N71-18005

NASA CR-116417

THE CONTROL OF NONLINEAR STOCHASTIC CONTROL SYSTEMS UNDER DISCOUNTED PERFORMANCE CRITERIA

by

Cliff Andrew Harris

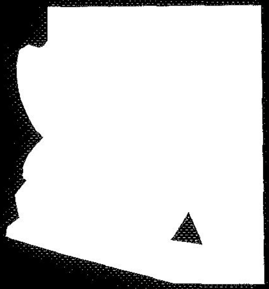
and

Theodore L. Williams

Prepared under Contract NGL-03-002-006

Donald G. Schultz, Principal Investigator

August, 1970



ENGINEERING EXPERIMENT STATION
COLLEGE OF ENGINEERING
THE UNIVERSITY OF ARIZONA
TUCSON, ARIZONA

THE CONTROL OF NONLINEAR STOCHASTIC CONTROL SYSTEMS UNDER DISCOUNTED PERFORMANCE CRITERIA

by

Cliff Andrew Harris

and

Theodore L. Williams

Prepared under Contract NGL-03-002-006

Donald G. Schultz, Principal Investigator

August, 1970

Engineering Experiment Station
College of Engineering
The University of Arizona
Tucson, Arizona

TABLE OF CONTENTS

TABLE OF CONTENTS--Continued

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS--Continued

THE CONTROL OF NONLINEAR STOCHASTIC CONTROL SYSTEMS
UNDER DISCOUNTED PERFORMANCE CRITERIA

ABSTRACT

This study is concerned with the optimal control of discrete time nonlinear stochastic control systems under discounted performance criteria. The dynamic equations, noise statistics, and stage cost are all time invariant. The discounted performance criteria apply when the present cost is weighted more heavily than future costs.

Systems of this type have classically been viewed as finite Markov chains. Howard's policy iteration and dynamic programming, the two basic methods of solution, are developed and discussed. Their successful application rests heavily upon the state space of the finite Markov chain being small. This study, however, investigates systems whose state is defined on the continuum. A grid is imposed on the state space, creating a finite state space which is very large. A numerical algorithm tailored to attack this problem is developed. The state space is partitioned into blocks, and the stationary cost function over each block is approximated with a quadratic surface. The algorithm then employs the dynamic programming functional equation iteratively to converge to the optimum control and cost functions. Several example problems are worked, including an inventory control problem and a nuclear reactor cooling problem. Both the high-speed memory requirement and computation time necessary for conventional dynamic programming are significantly reduced by the new algorithm.

CHAPTER 1

INTRODUCTION

1.1 The Problem

Often the state or position of a system can be observed perfectly but the behavior of the system is subject to uncertainty. Considerable effort has been expended in recent years to develop means of controlling such systems, which arise in many engineering and economic applications.

These systems can be explicitly defined by the dynamic transition equation

$$x(k+1) = g\left[x(k),\ \alpha(k),\ \zeta(k),\ k\right],\ k = 0,1,\ \ldots,\ N \qquad (1.1)$$

where,

k = time parameter

N = terminal time, which may be infinite

x = n-dimensional state vector

α = q-dimensional control vector

g = n-dimensional vector function

ζ = r-dimensional random variable with known probability distribution.

The objective is then to influence the evolution of the state $x(k)$ with the control $\alpha(k)$ in such a fashion as to minimize some criterion of system performance. The general form of the performance criterion is

$$\text{performance criterion} = E\left\{\sum_{k=0}^{N} \ell\left[x(k),\ \alpha(k),\ k\right] \Big| x(0)\right\} \tag{1.2}$$

where ℓ is a scalar function. This problem is called the stochastic control problem. It should be distinguished from the problem of combined estimation and control studied by Feldbaum (1960), Meier (1966), Aoki (1967), and others. The combined estimation and control problem allows the consideration of uncertainty of the system's present state as well as uncertainty in the system dynamics.

It has been established (Dreyfus, 1962; Kashyap, 1966) that the optimum control strategy for the stochastic control problem is pure (i.e., deterministic) and depends only on the present state of the system. Thus, the solution to the stochastic control problem is a controller which upon being informed of the state x(k) at time k generates the control $\alpha(k)$ to be applied. Control of this nature is called feedback control and the structure of the control system is illustrated in Figure 1.1. The controller is a function, u_k, which maps the state x(k) into the control $\alpha(k)$. It is the control function u_k which is sought.

This study considers a more specific problem, the control of a time invariant system under a discounted performance criterion. The system dynamics are described by

$$x(k+1) = g\left[x(k),\ \alpha(k),\ \zeta(k)\right],\ k = 0,1\ \ldots \tag{1.3}$$

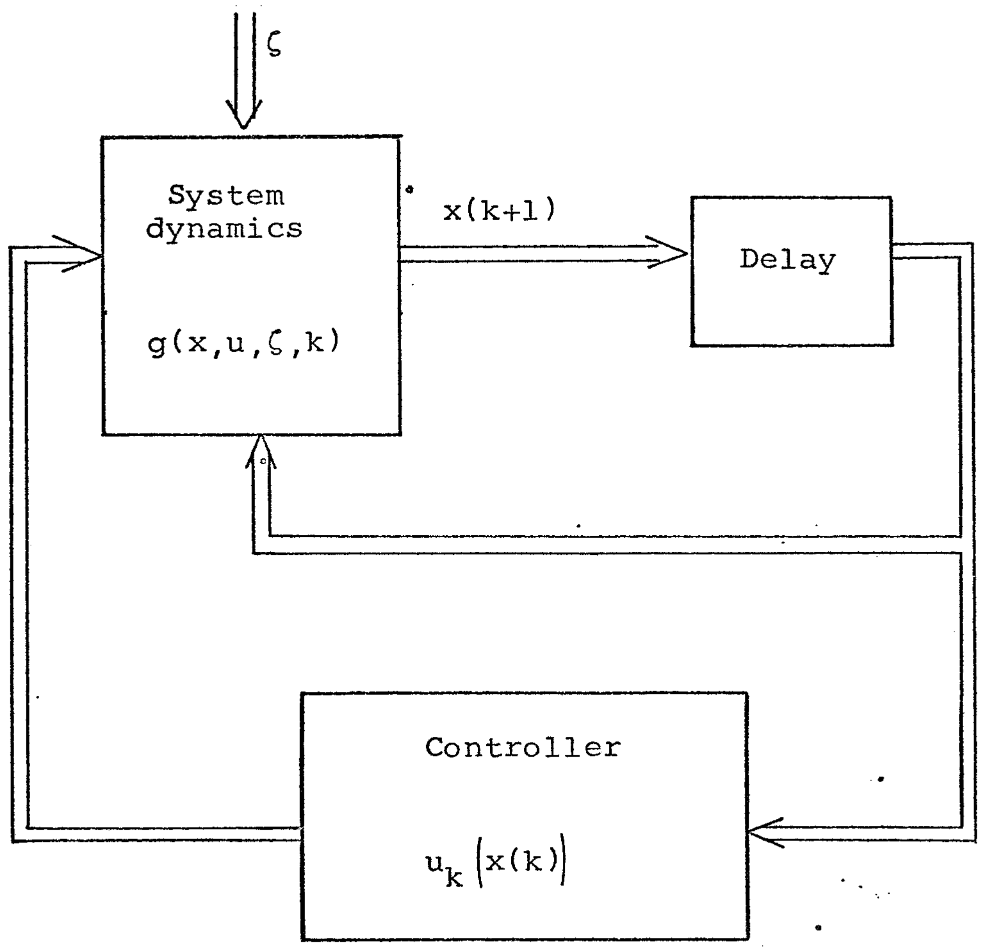


Figure 1.1 Stochastic control system

where the function g is time independent. The performance criterion becomes

$$\text{performance criterion} = E\left\{\sum_{k=0}^{\infty} \beta^k \ell\left[x(k),\ \alpha(k)\right] \Big| x(0)\right\} \tag{1.4}$$

where ℓ is a time independent scalar function and the discount factor β is a constant, $0 \leq \beta < 1$. The time span is infinite. Under these conditions it will be shown that both the control function, u_k, and the k-stage cost v_k (2.6) become stationary as $k \to \infty$. These properties are essential to the numerical techniques developed in this study.

The stochastic control problem described by (1.1) and (1.2) was first examined by Bellman (1958, 1961). The dynamic programming functional equation was generated by Bellman's work. It presented a means of numerical solution even though no general analytical solution resulted. A substantial research effort was directed at the stationary stochastic control problem of (1.3) and (1.4) with the additional constraint that the state space be finite. Howard (1960) presented his algorithm upon which most subsequent work was based. Blackwell (1962, 1965), Derman (1964), Wonham (1967), Miller and Veinott (1969), and Veinott (1969) considered extensions of the problems attacked by Howard and generated many existence theorems for optimal control. Miller (1968a, 1968b) considered the finite state space stochastic control problem with continuous time. Martin (1967) and Riordon (1969) obtained a solution for the adaptive stochastic control problem where the probabilistic description of the random variable ζ is unknown. However, work subsequent to Bellman and Howard relied on either dynamic programming or Howard's algorithm. Thus, from a practical point of view all subsequent work also shared the numerical difficulties these techniques possessed. In particular, Bellman's "curse of dimensionality" severely limited the problems for which a numerical solution could be achieved.

In this study a dynamic programming algorithm is presented to alleviate the problem of dimensionality by using quadratic approximation to the cost surface in a partitioned state space. The main contribution is this workable algorithm whose utility is demonstrated in several examples.

1.2 Organization

The second chapter discusses the control of finite Markov chains. Optimality is defined explicitly and behavior of the cost function under stationary and non-stationary control policies is considered, including the limiting case of $\beta = 1$. Dynamic programming and Howard's policy iteration are shown to result in an optimal control.

In Chapter 3 practical, numerical problems associated with solving the stochastic control problem for a continuous state space are considered. The numerical limitations of Howard's policy iteration and conventional dynamic programming are revealed. The algorithm which is the basic result of this work is developed. It is shown to dramatically reduce high-speed storage necessary and also to reduce computation time in comparison to Howard's and Bellman's techniques. The algorithm is then applied to several example problems in Chapter 4 including an inventory control problem and a nuclear reactor cooling problem. Chapter 5 contains conclusions and suggestions for further research. Appendices follow which contain program listings of the numerical algorithms used in this study.

1.3 Notation and Symbols

$\underset{\sim}{x}, \zeta$	=	random variable
$t, {}^{i}x$	=	state variable
X	=	state space
f, g	=	transition functions
$p_{ij}(\cdot)$	=	transition probability
$P(\cdot)$	=	stochastic matrix
μ_j	=	<u>a priori</u> probability
μ	=	<u>a priori</u> probability vector
α	=	control action
A	=	set of all α's
u	=	control function
U	=	set of all control functions

π	= control law for all time
u^{∞}	= stationary control law
$V(\cdot)$	= cost vector
$v(\cdot)$	= scalar cost function
$\bar{v}(\cdot)$	= quadratic approximation to v
L	= stage cost vector
ℓ	= scalar stage cost
B_{ℓ}	= state space partition or block

CHAPTER 2

THE CONTROL OF FINITE MARKOV CHAINS

2.1 Introduction

A meaningful analytical examination of the stochastic control problem is found in considering the control of finite Markov chains. The systems to be controlled are stochastic in that the evolution of the system state under a given control policy is uncertain; however, a knowledge of the current state and control action determine the probability distribution of the next state. The dynamic system, for analytical tractability is also considered to be described adequately by a finite number of states. A finite Markov chain representation follows naturally. Dynamic plant equations and plant noise are modeled by a set of transition probabilities over a finite state space. Each control law is associated with a set of transition probabilities, and a cost function is defined. It is found that the cost function may be minimized by either dynamic programming or Howard's policy iteration. This chapter examines both these methods and the properties of the cost function under various control laws.

2.2 Finite Markov Chains

Let $(\Omega, \mathcal{F}, \text{Prob})$ be a probability triple with Ω the set of elementary events, w, $\mathcal{F}$, the σ-algebra of subsets of Ω and Prob the probability measure on $\mathcal{F}$. The finite set of real numbers, $X = \{{}^1x, {}^2x, \ldots {}^Jx\}$ is called the state space and constitutes the range of the random variable $\underset{\sim}{x}$ mapping Ω onto X. A stochastic process is a sequence

$$\underset{\sim}{X} = \{\underset{\sim}{x}_n | n = 0, 1, 2, \ldots\}$$

of random variables.

The stochastic process $\underset{\sim}{X}$ is said to be a Markov chain if for

$$E_n \epsilon \mathcal{F} \text{ with } E_n = \{w | \underset{\sim}{x}_n(w) = {}^ix, {}^ix \epsilon X\},$$

$$\text{Prob } [E_n | E_0 \cap E_1 \cap E_2 \ldots \cap E_{n-1}] = \text{Prob } [E_n | E_{n-1}],$$

whenever Prob $[E_0 \cap E_1 \cap E_2 \ldots \cap E_{n-1}] \neq 0$. That is,

$$\text{Prob } [\underset{\sim}{x}_n = {}^ix \mid \underset{\sim}{x}_0 = {}^ax, \underset{\sim}{x}_1 = {}^bx, \ldots, \underset{\sim}{x}_{n-1} = {}^ix]$$

$$= \text{Prob } [\underset{\sim}{x}_n = {}^ix | \underset{\sim}{x}_{n-1} = {}^ix] \triangleq p_{ij}(n),$$

where the $p_{ij}(n)$ are the transition probabilities defining the chain. The transition (stochastic) matrix for the chain is

$$P(n) = [p_{ij}(n)] \quad n = 0, 1, \ldots$$

The transition probabilities are related by the Chapman-Kolmogorov equation

$$p_{ij}(m,n) = \sum_{k=1}^{J} p_{ik}(m,r)\, p_{kj}(r,n), \quad m \le r \le n \qquad (2.1)$$

where

$$p_{ij}(m,n) \triangleq \text{Prob}\,[\underset{\sim}{x}_n = {}^j x \,|\, \underset{\sim}{x}_m = {}^i x], \quad m \le n.$$

A chain is said to be homogeneous if

$$P(n) = P = \text{constant}$$

Then

$$P(0,n) = [p_{ij}(0,n)] = P^n.$$

Let $\mu j(n)$ = Prob $[\underset{\sim}{x}_n = {}^j x]$ be the <u>a priori</u> probability that the chain is at state ${}^i x$ at time n, and let

$$\mu(n) = \left|\mu_1(n), \ldots, \mu_J(n)\right|$$

be the row vector of all <u>a priori</u> probabilities at time n, then

$$\mu(n) = \mu(0)P(0,n)$$

and, for homogeneous chains, a simplified transition equation can be used,

$$\mu(n) = \mu(0)P^n.$$

The states are classified as

(a) ${}^j x$ is persistent if Prob $[\underset{\sim}{x}_n = {}^j x$ for some n$] = 1$,

(b) ^{j}x is <u>transient</u> if Prob $[\underset{\sim}{x}_n = {}^{j}x$ for some $n] < 1$,

(c) ^{j}x is <u>aperiodic</u> if the greatest common divisor of the set of all n such that $p_{jj}(n) > 0$ is one, i.e., g.c.d. $\{n \; p_{jj}(n) > 0\} = 1$,

and

(d) ^{j}x is ergodic if it is persistent and aperiodic (for finite chains).

A <u>chain</u> is said to be <u>ergodic</u> if all states are ergodic. Examples of state classification are given in Figure 2.1, where the transition probabilities are represented by arrows.

The following theorem will be useful in examining the stationary control of finite Markov chains.

Theorem 1: For a finite homogeneous ergodic Markov chain with transition matrix P, there exists a unique stationary probability distribution μ, and

$$p_{ij}(n) \rightarrow \mu_j \text{ as } n \rightarrow \infty \text{ geometrically fast.}$$

Or in matrix form,

$$P^n \rightarrow 1\mu = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} (\mu_1, \mu_2, \ldots, \mu_J) \text{ geometrically}$$

fast (Doob, 1953, Ch. 5 §2). Thus,

$$\mu(n) = \mu(0)P^n \rightarrow \mu(0)1\mu,$$

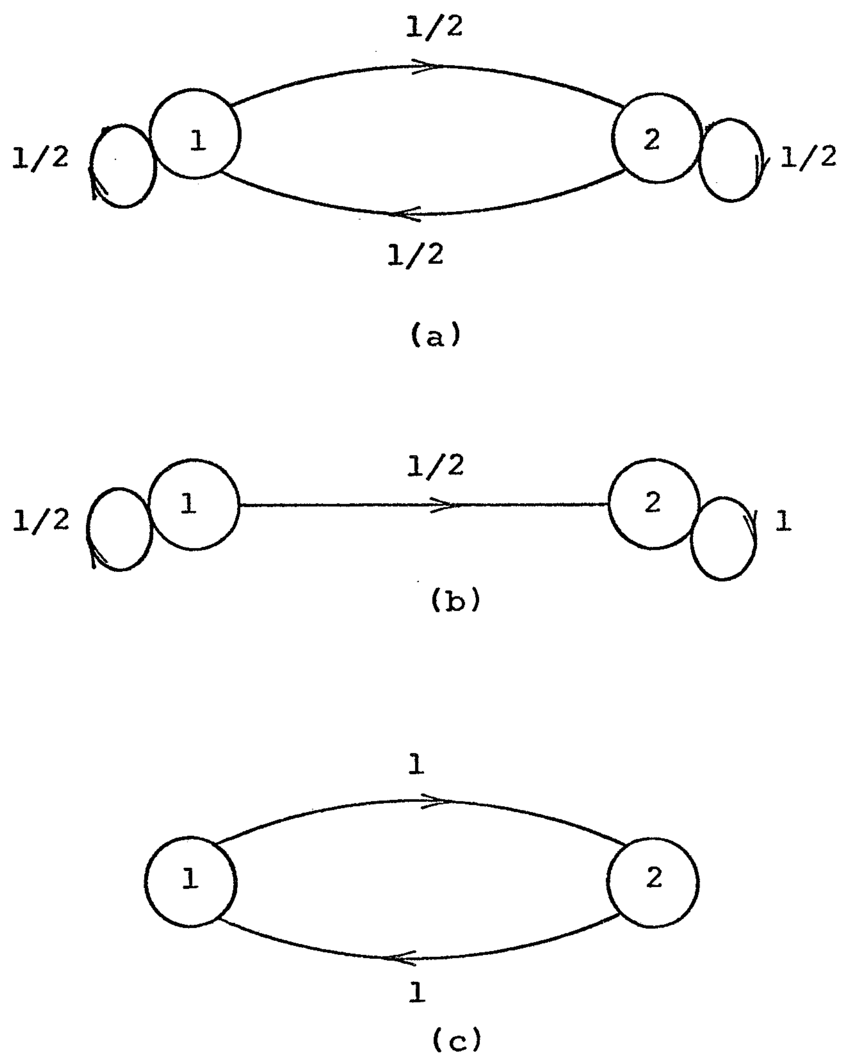


Figure 2.1 Finite Markov chains -- (a) ergodic chain; (b) non-ergodic chain, State 1 is transient, and State 2 is persistent; (c) non-ergodic chain, States 1 and 2 are persistent but chain is not aperiodic.

or

$$\mu(n) \rightarrow \mu \text{ as } n \rightarrow \infty,$$

and

$$\mu = \mu P.$$

2.3 Controlled Finite Markov Chains

The dynamic system to be controlled has a finite state space

$$X = \{{}^1x,\ {}^2x,\ \ldots,\ {}^Jx\}$$

and is observed periodically (at every discrete time period). At each time period a control, α, which influences the behavior of the system is applied from a set of possible controls A. As a result of the application of the control $\alpha_k \epsilon A$ with the system in state $x_k \epsilon X$ at time k there is a time independent,

(1) stage cost $0 < \ell(x_k, \alpha_k) < \infty$ incurred, and

(2) transition of the system from x_k at time $t = k$ to $x_{k+1} \epsilon X$ at time $t = k+1$ with

$$p_{ij}(\alpha_k) = \text{Prob } [\underset{\sim}{x}_{k+1} = {}^jx \mid x_k = {}^ix,\ \alpha_k].$$

There is also a discount factor, β, $0 \leq \beta < 1$; whereby, the cost $\ell(x,\alpha)$ for being in state x and applying a control α n periods into the future has a discounted cost of $\beta^n \ell(x,u)$ at the present.

Let U denote the set of control functions u from X into A [i.e., $u \epsilon U$ implies $u(x) \epsilon A$ for all $x \epsilon X$]. A policy, π, specifies a sequence of control functions for all time; $\pi = \{u_0, u_1, \ldots\}$. Thus, at time k, with the system in state x_k, the control $u_k \epsilon A$ is applied. A stationary policy is a policy for which $u_n = u$, $n = 0, 1, \ldots$, i.e., $\pi = \{u, u, \ldots\} \triangleq u^\infty$.

$$\text{Let } L(u) = [\ell(^1x, u(^1x)), \ldots, \ell(^Jx, u(^Jx))]^T$$
$$= [\ell_1(u), \ell_2(u), \ldots, \ell_T(u)]^T$$

be the column vector representation of the stage cost for all states under the control $u \epsilon U$. Let P(u) be the J x J Markov transition matrix for the control u in the Markov chain established by the policy π.

$$P(u) = [p_{ij}(u)] \tag{2.2}$$

Thus, by the Chapman-Kolmogorov equation, the transition matrix from time t = 0 to t = n is

$$P(n,\pi) = P(u_0)\ P(u_1) \ldots P(u_n).$$

For the policy π and the initial state x_0 the total expected cost vector is

$$V(\pi) = \left(v(1,\pi), v(2,\pi), \ldots, v(J,\pi)\right)^T$$

where

$$v(i,\pi) = E\left\{ \sum_{n=0}^{\infty} \beta^n \ell\left(\underset{\sim}{x}_n, u_n(\underset{\sim}{x}_n)\right) \middle| \underset{\sim}{x}_0 = {}^ix, \pi \right\} \tag{2.3}$$

or

$$= \ell_i(u_0) + \beta E \left\{ \Sigma \beta^{n-1} \left(\underset{\sim}{x}_n, u_n(\underset{\sim}{x}_n) \right) \middle| \underset{\sim}{x}_0 = {}^i x, \pi \right\}$$

$$= \ell_i(u_0) + \beta E \left\{ E \left\{ \sum_{n=1}^{\infty} \beta^{n-1} \ell\left(\underset{\sim}{x}_n, u_n(x_n) \right) \middle| \underset{\sim}{x}_1 \right\} \middle| \underset{\sim}{x}_0 = {}^i x, \pi \right\}$$

$$= \ell_i(u_0) + \beta E \left\{ v(\underset{\sim}{x}_1, \bar{\pi}) \middle| \underset{\sim}{x}_0 = {}^i x, u_0 \right\}$$

where $\bar{\pi} = \{u_1, u_2, \ldots\}$.

Thus, the important functional equation of dynamic programming results,

$$v(i, \pi) = \ell_i(u_0) + \beta \sum_{j=1}^{J} p_{ij}(u_0) v(j, \bar{\pi}), \tag{2.4}$$

or, in vector form,

$$V(\pi) = L(u_0) + \beta P(u_0) V(\bar{\pi})$$

$$= \sum_{n=0}^{\infty} \beta^n P(n,\pi) L(u_n), \quad P^o(\pi) = I.$$

It is desired, if possible, to obtain a policy π^* with the lowest cost,

$$V(\pi^*) \leq V(\pi) \text{ for all } \pi$$

where $\leq$ means $v(i,\pi^*) \leq v(i,\pi)$ for all π and $i = 1, \ldots, J$. Such a policy π^* is defined as optimal.

2.4 Effect of General Policies

The theorems in this section are due to Blackwell (1962) and Howard (1960) and cast light on the classes of policies that are profitable to be studied. The proofs which follow Blackwell almost verbatim are included for the insight they provide into the behavior of the expected cost function, $V(\pi)$, under different types of policies. Theorem 2 establishes that if any optimal policy exists then there is a stationary policy equally as good. Theorems 3 and 4 are used to obtain Theorem 5 which assures the existence of an optimal stationary policy for A finite and also a means of obtaining this optimal policy.

Lemma. There exists a $f \in U$ such that for an arbitrary $V = (v_1, v_2, \ldots, v\ )^T$, $v_i \geq 0$ and any $u \in U$

$$L(f) + \beta P(f)V \leq L(u) + \beta P(u)V.$$

Proof. Consider the i^{th} element

Let, $\Theta_i(\alpha) = \ell({}^i x, \alpha) + \beta \sum_j p_{ij}(\alpha) v_j$ for any $\alpha \in A$. Since $v_i \geq 0$, $\Theta_i(\alpha) > 0$. Thus, the set $\{\Theta_i(\alpha) | \alpha \in A\}$ has a lower bound of zero and hence a greatest lower bound for say $\alpha_i \in A$. The control function f such that $f({}^i x) = \alpha_i$ satisfies the lemma.

Theorem 2 (Blackwell) If there is an optimal policy $\pi^* = \{u_0, u_1, \ldots\}$, there is an optimal policy which is stationary.

Proof. By hypothesis,

$$V(\pi^*) \leq V(\pi) \text{ for all } \pi.$$

$$V(\pi^*) = L(u_0) + \beta P(u_0)V(\pi_1),$$

where $\pi_1 = \{u_1, u_2, \ldots\}$.

Also, $V(\pi^*) \leq V(\pi_1)$,

thus, $V(\pi^*) \geq L(u_0) + \beta P(u_0)V(\pi^*)$.

By the lemma, there exists a $f \varepsilon U$ such that

$$V(\pi^*) \geq L(f) + \beta P(f)V(\pi^*),$$

and, again,

$$\begin{aligned} V(\pi^*) &\geq L(f) + \beta P(f)[L(u_0) + \beta P(u_0)V(\pi_1)], \\ &\geq L(f) + \beta P(f)[L(u_0) + \beta P(u_0)V(\pi^*)] \\ &\geq L(f) + \beta P(f)[L(f) + \beta P(f)V(\pi^*)] \\ &\geq L(f) + \beta P(f)L(f) + \beta^2 P(f)V(\pi^*). \end{aligned}$$

By continuing this process,

$$V(\pi^*) \geq \sum_{n=0}^{n-1} \beta^n P^n(f)L(f) + \beta^N P^N(f)V(\pi^*).$$

As $N \rightarrow \infty$, $\beta^N P^N(f)V(\pi^*) \rightarrow 0$ since $\beta < 1$ and $P^N(f)$ is a stochastic matrix.

Thus, as $N \rightarrow \infty$

$$V(\pi^*) \geq V(f^\infty)$$

but since π^* is optimal

$$V(\pi^*) = V(f^\infty)$$

and f^∞ is an optimal stationary policy.

Theorem 3 (Blackwell) Let $\pi = \{u_0, u_1, \ldots\}$ and $\pi' = \{f, u_0, u_1, \ldots\}$. If

$$V(\pi) \leq V(\pi') \text{ for all } f\varepsilon U,$$

then π is optimal.

Proof. By hypothesis,

$$L(f) + \beta P(f)V(\mathring{\pi}) \geq V(\pi) \quad \text{for all } f\varepsilon U$$

Or, in particular,

$$L(f_N) + \beta P(f_N)V(\pi) \geq V(\pi) \quad \text{for all } f_N\varepsilon U$$

$$L(f_{N-1}) + \beta P(f_{N-1})V(\pi) \geq V(\pi) \quad \text{for all } f_{N-1}\varepsilon U$$

$$\vdots \qquad\qquad \vdots$$

$$L(f_1) + \beta P(f_1)V(\pi) \geq V(\pi) \quad \text{for all } f_1\varepsilon U$$

Thus,

$$L(f_1) + \beta P(f_1)[L(f_2) + \beta P(f_2)V(\pi)] \geq V(\pi),$$

or

$$L(f_1) + \beta P(f_1)L(f_2) + \beta^2 P(f_1)P(f_2)V(\pi) \geq V(\pi).$$

Continuing this substitution process for the policy $\pi'' = \{f_1, f_2, \ldots, f_N, u_0, u_1, \ldots\}$

$$L(f_1) + \beta P(f_1)L(f_2) + \dots + \beta^{N-1}P(f_1)P(f_2) \dots$$
$$P(f_{N-1})L(f_N) + \beta^N P(f_1) \dots P(f_N)V(\pi) \geq V(\pi),$$

or

$$V(\pi'') \geq V(\pi).$$

Again as $N \to \infty$, $\beta^N P(f_1)P(f_2) \dots P(f_N)V(\pi) \to 0$.

