c.1

COMPUTATIONAL METHODS FOR MARKOV DECISION PROBLEMS

by

MOON CHIRL SHIN

B.Sc., Seoul National University, 1969
M.Sc., Texas Tech University, 1974

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES

(Management Science)

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

August 1980

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the Head of my Department or by his representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Management Science

The University of British Columbia
2075 Wesbrook Place
Vancouver, Canada
V6T 1W5

Date August 1980

ABSTRACT

In this thesis we study computational methods for finite discounted Markov decision problems and finite discounted parametric Markov decision problems over an infinite horizon. For the former problem our emphasis is on finding methods to significantly reduce the effort required to determine an optimal policy. We discuss the implementation of Porteus' scalar extrapolation methods in the modified policy iteration algorithm and show that the results using only a final scalar extrapolation will be the same as those obtained by applying scalar extrapolation at each iteration and then using a final scalar extrapolation. Action elimination procedures for policy iteration and modified policy iteration algorithms are presented. The purpose of these techniques is to reduce the size of the action space to be searched in the improvement phase of the algorithm. A method for eliminating non-optimal actions for all subsequent iterations using upper and lower bounds on the optimal expected total discounted return is presented along with procedures for eliminating actions that cannot be part of the policy chosen in the improvement phase of the next iteration. A numerical comparison of these procedures on Howard's automobile replacement problem and on a large randomly generated problem suggests that using modified policy iteration together with one of the single iteration elimination procedures will lead to large savings in the computational time for problems with large state spaces. Modifications of the algorithm to reduce storage space are also discussed.

For the finite discounted Markov decision problems in which the reward vector is parameterized by a scalar we present an algorithm to determine the optimal policy for each value of the parameter within an interval. The algorithm is based on using approximations of values to resolve difficulties caused by roundoff error. Also, several action elimination procedures are presented for this problem. Bi-criterion Markov decision problems and Markov decision problems with a single constraint are formulated as parametric Markov decision problems. A numerical comparison of algorithms with and without action elimination procedures is carried out on a two criterion version of Howard's automobile replacement problem. The results suggest that the algorithm with one of the action elimination procedures will lead to efficient solution of this problem.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

I wish to thank Professor Martin L. Puterman for his help, advice and especially for his careful reading of my several drafts of this thesis; Sharon Haller, for turning hundreds of pages of scrawls and correction into this final manuscript; and my wife Yaeree Oh for encouraging me throughout this ordeal.

In addition, I am grateful to the National Science Engineering Research Council for the financial support through Grant A-3609.

CHAPTER I

INTRODUCTION

In this thesis we consider computational methods for finite state and action infinite horizon discounted Markov decision problems. In Chapter 2 we propose several methods to improve the efficiency of methods for solving these problems and in Chapter 3 present algorithms for solving finite discounted Markov decision problems parameterized by a single scalar.

The policy iteration method is usually attributed to Howard [14]. Later, Puterman and Shin [26] and van Nunen [29,30] developed independently the modified policy iteration to reduce computation for problems with large state spaces. In Chapter 2 we are concerned with increasing the efficiency of the policy iteration and modified policy iteration procedures. Porteus and Totten [23] and Porteus [21] investigated the use of extrapolation methods such as the trivial scalar extrapolation, the trivial lower bound extrapolation, row sum extrapolation and delayed scalar extrapolation to reduce the computational effort in the evaluation of expected total discounted return of a Markov and semi-Markov chain. All of these extrapolations reduce to scalar extrapolations in the discounted Markov decision problem considered here. We discuss the implementation of these scalar extrapolation methods in the modified policy iteration algorithm and show that the results using only a final scalar extrapolation will be the same as those obtained by applying scalar extrapolation at each iteration and then using a final scalar extrapolation.

To reduce computations in the improvement phase the idea of using bounds on the optimal return function to identify actions that are not part of an optimal policy was introduced by MacQueen [16]. Then Porteus [19,20], Hastings and Mello [10], Hastings [11], Hastings and van Nunen [12] and Hübner [15] studied action elimination procedures for value iteration algorithms and variants. Grinold [9] studied an action elimination procedure for policy iteration. Motivated by the discovery in Puterman and Shin [26] that modified policy iteration was considerably more efficient than value iteration and in many cases policy iteration, we introduce bounds and action elimination procedures for the policy iteration and modified policy iteration algorithms. A method for eliminating non-optimal actions for all subsequent iterations using upper and lower bounds on the optimal expected total discounted return is presented along with procedures for eliminating actions that cannot be part of the policy chosen in the improvement phase of the next iteration. A numerical comparison of these procedures on Howard's [14] automobile replacement problem and on a large structured randomly generated problem suggests that using modified policy iteration together with one of the single iteration elimination procedures will lead to large savings in the computational time for problems with large state spaces. Modifications of the algorithms to reduce storage space are also discussed.

