understanding of the organizational processes and institutionalized incentive structures.

5.4 Next Steps

There are a number of specific modeling enhancement considerations for the next phase of development work on the ADP financial execution simulation tool presented in this thesis. The realized exogenous information produced by the ADP model is the simulated actual expenditure amount for month t and the respective updated expenditure planning figures for month t+1 until the end of the fiscal year. During the simulation process, the realized actual expenditures are based on the plus-minus parameter assigned

to a project. At each individual month, this plus-minus parameter establishes the upper and lower bounds of a uniform distribution from which the actual expenditure is randomly selected. However, an alternative method for modeling uncertainty is to consider the possibility of assigning unique probability distributions to the various projects. Repetitive projects that are executed each year may have historical expenditure patterns that can be fitted to a probability distribution function. For those projects that are new and do not have historical expenditure patterns, one approach would be to work jointly with industry and contractors to build consensus on realistic expenditure forecasts and the associated distribution functions that could represent these planning figures.

Another modeling enhancement would include the incorporation of a penalty factor for each time the ADP recommends a commitment action that involves a new transaction. In some of the case studies examined, the recommended ADP commitment policy involved making an additive commitment action one month that was followed by a corrective de-commitment action the following month. Although this policy may appear superior since these adjustments reflect a greater sensitivity to the dynamics of the expenditure requirements, the reality is that program offices tend to want to minimize the number of de-commitment actions. A simple enhancement to the model would be to replace any ADP-recommended de-commitment actions with a commitment action of zero dollars. In general, each de-commitment as well as commitment action involves a cost associated with the manpower and coordination necessary to complete the transaction. The bottom line is to adjust the optimization function so that it more accurately reflects the realities that a commitment policy which achieves the same

objectives with fewer transactions is superior to a commitment policy that requires more transactions.

Lastly, in addition to the research areas mentioned earlier there are a number of next step techniques and enhancements that may prove beneficial to improve the run-times of the ADP financial execution model. Currently, the code does not take advantage of any parallel processing opportunities. Given the resources for parallel processing, a possible model implementation strategy that may improve run-times would be to separate the simulation of the financial execution process from the value update process. Here, the model first simulates an entire twelve-month FY execution cycle from a set Ω_n containing exogenous information for all time periods t = 1 through t = 12. Then the model performs the value update process for each of the PDS positions visited during the twelve month cycle simultaneously. In theory, this update approach is plausible as long as the simulation process never visited the same PDS twice during a single iteration. Furthermore, removing de-commitments as an action choice from the model will greatly improve the odds of satisfying this conditional.

Lastly, there are some relatively simple Matlab code adjustments that may provide further improvements on run-times. The first adjustment would be to switch out several of the matrix update and storage subroutines from a sequential search design to a binary search design. Although the impacts on problems with small state-space sizes could be negligible, there may be noticeable differences when applied to larger scale problems. Another simple modification would include the consistent use of pre-allocation commands that are used to set aside the memory requirements for any of the

matrices used in the ADP algorithm. Finally, one can further improve run-time performance through the use of vectorization of the for loop statements within the Matlab platform.

The research presented in this thesis provides an initial first step to considering the problem of finding efficient cash flow commitment policies for weapon system programs using a new and untested approach. First hand exposure to the processes and experiences within the Missile Defense Agency (MDA) served as the framework and motivation for constructing the ADP commitment policy model. It is the intention of the author to provide an overview of the modeling concept and initial findings to representatives within the Operations Directorate at MDA. The hope is to gain continued support for further refinement of the ADP model as well as the opportunity to implement the concept in parallel within a program office's current execution commitment planning and forecasting methodologies.

REFERENCES

REFERENCES

Bellman, Richard. 1954. “The Theory of Dynamic Programming”, *Bulletin of the American Mathematical Society* 60, no. 6: 503-515.

Bellman, Richard. 1957. Dynamic Programming. Princeton, NJ: Princeton University Press.

Bellman, Richard E. and Stuart E. Dreyfus. 1962. Applied Dynamic Programming. Princeton, NJ: Princeton University Press.