Each f_i is an arbitrary element of U; thus as $N \to \infty$ $V(\pi'')$ becomes the cost of any policy. That is

$$V(\pi) \leq V(\hat{\pi}) \text{ for any } \hat{\pi}.$$

Thus π is optimal.

Theorem 4 (Blackwell) Let $\pi = \{u_0, u_1, \dots\}$ and $\pi' = \{f, u_0, u_1, \dots\}$. If $V(\pi') < V(\pi)$, then for the stationary policy f^∞, $V(f^\infty) < V(\pi)$. ($<$ means $\leq$ for all elements with $<$ for some element.)

Proof. By hypothesis,

$$L(f) + \beta P(f)V(\pi) < V(\pi),$$

$$L(f) + \beta P(f)[L(f) + \beta P(f)V(\pi)] < V(\pi),$$

thus,

$$L(f) + \beta P(f)L(f) + \beta^2 P^2(f)V(\pi) < V(\pi).$$

Continuing this substitution process,

$$\sum_{n=0}^{N-1} \beta^n P^n(f)L(f) + \beta^N P^N(f)V(\pi) < V(\pi).$$

Once again, as $N \to \infty$, $\beta^N P^N(f)V(\pi) \to 0$, and

$$V(f^\infty) = \sum_{n=0}^{\infty} \beta^n P^n(f)L(f) < V(\pi),$$

completing the proof.

Theorem 5 (Howard) If A is finite, then there is an optimal stationary policy.

Proof. Consider any stationary policy g^∞, then either

(a) $\ell({}^i x, \alpha_i) + \beta \sum_j p_{ij}(\alpha_i) v(j, g^\infty) \geq v(i, g^\infty)$

for all $\alpha_i \varepsilon A$

$i = 1, 2, \ldots, J$

or

(b) $\ell({}^i x, \alpha_i) + \beta \sum_j p_{ij}(\alpha_i) v(j, g^\infty) < v(i, g^\infty)$

for some $\alpha_i \varepsilon A$

and some i

If (a) holds, then for any $f \varepsilon U$, $f({}^i x) = \alpha_i$, the policy $\pi' = (f, g, g, \ldots)$ is more costly than the stationary policy g^∞, i.e.,

$$V(g^\infty) \leq V(\pi')$$

and by Theorem 3 g^∞ is optimal. On the other hand, if g^∞ is not optimal, i.e., there is some i for which (b) holds, then a new control function, u, is defined such that for all i,

$$u({}^{i}x) = \begin{cases} g({}^{i}x), \text{ for case (a)} \\ \alpha_i, \text{ for case (b).} \end{cases}$$

Then by the construction of u, for the policy $\pi_u = \{u, g, g, \ldots\}$

$$V(\pi_u) < V(g^\infty)$$

By Theorem 4,

$$V(u^\infty) < V(g^\infty).$$

Thus, we have a policy, u^∞, which improves upon g^∞. Since A is finite, there are only a finite number of stationary policies. Thus, there is one which has no improvement and is optimal.

The motivation for restricting the class of control laws studied to those that are stationary is contained in Theorem 2; it is seen that any optimal policy may be replaced by a stationary optimal policy. Theorem 5 lays the basis for a constructive method of finding this optimal stationary policy, Howard's iteration in policy space. In the next section this procedure is explained. The set of admissible control policies is taken to be stationary; thus, for notational convenience the policy $u^\infty = \{u, u, \ldots\}$ and the control function $u \epsilon U$ are considered to be equivalent, and $V(u) = V(u^\infty)$.

2.5 Howard's Policy Iteration for $0 \leq \beta < 1$

Before the method of policy improvement contained in the proof of Theorem 5 can be applied, there must be a means of obtaining the expected cost vector, $V(u)$, for any $u \in U$. Consider any stationary policy, u, over n stages, then let,

$$v_n(i, u) \triangleq E\left\{ \sum_{k=0}^{n} \beta^k \ell\left(\underset{\sim}{x}_k, u(\underset{\sim}{x}_k)\right) \middle| \underset{\sim}{x}_0 = {}^i x \right\} \tag{2.5}$$

$$= \ell_i(u) + \beta E\left\{ E\left\{ \sum_{k=0}^{n} \beta^k \ell\left(\underset{\sim}{x}_k, u(x_k)\right) \middle| \underset{\sim}{x}_1 \right\} \underset{\sim}{x}_0 = {}^i x \right\}$$

$$v_n(i, u) = \ell_i(u) + \beta \sum_j p_{ij}(u) v_{n-1}(j, u). \tag{2.6}$$

or in matrix form

$$V_n(u) = L(u) + \beta P(u) V_{n-1}(u)$$

$$= L(u) + \beta P(u) L(u) + \ldots + \beta^n P^n(u) L(u) \tag{2.7}$$

The stage cost function $\ell\left({}^i x, u({}^i x)\right)$ is bounded for all i by definition. Let this bound be M. Then

$$V_n(u) \leq M1 + \beta P(u) M1 + \ldots + \beta^n P^n(u) M1$$

$$1 = (1, 1, \ldots, 1)^T$$

$$V_n(u) \leq (1 + \beta + \ldots + \beta^n) M1$$

$$< \frac{M}{1-\beta} 1 \text{ for all } 0 \leq \beta < 1$$

It is apparent that the sequence

$$v_0(i, u), v_2(i, u), \ldots, v_n(i, u), \ldots$$

is monotonically increasing for all i. Since $v_n(i, u)$ is bounded, it follows that the limit exists.

Say, $\lim_{n \to \infty} v_n(i, u) = v(i, u)$.

This limit is

$$v(i, u) = E\left\{ \sum_{k=0}^{\infty} \beta^n \ell\left(\underset{\sim}{x}_n, u(\underset{\sim}{x}_n) \right) \middle| \underset{\sim}{x}_0 = {}^i x \right\},$$

or the total excepted cost of applying the policy $u \epsilon U$ from (2.3).

Again taking the limit as $n \to \infty$ from (2.6),

$$v(i, u) = \ell_i(u) + \beta \Sigma p_{ij}(u) v(j, u), \tag{2.8}$$

or

$$V(u) = L(u) + \beta P(u) V(u) \tag{2.9}$$

Thus,

$$[I - \beta P(u)] V(u) = L(u),$$

and

$$V(u) = [I - \beta P(u)]^{-1} L(u), \tag{2.10}$$

if the inverse exists.

To establish the existence of the inverse, consider an arbitrary stochastic matrix P. $[I - \beta P]^{-1}$ exists if and only if $\det[I - \beta P] \neq 0$, or $\det[\lambda I - P] \neq 0$ where

$\lambda = \frac{1}{\beta}$, $\beta \neq 0$. However, for a stochastic matrix P all eigenvalues are of magnitude equal to or less than one. Thus, $\det[\lambda I - P] = 0$ only if $\lambda \leq 1$, but $0 \leq \beta < 1$ implies $\lambda > 1$. Therefore, $\det[\lambda I - P] \neq 0$, or equivalently $\det[I - \beta P]^{-1} \neq 0$, and the inverse exists.

Another useful result follows immediately. For a fixed policy $u \epsilon U$ the cost $V(u)$ is a continuous function of β. Consider

$$V(u) = [I - \beta P(u)]^{-1} L(u).$$

It is apparent that the elements of the inverse are rational functions of β with no singularities for $0 \leq \beta < 1$. Thus, $v(i, u)$ is a continuous function of β.

Howard's policy iteration is a two-step iterative process as follows:

(1) for a given stationary policy $u \epsilon U$ determine

$$V(u) = [I - \beta P(u)]^{-1} L(u),$$

and go to step 2 with $V = V(u)$;

(2) for the cost function $V = (v_1, v_2, \ldots, v_J)^T$ select $u \epsilon U$ such that $u({}^i x) \epsilon A$ minimizes

$$\ell\left({}^i x,\ u({}^i x)\right) + \beta \sum_j p_{ij}\left(u({}^i x)\right) v_j, \quad i = 1, \ldots, J$$

and repeat step 1.

The process is terminated when step 2 yields no further improvement. The resulting u is the optimal stationary policy by Theorem 4 for a finite control set A. The last V generated by the process is the total expected cost vector for the optimal policy u. The policy iteration procedure can be started at either step 1 or step 2. If there is no convenient policy to assume for initiating the process, that is, if there is no policy suspected to be near the optimum, then it is attractive to let $V = 0$ initially. This results in the first policy iteration improving upon the stage cost--a reasonable procedure if no additional knowledge is available about the optimum.

2.6 Direct Dynamic Programming

An alternative to considering the infinite duration process with a stationary control just solved by policy iteration is to examine a finite duration process. An optimal control sequence which minimizes the expected cost over n time periods is sought. The conventional dynamic programming functional equation results, and taking the limit as $n \to \infty$ the same control is obtained as by policy iteration. Consider,

$$v_n(i) \triangleq \min_{\{u_0, u_1 \ldots u_n\}} \left\{ E \left\{ \sum_{k=0}^{n} \beta^k \ell \left| \underset{\sim}{x}_k, u_k(\underset{\sim}{x}_k) \right| + \beta^{n+1} \Theta(\underset{\sim}{x}_{n+1}) \,\middle|\, \underset{\sim}{x}_0 = {}^i x \right\} \right\}$$

$$= \min_{u_0} \left\{ \ell_i(u_0) + \beta \sum_{j_1} p_{ij_i}(u_0) \min_{u_1} \left\{ \ell_{j1}(u_1) + \ldots \right.\right.$$

$$\left. \ldots + \min_{u_n} \left\{ \ell_{jn}(u_n) + \beta \sum_{jn+1} p_{j_n j_{n+1}}(u_n) \Theta({}^{j_{n+1}}x) \right\} \right\} \ldots \Big\}$$

$$v_n(i) = \min_{u_0} \left\{ \ell_i(u_0) + \beta \sum_j p_{ij}(u_0) v_{n-1}(j) \right\}, \tag{2.11}$$

where $0 \leq v_{-1}(i) = \Theta({}^i x) \leq M$ is an arbitrary terminal cost, $i = 1, \ldots, J$. As before, the set of cost functions $\{v_0(i), v_1(i), \ldots, v_n(i)\}$ is bounded for all $i = 1, \ldots, J$ since

$$0 < v_n(i) \leq M(1 + \beta + \ldots + \beta^{n+1}) < \frac{M}{1-\beta},$$

where $M = \max\limits_{i, \alpha \in A} \ell({}^i x, \alpha)$.

Let $v_{-1}({}^i x) = M/1-\beta$ for $i = 1, \ldots, J$; for this terminal cost $v_n(i)$ decreases monotonically. To show this, observe that

$$v_0(i) \leq v_{n-1}(i) \text{ for all } i.$$

Now assume

$$v_n(i) \le v_{n-1}(i) \text{ for all } i,$$

and show

$$v_{n+1}(i) \le v_n(i) \text{ for all } i.$$

$$v_n(i) = \min_u \left\{ \ell_i(u) + \beta \sum_j p_{ij}(u) v_{n-1}(j) \right\}$$

$$= \ell_i(u^*) + \beta \sum_j p_{ij}(u^*)\, v_{n-1}(j), \text{ say.}$$

$$v_{n+1}(i) = \min_u \left\{ \ell_i(u) + \beta \sum_j p_{ij}(u)\, v_n(j) \right\}$$

$$\le \ell_i(u^*) + \beta \sum_j p_{ij}(u^*)\, v_n(j)$$

$$\le \ell_i(u^*) + \beta \sum_j p_{ij}(u^*)\, v_{n-1}(j)$$

Thus $v_{n+1}(i) \le v_n(i)$, and $v_n(i)$ is seen to decrease monotonically. Again, since the sequence $\{v_0(i), v_1(i), \ldots\}$ is monotonically decreasing and bounded below by zero, it has a limit as $n \to \infty$, say, $v(i)$. Taking this limit in (2.11)

$$v({}^i x) = \min_u \left\{ \ell_i(u) + \beta \sum_j p_{ij}(u)\, v(j) \right\}. \tag{2.12}$$

Thus, (2.12) defines the expected cost function for an optimal policy over an infinite duration. Furthermore, it can be established that the solution to the equation is unique. Assume to the contrary that two solutions, v(i) and y(i) exist with associated control functions u and s. Then

$$v(i) = \ell_i(u) + \beta \sum_j p_{ij}(u)\, v(j),$$

$$y(i) \leq \ell_i(u) + \beta \sum_j p_{ij}(u)\, y(j).$$

Subtracting yields,

$$[y(i) - v(i)] \leq \beta \sum_j p_{ij}(u) \quad [y(j) - v(j)].$$

By successive substitution,

$$[y(i) - v(i)] \leq \beta^n \sum_j p_{ij}(n,u)\, [y(j) - v(j)].$$

Taking the limit as $n \rightarrow \infty$,

$$y(i) - v(i) \leq 0 \text{ for all } i.$$

By a symmetrical argument,

$$y(i) - v(i) \geq 0 \text{ for all } i.$$

Thus,

$$y(i) = v(i),$$

and the solution $V = \left| v(1), \ldots, v(J) \right|^T$ to (2.12) is seen to be unique. Also, by letting $n \rightarrow \infty$ the control which results from dynamic programming is optimal for the original cost function (2.3), since

$$v(i) = \min_{u} \left\{ \ell_i(u) + \beta \sum_j p_{ij}(u)\ v(j) \right\}$$

$$= \ell_i(u^*) + \beta \sum_j p_{ij}(u)\ v(j), \text{ say.}$$

There exists no $g \epsilon U$ such that,

$$\ell_i(g) + \beta \sum_j p_{ij}(g)\ v(j) \leq \ell_i(u^*) + \beta \sum_j p_{ij}(u^*)\ v(j).$$

It is seen that the solution to (2.12) is the same as the solution of Howard's policy iteration procedure. Thus, the solution of the dynamic programming iterative equation

$$V_n = \min_{u} \left\{ L_1(u) + \beta P(u)\ V_{n-1} \right\}$$

as $n \rightarrow \infty$ yields the same cost function as does policy iteration. It is also apparent that, if the limiting control function resulting from dynamic programming is used as a stationary policy, then this policy is the same as the one resulting from policy iteration.

One important question still remains unanswered. What is the rate of convergence of the dynamic programming solution to the stationary optimum? As before, the sequence

$$v_0(i),\ v_1(i),\ v_2(i)\ \ldots$$

decreases monotonically to $v(i)$, where

$$v(i) = \min_{u} \left\{ \ell_i(u) + \beta \sum_j p_{ij}(u)\, v(j) \right\}$$

$$= \ell_i(u^*) + \beta \sum_j p_{ij}(u^*)\, v(j).$$

$$v_n(i) = \min_{u} \left\{ \ell_i(u) + \beta \sum_j p_{ij}(u)\, v_{n-1}(j) \right\}$$

$$\leq \ell_i(u^*) + \beta \sum_j p_{ij}(u^*)\, v_{n-1}(j).$$

Thus,

$$v_n(i) - v(i) \leq \beta \sum_j p_{ij}(u^*)\, [v_{n-1}(j) - v(i)]$$

Let,

$$\epsilon_n \triangleq \max_i \{v_n(i) - v(i)\} > 0,$$

then,

$$\epsilon_n \leq \beta\, \epsilon_{n-1}. \tag{2.13}$$

The maximum deviation of V_n from V thus decreases at a rate of at least β. Practical experience (Beckman, 1968) shows that this estimate of the rate of improvement is quite close. It is seen that for β less than about .7 the rate of convergence is very rapid.

The maximum error, ϵ_n, is, of course, impossible to obtain during the dynamic programming algorithm since the final cost V is unknown. A bound on ϵ_n can however be found.

Let

$$\delta_n = \max_i \{v_{n-1}(i) - v_n(i)\} > 0$$

As before, for

$$v_n(i) = \ell_i(u^*) + \beta \sum_j p_{ij}(u^*)\, v_{n-1}(j),$$

$$v_{n-1}(i) \leq \ell_i(u^*) + \beta \sum_j p_{ij}(u^*)\, v_{n-2}(j).$$

Thus,

$$v_{n-1}(i) - v_n(i) \leq \beta \sum_j p_{ij}(u^*)\, [v_{n-1}(j) - v_{n-2}(j)],$$

and

$$\delta_n \leq \beta\, \delta_{n-1}.$$

However,

$$\epsilon_n = \delta_{n+1} + \delta_{n+2} + \dots$$

$$\leq \delta_n\, [\beta + \beta^2 + \dots]$$

or

$$\epsilon_n \leq \delta_n \frac{\beta}{1-\beta} \qquad (2.14)$$

Thus the error ϵ_n is bounded by the observable stage difference, δ_n. The dynamic programming algorithm can be terminated when δ_n gets sufficiently small.

2.7 Howard's Policy Iteration for $\beta = 1$

The control of finite Markov chains with $\beta = 1$ (i.e., no discounting) is somewhat more difficult to examine than the discounted cost chains. It is convenient to assume not only a finite set of stationary control laws, but also to restrict A such that for any $u \in U$ the resulting Markov chain is ergodic. Before defining what optimal control means for the undiscounted costs, the behavior of the cost function is examined.

Let,

$$v_n(i,u) = E\left\{\sum_{k=0}^{n} \ell\left(\underset{\sim}{x}_k, u(\underset{\sim}{x}_k)\right) \middle| \underset{\sim}{x}_0 = {}^i x\right\} \tag{2.15}$$

be the undiscounted expected cost function for the stationary policy, u, applied to n stages. Then, as before,

$$v_n(i,u) = \ell_i(u) + \sum_j p_{ij}(u)\, v_{n-1}(j,u) \tag{2.16}$$

with

$$v_{-1}(i) = 0,$$

or

$$v_n(i,u) = \ell_i(u) + \sum_j p_{ij}(u)\, \ell_j(u) + \dots$$
$$+ \sum_j p_{ij}(n,u)\, \ell_j(u)$$

In matrix form

$$V_n(u) = L(u) + P(u)\, V_{n-1}(u)$$

$$= L(u) + P(u)\, L(u) + \ldots + P^n(u)\, L(u) \qquad (2.17)$$

By Theorem 1,

$$P^n(u) = 1\mu + Q_n(u)$$

where $Q_n(u) \rightarrow 0$ as $n \rightarrow \infty$, geometrically fast.

Consider,

$$\lim_{n\to\infty} n^{-1} V_n(u) = \lim_{n\to\infty} n^{-1} \sum_{m=0}^{n} P^m(u)\, L(u), \text{ if the limit exists,}$$

$$= \lim_{n\to\infty} n^{-1} \sum_{m=0}^{n} 1\mu\, L(\mu) + \lim_{n\to\infty} n^{-1} \sum_{m=0}^{n} Q_m(u)\, L(u)$$

However,

$$\lim_{n\to\infty} n^{-1} \Sigma\, Q_m(u)\, L(u) = 0 \text{ since } \lim_{n\to\infty} Q_n(u) = 0.$$

Thus,

$$\lim n^{-1}\, V_n(u) = [\mu L(u)]1$$

and for large n,

$$V_n(n) \simeq n[\mu L(u)]1 + \text{constant}$$

$$V_n(u) \simeq ng(u)1 + W(u), \text{ say.} \qquad (2.18)$$

The scalar $g(u) = \mu L(\mu)$ is the stationary average cost of the policy u, and the vector W(u) is called the potential of the policy. Also, a n-stage potential $W_n(u)$ can be defined as,

$$W_n(u) = \sum_{m=0}^{n} Q_m(u)\ L(u) \tag{2.19}$$

so that

$$V_n(u) = ng(u)1 + W_n(u). \tag{2.20}$$

Thus,

$$\lim_{n\to\infty} W_n(u) = W(u),$$

and substituting (2.17) into (2.20),

$$ng(u)1 + W_n(u) = L(u) + P(u)\ [(n-1)g(u)1 + W_{n-1}(u)],$$

or

$$W_n(u) + g(u)1 = L(u) + P(u)\ W_{n-1}(u),$$

with $W_{-1}(u) = 0$.

In the limit as $n \to \infty$,

$$W(u) + g(u)1 = L(u) + P(u)\ W(u) \tag{2.21}$$

A stationary policy $u^*\epsilon U$ is said to be optimal if

$$g(u^*) \leq g(u) \text{ for all } u\epsilon U.$$

That is, the optimal policy for $\beta = 1$ is the one which accrues the least average cost.

The question arises, does (2.21) determine $g(u)$ and $W(u)$ uniquely? To answer this, consider two solutions, W,g

and Y,a for the same policy u. Equation (2.21) immediately yields,

$$W-Y + (g-a)1 = P(u)[W-Y]$$

or

$$Z = P(u)Z + C$$

where

$$C = (a-g)1, \ Z = W-Y.$$

Thus,

$$Z = nC + P(u)^n Z$$

$$\rightarrow nC + 1\mu Z \text{ as } n \rightarrow \infty.$$

However, the elements of Z are bounded as $n \rightarrow \infty$, thus $C \equiv 0$ and $g = a$. Therefore, the stationary average cost is determined uniquely by (2.21). Now, with $C \equiv 0$, in the limit as $n \rightarrow \infty$,

$$Z = 1\mu Z, \ \sum_{j=1}^{J} \mu_j = 1 \text{ with } \mu_j > 0.$$

The only solution to this equation is

$$Z_i = \text{constant}.$$

Therefore, the potential, W(u), for a given policy, $u \epsilon U$, is determined up to an additive constant by (2.21).

Howard's policy iteration for undiscounted cost may now be specified as follows:

1. For a given stationary policy, $u \epsilon U$, determine g(u) and W(u) from

$$W(u) + g(u)1 = L(u) + P(u)\ W(u)$$

and go to step (2) with $W = W(u)$.

2. For the potential function, W, select u such that $u({}^{i}x)$ minimizes

$$\ell\left({}^{i}x, u({}^{i}x)\right) + \sum_j p_{ij}(u) W_j \quad i = 1, 2, \ldots, J$$

and repeat step (1).

Again, the process is terminated when there is no further improvement in $g(u)$, or equivalently when the policy u ceases to change in step 2. To show that the policy iteration indeed yields an optimum stationary policy, consider any policy $u \varepsilon U$, then

$$W(i,u) + g(u) = \ell_i(u) + \sum_j p_{ij}(u)\ W(j,u) \tag{2.22}$$

A new policy, $\hat{u} \varepsilon U$ is generated by minimizing the right hand side of (2.22). It is apparent that the additive constant in W as determined in (2.22) does not affect $\hat{u}$. Now,

$$W(i,\hat{u}) + g(\hat{u}) = \ell_i(\hat{u}) + \sum_j p_{ij}(\hat{u})\ W(j,\hat{u}),$$

where

$$W(i,u) + g(u) \geq \ell_i(\hat{u}) + \sum_j p_{ij}(\hat{u})\ W(j,u),$$

and $>$ applies for some i.

Thus,

$$W(i,u) - W(i,\hat{u}) + g(u) - g(\hat{u}) \geq \sum_j p_{ij}(\hat{u})\ [W(j,u) - W(j,\hat{u})] \tag{2.23}$$

Recalling that for the stationary probability distribution, $\hat{\mu}$, associated with $\hat{u}$,

$$\sum_{k=1}^{J} \hat{\mu}_k \, p_{ij}(\hat{u}) = \hat{\mu}_j$$

and multiplying, (2.23) by μ_i and summing yields,

$$\sum_i \hat{\mu}_i[W(i,u) - W(i,\hat{u})] + g(u) - g(\hat{u}) > \sum_j \hat{\mu}_j[W(j,u) - W(j,\hat{u})]$$

or

$$g(u) - g(\hat{u}) > 0$$

Therefore, $g(\hat{u}) < g(u)$ and the policy, $\hat{u}$, generated by policy iteration is superior to u, the policy which preceded it. Since there are only a finite number of policies eventually there occurs a policy which can not be improved upon in step (2). This policy is the optimal policy.

2.8 The Optimal Control as $\beta \rightarrow 1$

The question arises whether the optimal control for $\beta < 1$ but sufficiently close to one is the same as the control for $\beta = 1$. The answer rests in the continuity of the cost surfaces with respect to β; however, the mathematical structure necessary to examine the question is extensive so the result is stated. The control is indeed the same for $\beta = 1$ and $\beta < 1$ but close to one. That is, the optimal control derived under the discounted expected cost criteria for β close to one is the same as the control derived under the stationary average cost criteria for $\beta = 1$. This result was obtained by Blackwell (1962) and later considered more extensively by Miller and Veinott (1969). Thus the undiscounted problem can be solved by solving the discounted problem for β sufficiently close to one. This result is stated for completeness. In the remaining three chapters only discounted problems are considered.

CHAPTER 3

A NUMERICAL ALGORITHM FOR OPTIMAL CONTROL

3.1 Introduction

In Chapter 2 the characteristics of the expected cost function were examined, and two methods, Howard's policy iteration and dynamic programming, were developed for obtaining the optimal control of finite Markov chains. In this chapter stochastic systems whose state space is defined on the continuum are considered. However, rather than view these systems rigorously as infinite state diffusion processes, they will be considered as finite Markov chains with a large but finite discrete state space. A numerical algorithm which employs a quadratic approximation to the expected cost function for a partitioned state space will be developed.

3.2 System Description

The systems to be studied are defined by a set of difference equations

$$x(k+1) = f\left[x(k),\ \alpha(k)\right] + \zeta(k) \qquad (3.1)$$

called the plant equation, where

k = time parameter

x = n-dimensional state vector

α = q-dimensional control vector

ζ = n-dimensional random vector, plant noise

f = n-dimensional vector function.

The state $x = (x_1, x_2, \ldots, x_n)^T$ is restricted to the state space $X = \{x | xmin_i \le x_i \le xmax_i, \ i = 1, \ldots, n\}$ and any transition out of this region under (3.1) is not considered. The control $\alpha = (\alpha_1, \ldots, \alpha_q)^T$ is restricted to the control space A. The random variable ζ, called the plant noise, has a known probability density function, $p_\zeta(\zeta)$, which is time invariant, and ζ is independent from one time instant to another. If it is desired to model a system with correlation between plant noise from one time instant to the next, it is possible to define additional state variables and new random variables for which the plant noise is independent (Meier, 1966). Also, with no loss of generality ζ is considered to have zero mean.