In Chapter 3 we study finite discounted Markov decision problems in which the reward vector is parameterized by a scalar and present an algorithm to determine the optimal policy for each value of the parameter within an interval. The algorithm is based on a modification of parametric linear programming in which approximations of values are used to resolve difficulties caused by roundoff error. Also, several action elimination

procedures are presented for this problem. Bi-criterion Markov decision problems and Markov decision problems with a single constraint are then formulated as parametric Markov decision problems. A numerical comparison of algorithms with and without action elimination procedures is carried out on a two criterion version of Howard's [14] automobile replacement problem. The results suggest that using the algorithm with one of the action elimination procedures is an efficient method for solving this problem.

CHAPTER II

IMPROVED COMPUTATIONAL METHODS FOR DISCOUNTED MARKOV DECISION PROBLEMS

1. Introduction.

In this chapter we consider methods for increasing the efficiency of the policy iteration (Howard [14]) and modified policy iteration (Puterman and Shin [26] and van Nunen [31,32]) algorithms for solving finite discounted Markov decision problems over an infinite horizon. The modified policy iteration algorithm was suggested by Porteus [19] and Morton [18] and its properties were studied by van Nunen [31,32], Puterman and Brumelle [24,25], Puterman and Shin [26], van der Wal [30] and Rothblum [27].

In a series of papers Porteus [21] and Porteus and Totten [23] investigated the use of extrapolation methods to reduce the computational effort to evaluate the expected total discounted return of a Markov or semi-Markov chain. Porteus and Totten [23] sought improved iterative methods such as the trivial scalar extrapolation and the trivial lower bound extrapolation to compute the expected discounted return in a Markov chain. Recently Porteus [21] studied other improved iterative schemes such as row sum extrapolation and delayed scalar extrapolation for Markov and semi-Markov chains. A relevant survey is that of Porteus [22]. All of the extrapolations reduce to scalar extrapolations in the discounted Markov decision problem considered here. The implementation of these scalar extrapolation schemes in the modified policy iteration algorithms is studied in Section 3 and it is shown that the policy selected using modified policy iteration with and without scalar extrapolations is identical and the value obtained by the algorithm using only a final scalar extrapolation will give exactly the same result as that obtained by using scalar extrapolations at each iteration in the evaluation phase and then applying a final scalar extrapolation. However, when there are unequal row sums, such as in the case of semi-Markov decision problems, scalar extrapolations may have an important influence on the results.

The idea of using bounds on the optimal return function to identify actions that are not part of an optimal policy and reduce computations in the improvement phase was introduced by MacQueen [16]. At each iteration of a value iteration algorithm such actions are identified and discarded from the action set for <u>all</u> subsequent iterations. Porteus [19] and [20] strengthened MacQueen's bounds and extended them to more general decision processes. Later, Hastings and Mello [10] and Porteus [20] considered a modification of MacQueen's and Porteus' [19] procedures that reduces storage requirements. Grinold [9] used bounds similar to those of Porteus [20] to develop an action elimination procedure for policy iteration. Much of this material is surveyed in an article by White [35].

In the above papers, actions are eliminated only if they are nonoptimal. Hastings [11] proposed a procedure to eliminate an action for one or more iterations after which, it could reenter the decision set. His work was concerned with the average reward criteria and was extended by Hübner [15] to discounted problems with arbitrary discount factors. Hastings and van Nunen [12] applied Hübner's results to the case where the discount factor was less than one and through an example showed that this procedure eliminated more actions than that of Hastings and Mello [10].

In all of these papers, with the exception of that of Grinold [9], elimination procedures were developed for value iteration algorithms and variants. Motivated by the discovery in [26] that modified policy iteration was considerably more efficient than value iteration and in many cases policy iteration, we introduce bounds and elimination procedures for the modified policy iteration and policy iteration algorithms. The bounds are motivated by the Newton method representation for modified policy iteration in [26]. However, as pointed out by Porteus in a private communication, they can be derived from results in [20].