Bertsekas, Dimitri P. 1995. Dynamic Programming and Optimal Control. Nashua, NH: Athena Scientific

Bertsekas, D. and J. Tsitsiklis. 1996. Neuro-Dynamic Programming. Belmont, Massachusetts: Athena Scientific

Borkar, V.S. 2002. “Q-Learning for Risk-Sensitive Control”, *Mathematics of Operations Research* 27, no. 2: 294-311.

Darken, C., J. Chang, and J. Moody. 1992. “Learning rate schedules for faster stochastic gradient search”, In *Neural Networks for Signal Processing 2, Proceedings of the 1992 IEEE Workshop.* Piscataway, NJ. IEEE Press.

Das, Tapas K., Abhijit Gosavi, Sridhar Mahadevan, and Nicholas Marchalleck. 1999. “Solving Semi-Markov Decision Problems Using Average Reward Reinforcement Learning”, *Management Science* 45, no. 4: 560-574.

Defense Acquisition Guidebook. Defense Acquisition University.
Available from https://dag.dau.mil/

Denardo, Eric V. 2003. Dynamic Programming: Models and Applications. Mineola, NY: Dover Publications, Inc.

Department of Defense (DoD) Financial Management Regulation Glossary
Available From: http://comptroller.defense.gov/fmr/glossary.pdf

George, Abraham, Warren B. Powell, and Sanjeev R. Kulkarni. 2008. “Value Function Approximation Using Multiple Aggregation for Multiattribute Resource Management”, *Journal Machine Learning Research* 9, 2079-2111.

Gosavi, A. 2003. Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning. Boston, Massachusetts. Kluwer Academic Publishers.

Gosavi, Abhijit. 2005. “Boundedness of Iterates in Q-Learning”, *Systems & Control Letters*, no. 55: 347-349.

Gosavi, Abhijit. 2009. “Reinforcement Learning: A Tutorial Survey and Recent Advances”, INFORMS Journal on Computing 21, no. 2: 178-192.

Howard. R. 1960. Dynamic Programming and Markov Processes. Cambridge, MA. MIT Press.

Iyengar, Garud N. 2005. “Robust Dynamic Programming”, *Mathematics of Operations Research* 30, no. 2: 257-280.

Powell, W.B. 2007. Approximate Dynamic Programming: Solving the Curse of Dimensionality. Hoboken, NJ: John Wiley & Sons, Inc.

Powell, Warren B. 2009. “What You Should Know About Approximate Dynamic Programming”, Naval Research Logistics 56: 239-249.

Powell, Warren B. 2010. “Merging AI and OR to Solve High-Dimensional Stochastic Optimization Problems Using Approximate Dynamic Programming”, *INFORMS Journal on Computing* 22, no. 1: 2-17.

Puterman, Martin L. 1994. Markov Decision Processes: Discrete Stochastic Dynamic Programming. New York: Wiley-Interscience Publication.

Robbins, Herbert and Sutton Monro. 1951. “A Stochastic Approximation Method”, *The Annals of Mathematical Statistics* 22, no. 3: 400-407.

Sutton, R. S. and A. G. Barto. 1998. Reinforcement Learning. Cambridge, MA: MIT Press.

Tsitsiklis, John N. 1994. “Asynchronous Stochastic Approximation and Q-Learning”, *Machine Learning*, no. 16: 185-202.

Tskitsiklis, John N. 2010. “Perspectives on Stochastic Optimization Over Time”, *INFORMS Journal on Computing* 22, no. 1: 18-19.

Watkins, C.J.C.H. 1989. Learning from Delayed Rewards. Ph.D. Thesis, University of Cambridge, England.

Watkins, Christopher J.C.H. & Peter Dayan. 1992. “Technical Note: Q-Learning”, *Machine Learning*, no. 8:279-292.

Winston, Wayne L. 1994. Operations Research: Applications and Algorithms. Belmont, California: Duxbury Press.

CURRICULUM VITAE

Erich D. Morman obtained a Bachelor's degree with a double major in mathematics and economics from the University of Dayton in May 1996. He received his Masters degree in Systems Engineering and Operations Research from George Mason University in May 2002. From March 1997 until the present he has worked for the Missile Defense Agency as a support contractor. His professional experience includes the areas of wargaming and simulation, cost estimating and analysis, and Planning, Programing, Budgeting, and Execution (PPBE). He currently works in Arlington, VA as a senior analyst for Electronic Consulting Services, Inc.