Stochastic control systems with continuous state space can be considered, as an approximation, to be finite Markov chains by establishing a grid on the state space X. The grid points are states of the finite Markov chain and the transition probabilities, p_{ij}, defining the chain under a stationary control law, are obtained by determining the probability of a state transition from ${}^i x$ on the grid to a hypercube about ${}^j x$ on the grid. To better illustrate this, consider the second order system in Figure 3.1. The

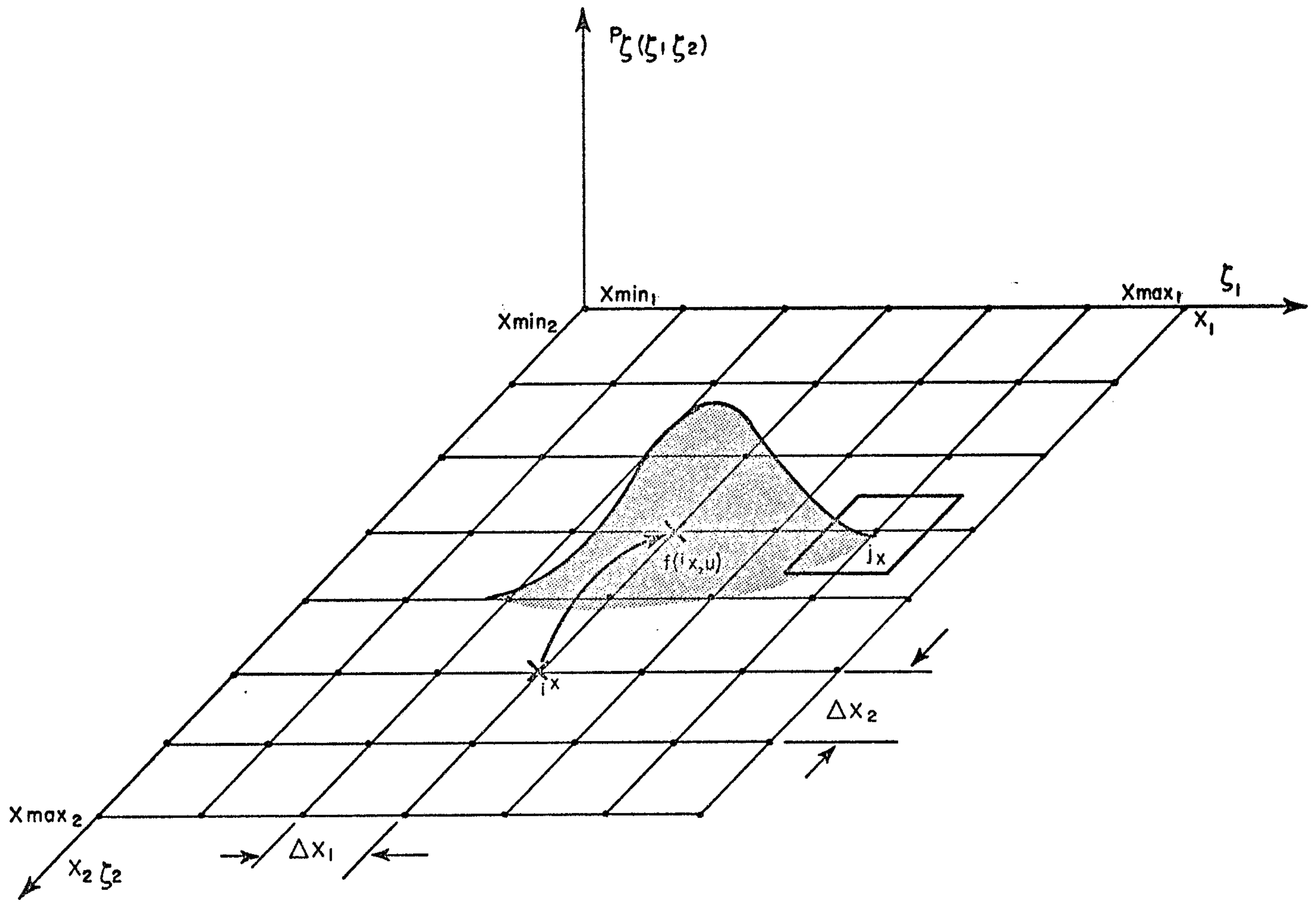


Figure 3.1 Grid for discrete state approximation to a continuous state system.

transition probability $p_{ij}(\alpha)$ under control α is defined as

$$p_{ij}(\alpha) = \int_{{}^{j}x_1 - \frac{1}{2}\Delta x_1 - f_1({}^{i}x,\alpha)}^{{}^{j}x_1 + \frac{1}{2}\Delta x_1 - f_1({}^{i}x,\alpha)} \int_{{}^{j}x_2 - \frac{1}{2}\Delta x_2 - f_2({}^{i}x,\alpha)}^{{}^{j}x_2 + \frac{1}{2}\Delta x_2 - f_2({}^{i}x,\alpha)} p_{\zeta}(\zeta_1,\zeta_2)d\zeta_1\, d\zeta_2 \qquad (3.2)$$

The stage cost at time k is defined, as before,

$$0 < \ell\,\big(x(k),\ \alpha(k)\big) < M.$$

The total expected cost function is, as in Chapter 2, for a stationary control law, u,

$$v(x) = E\left\{ \sum_{k=0}^{\infty} \beta^k \ell\Big(\underset{\sim}{x}(k),\ u\big(\underset{\sim}{x}(k)\big)\Big) \;\Big|\; \underset{\sim}{x}(0) = x_0 \right\}$$

Again the control law u with $u\big(x(k)\big) \epsilon A$ is sought which minimizes v(x) for all $x \epsilon X$, or, for the finite Markov chain representation, all x which are grid points.

As before,

$$v(x) = \ell\big(x_0,\ u(x_0)\big) + \beta E\Big\{ v\Big(f\big(x_0,\ u(x_0)\big) + \zeta_0\Big)\Big\} \qquad (3.3)$$

3.3 Solution by Howard's Policy Iteration

To find the optimal control via policy iteration it is first necessary to model the system as a finite Markov chain. A grid must be established which is sufficiently fine to approximate the behavior of the system defined on the continuum. Dividing each coordinate x_i into N_i equal increments Δx_i wide accomplishes this for Δx_i small enough,

$$N_i = \frac{xmax_i - xmin_i}{\Delta x_i}, \; i = 1, \ldots, n,$$

and defines $J = \prod_{i=1}^{n} N_i$ grid points.

Now to obtain transition probabilities $p_{ij}(u)$ under the stationary control u it will be necessary to perform the integration in (3.2) J^2 times. Then having obtained the J x J transition matrix

$$P(u) = [p_{ij}(u)],$$

step one of the policy iteration procedure (Section 2.5) requires inverting

$$[I - \beta P]$$

also, a J x J matrix. In the minimization in step two, it will again be necessary to evaluate (3.2) J^2 times for each control law considered. The number of control laws considered will depend on the numerical minimization technique used, but it is evident that this number could be large even for limited control spaces. To see the prodigious labor necessary to employ Howard's policy iteration for systems with continuous state space, consider a second order example with

$$X = \{1 \leq x_1 \leq 100,\ 1 \leq x_2 \leq 100\}$$

and let $\Delta x_1 = \Delta x_2 = 1$. Then $N_1 = N_2 = 100$, and $J = 10^4$. Thus P has 10^8 elements as does $[I - \beta P]$. Already it is evident that while Howard's policy iteration is a valuable technique for finding the optimal control of finite Markov chains with very few states and a useful theoretical tool, it is impractical to employ it on the systems defined in this chapter. It would be necessary in the present example to store 100 million transition probabilities in computer storage and invert a $10^4 \times 10^4$ matrix to achieve only step one of the first iteration of Howard's method--clearly an overwhelming computational task. On the other hand, it will be shown in the next section that dynamic programming as developed in Section 2.6 offers a more palatable numerical technique.

3.4 Solution by Dynamic Programming

To employ dynamic programming, as before, a N-stage minimum expected cost function is defined,

$$v(x_0,N) = \min_{\{u_0,u_1,\ldots u_n\}} E\left\{\sum_{k=0}^{N} \beta^k \ell\left(\underset{\sim}{x}(k),u_k\left(x(k)\right)\right)\Big|\underset{\sim}{x}(0) = x_0\right\}$$

or

$$v(x\ ,N) = \min_{u_0}\left\{\ell\left(x_0,u_0(x_0)\right) + \beta E\left\{v\left(f\left(x_0,u_0(x_0)\right)\right.\right.\right.$$

$$\left.\left.\left. + \zeta_0,N-1\right)\Big|x_0\right\}\right\} \quad (3.4)$$

with $v(x,0) = 0$. Again a grid is imposed on the state space with N_i increments along the x_i axis and $J = \prod_{i=1}^{n} N_i$ total grid points. It would now be possible to employ (3.2) to define the J x J transition matrix P and (3.3) would become, as in the last chapter,

$$v({}^{i}x,N) = \min_{u} \left\{ \ell\left[{}^{i}x,u({}^{i}x)\right] + \beta \sum_{j=1}^{J} p_{ij}(u)v({}^{i}x,N-1) \right\}$$

for all the grid points. However, to avoid the difficulty of obtaining P, a more convenient approximation is to quantify the noise in a manner similar to imposing a grid on the state space. That is, the probability density function $p_\zeta(\zeta)$ is approximated by imposing a grid on the domain of p_ζ and attaching a probability to each grid point. Then the noise is described by the set of noise values $\{{}^{i}\zeta,\ i=1,\ \ldots,\ N_\zeta\}$ and the associated probabilities, $\{p({}^{i}\zeta),\ i=1,\ \ldots,\ N_\zeta\}$. Now equation (3.4) becomes,

$$v({}^{i}x,N) = \min_{u} \left\{ \ell\left[{}^{i}x,u({}^{i}x)\right] + \beta \sum_{j=1}^{N_\zeta} p({}^{j}\zeta)v\left[f({}^{j}x,u)+{}^{j}\zeta,N-1\right] \right\} \tag{3.5}$$

$v({}^{i}x,0) = 0$ for $i = 1, 2, \ldots, J$. Equations (3.1) and (3.5) describe the dynamic programming numerical algorithm for the solution of the stochastic control problem with discounted cost. While the dynamic programming functional equation (3.5) offers a solution to a wide range of problems analytically, the computational requirements of high-speed computer memory and computing time can become excessive except for simple problems. The memory requirements are the same as for deterministic problems while the computation time is more severe. To better observe these difficulties and to see that Bellman's "curse of dimensionality" not only affects memory requirements but also computing time in the stochastic control problem a more detailed examination of the algorithm is in order.

Since it was shown in the previous chapter that $\lim_{n\to\infty} v({}^{i}x,N) = v({}^{i}x)$ there is no necessity to store all the cost functions and control functions generated as (3.5) is solved. Only the last cost function and the present cost function, and control function that is being generated, need be stored. Thus, $3 \cdot J = 3 \prod_{i=1}^{n} N_i$ memory locations are required to store the information vital to the iteration of (3.4). Further, for economy in computation time, these values should be stored in high-speed memory (Larson, 1968) which for most computers is limited to about 10^5 words. Thus for the second order example of Section 3.3 it would be necessary to have available $3 \cdot 10^4$ high-speed memory locations. For a three dimensional state space with $N_i = 100$, $i = 1, 2, 3$, $3 \cdot 10^6$ storage locations would be necessary, overwhelming the capacity of nearly any computer. This "curse of dimensionality" is a severe limitation to the problems solvable by dynamic programming. A first order problem is shown in Figure 3.2. To evaluate $v({}^{i}x,k)$ with

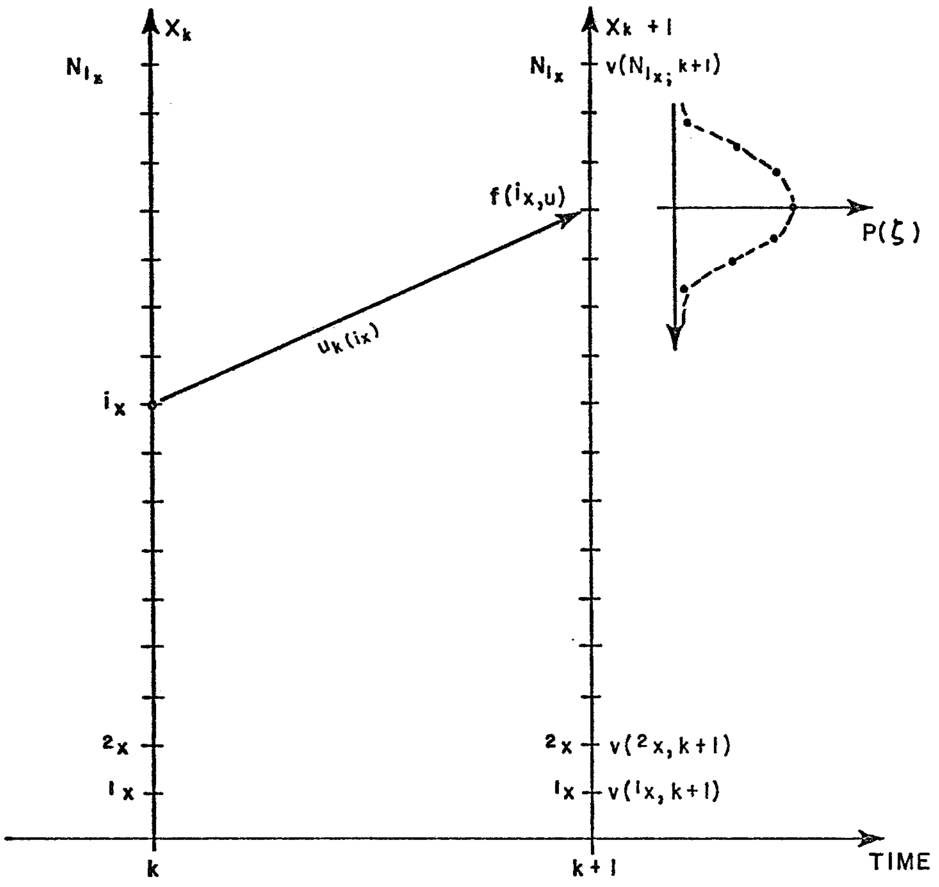


Figure 3.2 The dynamic programming numerical algorithm for first order problem.

the control $u_k({}^i x)$ applied, it is necessary to evaluate $v(x_{k+1},k+1)$ by interpolation of the stored cost function at time k+1 N_ζ times where N_ζ is the number of discrete noise levels used to approximate the probability density function $p_\zeta(\zeta)$.

For a second order plant with

$$x_1(k+1) = f_1\left|x(k),u\right| + \zeta_1(k)$$
$$x_2(k+1) = f_2\left|x(k),u\right| + \zeta_2(k),$$

and ζ_1 independent of ζ_2, both ζ_1 and ζ_2 could be quantified separately into say M_1 and M_2 levels. Thus, $N_\zeta = M_1 M_2$ and in general for a nth order plant $N_\zeta = \prod_{i=1}^{n} M_i$, and the cost function must be evaluated $J \prod_{i=1}^{n} M_i$ times for each iteration of (3.4). Consider each noise element quantified into, say, five levels where each state is perturbed by noise. The number of cost function evaluations necessary for the stochastic problem as opposed to the deterministic problem (Prob[ζ=0]=1) increases by a factor of five for each increase in dimensionality. Thus the "curse of dimensionality" affects the computation time of the stochastic problem with respect to the quantization of the noise. It is the main purpose of this work to develop an algorithm which alleviates the high-speed memory requirement and long computational time intrinsic to a straightforward application of dynamic programming to the stochastic control problem. The next section begins the development of this algorithm. A flow diagram of the dynamic programming algorithm is contained in Appendix B.

3.5 Dynamic Programming with a Partitioned State Space

The problem of excessive high-speed storage which is attendant to the dynamic programming algorithm was attacked with considerable success by Larson (1964, 1968) for the case of a deterministic plant and continuous time, i.e.,

$$\dot{x}(t) = f\left[x(t),u(t),t\right].$$

Larson's method, called state increment dynamic programming, took specific advantage of time being defined on the continuum. This restriction and the deterministic nature of his plant equation thwart a direct application of his technique to the discrete time stochastic problem under study. However, a basic concept of Larson's method will be employed for the problem at hand. State space will be partitioned into blocks, and these blocks will be treated individually in calculating the optimal control and cost function. The expected cost function, v(x), over each of these blocks will be approximated by a quadratic surface. The effect of this partition and the quadratic surfaces is to substantially reduce the amount of high-speed memory necessary and also to reduce the computation time. The price paid for these advantages is a more approximate control law than that achieved by conventional dynamic programming. However, the classes of systems examined will be restricted such that this loss of accuracy is not substantial.

To better illustrate these concepts, consider the second order problem and two dimensional state space in Figure 3.3. Here the state space has been partitioned into 25 blocks of equal dimension. There is no advantage in unequal dimensions, so for simplicity equal dimension blocks are used for the partition. The expected cost function is also partitioned into the surfaces above each block. In the figure the surface partitions above blocks 0 and 6 are illustrated. These surfaces are then to be approximated by a quadratic fit which in the two dimensional case will be, for block ℓ,

$$\bar{v}_{\ell}(x) = \alpha(\ell) + \beta_1(\ell)x_1 + \beta_2(\ell)x_2 + \gamma_{11}(\ell)x_1^2 + \gamma_{12}(\ell)x_1x_2 + \gamma_{22}(\ell)x_2^2$$

and for the nth order system,

$$\bar{v}_{\ell}(x) = \alpha(\ell) + \sum_{i=1}^{n} \beta_i(\ell)x_i + \sum_{i=1}^{n}\sum_{j=1}^{i} \gamma_{ij}(\ell)x_1x_2. \qquad (3.6)$$

The block size is selected such that, as illustrated in Figure 3.4, when ${}^{i}x$ is under consideration and control u is applied $f({}^{i}x,u)$ lies in the block containing ${}^{i}x$ or an adjacent block. This condition can be met easily enough by making the block size very large. However, since the cost function or surface over each block is to be approximated

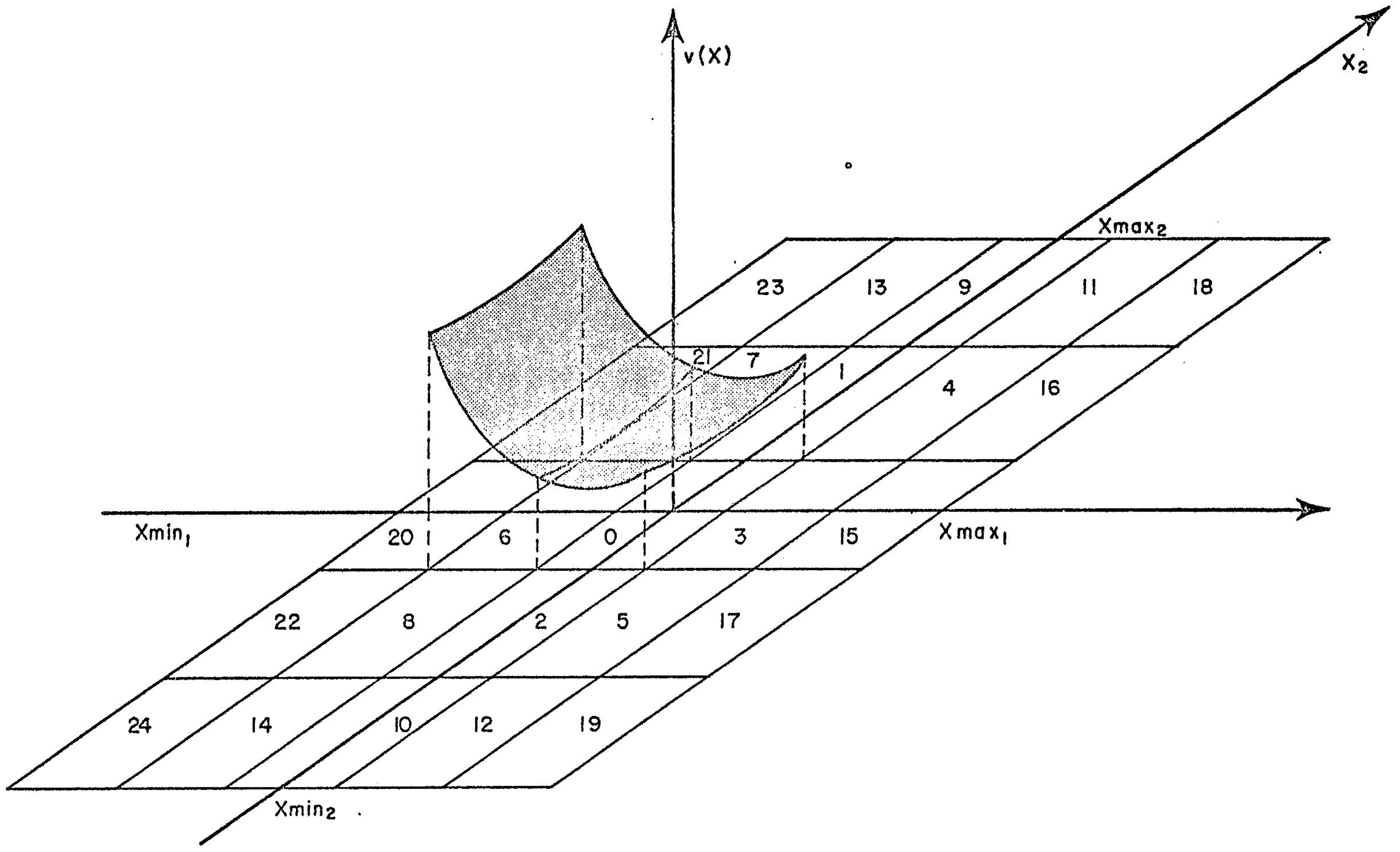


Figure 3.3 Partitioned two dimensional state space.

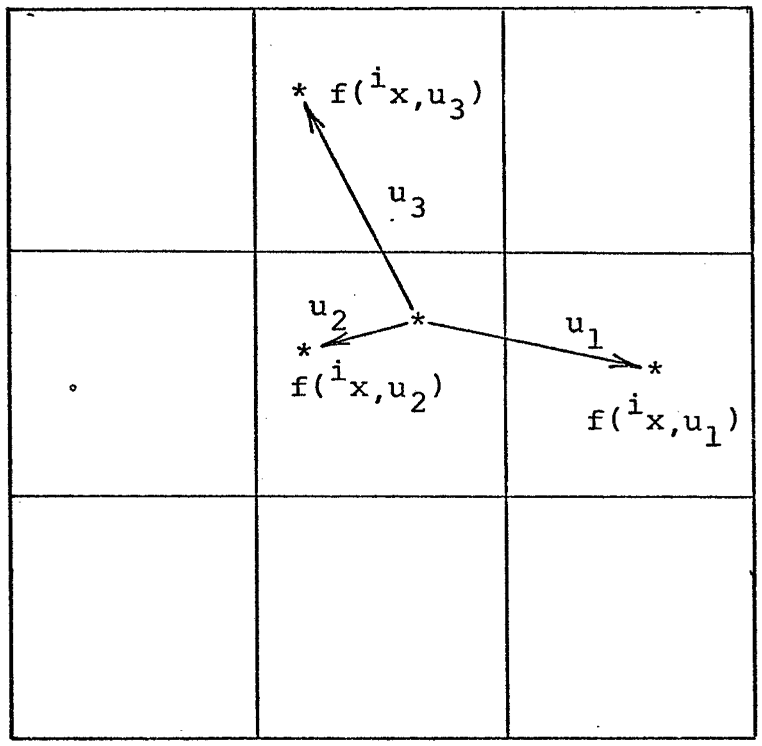


Figure 3.4 Transitions from the state ${}^ix$.

by a quadratic surface, it is also desirable to have the blocks small in size. Thus, a compromise must be reached, and this compromise obviously depends upon the problem being solved. A reflective examination of the system equations is usually adequate to determine an appropriate block size.

Consider for example that the state space in Figure 3.3 is $xmin_1 = xmin_2 = -25$ and $xmax_1 = xmax_2 = 25$ and that $\Delta x_1 = \Delta x_2 = 1$. Thus, each block would have 100 points in it (including its boundaries) with 10 increments to a side. The cost surface above each block would be described by 6 numbers, α, β's, and γ's. Since for each x a member of the ℓ^{th} block $(x \epsilon B_\ell)$, $f(x,u)$ is restrained to be a member of either B_ℓ or a block adjacent, it is possible to evaluate (3.3) for all points in B_ℓ with only the parametric description of B_ℓ and its adjacent blocks in high-speed memory. Thus, recalling Figure 3.3, only $9 \cdot 6 = 54$ high-speed memory locations are necessary to store the cost surface for the partitioned state space algorithm. For conventional dynamic programming $50^2 = 2500$ high-speed memory locations would be necessary.

Obviously, even for conventional dynamic programming it would be possible to store the entire cost function in low-speed memory (tape, disc, or drum storage). However, then it would be necessary to go to low-speed memory for each cost function evaluation. This is a time-consuming process which would involve $K \cdot \prod_{i=1}^{n} N_i \prod_{i=1}^{n} M_i$ accesses to low-speed storage where K is the number of controls evaluated at each state point. With K = 10 the example considered in Section 3.3 would require $10 \cdot 100^2 \cdot 25 = 2.5 \cdot 10^6$ accesses to low-speed memory. For the partitioned state space (PSS) algorithm only N_B accesses would have to be made to low-speed memory, where N_B is the number of blocks (25 accesses for the problem in Figure 3.2). In the next section the PSS algorithm is shown to reduce computation time as well as high-speed storage.

3.6 The Quadratic Approximation of the Cost Surface

The criteria for fitting the quadratic surface to the cost function over a given block is taken to be unweighted least squares regression. For block B_ℓ recall equation (3.6),

$$\bar{v}_\ell(x) = \alpha(\ell) + \sum_{i=1}^{n} \beta_i(\ell)x_i + \sum_{i=1}^{n} \sum_{j=1}^{i} \gamma_{ij}(\ell)x_i x_j$$

and the functional to be minimized is,

$$J(\alpha_1, \beta_1, \ldots, \beta_n, \gamma_n, \ldots, \gamma_{nn}) = \sum_{x \in B_\ell} \left| v(x) - \bar{v}_\ell(x) \right|^2$$

for the $M = \frac{n^2+3n+2}{2}$ parameters of the quadratic surface. Thus

$$\frac{\partial J}{\partial \alpha} = \frac{\partial J}{\partial \beta_i} = \frac{\partial J}{\partial \gamma_{ij}} = 0,$$

which yields,

$$\sum_{x\epsilon B_\ell}\left\{\alpha + \sum_{i=1}^{n} \beta_i x_i + \sum_{i=1}^{n}\sum_{j=1}^{i} \gamma_{ij}x_i x_j\right\} = \sum_{x\epsilon B_\ell} v(x), \quad (3.7a)$$

$$\sum_{x\epsilon B_\ell}\left\{\alpha x_k + \sum_{i=1}^{n} \beta_i x_i x_k + \sum_{i=1}^{n}\sum_{j=1}^{i} \gamma_{ij}x_i x_j x_k\right\} = \sum_{x\epsilon B_\ell} v(x)x_k$$

$$k=1, \ldots, n \quad (3.7b)$$

and

$$\sum_{x\epsilon B_\ell}\left\{\alpha x_k x_m + \sum_{i=1}^{n} \beta x_i x_k x_m + \sum_{i=1}^{n}\sum_{j=1}^{i} \gamma_{ij}x_i x_j x_k x_m\right\}$$

$$= \Sigma\, v(x)x_n x_m \qquad k=1, \ldots, n; \; m=1, \ldots, k. \quad (3.7c)$$

Equation (3.6) may be summarized in matrix form as,

$$SZ(\ell) = T(\ell) \quad (3.8)$$

where

$$Z(\ell) = (\alpha, \beta_1, \ldots, \beta_n, \gamma_{11}, \ldots, \gamma_{nn})^T$$

and

$$T(\ell) = \sum_{x\epsilon B_\ell} \left\{v(x), x_1 v(x), \ldots, x_n v(x), x_1^2 v(x),\right.$$

$$\left.\ldots, x_n^2 v(x)\right\}^T$$

are 1 x M column vectors, and S is the M x M matrix described by (3.8) such that (3.7) holds. Thus, the column vector, Z, describing the quadratic surface is

$$Z(\ell) = S^{-1}T(\ell).$$

It is not necessary to invert an S matrix for each block; instead, since all blocks have the same dimensions, S,T may be calculated for a block with standard coordinates, and v(x) transformed to this block. Thus the M x M matrix S need be inverted only once. Further, the storage for the surface for B_ℓ and adjacent blocks is $N_s = 3^n \cdot M$ locations. Thus,

$$n = 2 \Rightarrow N_s = 54$$

$$n = 3 \Rightarrow N_s = 270$$

$$n = 4 \Rightarrow N_s = 1215, \text{ etc.}$$

To see that the quadratic approximation not only reduces high-speed storage requirements but also computation time, recall equation (3.4),

$$v(x,N) = \min_u \left\{ \ell(x,u) + \beta E \left\{ v\left[f(x,u) + \zeta, N-1\right] \right\}\right\}.$$

For the noise quantified into N_ζ values (3.3) becomes (3.4),

$$v(x,N) = \min_u \left\{ \ell(x,u) + \beta \sum_{j=1}^{N_\zeta} P({}^j\zeta) v\left[f(x,u) + {}^j\zeta,\ N-1\right] \right\}.$$

Therefore, it is necessary to evaluate v(x,N-1) N_ζ times for each control considered, where N_ζ will have a tendency to increase geometrically with the dimension, n. On the other hand, for PSS dynamic programming with f(x,u) lying in block ℓ and parameters $Z(\ell)$ describing $v(x,N-1)$ for $x \varepsilon B_\ell$,

$$v(x,N) = \min_u \left\{ \ell(x,u) + \beta E \left\{ v\left[f(x,u) + \zeta, N-1 \right] \right\} \right\},$$

or, approximately,

$$v(x,N) = \min_u \left\{ \ell(x,u) + \beta E \left\{ \alpha(\ell) + \sum_{i=1}^{n} \beta_i(\ell) \left[f_i(x,u) + \zeta_i \right] \right. \right.$$

$$\left. \left. + \sum_{i=1}^{n} \sum_{j=1}^{i} \gamma_{ij}(\ell) \left[f_i(x,u) + \zeta_i \right] \left[f_j(x,u) + \zeta_j \right] \right\} \right\}$$

$$= \min_u \left\{ \ell(x,u) + \beta \left\{ \alpha(\ell) + \sum_{i=1}^{n} \beta_i(\ell) f_i(x,u) + \sum_{i=1}^{n} \sum_{j=1}^{i} \right. \right.$$

$$\gamma_{ij}(\ell) f_i(x,u) f_j(x,u)$$

$$\left. \left. + \sum_{i=1}^{n} \sum_{j=1}^{i} \gamma_{ij}(\ell) E \left\{ \zeta_i \zeta_j \right\} \right\} \right\}$$

$$v(x,N) = \min_u \left\{ \ell(x,u) + \beta \left\{ \bar{v}_\ell \left[f(x,u), N-1 \right] \right. \right.$$

$$\left. \left. + \sum_{i=1}^{n} \sum_{j=1}^{i} \gamma_{ij} E[\zeta_i \zeta_j] \right\} \right\}. \qquad (3.9)$$

Thus, only one cost function evaluation must be made for each control and the additional term,

$$\sum_{i=1}^{n} \sum_{j=1}^{i} \gamma_{ij}(\ell) E[\zeta_i \zeta_j]$$

calculated using known covariances, $E\{\zeta_i\zeta_j\}$. The cost function evaluation is of $\overline{v}(x,N-1)$ rather than $v(x,N-1)$; however, the computation time of the two evaluations is comparable. Thus, for PSS dynamic programming the computational task of N_ζ evaluations of $v(x,N-1)$ is traded for the computation of the quadratic surface,

$$Z(\ell) = S^{-1}T(\ell).$$

For most problems of dimension greater than one it is considerably less time consuming to find the parametric surface $\overline{v}(x,N-1)$ for a block than to evaluate $v(x,N-1)$ N_ζ times. The quadratic approximation to the cost function, therefore, affords a significant savings in cost function evaluations and computation time.

3.7 PSS Algorithm

Once the state space has been partitioned, the PSS dynamic programming algorithm can be applied. A flow diagram of the basic procedure is contained in Figure 3.5 while a FORTRAN program listing is to be found in Appendix A.

A particular block is designated as the origin block (for example, block 0 of Figure 3.3) and the cost surface associated with it is determined by techniques to be discussed in Section 3.9. The origin block is generally selected to contain the minimum of the cost function over

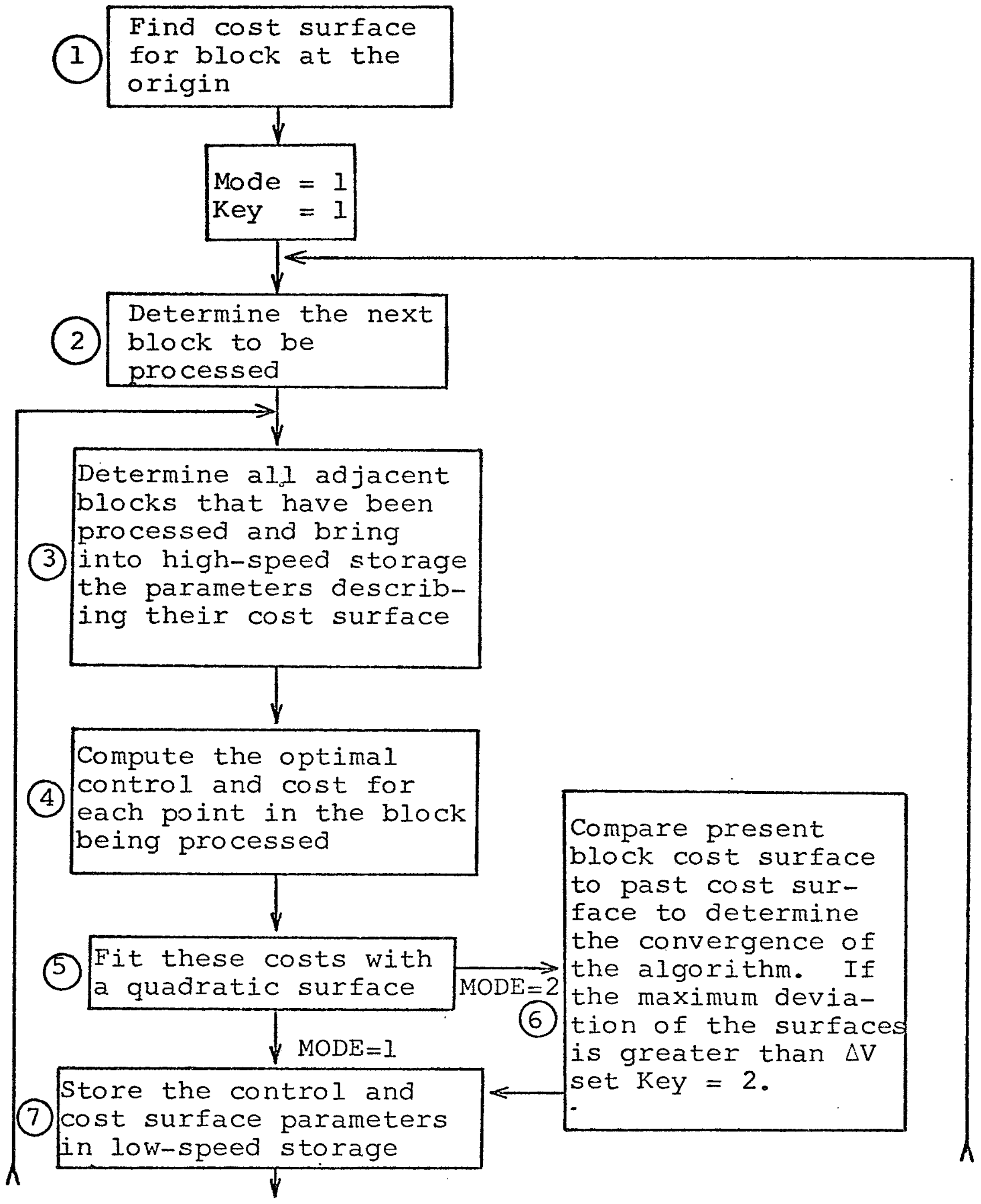

1
Find cost surface for block at the origin
Mode = 1
Key = 1
2
Determine the next block to be processed
3
Determine all adjacent blocks that have been processed and bring into high-speed storage the parameters describing their cost surface
4
Compute the optimal control and cost for each point in the block being processed
5
Fit these costs with a quadratic surface
MODE=2
6
Compare present block cost surface to past cost surface to determine the convergence of the algorithm. If the maximum deviation of the surfaces is greater than ΔV set Key = 2.
MODE=1
7
Store the control and cost surface parameters in low-speed storage

Figure 3.5 Flow diagram for dynamic programming with partitioned state space.

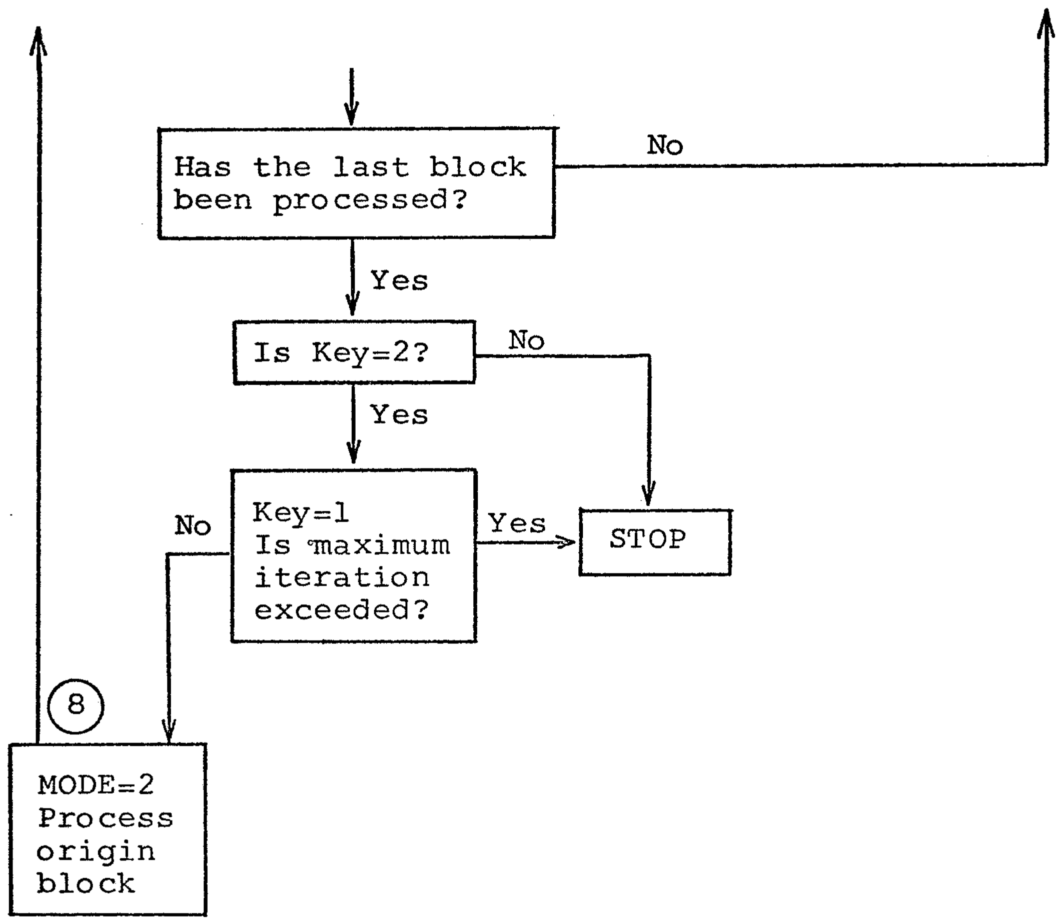

Has the last block been processed?
No
Yes
Is Key=2?
No
Yes
Key=1
Is maximum iteration exceeded?
No
Yes
STOP
8
MODE=2
Process origin block

Figure 3.5--*Continued*

all state space, if possible. For many problems it is easy to define the origin block appropriately, such as the stochastic regulator problem where the system is to be driven to the origin of state space.

With the cost surface for the origin block obtained, another block, say B_1, is considered for processing (Step 2). Both this block and all adjacent calculated blocks are brought into high-speed storage. The block being processed must have at least one calculated block next to it. This is not a significant restriction on the method, as, in general, the blocks are ordered in such a manner that they radiate out from the origin block as they are considered (Figure 3.3).

The optimal control and cost of each point $x \epsilon B_\ell$ is calculated (Step 4) by

$$v(x) = \min_u \left\{ \ell(x,u) + \beta \left\{ \overline{v}_m \left| f(x,u) \right| + \Sigma \, \Sigma \, \gamma_{ij}(m) E[\zeta_i \zeta_j] \right\} \right\} \tag{3.10}$$

where $f(x,u) \epsilon B_m$ with $Z(m)$ known, or B_m is the closest block to the point $f(x,u)$ for which $Z(m)$ is calculated. The iteration variable N has been suppressed since the blocks will be stored back in the same location after they are processed. That is, the stage identity is destroyed. The set of costs, $\{v(x) | x \epsilon B_\ell\}$, is then fitted (Step 5) with a quadratic surface, $Z(\ell)$. During the first pass through state space (MODE = 1), the control for B_ℓ and the parameters of adjacent blocks are then placed in low-speed storage (Step 7) and Step 2 is repeated.

After all of state space has been considered once, the algorithm goes into MODE = 2 (Step 8). For all subsequent calculations f(x,u) is assured of lying in a calculated block for the evaluation of (3.9). Also, a comparison of the present cost surface and the previous cost surface over the block is made (Step 6) to determine the convergence of the algorithm. Convergence is guaranteed for $\beta < 1$ by (2.12). The process is continued until convergence is attained over all of state space or until a maximum number of iterations is reached.

3.8 Block Processing Order

Before the algorithm described in the last section may be applied, the partition of state space must be ordered; i.e., an integer must be associated with each block which determines when it will be processed during a pass through state space. The only restriction upon this ordering is that each block be adjacent to a block previously processed during the current processing sequence. This restriction causes the blocks to tend to radiate out through state space from the origin block as they are considered. There is, however, reason to be more selective in the ordering. Namely, it would be ideal if the optimal control, u, at a point x always caused f(x,u) to lie in a block which had already been processed during that pass through state space. This could be accomplished if the optimal control were already known. The block ordering could be taken opposite to the direction [f(x,u)-x], that is, opposite to the direction of the expected transition from x under optimal control. Obviously, if the optimal control were known, the problem would be solved; however, in many problems although the optimal solution is not known, there is some knowledge as to the manner in which the system should be controlled.

This idea was made explicit by Larson with the concept of preferred direction of motion. The preferred direction of motion is, basically, the expected direction in which the trajectories of the system tend under optimal control. The information used in establishing the preferred direction is *a priori* and rests on an intuitive feeling for the system's behavior. The blocks are then processed opposite to the preferred direction.

If the preferred direction is not known, the algorithm still works and will converge, although more iterations over state space may be necessary. Thus a general technique for ordering the blocks in the absence of a preferred direction is desired. This objective can be achieved in the following way, again suggested by Larson. Let the blocks be designated as in Figure 3.3 where B_0 is the origin block and is defined to have coordinates

$B_0 = (0,0)$. The blocks $B_1, \ldots, B_8$ are said to lie in layer one (L=1), $B_9, \ldots, B_{24}$ in layer two, etc. These blocks have coordinates, $B_1 = (0,1)$, $B_2 = (0,-1)$, ..., $B_{24} = (-2,2)$. The ordering is achieved by counting with 2-digits modulo $M = 2L + 1$ for the blocks in layer L. Take for example layer one; counting MOD/3 yields 00, 01, 02, 10, 11, 12, 20, 21, 22. The digits MOD/3 are associated with the block coordinate elements as follows:

$$\begin{array}{lll} 0 & \text{MOD/3} & \Rightarrow 0 \\ 1 & \text{MOD/3} & \Rightarrow 1 \\ 2 & \text{MOD/3} & \Rightarrow -1. \end{array}$$

Thus, the numbers MOD/3 are associated with the block coordinates (0,0), (0,1), (0,-1), (1,0), (1,1), (1,-1), (-1,0), (-1,1), (-1,-1), respectively, and the block ordering through the first layer is achieved. For the second layer counting MOD/5 yields 00, 01, 02, 03, 04, 10, 11, 12, 13, 14, 20, 21, 22, 23, 24, 30, 31, 32, 33, 34, 40 41, 42, 43, 44. Again the MOD/5 digits are associated with the block coordinate elements as follows:

$$\begin{array}{lll} 0 & \text{MOD/5} & \Rightarrow 0 \\ 1 & \text{MOD/5} & \Rightarrow 1 \\ 2 & \text{MOD/5} & \Rightarrow -1 \\ 3 & \text{MOD/5} & \Rightarrow 2 \\ 4 & \text{MOD/5} & \Rightarrow -2 \end{array}$$

Thus, the sequence of MOD/5 numbers corresponds to the block coordinates (0,0), (0,1), (0,-1), (0,2), (0,-2), (1,0), (1,1), (1,-1), (1,2), (1,-2), (-1,0), (-1,1), (-1,-1), (-1,2), (-1,-2), (2,0), (2,1), (2,-1), (2,2), (2,-2), (-2,0), (-2,1), (-2,-1), (-2,2), (-2,-2). Deleting those coordinates in layers lower than layer two results in the sequence, (0,2), (0,-2), (1,2), (1,-2), (-1,2), (-1,-2), (2,0), (2,1), (2,-1), (2,2), (2,-2), (-2,0), (-2,1), (-2,-1), (-2,2), (-2,-2) with the associated blocks B_9, B_{10}, ..., B_{24}. This counting procedure can be carried out through an arbitrary number of layers and for a n^{th} order system. The n^{th} order system would require counting with n-digits MOD/M.

3.9 Calculating the Origin Block

To initiate the PSS algorithm it is necessary to calculate the quadratic cost surface associated with the origin block for the first pass through state space. This can be done either by dynamic programming using quadratic approximation over the origin block or by policy iteration also employing quadratic approximation.

Howard's policy iteration has application in finding the cost function of the origin block for the continuous state space stochastic control problem. Again, let the quadratic cost surface over the origin block be described by

$$\bar{v}(x) = \alpha + \sum_{i=1}^{n} \beta_i x_i + \sum_{i=1}^{n} \sum_{j=1}^{i} \gamma_{ij} x_i x_j .$$

Then for a fixed policy $u_0 \epsilon U$ defined for all grid points in the block, it is desired that

$$\bar{v}(x) = \ell(x,u_0) + \beta E \left\{ \bar{v}\left[f(x,u_0) + \zeta\right] \right\}. \qquad (3.11)$$

However, there are, in general, more than $(n^2+3n+2)/2$ points in a block for a n^{th} order system. Thus, a least square equation error criteria is used to determine the quadratic fit for the cost function. That is, letting

$$e_x = \ell(x,u_0) + \beta E \left\{ \bar{v}\left[f(x,u_0)+\zeta\right] \right\} - \bar{v}(x),$$

the functional,

$$J = \sum_{x \epsilon B_0} e_x^2$$

is minimized with respect to $\alpha_1, \beta_1, \ldots, \beta_n, \zeta_{11}, \ldots, \zeta_{nn}$. This minimization determines a set of linear equations which in turn define the quadratic surface, $\bar{v}_0(x)$, associated with the policy u_0. This surface is then used in step two of Howard's policy iteration to determine a new policy $u_1 \epsilon U$. Thus,

$$\min_{u} \left\{ \ell(x,u) + \beta \left\{ \bar{v}_0\left[f(x,u)\right] + \sum_{i=1}^{n} \sum_{j=1}^{i} \gamma_{ij} E[\zeta_i \zeta_j] \right\} \right\}$$

for all $x \epsilon B_0$ determines the new policy u, which in turn determines a new cost surface $\bar{v}_1(x)$. The policy iteration is carried out until convergence. It has been found numerically that while this procedure works well at the origin block (containing the minimum point of the cost surface) it does not converge well for other blocks. Thus, it cannot be used to find the cost surface for blocks other than the origin.

A second technique to find the cost surface of the origin block is to employ dynamic programming. Assuming a terminal cost of zero, the dynamic programming algorithm can be applied to each point in B_0, i.e.,

$$v(\overset{\circ}{x}) = \min \{\ell(x,u)\} \text{ for all } x \varepsilon B_0.$$

This cost function is fitted with a quadratic surface $\overline{v}(x)$, then,

$$v(x) = \min_u \left\{ \ell(x,u) + \beta E \left\{ \overline{v}_0 \left| f(x,u) + \zeta \right| \right\} \right\} \tag{3.12}$$

is calculated for all $x \varepsilon B_0$. Again a quadratic surface $\overline{v}(x)$ is fitted to the cost function $v(x)$ and (3.10) applied. This procedure is carried out until convergence, with the speed of convergence described in (2.13).

CHAPTER 4

EXAMPLES

4.1 Introduction

In this chapter the partitioned state space algorithm developed in the last chapter is applied to several nonlinear stochastic control problems. Since no exact, analytical solution has been achieved for this class of problems, conventional dynamic programming is used to obtain an alternative solution for comparison. The control and cost obtained by both methods are compared. The problems are first and second order and have scalar control. A golden section search was used to search the control space (Wilde and Beightler, 1967). The CDC 6400 digital computer was used to achieve the numerical results.

4.2 Examples

Example 4.1. This example and the following one are simple scalar test examples. The objective is to drive the state to the origin with a bounded control. The problem is specified by,

plant equation	-- $x(k+1) = x(k) + u(k) + \zeta(k)$
stage cost	-- $\ell(x,u) = x^2 + u^2$
noise	-- $\zeta(k)$ is normal with mean zero and variance σ^2

discount factor	-- $\beta = .7$
state space	-- $10 \leq x \leq + 10$
grid	-- $\Delta x = .5$
control	-- $-2.2 \leq u \leq 2.2$.

The state space is partitioned into

$B_0 = \{x | -2 \leq x \leq 2\}$, $B_1 = \{x | 2 \leq x \leq 6\}$,

$B_2 = \{x | -6 \leq x \leq -2\}$, $B_3 = \{x | 6 \leq x \leq 10\}$, and

$B_4 = \{x | -10 \leq x \leq -6\}$.

The control and cost functions are plotted in Figures 4.1 and 4.2, respectively. The percentage difference between dynamic programming and PSS dynamic programming is less than 2% for both $\sigma^2 = 1.0$ and $\sigma^2 = 2.5$ in the cost function and the control function was the same within the accuracy of the golden section search routine.

Figure 4.1 indicates that the optimal control is very nearly linear until saturation at $u = \pm 2.2$. The control is approximately $u_k = -.5x_k$. The question naturally arises, is this the optimal control for the linear (unbounded control) case? The solution to this linear regulator problem is well documented (Sage, 1968), and the solution to a Ricatti type equation yields $u_k = -.4x_k$. The bounded control problem thus controls more heavily in its linear region than the linear problem. To achieve the

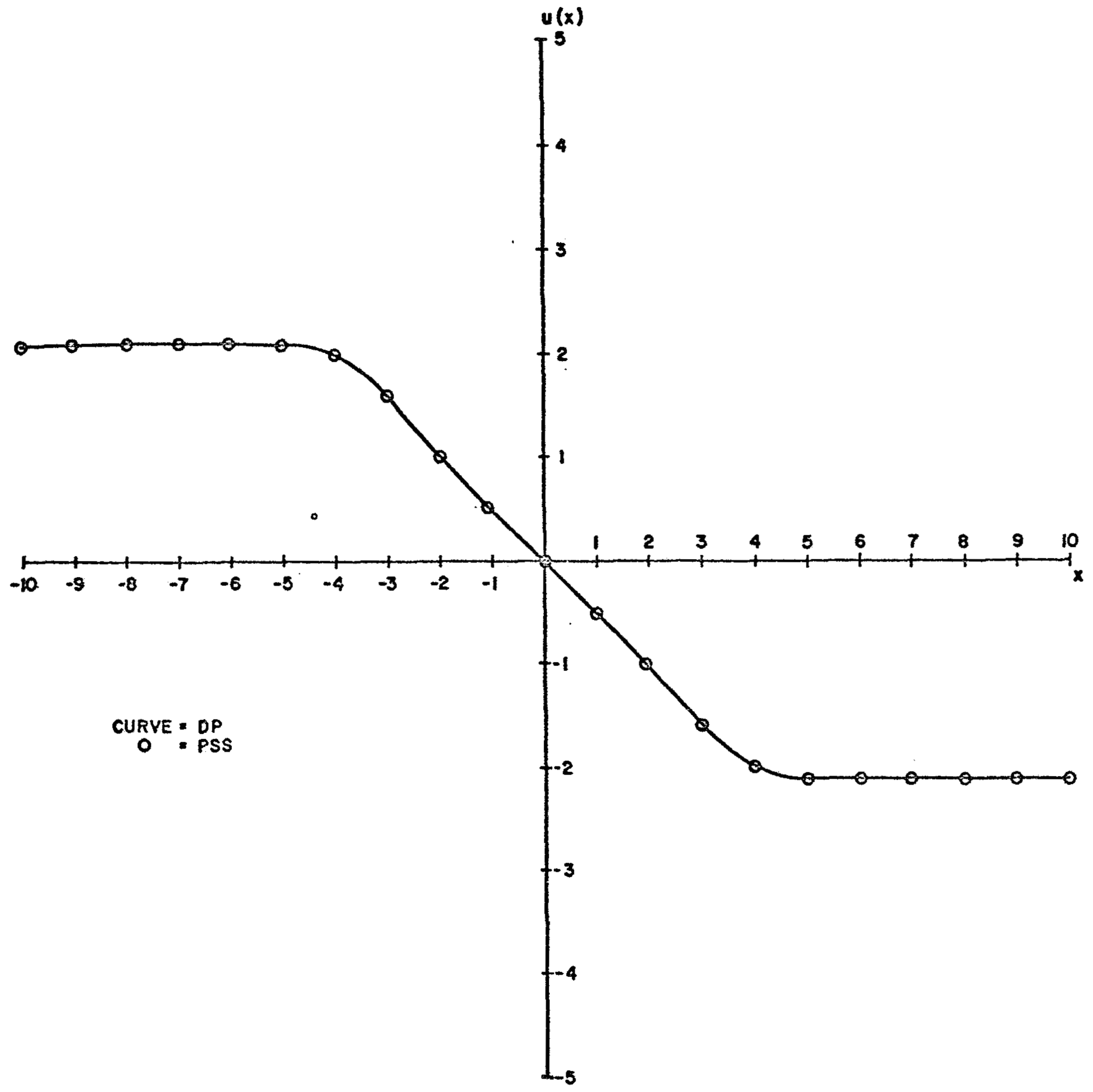


Figure 4.1 Control for Example 4.1, $\sigma^2 = 1.0$.

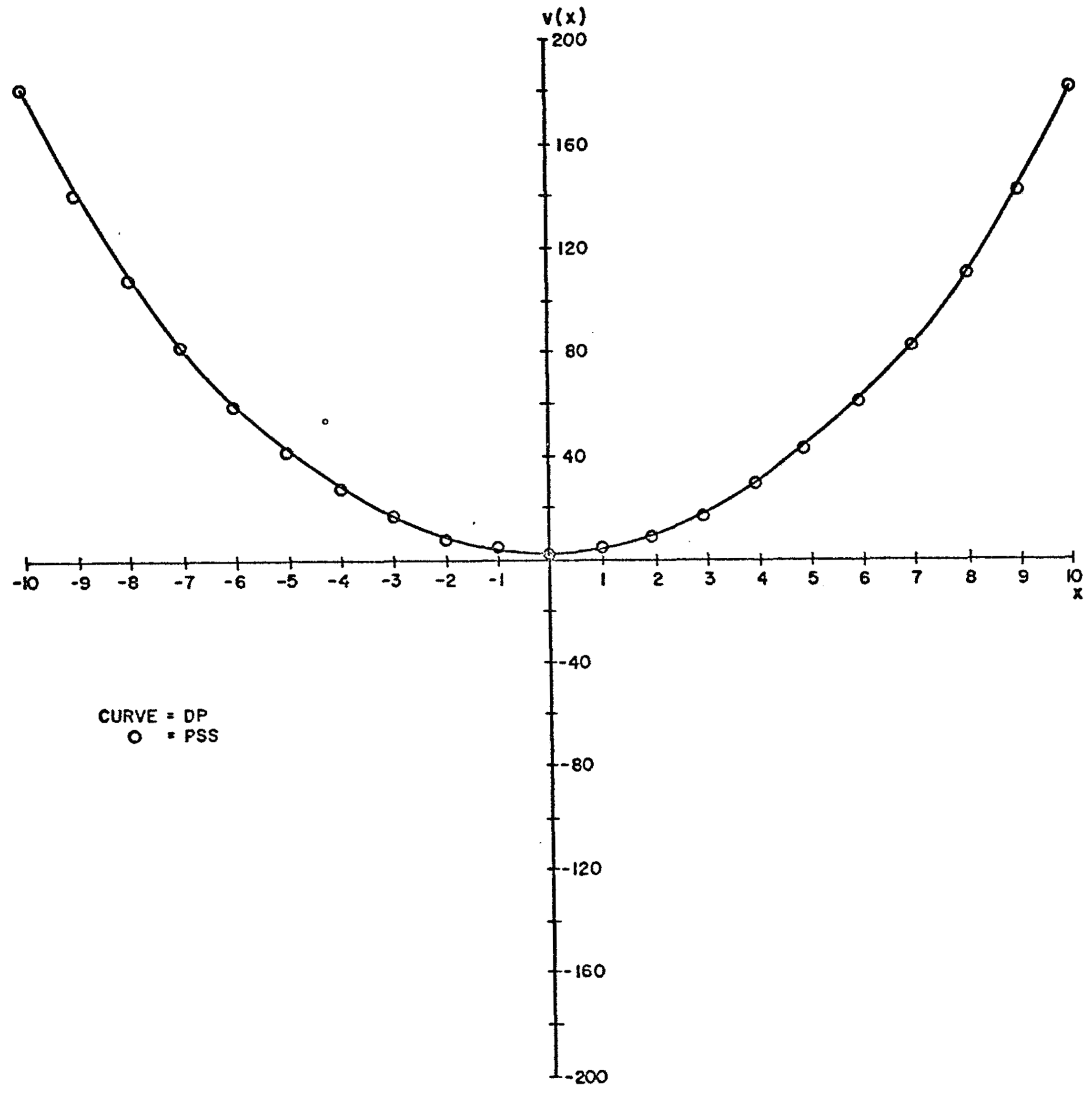


Figure 4.2 Cost for Example 4.1, $\sigma^2 = 1.0$.

optimal solution, the bounded control problems must be considered as nonlinear.

Example 4.2. The specifications for this example are the same as Example 4.1 except the plant equation is changed to a nonlinear equation,

$$x(k+1) = \frac{1}{2}x(k) + \frac{1}{20}x(k)^2 + u(k) + \zeta(k).$$

The control and cost functions are plotted in Figures 4.3 and 4.4. The percentage difference between cost functions by the two methods was less than 3%, and the control functions never deviated more than .13, or one interval of the search routine, for both $\sigma^2 = 1$ and $\sigma^2 = 2.5$.

It is interesting to note that for $x < 0$ the $\frac{1}{2}x$ and $\frac{1}{20}x^2$ terms tend to cancel and for $x = -10$ they cancel exactly. Thus, the control for $x < 0$ rises and then drops to zero at $x = -10$. This effect is also noticed in comparing the cost functions for Example 4.1 and 4.2. The cost function for Example 4.2 is asymmetrical and lower for $x < 0$ while the cost function for Example 4.1 is symmetrical.

Example 4.3. Again it is desired to drive the state to the origin with a bounded control. The specifications are,

$$\text{plant equation} \quad -- \quad x_1(k+1) = x_1(k) + \frac{1}{10}x_2(k) + \zeta_1(k)$$

$$x_2(k+1) = x_2(k) + u(k) + \zeta_2(k)$$

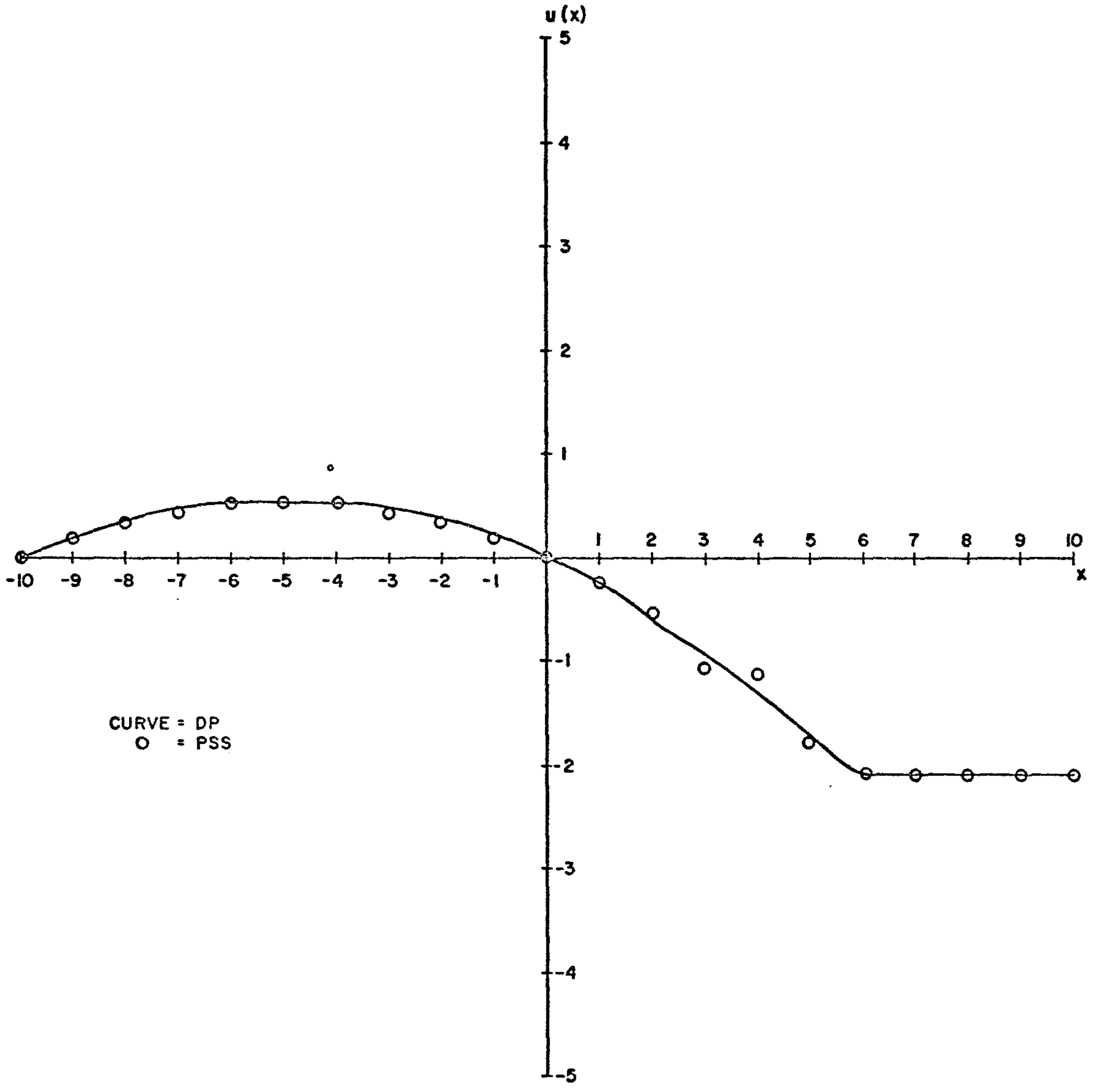


Figure 4.3 Control for Example 4.2, $\sigma^2 = 1.0$.

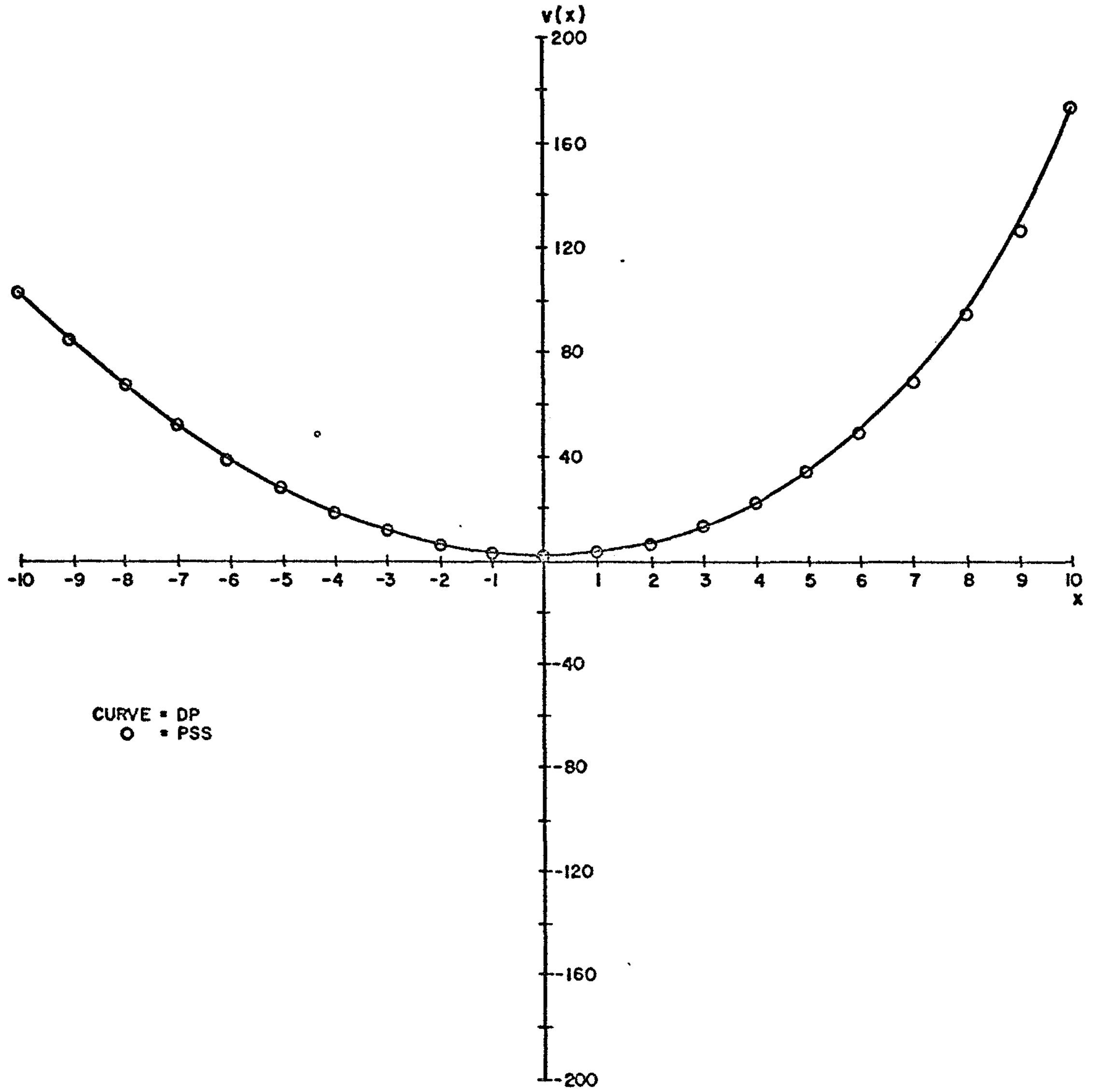


Figure 4.4 Cost for Example 4.2, $\sigma^2 = 1.0$.

noise	-- $\zeta_1(k)$ and $\zeta_2(k)$ are independent and gaussian with zero mean and variance σ^2
stage cost	-- $\ell(x,u) = x_1^2 + x_2^2 + u^2$
discount factor	-- $\beta = .7$
state space	-- $-5 \leq x_1 \leq 5$ $-5 \leq x_2 \leq 5$
control	-- $-2.2 \leq u \leq 2.2$
grid	-- $\Delta x_1 = \Delta x_2 = .5$
partition	-- the state space is partitioned into 25 blocks ordered as in Figure 3.3 with sides 2 units long, i.e., 25 grid-points per block.

Figure 4.5 presents a isometric plot of the cost surface in the right half plane. The control and cost are symmetrical about the origin so that knowledge of the right half plane is adequate. The cost surface is very smooth and quadratic in nature thus the quadratic approximation should apply. Figure 4.6 gives the control in the right half plane at the block corners and Figure 4.7 gives the cost at these points. For $\sigma^2 = 1.0$ the percentage difference of the cost functions is less than 6% while the control functions are identical within the accuracy of the search. For $\sigma^2 = 2.5$ the cost functions are within 10% of each other and the accuracy of the control is unaffected.

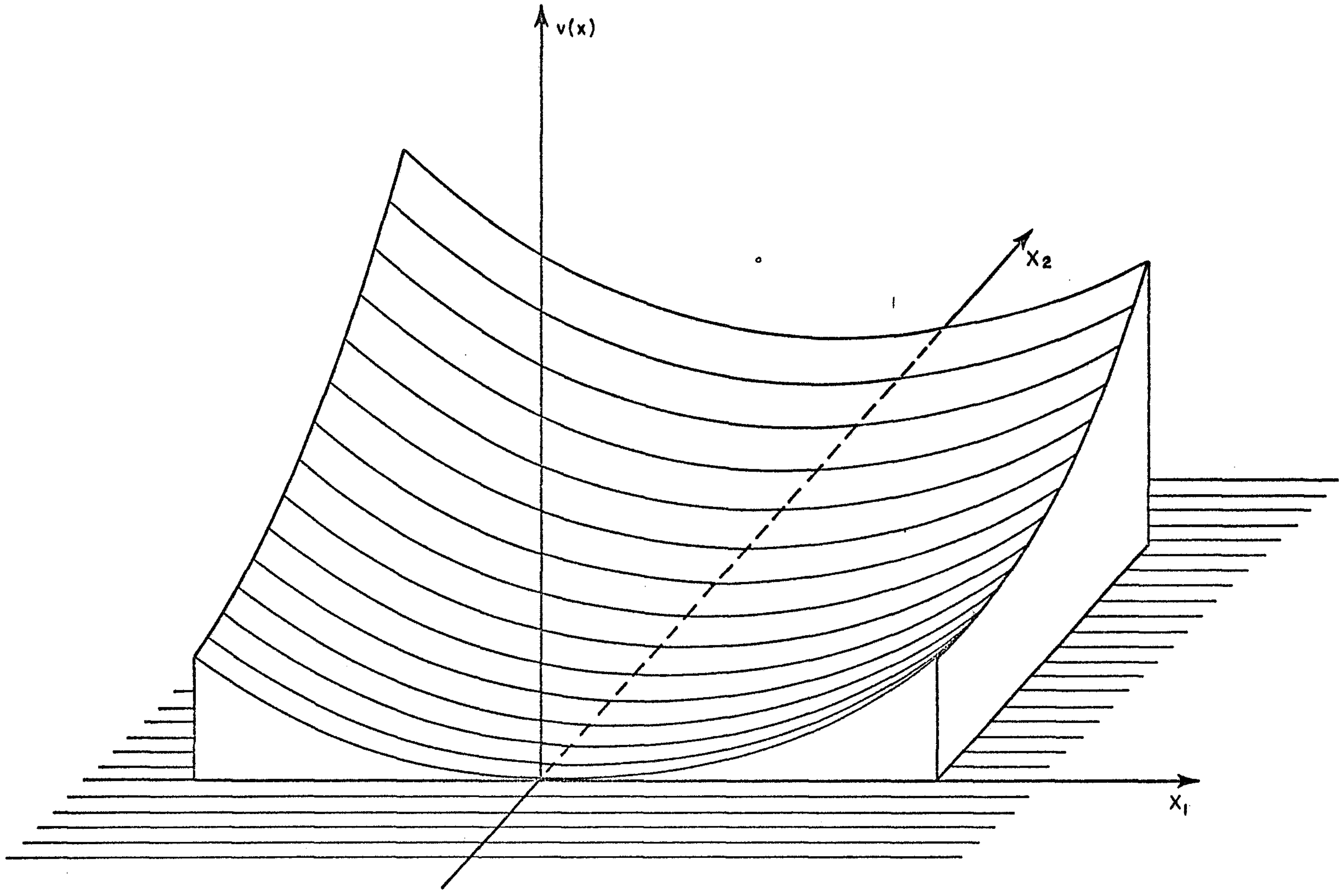


Figure 4.5 Isometric plot of the cost surface for Example 4.3.

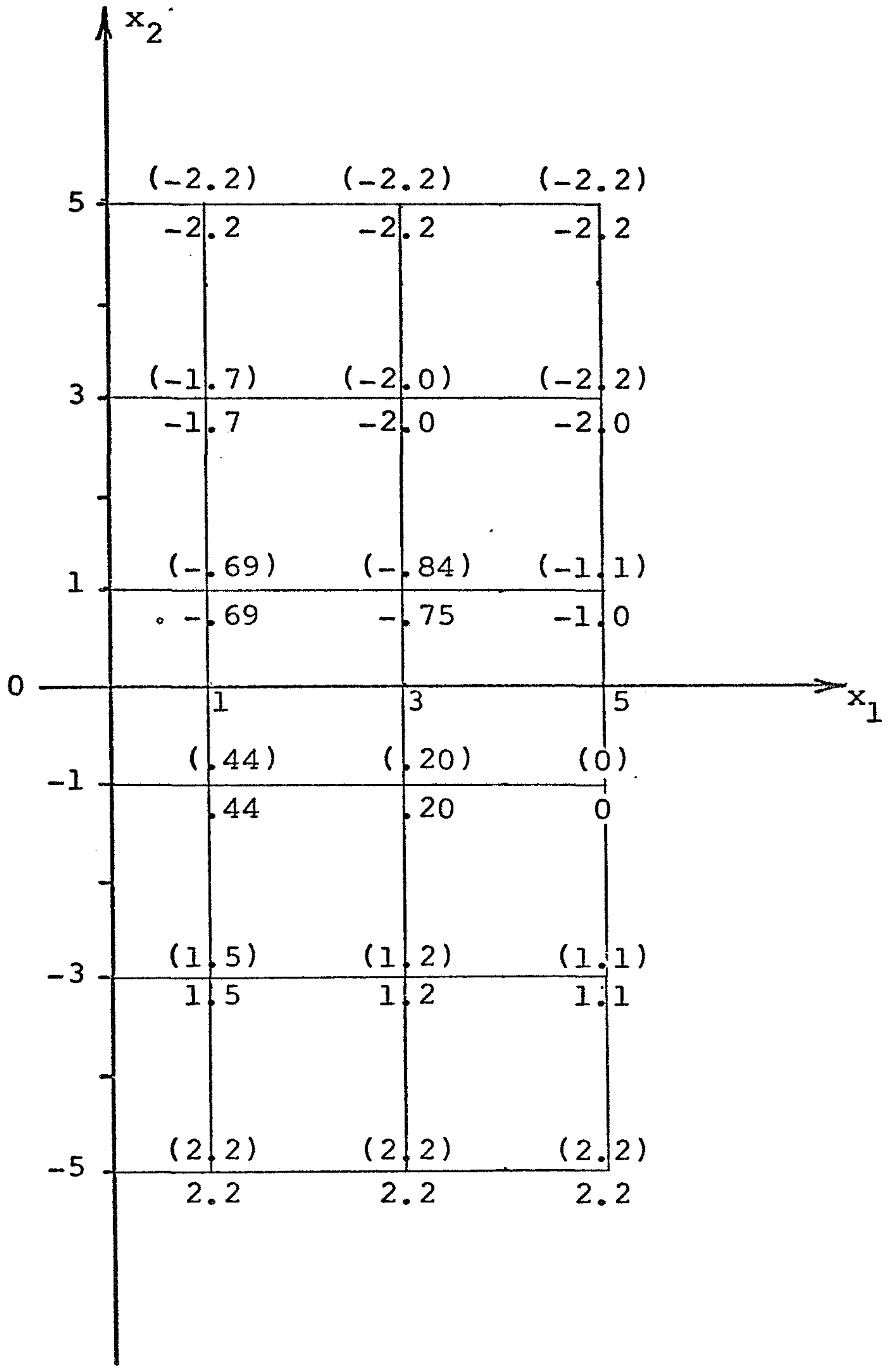


Figure 4.6 Control over right half plane for Example 4.3, $\sigma^2 = 1.0$, parentheses indicate PSS.

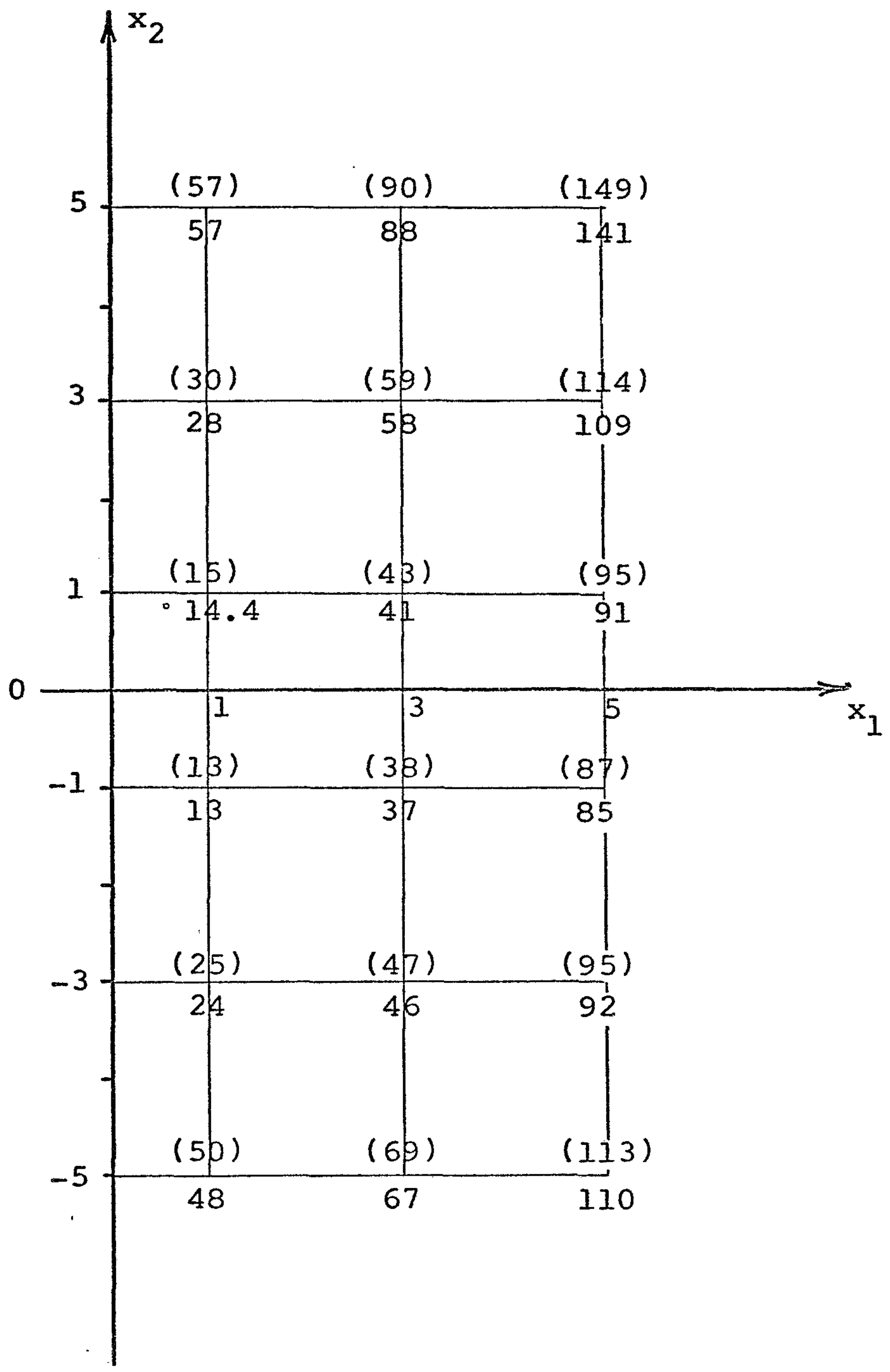


Figure 4.7 Cost over right half plane for Example 4.3, $\sigma^2 = 1.0$, parentheses indicate PSS.

The PSS algorithm takes 9 sec per pass through state space while conventional dynamic programming (DP) requires 33 sec/pass. For DP the noise is incremented into 7 values for each noise element. PSS converges in 3 iterations while DP requires 13 iterations. Thus, the solution time for PSS is 30 sec. and DP 460 sec. The "curse of dimensionality" is seen to affect solution time as stated in the last chapter. The PSS algorithm, thus, not only has a high-speed storage advantage but also reduces the computation time with respect to the conventional DP algorithm. This effect will also be seen in the next example which has noise on both state variables. However, in Example 4.5, whose noise is imposed on only one state variable, the computation times of the two methods are comparable.

Example 4.4. A classical stochastic control problem is that of inventory control. An appropriate quantity of goods is ordered to supply a stochastic demand and to minimize an expected cost function. This problem was formulated by Arrow, Harris, and Marshak (1951). Scarf (1960) demonstrated the optimality of the (s,S) policy for certain classes of stage costs. The problem is formulated as follows:

x	-- state, quantity of goods in storage
u	-- control, quantity of goods ordered

$\ell(x,u)$ -- stage cost, reflecting ordering, storage, and shortage costs

ζ -- noise, the demand for the goods

$x(k+1) = f\left[x(k),\ u(k)\right] + \zeta(k)$ -- plant equation, determines the number of goods in inventory at the end of a selling period.

Scarf showed that for x scalar and the restricted plant dynamics,

$$x(k+1) = x(k) + u(k) + \zeta(k)$$

and $\ell(x,u)$ enjoying certain convexity properties the optimal policy had the form

$$u = 0 \text{ when } x > s$$
$$u = S - x \text{ when } x \leq s.$$

This type of policy is called (s,S). The following example has a more general two dimensional plant equation and bounded control and, thus, is not (s,S) optimal.

A two dimensional inventory of perishables is considered where goods in class x_1 degrade to class x_2 and class x_2 goods eventually perish. The dynamics for this example are specified as,

$$x_1(k+1) = .7\ x_1(k) + .8\ u(k) + \zeta_1(k)$$
$$x_2(k+1) = .3\ x_1(k) + .7\ x_2(k) + .2\ u(k) + \zeta_2(k).$$

The commodity inventoried could be interpreted as, say, eggs with,

x_1 -- eggs of grade A

x_2 -- eggs of grade B.

The stage cost $\ell(x,u)$ is the sum of the <u>ordering cost</u>, the <u>storage cost</u>, and the <u>shortage cost</u>. The <u>ordering cost</u> is,

$$\ell_1(u) = \begin{cases} 0 & u=0 \\ k+cu & u>0. \end{cases}$$

The <u>storage cost</u>, ℓ_2, is the cost to store the good accumulated at the beginning of a period. Let

$$z = u\delta(u) + x_1\delta(x_1) + x_2\delta(x_2)$$

where

$$\begin{aligned} \delta(y) &= 0 \quad y < 0 \\ &= 1 \quad y \leq 0, \end{aligned}$$

then

$$\ell_2(x,u) = \begin{cases} H\cdot z & z \leq d \\ H\left|d-(z-d)^2\right| & z > d \end{cases}$$

where d is the designed storage capacity which can only be exceeded by implementing expensive temporary storage. The <u>shortage cost</u>, ℓ_3, is the cost attached to failure to meet the demand (i.e., failure to supply customers, thus, angering them).

$$\ell_3(x,u) = A_1 \int_0^{f_1(x,u)} \left|\zeta_1 - f_1(x,u)\right|^2 p_{\zeta_1}(\zeta_1)d\zeta_1$$

$$+ A_2 \int_0^{f_2(x,u)} \left|\zeta_2 - f_2(x,u)\right|^2 p_{\zeta_2}(\zeta_2)d\zeta_2.$$

Thus,

$$\ell(x,u) = \ell_1(u) + \ell_2(x,u) + \ell_3(x,u).$$

The parameters for this example are:

$$k = 1 \quad c = 1$$

$$H = 1 \quad d = 1$$

$$A_1 = 10 \quad A_2 = 10.$$

The demand, ζ, is a gaussian random variable with ζ_1 and ζ_2 independent and

$$E\{\zeta_1\} = -4.0 \qquad Var\{\zeta_1\} = 1.0,$$

$$E\{\zeta_2\} = -2.0 \qquad Var\{\zeta_2\} = 1.0.$$

The state space is,

$$-4 \leq x_1 \leq 8,$$

$$-3 \leq x_2 \leq 6,$$

and the control space is,

$$0 \leq u \leq 8.$$

The grid is $\Delta x_1 = \Delta x_2 = .5$. The discount factor is

$$\beta = \frac{1}{1+\rho} = .9$$

where $\rho = .111 = 11.1\%$ is the interest rate. The dynamic programming functional equation is then, as before,

$$v(x) = \min_{u} \left\{ \ell(x,u) + \beta E \left\{ v\left[f(x,u) + \zeta \right] \right\} \right\}$$

and the PSS and DP algorithms may be applied to find the optimal ordering policy.

An isometric plot of the cost surface is displayed in Figure 4.8. It is seen that the cost surface is not quadratic in shape. However, by partitioning the state space as in Figure 4.9 the quadratic approximation achieved a cost surface within 4% of the DP solution. The noise was incremented into five values in each variable for the DP algorithm. The control found by the two methods was once again within one search increment. The control and cost is displayed in Figures 4.10 and 4.11, respectively.

The PSS algorithm takes 9 seconds per pass through state space while DP takes 40 sec/pass. The DP algorithm takes 35 passes to converge. Convergence is slow since

$$\epsilon_n \leq .9\ \epsilon_{n-1}. \tag{2.13}$$

The PSS algorithm beginning with a good origin block obtained from the DP solution takes only 5 iterations to converge since it takes advantage of extrapolation on its

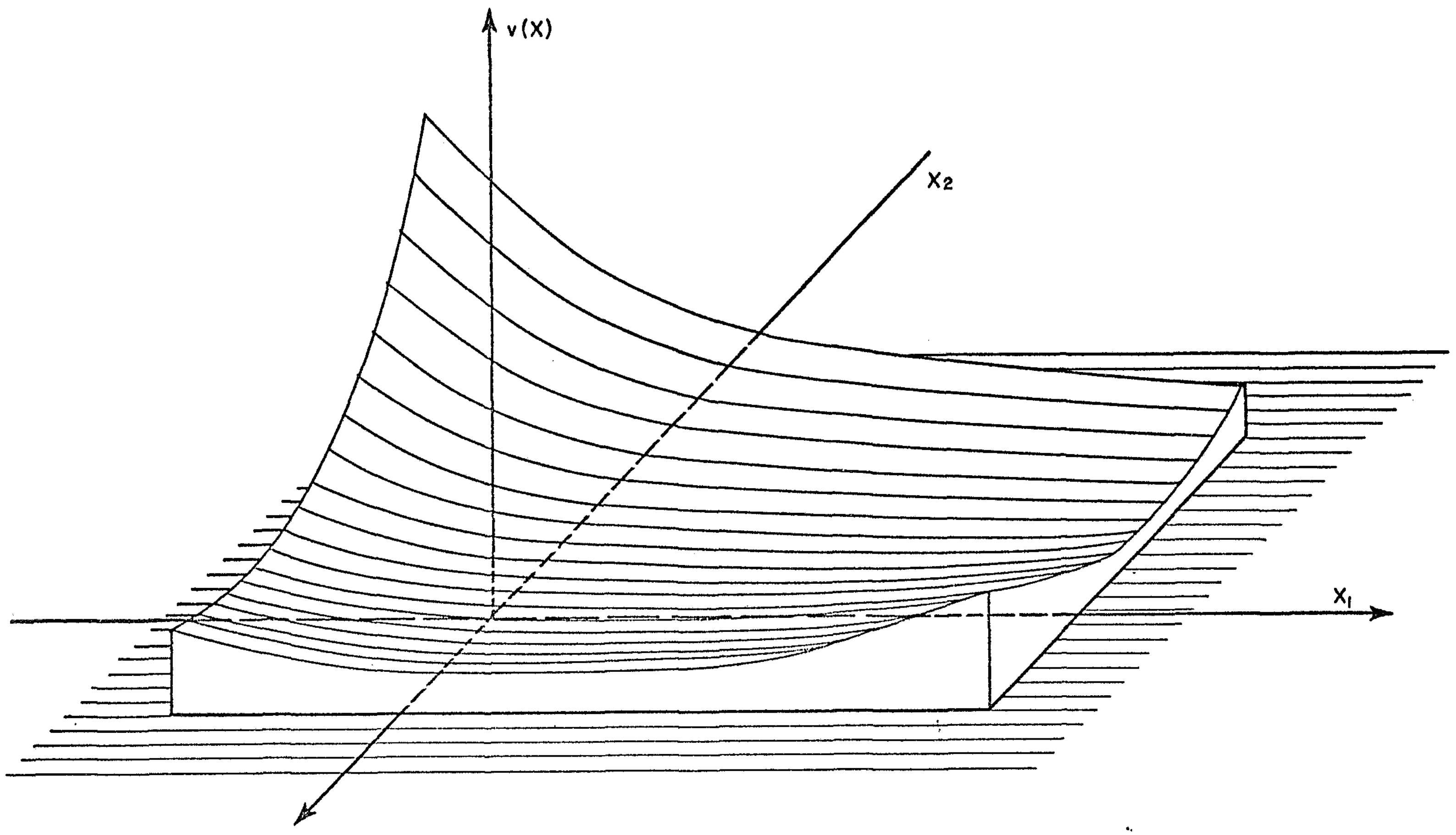


Figure 4.8 Isometric plot of the cost surface for Example 4.4.

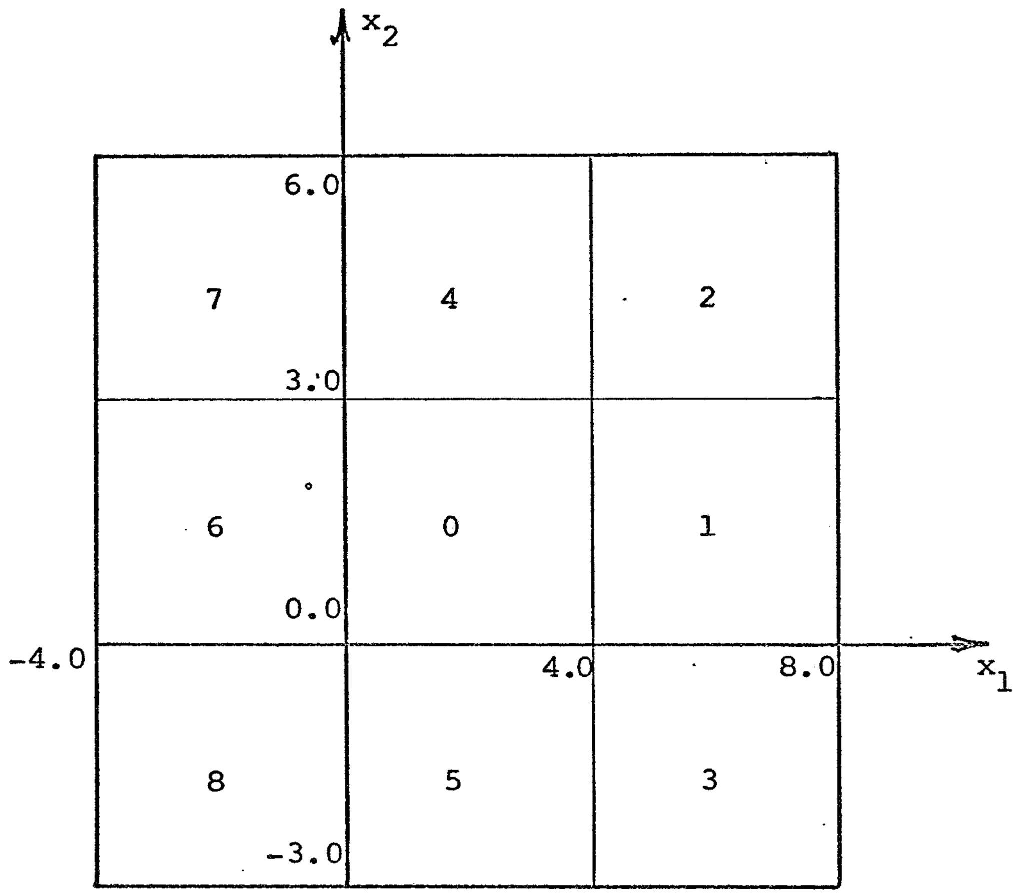


Figure 4.9 Block ordering for Example 4.4.

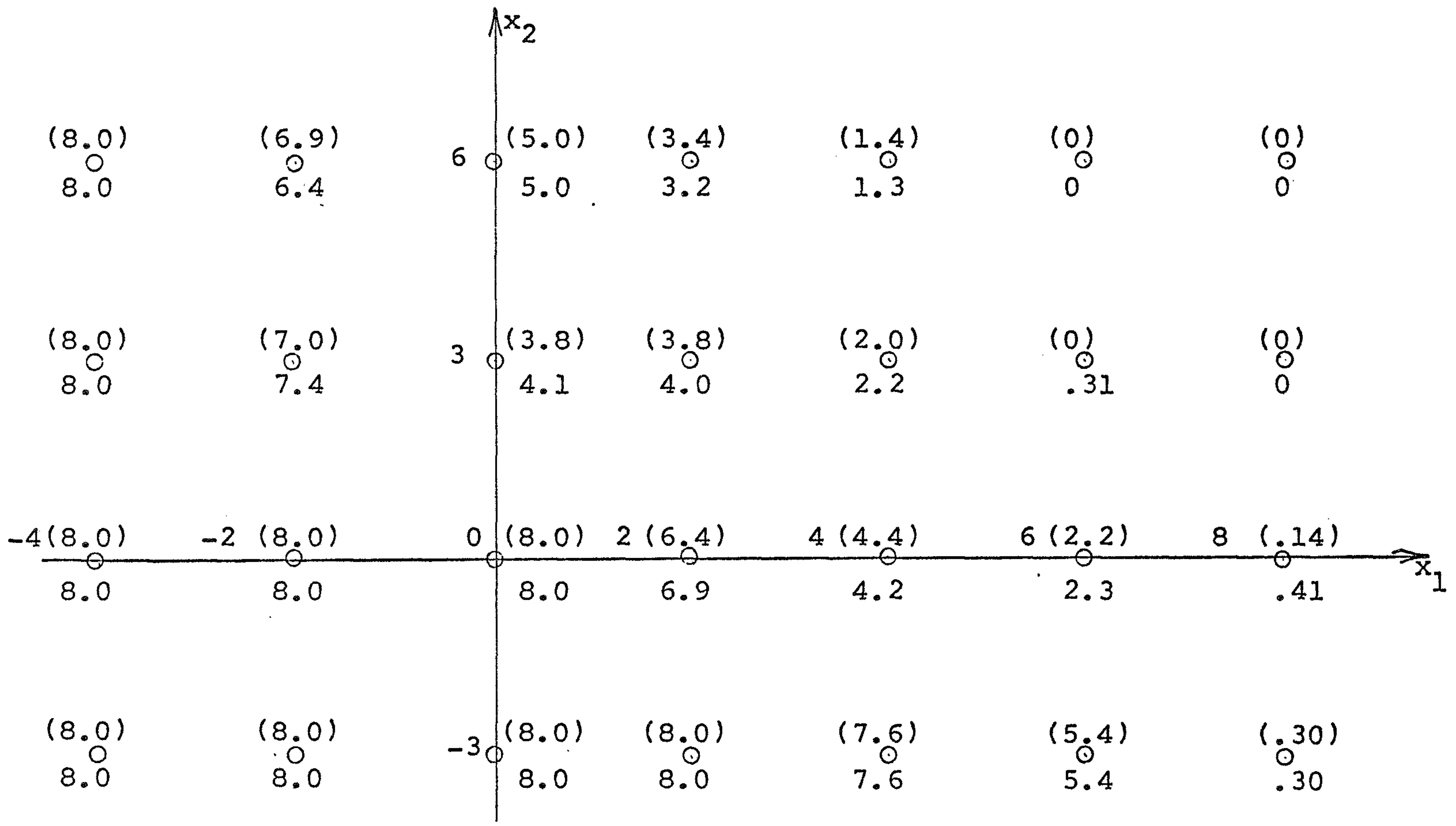


Figure 4.10 Control for Example 4.4, parentheses indicate PSS.

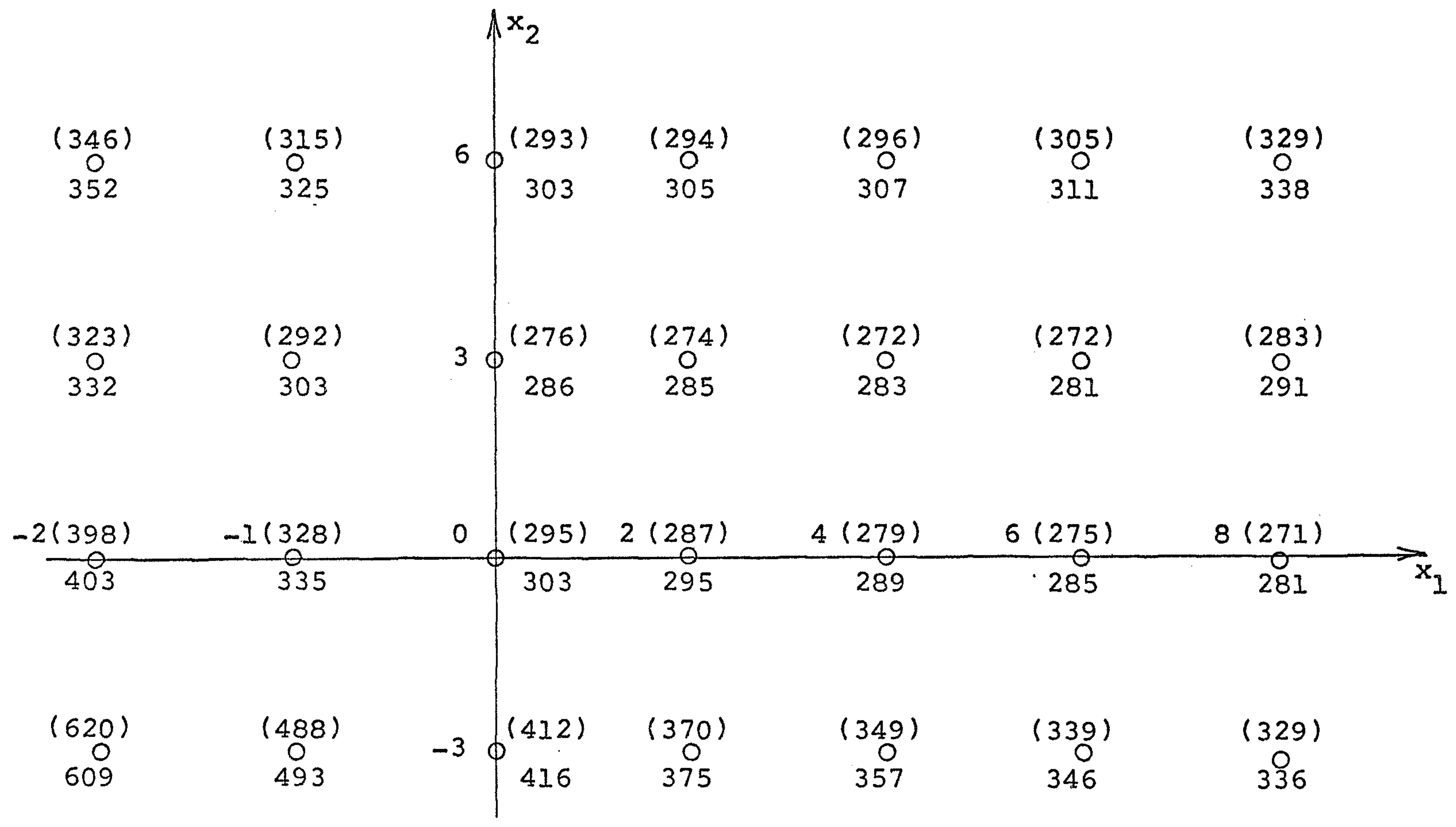


Figure 4.11 Cost for Example 4.4, parentheses indicate PSS.

first pass through state space. The PSS algorithm takes about 1/15 the calculation time of DP.

Figure 4.10 demonstrates conclusively that the optimal policy is not a generalized (s,S) policy. There is no point S in two dimensional space which the inventory always achieves; also, the order is on the upper bound at several state points. Thus, the purchasing agent must, to operate optimally, inventory his goods at the end of each period and then order that number of goods indicated by his two dimensional control function, u(x).

Example 4.5. In this example it is desired to reduce the core temperature of a nuclear reactor after shut down by a pulse of coolant at fixed flow rate through the core. At shut down the control rods are withdrawn and the reactor is heated by γ-heating. The γ-heating is determined by dynamics of the following nature

$$\dot{Q} = -aQ,$$

where Q is the heat generated by γ-heating and a is a fixed time constant. The temperature dynamics are given by,

$$\dot{T} = -kwT + \frac{1}{m_c}Q$$

where,

T -- temperature

w -- flow rate

k -- heat transfer coefficient

m_c -- effective core mass heat capacity.

The temperature is to be controlled by pulsing the flow rate w at fixed time intervals for any time up to the time interval duration. This scheme is illustrated in Figure 4.12. For $0 \leq t \leq T$,

$$\begin{aligned} w &= \bar{w} \quad 0 \leq t \leq u_0 \\ &= 0 \quad u_0 < t < T, \text{ etc.} \end{aligned}$$

The temperature ranges from 400°R to 4000°R and the heat generated from 0 Btu to 300,000 Btu. State variables are defined as,

$$x_1 = \frac{1}{m_c} Q$$

$$x_2 = \frac{1}{100} T.$$

It is desired to drive the temperature to 500°R and to restrict it to remain below 4200°, the maximum allowable core temperature. Also, the amount of coolant used should be weighted in the cost function. To achieve these specifications the following stage cost is defined,

$$\ell(x,u) = \left(\frac{x_2-5}{3.5}\right)^2 + u^2 + \frac{1}{\left(\frac{x_2-41}{2}\right)^2} .$$

The parameters of the problem are defined as,

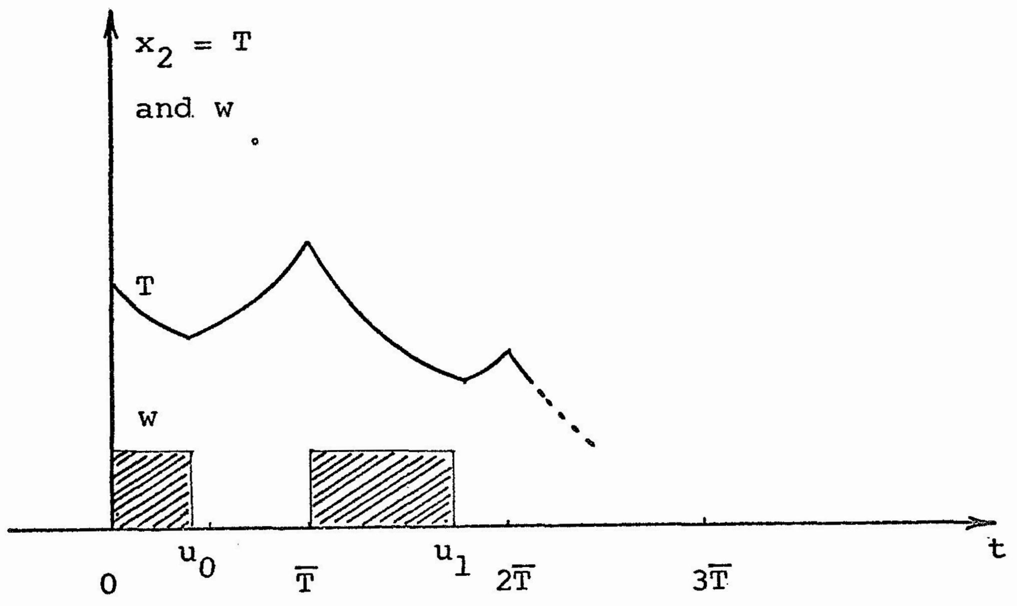


Figure 4.12 Temperature control method.

$$\overline{T} = 5 \text{ sec} \qquad \overline{w} = 4 \text{ lb/sec}$$

$$k = .025 \text{ sec/lb} \qquad a = .01/\text{sec}$$

$$m_c = 1000 \text{ Btu}/^{\circ}\text{R}$$

Since w is piecewise constant, the plant equations can be integrated analytically over one period $\overline{T}$, yielding

$$Q(\overline{T}) = Q(0)e^{-a\overline{T}},$$

and

$$T(\overline{T}) = T(0)e^{-k\overline{w}u} + \frac{Q(0)}{k\overline{w}-a}\left[e^{-au} - e^{-k\overline{w}u}\right]$$

$$+ \frac{Q(0)}{a}\left[e^{-au} - e^{-a\overline{T}}\right].$$

For $a = .01/\text{sec}$ and $k\overline{w} = .1$ the last equation becomes, to a very good approximation,

$$T(\overline{T}) = T(0)e^{-.1u} + Q(0)\left[16.1 - 11.1\, e^{-.1u} - u\right].$$

Employing state variable notation the plant equations become,

$$x_1(k+1) = .95\, x_1(k)$$

$$x_2(k+1) = e^{-.1u}\, x_2(k) + \left[.161 - .111\, e^{-.1u} - .01u\right] x_1(k).$$

The random effects of the nuclear γ-heating and the effect of suppressed state variables are accounted for by imposing plant noise, ζ, on the plant equations. Noise is only added to the temperature equation; thus, the plant equations are

$$x_1(k+1) = .95\ x_1(k)$$

$$x_2(k+1) = e^{-.1u}\ x_2(k) + \left[.161 = .111\ e^{-.1u} + .01u\right] x_1(k)$$

$$+ \zeta_2(k).$$

The discounted stochastic control problem is now completely defined by specifying $\beta = .7$ and ζ_2 to be normal mean zero and variance one.

An isometric plot of the cost surface is given in Figure 4.13. The cost surface has a minimum at $x = (0,4)$ and the cost surface is a rather gentle sloping surface with a low edge at $x_1 = 0$. This shape motivates the state space partition in Figure 4.14. The large block sizes are acceptable since the cost surface has little curvature. The long blocks in the x_1 direction are permissible since,

$$x(k+1) = .95\ x_1(k),$$

and there is little transition in the x_1 direction. In this problem the block ordering does not spiral out through state space, but rather a linear ordering of the blocks is used. No attempt was made in this problem to obtain the origin block and then extrapolate (MODE = 1) on the first pass through state space. Instead both PSS and DP were initialized by a terminal cost of zero.

The comparative results are tabulated in Figures 4.15 and 4.16. Again the control by the two methods is

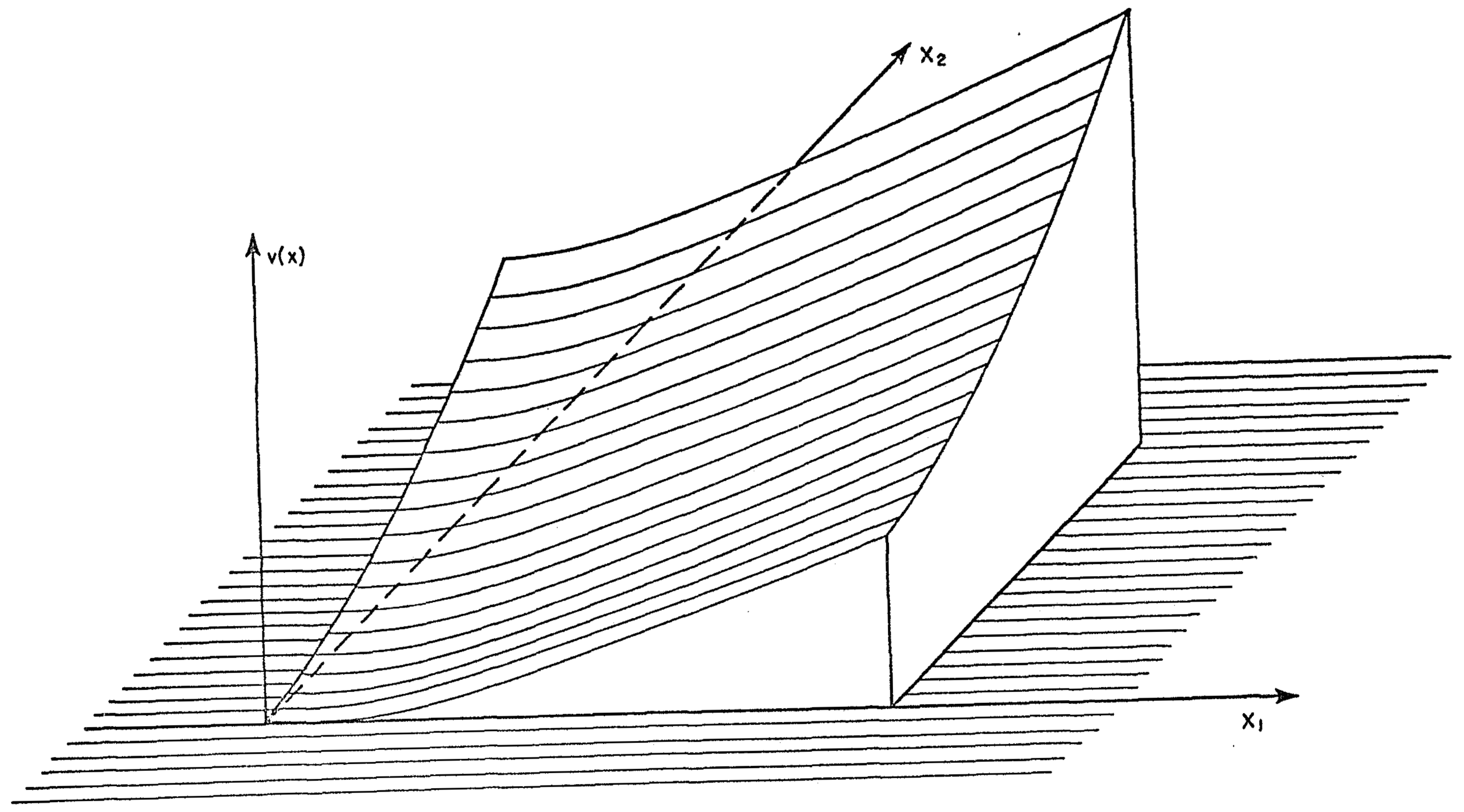


Figure 4.13 Isometric plot of the cost surface for Example 4.5.

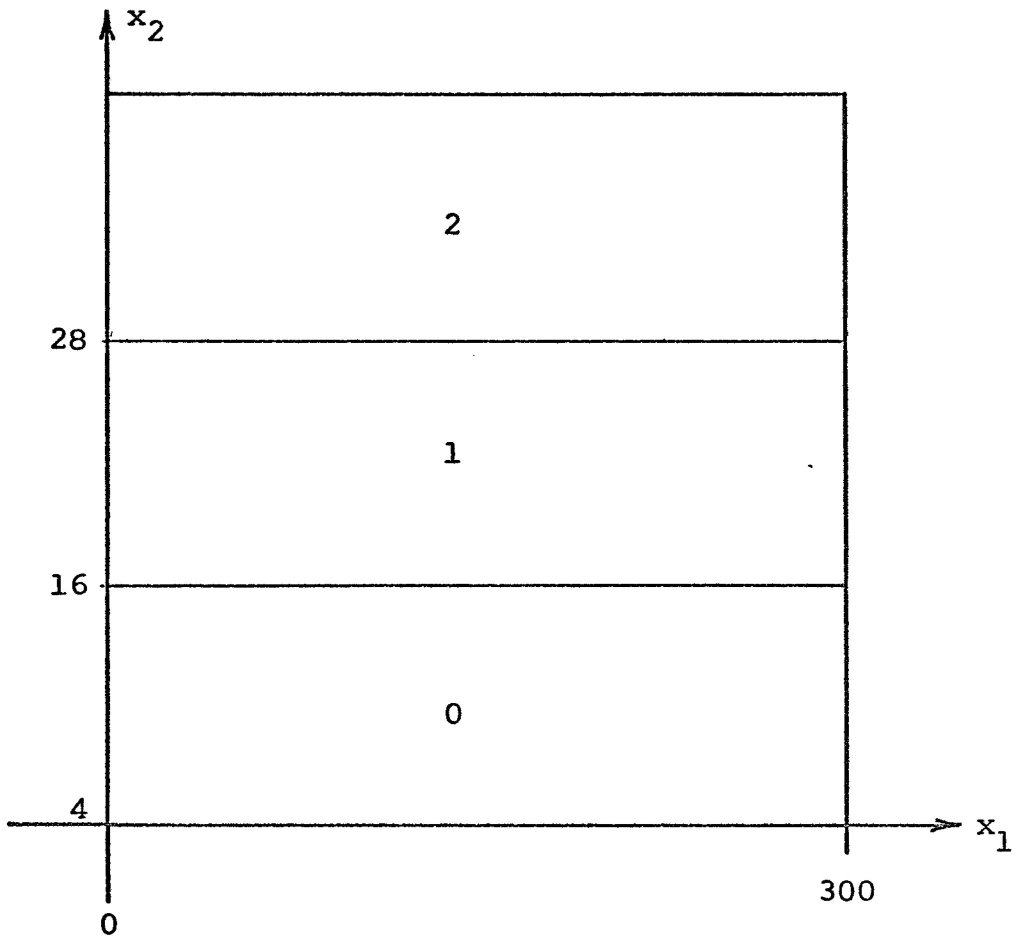


Figure 4.14 Block order for Example 4.5.

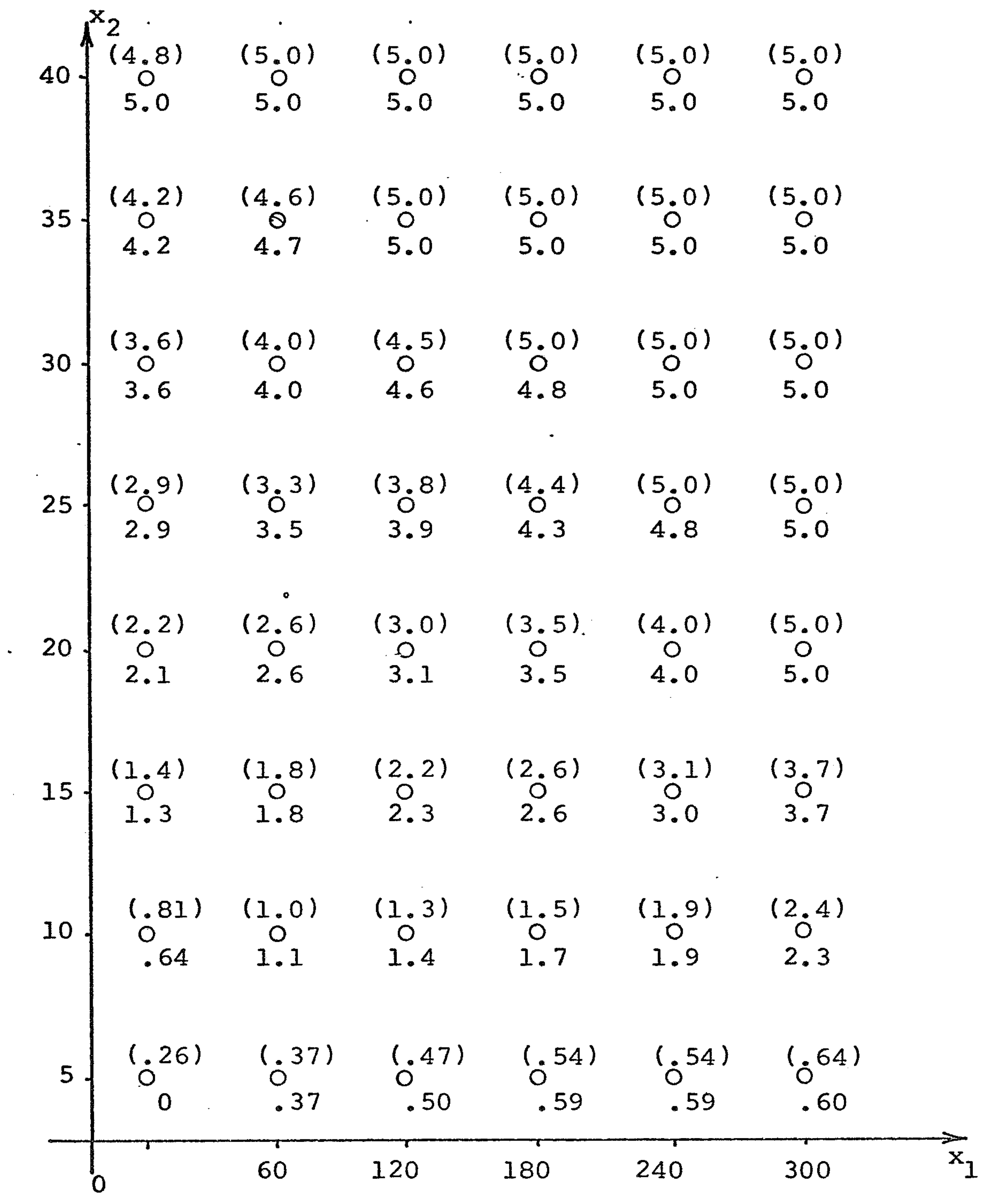


Figure 4.15 Control for Example 4.5, parentheses indicate PSS.

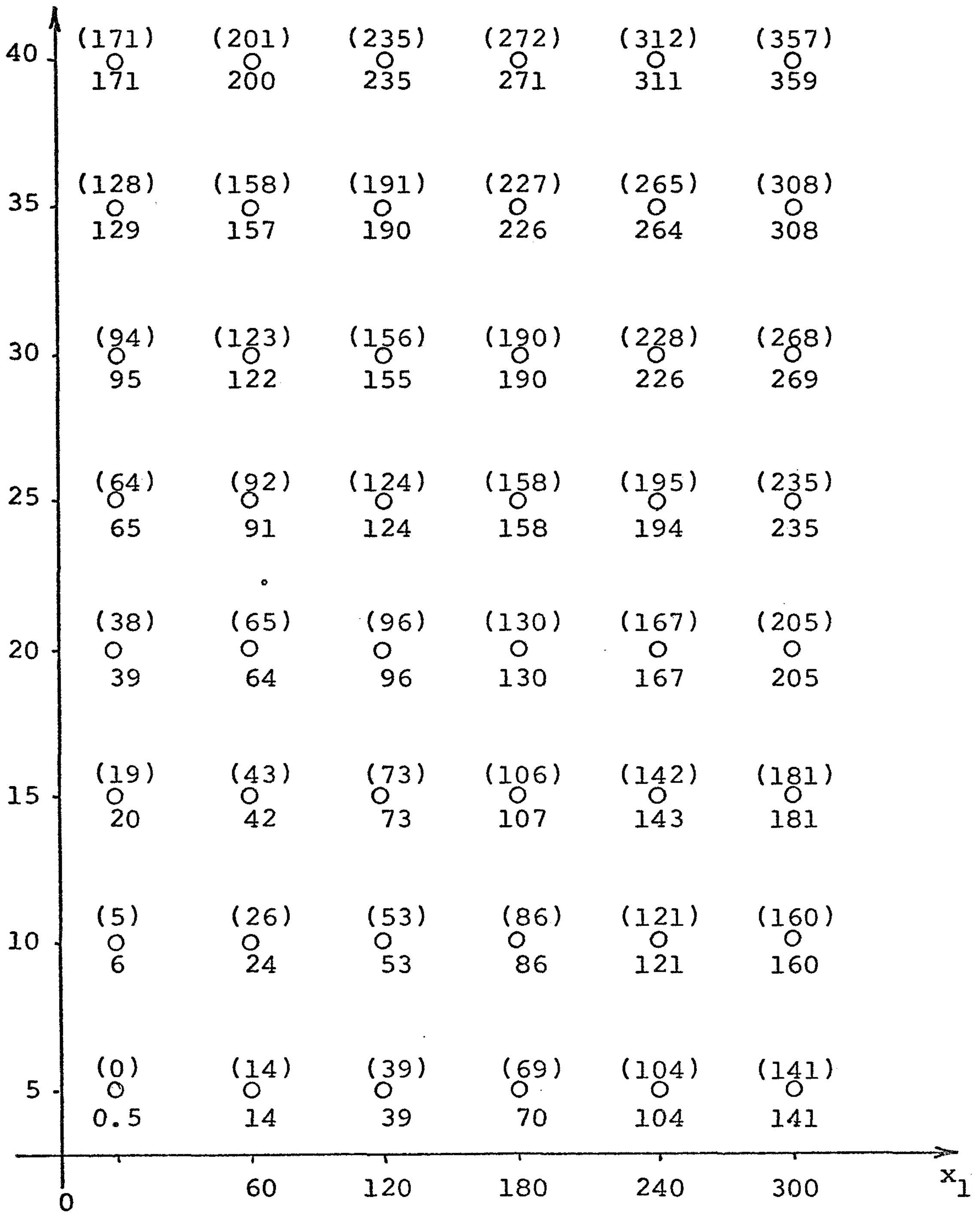


Figure 4.16 Cost for Example 4.5, parentheses indicate PSS.

always within one search increment while the cost surfaces are almost coincident. The PSS algorithm requires 64 seconds while DP takes 78 seconds. Since plant noise, ζ, is only imposed on one state variable the time advantage of PSS is suppressed (cf. Section 3.6). Further, the PSS bias term due to noise,

$$\sum_{i=1}^{n} \sum_{j=1}^{i} \gamma_{ij}(\ell) E\left\{\zeta_i \zeta_j\right\} = \gamma_{22}(\ell)\ E\left\{\zeta_2^2\right\},$$

is found to be negligible (always less than one). Thus, this problem could be viewed as a deterministic control problem for the stage cost defined. The noise, ζ_2, is represented by five increments in the DP algorithm; thus, the DP solution time would be cut by a factor of five if the problem were viewed as deterministic. In the previous four examples, however, the γ_{ij} quadratic cost coefficients were not negligible. The effect of noise, therefore, contributes significantly to the cost and control functions for these problems.

It is interesting to note in Figure 4.15 that the control is approximately linear up to saturation.

CHAPTER 5

CONCLUSIONS

5.1 Summary

This study examines the control of stochastic systems under discounted performance criteria. The general stochastic control problem and the more specific time invariant problem considered in this study are defined in Chapter 1. The stochastic control problem is viewed as a finite state Markov chain in Chapter 2. It is established that stationary control laws should be considered for the infinite duration process. Howard's policy iteration and dynamic programming are developed as methods of solution. Currently, they are the only feasible basic approaches to the problem.

In Chapter 3 the state space is defined on the continuum. To attack this new problem a grid is imposed on the state space and a large but finite state space results. It is shown that Howard's policy iteration is unsuited for this problem. While dynamic programming offers a more practical solution numerically, Bellman's "curse of dimensionality" restricts the problems to which it applies. To alleviate the difficulties associated with conventional dynamic programming a modified dynamic programming algorithm is developed. This practical algorithm, referred to as partitioned state space dynamic programming, is the main contribution of this study. It greatly reduces the high-speed memory requirement and the computation time necessary to achieve a numerical solution.

The essential distinction of the PSS algorithm is the partitioning of state space into blocks and the quadratic approximation of the cost surface over these blocks. The reduction of computation time is demonstrated in the examples in Chapter 4. For second order problems PSS requires as little as one-fifteenth the computation time of DP. Also, the new algorithm is accurate. The cost and control functions calculated by both methods have very small differences. The high-speed memory necessary to store the cost function in, say, Example 4.3 is 54 words for PSS and 800 words for DP. Thus, while high-speed memory is reduced neither algorithm begins to strain the limits of the CDC-6400 computer system. It is for 3rd and higher order problems that the high-speed memory reduction becomes important.

The description of the PSS algorithm in Chapter 3 and its application in Chapter 4 point out the main requirements of its application. They are,

1. Partitioning of state space into blocks.
2. Ordering the blocks.
3. Obtaining the cost surface of the origin block.

Partitioning and block ordering require insight into the specific problem being solved. These difficulties are intrinsic to the algorithm. If there is not sufficient *a priori* knowledge of the cost function and dynamical behavior to partition and order adequately then the problem may have to be solved several times. The solution of the origin block before the PSS algorithm is employed is very desirable since computation time is reduced, but it is not necessary. If difficulty is encountered in solving the origin block all of state space can be initialized to zero and the PSS algorithm begun in MODE = 2, as in Example 4.5.

A basic requirement on the stochastic systems studied is that the state not change excessively over one time period, i.e., not more than one block. It is seen in the examples that for such a restriction on the system dynamics, if the stage cost function does not result in appreciable second order curvature in the total cost function then the system can be modeled adequately as deterministic. Example 4.5 with a rather linear cost surface is of this nature.

The PSS algorithm finds its best application in problems for which the "curse of dimensionality" becomes prohibitive or in problems which must be solved repetitively with different parameters. In the second case, once the PSS algorithm is set up then its time advantage becomes very significant with repetition.

5.2 Further Research

The last two examples in Chapter 4 are related to reality; however, they are not truly practical problems dictated by an existing system. Probably the most fruitful extension of this study would be the application of the concepts developed here to practical systems whose solution is important.

The PSS algorithm takes advantage of the stationary cost surfaces resulting from the discount factor, $0 \leq \beta < 1$. In Section 2.8 it is shown that the control for $\beta = 1$ can be obtained by examining the control as $\beta \rightarrow 1$. However, in practice this is difficult since,

$$V_n(u) \simeq ng(u)1 + W(u), \; (\beta = 1) \tag{2.18}$$

and

$$\epsilon_n \leq \beta \epsilon_{n-1}. \tag{2.13}$$

Thus, for β very near to one the cost surface becomes very large, the convergence becomes very slow, and the PSS algorithm does not work well. An algorithm which handles the undiscounted case adeptly would be significant. The undiscounted cost criteria is of importance for continuous time systems being treated in discrete time.

Finally, both Riordon (1969) and Martin (1967) in treating the adaptive stochastic control problem employed variations of Howard's policy iteration. Thus, their methods are severely limited to a small finite state space. An investigation of methods to reduce the high-speed memory requirements for the adaptive problem would be worthwhile.

APPENDIX A

PARTITIONED STATE SPACE DYNAMIC PROGRAMMING ALGORITHM

The general explanation and flow diagram of the PSS algorithm is contained in Section 3.7 and Figure 3.5. This appendix presents a FORTRAN program listing of the PSS program employed to solve the first and second order examples of Chapter 4. Since the high-speed memory requirements of these examples do not exceed the capabilities of the CDC 6400 no referral is made to low-speed memory. All operations are performed in high-speed memory in this program. The logic necessary to implement the storage of the control and cost functions in low-speed memory would be contained in the main program, PSSDP (Figure A.1). All subsequent subroutines (Figures A.1, A.2, A.3) would remain unchanged.

The program listed is the one used to solve Example 4.5. The subroutines SLOSS, TRANFN, LOCATE, BLOCK, and CORNER are supplied by the user. SLOSS defines the stage cost, COST = $\ell(x,u)$ in terms of the state, X(I) = x_i, and the control, u. TRANFN defines the deterministic portion of the system dynamics, F(I) = $f_i(x,u)$. LOCATE is a subroutine which determines the block processing and storage location (LOC) from the block coordinate position (IX). BLOCK performs the inverse operation of finding the block storage location (LOC) given the block coordinate position (IX). CORNER finds the state coordinates of corner of the block (XLOW) such that,

$$\text{XLOW(I)} = \min \left\{ x_i \middle| x_i \varepsilon \text{ block} \right\}.$$

All other subroutines mentioned later are independent of the specific problem.

The problem being solved is completely specified by the subroutines mentioned in the previous paragraph and by the data input to the subroutine INITIAL. The input parameters to INITIAL are:

TITLE	-- title of the problem
M	-- order of the system
MAXLOC	-- maximum block location
NITER	-- number of iterations of the search routine (GOLDEN)
BETA	-- discount factor, β
UMIN,UMAX	-- bounds on the control
RESU	-- convergence criterion for the cost surface (ΔV in Figure 3.5)
IPB(I)	-- increments per block in the ith coordinate, this number must be even.
DELTAX(I)	-- Δx_i, state increment in the ith coordinate
R(I)	-- $R(1) = E\{\zeta_1\zeta_1\}$, $R(2) = E\{\zeta_1\zeta_2\}$, $R(3) = E\{\zeta_2\zeta_2\}$, $R(4) = E\{\zeta_3\zeta_1\}$, $\ldots, R\left(\frac{M(M+1)}{2}\right) = E\{\zeta_M\zeta_M\}$.

Also, in the INITIAL program listed in Figure A.1 the arrays determining block order and location for BLOCK and LOCATE are specified by FORTRAN statements. These statements may be modified if the user wishes to define BLOCK and LOCATE in a different way. If no a priori knowledge of the system motivates a preferred direction and hence a block ordering then the general block ordering techniques of Section 3.8 can be used.

The remaining program components found in Figure A.1 are:

PSSDP	-- main control program
IMPROVE	-- performs the minimization $\min_u \left\{ \ell(x,u) + \beta E\left\{\bar{v}\left[f(x,u) + \zeta\right]\right\}\right\}$ over all grid points in a block.
POLY	-- evaluates the parametric cost surface, $\bar{v}(x)$
Y(u)	-- evaluates $\left\{\ell(x,u) + \beta\left\{\bar{v}\left[f(x,u)\right] + \sum_{i=1}^{n}\sum_{j=1}^{i} \gamma_{ij}(\zeta_i\zeta_j)\right\}\right\}$
INWARD	-- a subroutine used by Y(u)
SCAL,TCAL	-- calculate S and $T(\ell)$ in $S\,Z(\ell) = T(\ell)$, (3.8) for a standard block with XLOW = 0
SIMEQ	-- Matrix inversion routine which solves (3.8),

$$Z(\ell) = S \quad T(\ell)$$

(not listed in Figure A.1)

PARAM	--	makes a linear transformation of $Z(\ell)$ from the standard block to the block being calculated
GOLDEN	--	golden section search routine.

The principal variables and arrays still undefined are:

Z(100,6)	--	storage for cost surface parameters defined in Section 3.6, $Z(\ell) = \left[Z(\ell,1),\ Z(\ell,2),\ \ldots,\ Z(\ell,6)\right]^T$
ZZ(6)	--	buffer storage for the current cost surface
UU(20,20)	--	control over current block
VV(20,20)	--	cost over current block
MODE,KEY	--	described in Section 3.7.

The origin block cost surface and the rest of state space may be initially set to zero and the PSS algorithm operated in MODE = 2 (cf. Example 4.). An alternative method is to calculate the origin block by some technique and then make one pass through state space in MODE = 1 as was done in Examples 4.1-4.4. The two origin block algorithms discussed in Section 3.9 are listed in Figures A.2 and A.3. The policy iteration algorithm is listed in Figure A.2 and the DP algorithm in Figure A.3. These subroutines need no input and introduce no new variables.

```
      PROGRAM   PSSDP (INPUT, OUTPUT, TAPE 5=INPUT, TAPE 6=OUTPUT)
         COMMON/SPEC1/M,MAXLOC,BETA     ,DELTAX(4),R(10),IPB(4)
         COMMON/SPEC2/UMIN,UMAX         ,RESU,NITER
         COMMON/ZSTORE/NZ,IS(10),JS(10),Z(100,6),ZZ(6)
         COMMON/CONTROL/UU(20,20),VV(20,20)
         DIMENSION S(6,6), T(6), SINV(6,6), IX(4)
         COMMON/PARIMP/XLOW(4)
         DIMENSION X(4),XX(4)
 1111    CALL INITIAL
 C ***   CALCULATE CONSTANTS AND INITIALIZE
         N1 = IPB(1) + 1
         N2 = IPB(2) +1
         NZ = (M**2 + 3*M + 2)/2
         LS=0
         DO 33 I=1,M
         IX(I) = 0
         DO 33 J=1,I
         LS = LS + 1
         IS(LS) = I
 33      JS(LS) = J
         MM=(M*(M+1))/2
         WRITE(6,261) (IS(I),JS(I),I=1,MM)
 261     FORMAT(5X,*IS AND IJ *,I2,2X,I2)
         WRITE(6,262) NZ
 262     FORMAT(5X,*NZ*,I2)
         CALL ORIGIN
         CALL SCAL(S)
         CALL SIMEQ (S, T, NZ, SINV, ZZ)
         GO TO 2001
         DO 1000 LOC=2,MAXLOC
         CALL BLOCK (LOC, IX)
```

Figure A.1 PSS algorithm FORTRAN listing.

```
          CALL IMPROVE (IX, LOC, 1)
          CALL TCAL (T)
          CALL PARAM(LOC,NZ,SINV,T,ZZ)
          WRITE(6,222) (ZZ(I), I=1,NZ)
222       FORMAT(//,*      ZZ  *,6(E12.5,2X))
          DO 50 I = 1, NZ
50        Z(LOC, I) = ZZ(I)
1000      CONTINUE
2001      DO 2000 IJK=2,14
          KEY=1
          WRITE(6,301) IJK
301       FORMAT(//,*      PASS *,I2,* THROUGH STATE SPACE*,//)
          DO 2002 LOC=1,MAXLOC
          CALL BLOCK (LOC, IX)
          CALL IMPROVE (IX,LOC,2)
          CALL TCAL (T)
          CALL PARAM(LOC,NZ,SINV,T,ZZ)
          WRITE(6,222) (ZZ(I), I=1,NZ)
          GO TO (79,49),KEY
79        DO 80 I1=1,N1
          F1=I1-1
          XX(1)=XLOW(1)+F1*DELTAX(1)
          DO 80 I2=  1,N2
          F2=I2-1
          XX(2)=XLOW(2)+F2*DELTAX(2)
          CALL POLY(XX,LOC,P)
          TEST=P-ZZ(1)-ZZ(2)*XX(1)-ZZ(3)*XX(2)-ZZ(4)*(XX(1)**2)
     1    -ZZ(5)*XX(1)*XX(2)-ZZ(6)*(XX(2)**2)
          IF(ABS(TEST)-RESU) 80,80,81
80        CONTINUE
          GO TO 49
81        KEY=2
49        DO 51 I=1,NZ
```

Figure A.1--<u>Continued</u> PSS algorithm FORTRAN listing.

```
51          Z(LOC,I)=ZZ(I)
2002        CONTINUE
            GO TO (1111,2000),KEY
2000        CONTINUE
            GO TO 1111
            END


         SUBROUTINE SLOSS(COST,X,U)
            DIMENSION X(4)
            COST=((X(2)-5.)/3.5)**2 + U**2 + 1.0/(((X(2)-41.)/2.)**2)
            RETURN
            END


         SUBROUTINE TRANFN(F,X,U )
            DIMENSION F(4), X(4)
            F(1)=X(1)
            UT=-.1*U $ FU=EXP(UT)
            F(2)=X(2)*FU + X(1)*(.161 - .111*FU - .01*U)
            RETURN
            END


          SUBROUTINE LOCATE (IX,LOCX)
            COMMON/SPEC1/M,MAXLOC,BETA    ,DELTAX(4),R(10),IPB(4)
            COMMON/LOCSTR/III,LOC(5,5)
            DIMENSION IX(4)
            I=IX(1)+III
            J=IX(2)+III
            MAXLAY=3
            DO 1 K=1,M
            IF(IABS(IX(K)) - MAXLAY) 1,1,2
```

Figure A.1--<u>Continued</u> PSS algorithm FORTRAN listing.

```
1        CONTINUE
         LOCX = LOC(I,J)
         RETURN
2        LOCX = 10000
         RETURN
         END


      SUBROUTINE BLOCK (LOCX,IX)
         COMMON/BLKSTR/IB(2,25)
         DIMENSION IX(4)
         IX(1)=IB(1,LOCX)
         IX(2)=IB(2,LOCX)
         RETURN
         END


      SUBROUTINE CORNER (IX,XLOW)
         DIMENSION IX(4),XLOW(4)
         XLOW(1)=0.0
         FIX=IX(2)-1
         XLOW(2)=4.0 + FIX*12.0
         RETURN
         END


      SUBROUTINE INITIAL
         COMMON/SPEC1/M,MAXLOC,BETA     ,DELTAX(4),R(10),IPB(4)
         COMMON/SPEC2/UMIN,UMAX         ,RESU,NITER
         COMMON/LOCSTR/III,LOC(5,5)
         COMMON/BLKSTR/IB(2,25)
         DIMENSION TITLE(8)
         READ(5,99) TITLE
```

Figure A.1--Continued PSS algorithm FORTRAN listing.

```
99        FORMAT(8A10)
          WRITE(6,199) TITLE
199       FORMAT(1H1,5X,8A10,///)
          READ(5,100) M, MAXLOC, NITER
100       FORMAT(3I10)
          IF(M) 2222,2222,2
2222      STOP
2         READ(5,101) BETA    ,UMIN,UMAX,RESU
101       FORMAT(4F10.0)
          WRITE(6,200) M, MAXLOC,BETA,    UMIN,UMAX,        RESU ,NITER
200       FORMAT(5X,*M = *,I2,*  MAXLOC = *,I2,*  BETA = *,F10.5,
     1          /,5X,*UMIN = *,F10.5,*  UMAX = *,F10.5,
     2*   RESU = *,F10.5,*  NITER = *,I2,//)
          READ(5,102) (I,IPB(I),DELTAX(I),J=1,M)
102       FORMAT(2I10,F10.0)
          WRITE(6,201) (I,IPB(I),DELTAX(I),I=1,M)
201       FORMAT(5X,I2,2X,I2,2X,F10.5)
          MM = (M*(M+1))/2
          READ(5,103) (I,R(I),J=1,MM)
103       FORMAT(I10,F10.0)
          WRITE(6,202) (R(I),I=1,MM)
202       FORMAT(5X,10(2X,F10.5))
          III=0
          LOC(1,1)=1
          LOC(1,2)=2
          LOC(1,3)=3
          IB(1,1)=1 $IB(2,1)=1
          IB(1,2)=1 $ IB(2,2)=2
          IB(1,3)=1 $ IB(2,3)=3
          RETURN
          END
```

Figure A.1--Continued PSS algorithm FORTRAN listing.

```
      SUBROUTINE IMPROVE(IX, LOC, MODE)
         COMMON/SPEC1/M,MAXLOC,BETA     ,DELTAX(4),R(10),IPB(4)
         COMMON/SPEC2/UMIN,UMAX         ,RESU,NITER
         COMMON/CONTROL/UU(20,20),VV(20,20)
         COMMON/YSTORE/X(4),LOCY , MODEY
         COMMON/TEMPORY/LOCX1,F(4),COST  ,P,SUM
         COMMON/PARIMP/XLOW(4)
         DIMENSION IX(4)
         MODEY=MODE
         LOCY=LOC
         KEY=1
         CALL CORNER (IX,XLOW)
         WRITE(6,198) LOC, (IX(I), I=1,M)
198      FORMAT(///,*     LOC *,I2,*    BLOCK  *,4I4)
         WRITE(6,199)
199      FORMAT(///,5X,*     X(1)        X(2)        U                  V*)
         N1=IPB(1)+1
         N2=IPB(2)+1
         DO 10 I1=1,N1
         F1 = I1 -1
         X(1) = XLOW(1) + F1*DELTAX(1)
         DO 10 I2=1,N2
         F2=I2-1
         X(2)= XLOW(2) + F2*DELTAX(2)
         CALL GOLDEN(UMIN,UMAX,UU(I1,I2),VV(I1,I2),NITER)
         WRITE(6,200) X(1), X(2), UU(I1,I2), VV(I1,I2), LOCX1, F(1),F(2)
     1   ,COST     ,P,SUM
200      FORMAT(5X,F6.2,5X,F6.2,2X,F10.5,5X,E12.4,*  LOCX1 *,I2,* F*,F8.
     12,2X,F8.2,2X,F8.2,,2X,F8.2,2X,F8.2)
10       CONTINUE
         RETURN
         END
```

Figure A.1--Continued PSS algorithm FORTRAN listing.

```
      SUBROUTINE POLY(X,LOC,P)
         COMMON/SPEC1/M,MAXLOC,BETA     ,DELTAX(4),R(10),IPB(4)
         COMMON/ZSTORE/NZ,IS(10),JS(10),Z(100,6),ZZ(6)
         DIMENSION X(4)
         MP1=M+1
         MP2 = M+2
         P = Z(LOC,1)
         DO 1 I=1,M
1        P = P + Z(LOC,I+1)*X(I)
         DO 2 I=MP2,NZ
         ISI=IS(I-MP1)
         JSI=JS(I-MP1)
2        P=P + Z(LOC,I)*X(ISI)*X(JSI)
         RETURN
         END


      FUNCTION Y(U)
         COMMON/SPEC1/M,MAXLOC,BETA     ,DELTAX(4),R(10),IPB(4)
         COMMON/SPEC2/UMIN,UMAX         ,RESU,NITER
         COMMON/ZSTORE/NZ,IS(10),JS(10),Z(100,6),ZZ(6)
         COMMON/YSTORE/X(4),LOCY , MODEY
         COMMON/TEMPORY/LOCX1,F(4),COST  ,P,SUM
         DIMENSION       IX1(4), IDUM(4)
         CALL SLOSS(COST,X,U)
         CALL TRANFN(F,X,U)
C   ***   FIND BLOCK OF F
         IX1(1)=1
         IF(F(2)-4.0) 6,6,7
6        IX1(2)=1
         GO TO 8
```

Figure A.1--Continued PSS algorithm FORTRAN listing.

```
7          IX1(2)=(F(2)-4.0)/12.0 + 1.0
8          CALL LOCATE(IX1,LOCX1)
           GO TO (1001,1003),MODEY
1001       IF(LOCY-1) 18,18,19
18         LOCX1=LOCY
           GO TO 333
19         IF(LOCX1 - LOCY) 333,401,401
401        CALL INWARD(IX1,IDUM)
           DO 21 I=1,M
21         IX1(I)=IDUM(I)
           CALL LOCATE(IX1,LOCX1)
           GO TO 19
1003       IF(LOCX1-MAXLOC) 333,333,403
403        CALL INWARD(IX1,IDUM)
           DO 23 I=1,M
23         IX1(I)=IDUM(I)
           CALL LOCATE (IX1,LOCX1)
           GO TO 1003
333        CALL POLY(F,LOCX1,P)
           MP1=M+1
           MP2=M+2
           SUM=0.0
           DO 1 I=MP2,NZ
1          SUM=SUM+R(I-MP1)*Z(LOCX1,I)
           Y= COST + BETA*(P+SUM)
           RETURN
           END

        SUBROUTINE INWARD (IN,IO)
           COMMON/SPEC1/M,MAXLOC,BETA     ,DELTAX(4),R(10),IPB(4)
           DIMENSION IN(4), IO(4)
           MAX = 0
```

Figure A.1--Continued PSS algorithm FORTRAN listing.

```
         DO 10 I = 1, M
         ITEMP=IABS(IN(I))
         IF(ITEMP - MAX) 10,10,11
11       MAX = ITEMP
10       CONTINUE
         DO 20 I = 1, M
         IF(IN(I)) 21, 24, 22
21       IF (IN(I) + MAX) 23, 23, 24
23       IO(I) = IN(I) + 1
         GO TO 20
22       IF (IN(I) - MAX)  24, 25, 25
25       IO(I) = IN(I) - 1
         GO TO 20
24       IO(I) = IN(I)
20       CONTINUE
         RETURN
         END


      SUBROUTINE SCAL(S)
         COMMON/SPEC1/M,MAXLOC,BETA    ,DELTAX(4),R(10),IPB(4)
         COMMON/ZSTORE/NZ,IS(10),JS(10),Z(100,6),ZZ(6)
         DIMENSION S(6,6)    , X(4)
         DO 5 I=1,NZ
         T(I)=0.0
         DO 5 J=1,NZ
5        S(I,J) = 0.0
         MP1=M+1
         MP2=M+2
C   ***   THE FOLLOWING GROUP OF NESTED DO LOOPS
C   ***   MUST BE CHANGED WHEN M CHANGES
         N1=IPB(1)+1
         N2=IPB(2)+1
```

Figure A.1--<u>Continued</u> PSS algorithm FORTRAN listing.

```
            DO 10 I1=1,N1
            F1 = I1-1
            X(1) = F1*DELTAX(1)
            DO 10 I2=1,N2
            F2=I2-1
            X(2) = F2*DELTAX(2)
C     ***    CALCULATE ROW 1 OF S
            S(1,1)=S(1,1) + 1
            DO 11  I=1,M
11          S(1,I+1) = S(1,I+1) + X(I)
            DO 12  LS=MP2,NZ
            ISLS=IS(LS-MP1)
            JSLS=JS(LS-MP1)
12          S(1,LS)=S(1,LS) + X(ISLS  )*X(JSLS  )
C     ***    CALCULATE ROW 2......M+1
            DO 13 LL=1,M
            S(LL+1,1) =S(LL+1,1) + X(LL)
            DO 14 I=1,M
14          S(LL+1,I+1) = S(LL+1,I+1) + X(I)*X(LL)
            DO 13 LS = MP2,NZ
            ISLS=IS(LS-MP1)
            JSLS=JS(LS-MP1)
13          S(LL+1,LS) = S(LL+1,LS) + X(ISLS  )*X(JSLS  )*X(LL)
C     ***    CALCULATE ROW M+2.........,NZ
            DO 15 K=MP2,NZ
            ISK=IS(K-MP1)
            JSK=JS(K-MP1)
            TEMP=X(ISK)*X(JSK)
            S(K,1) = S(K,1) + TEMP
            DO 16 I=1,M
16          S(K,I+1)=S(K,I+1) + X(I)*TEMP
            DO 15 LS=MP2,NZ
            ISLS=IS(LS-MP1)
```

Figure A.1--Continued PSS algorithm FORTRAN listing.

```
        JSLS=JS(LS-MP1)
15      S(K,LS) = S(K,LS) +             TEMP*X(ISLS  )*X(JSLS  )
10      CONTINUE
        RETURN
        END


      SUBROUTINE TCAL(T)
        COMMON/SPEC1/M,MAXLOC,BETA    ,DELTAX(4),R(10),IPB(4)
        COMMON/ZSTORE/NZ,IS(10),JS(10),Z(100,6),ZZ(6)
        COMMON/CONTROL/UU(20,20),VV(20,20)
        DIMENSION T(6),X(4)
        DO 5 I=1,NZ
5       T(I)=0.0
        MP1=M+1
        MP2=M+2
C   ***  THE FOLLOWING GROUP OF NESTED DO LOOPS
C   ***  MUST BE CHANGED WHEN M CHANGES
        N1=IPB(1)+1
        N2=IPB(2)+1
        DO 10 I1=1,N1
        F1 = I1-1
        X(1)=F1*DELTAX(1)
        DO 10 I2=1,N2
        F2=I2-1
        X(2)=F2*DELTAX(2)
        V = VV(I1,I2)
C   ***  CALCULATE ROW 1
        T(1)=T(1) + V
        DO 13 LL=1,M
13      T(LL+1)=T(LL+1) + V*X(LL)
        DO 15 K=MP2,NZ
        ISK=IS(K-MP1)
```

Figure A.1--<u>Continued</u> PSS algorithm FORTRAN listing.

```
      JSK=JS(K-MP1)
15    T(K)=T(K)+V*X(ISK)*X(JSK)
10    CONTINUE
      RETURN
      END


      SUBROUTINE PARAM(LOC,NZ,SINV,T,ZZ)
      COMMON/SPEC1/M,MAXLOC,BETA    ,DELTAX(4),R(10),IPB(4)
      COMMON/PARIMP/XLOW(4)
      DIMENSION SINV(6,6),T(6),ZZ(6)
      DO 1 I=1,NZ
      ZZ(I)=0.0
      DO 1 J=1,NZ
1     ZZ(I)=ZZ(I)+SINV(I,J)*T(J)
      ZZ(1)=ZZ(1)-ZZ(2)*XLOW(1)-ZZ(3)*XLOW(2)+ZZ(4)*(XLOW(1)**2)
     1 +ZZ(5)*XLOW(1)*XLOW(2)+ZZ(6)*(XLOW(2)**2)
      ZZ(2)=ZZ(2)-2.0*ZZ(4)*XLOW(1)-ZZ(5)*XLOW(2)
      ZZ(3)=ZZ(3)-2.0*ZZ(6)*XLOW(2)-ZZ(5)*XLOW(1)
      RETURN
      END


      SUBROUTINE GOLDEN(UMIN,UMAX,UGOLD,YGOLD,NITER)
C   ***   SEARCH IS MADE BY GOLDEN SECTION FOR THE MINIMUM OF Y(U),
C   ***   UMIN.LE.U.LE.UMAX,  A FUNCTION SUBROUTINE Y(.) IS NECESSARY
C   ***   THE MINIMUM IS LOCATED WITHIN AN INTERVAL  ( UI,UF)
C   ***   ABS(UI-UF) = (R**NITER)*ABS(UMAX-UMIN)      R=.618033989...
C   ***   (NITER,R**NITER),(1,.618), (2,.382), (3,.236), (4,.146), (5,.09),
C   ***   (6,.055), (7,.034), (8,.021), (9,.013), (10,.008),(11,.005),
C   ***   (12,.003), (13,.002), (14,.0012), (15,.0007)
C   ***   GOLDEN MAKES (NITER + 1) FUNCTION EVALUATIONS
      R=.618033989
```

Figure A.1--Continued PSS algorithm FORTRAN listing.

```
            ITER=0
C    ***     FIRST ITERATION ... INITIALIZATION
            UI=UMIN
            UF=UMAX
            DELTA=UMAX-UMIN
            U1=UF-R*DELTA
            U2=UI+R*DELTA
            Y1=Y(U1)
            Y2=Y(U2)
C    ***     ITERATION CYCLE BEGINS
101         ITER=ITER+1
            IF(Y1-Y2) 1, 1, 2
1           UF=U2
            IF(NITER-ITER) 1111, 1111, 4
C    ***     THE SEARCH IS COMPLETE,  ITER=NITER
1111        UGOLD = .5*(UI+UF)
            YGOLD = Y1
            RETURN
C    ***     SEARCH CONTINUES, REDEFINE POINTS
4           U2=U1
            Y2=Y1
            DELTA=UF-UI
            U1=UF-R*DELTA
            Y1=Y(U1)
            GO TO 101
2           UI=U1
            IF(NITER-ITER) 1112, 1112, 8
C    ***     THE SEARCH IS COMPLETE,  ITER=NITER
1112        UGOLD = .5*(UI+UF)
            YGOLD = Y2
            RETURN
C    ***     SEARCH CONTINUES, REDEFINE POINTS
8           U1=U2
```

Figure A.1--<u>Continued</u> PSS algorithm FORTRAN listing.

```
Y1=Y2
DELTA=UF-UI
U2=UI+R*DELTA
Y2=Y(U2)
GO TO 101
END
```

Figure A.1--Continued PSS algorithm FORTRAN listing.

```
      SUBROUTINE ORIGIN
         COMMON/SPEC1/M,MAXLOC,BETA      ,DELTAX(4),R(10),IPB(4)
         COMMON/SPEC2/UMIN,UMAX          ,RESU,NITER
         COMMON/ZSTORE/NZ,IS(10),JS(10),Z(100,6),ZZ(6)
         COMMON/CONTROL/UU(20,20),VV(20,20)
         DIMENSION F(4), XLOW(4), X(4), SS(6,6), TT(6), SINV(6,6)
         DIMENSION UOLD(20,20),IX(4)
         LCAL=1
         MP1=M+1
         WRITE(6,250) LCAL, (IX(I),     I=1,M)
250      FORMAT(///,5X,*LOCATION =*,I3,*   BLOCK   *,4(2X,I3,2X)///)
         ITER = 0
         N1=IPB(1)+1
         N2=IPB(2)+1
         DO 60  I1=1,N1
         DO 60  I2=1,N2
60       UU(I1,I2)=0.0
501      CALL CURRENT (IX,SS,TT)
         CALL NORMAL (SS,TT,NZ)
         CALL SIMEQ (SS,TT,NZ,SINV,ZZ)
         DO 50 I = 1,NZ
50       Z(1,I) = ZZ(I)
         WRITE(6,270) (Z(1,I), I=1,NZ)
         ITER = ITER + 1
         WRITE (6,255) ITER
255      FORMAT (///,5X,*ITERATION = *,I3,///)
         DO 61  I1=1,N1
         DO 61  I2=1,N2
61       UOLD(I1,I2)=UU(I1,I2)
         CALL IMPROVE(IX, 1, 1)
         DO 62 I1=1,N1
```

Figure A.2 Policy iteration origin block algorithm.

```
         DO 62 I2=1,N2
         IF(ABS(UOLD(I1,I2) - UU(I1,I2)) - RESU) 62,62,52
62       CONTINUE
         GO TO  222
52       IF(15 - ITER) 222,  501,  501
222      CALL IMPROVE (IX, 1, 1)
         CALL BNDRY (IX,SS,TT)
         CALL SIMEQ (SS, TT, NZ, SINV, ZZ)
         DO 55 I=1,NZ
55       Z(1,I)=ZZ(I)
         WRITE(6,270)  (Z(1,I), I=1,NZ)
270      FORMAT(//,5X,*ZZ *,6(E12.5,2X))
         RETURN
         END


      SUBROUTINE CURRENT(IX,S,T)
         COMMON/SPEC1/M,MAXLOC,BETA     ,DELTAX(4),R(10),IPB(4)
         COMMON/ZSTORE/NZ,IS(10),JS(10),Z(100,6),ZZ(6)
         COMMON/CONTROL/UU(20,20),VV(20,20)
         DIMENSION S(6,6), T(6), IX(4), F(4),XLOW(4),X(4)
         DO 5 I=1,NZ
         T(I)=0.0
         DO 5 J=1,NZ
5        S(I,J) = 0.0
         DO 1 I=1,M
         FLT = 2*IX(I)*IPB(I) - IPB(I)
1        XLOW(I) = 0.5*FLT*DELTAX(I)
         MP1=M+1
         MP2=M+2
C    ***  THE FOLLOWING GROUP OF NESTED DO LOOPS
C    ***  MUST BE CHANGED WHEN M CHANGES
         N1=IPB(1)+1
```

Figure A.2--<u>Continued</u> Policy iteration origin block algorithm.

```
      DO 10 I1=1,N1
      F1 = I1-1
      X(1) = XLOW(1) + F1*DELTAX(1)
      N2=IPB(2)+1
      DO 10 I2=1,N2
      F2=I2-1
      X(2) = XLOW(2) + F2*DELTAX(2)
      CALL TRANFN (F,X,UU(I1,I2))
      CALL SLOSS(COST,X,UU(I1,I2))
C   ***  CALCULATE ROW 1 OF S AND T
      S(1,1)=S(1,1) + 1.0 - BETA
      DO 11 I=1,M
11    S(1,I+1)=S(1,I+1) + X(I) - BETA*F(I)
      DO 12 LS=MP2,NZ
      ISLS=IS(LS-MP1)
      JSLS=JS(LS-MP1)
12    S(1,LS) = S(1,LS) + X(ISLS  )*X(JSLS  ) - BETA*(F(ISLS  )*F(JSL
     1S  ) + R(LS-MP1))
      T(1)=T(1) + COST
C   ***  CALCULATE ROW 2......M+1
      DO 13 LL=1,M
      TEMP = X(LL) - BETA*F(LL)
      S(LL+1,1) = S(LL+1,1) + TEMP*(1.0-BETA)
      T(LL+1)=T(LL+1) +COST*TEMP
      DO 14 I=1,M
14    S(LL+1,I+1)=S(LL+1,I+1) + (X(I) - BETA*F(I))*TEMP
      DO 13 LS=MP2,NZ
      ISLS=IS(LS-MP1)
      JSLS=JS(LS-MP1)
13    S(LL+1,LS) = S(LL+1,LS) + (X(ISLS  )*X(JSLS  ) - BETA*(F(ISLS
     1)*F(JSLS  ) + R(LS-MP1)))*TEMP
C   ***  CALCULATE ROW M+2.........NZ
```

Figure A.2--<u>Continued</u> Policy iteration origin block algorithm.

```
      DO 16 K=MP2,NZ
      ISK=IS(K-MP1)
      JSK=JS(K-MP1)
      TEMP = X(ISK  )*X(JSK  ) - BETA*(F(ISK  )*F(JSK  ) + R(K-MP1))
      S(K,1) = S(K,1) + (1.0-BETA)*TEMP
      T(K)=T(K) + COST*TEMP
      DO 17 I=1,M
17    S(K,I+1) = S(K,I+1) + (X(I) - BETA*F(I))*TEMP
      DO 16 LS=MP2,NZ
      ISLS=IS(LS-MP1)
      JSLS=JS(LS-MP1)
16    S(K,LS) = S(K,LS) + (X(ISLS  )*X(JSLS  ) - BETA*(F(ISLS  )*F(JS
     1LS  ) + R(LS-MP1)))*TEMP
10    CONTINUE
      RETURN
      END
     SUBROUTINE NORMAL (A,B,NZ)
      DIMENSION A(6,6), B(6)
      DO 50 I =1,NZ
      AMAX = 0.0
      DO 51 J = 1,NZ
      TEMP = ABS (A(I,J))
      IF (TEMP-AMAX) 51,51,2
2     AMAX = TEMP
51    CONTINUE
      DO 52 J = 1,NZ
52    A(I,J) = A(I,J) / AMAX
50    B(I) = B(I) / AMAX
      RETURN
      END
```

Figure A.2--<u>Continued</u> Policy iteration origin block algorithm.

```
      SUBROUTINE ORIGIN
      COMMON/SPEC1/M,MAXLOC,BETA      ,DELTAX(4),R(10),IPB(4)
      COMMON/SPEC2/UMIN,UMAX          ,RESU,NITER
      COMMON/ZSTORE/NZ,IS(10),JS(10),Z(100,6),ZZ(6)
      COMMON/CONTROL/UU(20,20),VV(20,20)
      COMMON/PARIMP/XLOW(4)
      DIMENSION SINV(6,6),ZOLD(6),T(6)   , IX(4)   , S(6,6)  ,XX(4)
      CALL SCAL(S)
      CALL SIMEQ(S,T,NZ,SINV,ZZ)
      ITER=0
      DO 2 I=1,M
2     IX(I)=0
      DO 1 I=1,NZ
1     Z(1,I)=0.0
501   ITER=ITER+1
      KEY=1
      DO 3 I=1,NZ
3     ZOLD(I)=Z(1,I)
      WRITE(6,255) ITER
255   FORMAT(///,5X,*ITERATION = *,I3,///)
      CALL IMPROVE (IX,1,1)
      CALL TCAL(T)
```

Figure A.3 DP origin block algorithm.

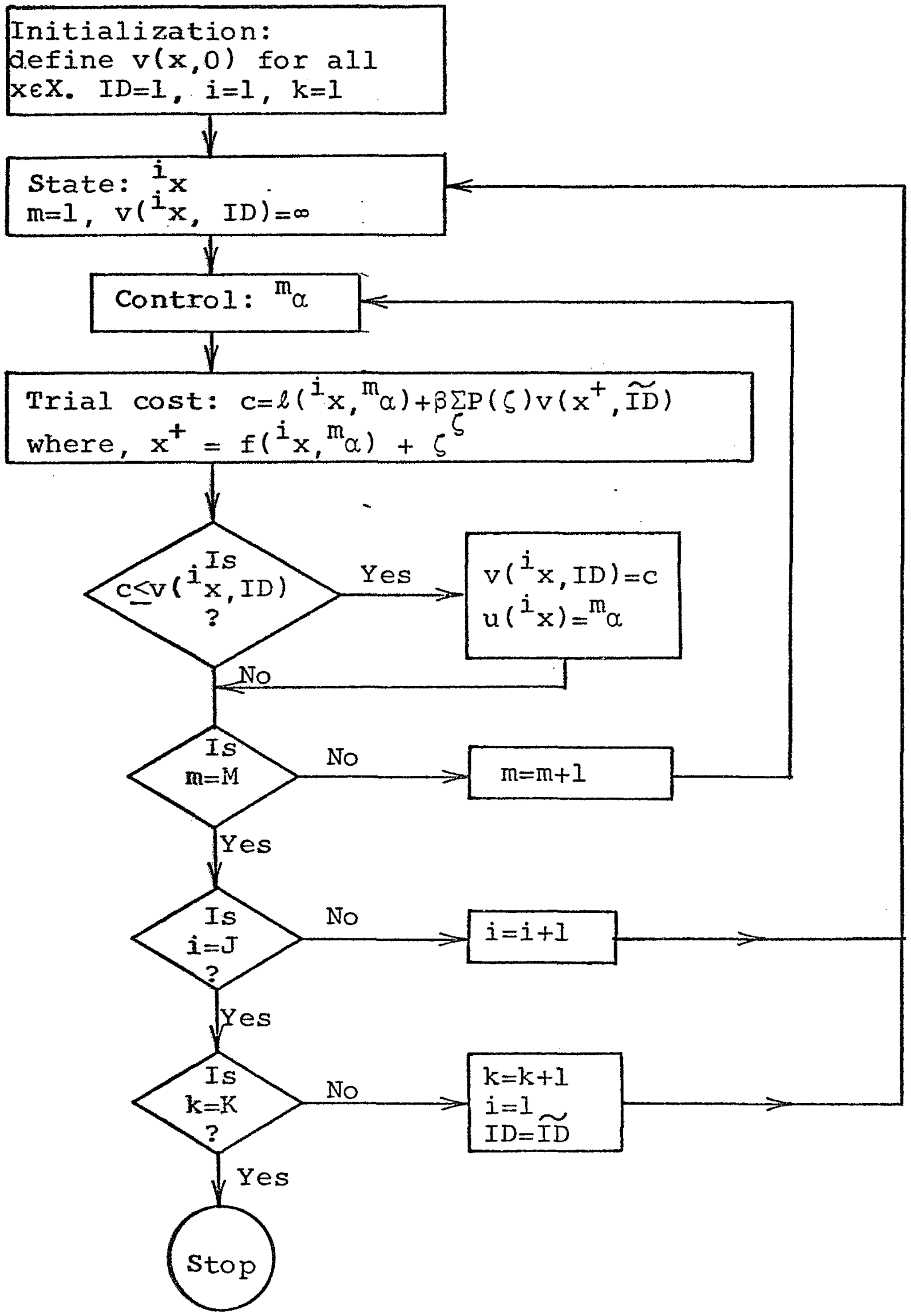
```
        CALL PARAM (1,NZ,SINV,T,ZZ)
        WRITE(6,222) (ZZ(I),I=1,NZ)
222     FORMAT(//,*       ZZ   *,6(E12.5,2X))
        GO TO (79,49),KEY
79      DO 80 I1=1,N1
        F1=I1-1
        XX(1)=XLOW(1)+F1*DELTAX(1)
        DO 80 I2=  1,N2
        F2=I2-1
        XX(2)=XLOW(2)+F2*DELTAX(2)
        TEST=ZOLD(1)+ZOLD(2)*XX(1)+ZOLD(3)*XX(2)+ZOLD(4)*(XX(1)**2)
     1  +ZOLD(5)*XX(1)*XX(2)+ZOLD(6)*(XX(2)**2)-ZZ(1)-ZZ(2)*XX(1)
     2  -ZZ(3)*XX(2)-ZZ(4)*(XX(1)**2)-ZZ(5)*XX(1)*XX(2)-ZZ(6)*(XX(2)**2
     3  )
        IF(ABS(TEST)-RESU) 80,80,81
80      CONTINUE
        GO TO 49
81      KEY=2
49      DO 51 I=1,NZ
51      Z(1,I)=ZZ(I)
        IF(15-ITER) 2222,501,501
2222    RETURN
        END
```

Figure A.3--Continued

APPENDIX B

CONVENTIONAL DYNAMIC PROGRAMMING ALGORITHM

For the time invariant stochastic control problem with discounted performance criterion both the control and cost function are stationary. Thus, only two cost functions must be stored to employ the DP functional equation (3.5) iteratively. This is accomplished by counting in MOD/2 the subscript ID which identifies the current and previously calculated cost surface. Figure B.1 gives a flow chart of the DP algorithm. The state space is $X = \left\{ {}^{1}x,\ {}^{2}x,\ \ldots {}^{J}x \right\}$ and the control space is $A = \left\{ {}^{1}\alpha,\ {}^{2}\alpha,\ \ldots,\ {}^{M}\alpha \right\}$. K is the number of iterations of the functional equation. The random variable ζ has been quantized and has a probability mass function $p(\zeta)$.



Initialization: define $v(x,0)$ for all $x \epsilon X$. ID=1, i=1, k=1
State: ${}^i x$
m=1, $v({}^i x, \text{ID}) = \infty$
Control: ${}^m \alpha$
Trial cost: $c = \ell({}^i x, {}^m \alpha) + \beta \sum_\zeta P(\zeta) v(x^+, \widetilde{\text{ID}})$
where, $x^+ = f({}^i x, {}^m \alpha) + \zeta$
Is $c \leq v({}^i x, \text{ID})$?
Yes
$v({}^i x, \text{ID}) = c$
$u({}^i x) = {}^m \alpha$
No
Is m=M
No
m=m+1
Yes
Is i=J ?
No
i=i+1
Yes
Is k=K ?
No
k=k+1
i=1
$\text{ID} = \widetilde{\text{ID}}$
Yes
Stop

Figure B.1 Dynamic programming flow chart.

REFERENCES

Aoki, M. Optimization of Stochastic Systems. Academic Press, New York, 1967.

Arrow, K. J., T. Harris, and J. Marshak. "Optimal Inventory Policy." Econometrica, Vol. 19, No. 3, 1951.

Beckman, M. J. Dynamic Programming of Economic Decisions. Springer-Verlag New York Inc., New York, 1968.

Bellman, R. "Dynamic Programming and Stochastic Control Processes." Information and Control, Vol. 1, 1958.

Bellman, R. Adaptive Control Processes: A Guided Tour. Princeton University Press, Princeton, 1961.

Blackwell, D. "Discrete Dynamic Programming." Annals of Mathematical Statistics, Vol. 33, pt. 1, 1962.

Blackwell, D. "Discounted Dynamic Programming." Annals of Mathematical Statistics, Vol. 36, pt. 1, 1965.

Derman, C. "On Sequential Control Processes." Annals of Mathematical Statistics, Vol. 35, pt. 1, 1964.

Doob, J. L. Stochastic Processes. Wiley, New York, 1953.

Dreyfus, S. E. "Some Types of Optimal Control of Stochastic Systems." S.I.A.M. Journal on Control, Ser. A, Vol. 2, No. 1, 1962.

Feldbaum, A. A. "Dual Control Theory, I." Automatika i Telemekhanika, September 1960.

Howard, R. A. Dynamic Programming and Markov Processes. M.I.T. Press, Cambridge, 1960.

Kashyap, R. L. "Optimization of Stochastic Finite State Systems." IEEE Transactions on Automatic Control, Vol. AC-11, No. 4, October, 1966.

Larson, R. E. "Dynamic Programming with Continuous Independent Variable," Tech. Report No. 6302-6, Stanford Electronics Laboratories, Stanford, Calif., 1964.

Larson, R. E. State Increment Dynamic Programming. American Elsevier Publishing Company, Inc., New York, 1968.

Martin, J. J. Bayesian Decision Problems and Markov Chains. John Wiley and Sons, Inc., New York, 1967.

Meier, L. "Combined Optimum Control and Estimation Theory." NASA Contractor Report, NASA CR-426, April, 1966.

Miller, B. L. "Finite State Continuous Time Markov Decision Processes with Finite Planning Horizon." S.I.A.M. Journal on Control, Vol. 6, No. 2, 1968a.

Miller, B. L. "Finite State Continuous Time Markov Decision Processes with Infinite Planning Horizon." Journal of Mathematical Analysis and Applications, Vol. 22, 1968b.

Miller, B. L. and A. F. Veinott. "Discrete Dynamic Programming with a Small Interest Rate." Annals of Mathematical Statistics, Vol. 40, No. 2, 1969.

Riordon, J. S. "An Adaptive Automation Controller for Discrete-Time Markov Processes." Automatica, Vol. 5, No. 6, November, 1969.

Sage, A. P. Optimum Systems Control, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1968.

Scarf, H. E. "The Optimality of (S,s) Policies in the Dynamic Inventory." in Mathematical Methods in the Social Sciences, K. J. Arrow, S. Karlin, and P. Suppes, eds., Stanford University Press, Stanford, 1960.

Veinott, A. F. "Discrete Dynamic Programming with Sensitivity Discount Optimality Criteria." Annals of Mathematical Statistics, Vol. 40, No. 5, 1969.

Wilde, J. D. and Beightler, C. S. Foundations of Optimization, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1967.

Wonham, W. M. "Lecture Notes on Stochastic Control." Brown University, 1967.