In Section 4 a method for eliminating non-optimal actions for all subsequent iterations using upper bounds and lower bounds on the optimal expected total discounted return at each iteration of modified policy iteration and policy iteration algorithms is presented. In Section 5 procedures for eliminating actions that cannot be part of the policy chosen in the improvement phase of the next iteration are developed. The relationship with the action elimination procedure of Hastings and van Hunen [12] and Hübner [15] is discussed.

Section 6 presents modified policy iteration algorithms which incorporate the above action elimination procedures. The storage requirements of these algorithms are discussed and some methods to reduce them are proposed. Numerical results appear in Section 7 where a comparison of these procedures on Howard's [14] automobile replacement problem and on a sparse randomly generated test problem with a large action set are presented.

2. Preliminaries.

We consider a discrete time finite state and action discounted Markov decision problem over an infinite horizon. Let I be the set of finite states labeled by integers $i=1,2,\cdots,N$. For each $i\varepsilon I$, we have a set K_i of finite actions denoted by $k=1,2,\cdots,M_i$. The policy space is defined by the Cartesian product of each action set, that is, $\Delta = K_1 \times K_2 \times \cdots \times K_N$. By a policy $\delta\varepsilon\Delta$, we mean $\delta(i) \varepsilon K_i$. Then a strategy π is defined by a sequence $\{\delta^t \varepsilon \Delta, t=1,2,\cdots\} = \{\delta^1,\delta^2,\cdots,\delta^t,\cdots\}$ and is an element of strategy space denoted by $\Pi = \Delta \times \Delta \times \cdots$. A strategy π is stationary if $\pi = (\delta,\delta,\cdots)$.

Let us consider a sequential decision problem where at each stage $t = 0,1,2,\cdots$ we observe the state of the system and then choose an action. More precisely, if the system is in state $i\varepsilon I$ at stage t and an action $k\varepsilon K_i$ is selected, the expected one period reward is $r(i,k)$ and the system moves to state $j\varepsilon I$ at stage t+1 with probability $P(i,j,k)$. We assume that

$$\sum_{j\varepsilon I} P(i,j,k) = 1 \qquad \text{for } i\varepsilon I \text{ and } k\varepsilon K_i$$

and

$$\max_{i\varepsilon I} \max_{k\varepsilon K_i} |r(i,k)| \leq M < +\infty.$$

Denote the one period discount rate by $\lambda \varepsilon [0,1)$. For $\delta\varepsilon\Delta$ define P_δ by $P_\delta(i,j) = P(i,j,\delta(i))$ and r^δ by $r^\delta(i) = r(i,\delta(i))$. Let V denote the Banach space of bounded real valued functions on I with norm $||v|| = \max_{i\varepsilon I} |v(i)|$.

For each $\pi\varepsilon\Pi$, let v^π be the expected total discounted reward vector. For each stationary strategy $\pi = (\delta,\delta,\cdots)$, let v^δ be the expected total discounted reward vector. Then the problem we study is that of finding an optimal policy $\delta^* \varepsilon \Delta$ and a $v^* \varepsilon V$ such that

(1) $$v^* = v^{\delta^*} = \underset{\delta \varepsilon \Delta}{\mathrm{Max}}\ v^{\delta}.$$

That there exists an optimal stationary strategy $\pi^* = (\delta^*, \delta^*, \cdots)$, i.e., there exists an optimal policy δ^*, in the discounted Markov decision problem, is shown in Blackwell [2,3] and Denardo [6].

For a given $v \varepsilon V$ and $\delta \varepsilon \Delta$ define the linear operators B, T, B_δ and T_δ mapping V to V by

(2) $$Bv(i,\delta(i)) \equiv r(i,\delta(i)) + \lambda \sum_{j \varepsilon I} P(i,j,\delta(i))\ v(j) - v(i)$$

(3) $$Tv(i,\delta(i)) \equiv r(i,\delta(i)) + \lambda \sum_{j \varepsilon I} P(i,j,\delta(i)) v(j)$$

(4) $$B_\delta v(i) \equiv Bv(i,\delta(i))$$

(5) $$T_\delta v(i) \equiv Tv(i,\delta(i))$$

and define the policy δ_v and the operator $H: V \to V$ by

(6) $$Hv \equiv B_{\delta_v} v = \underset{\delta \varepsilon \Delta}{\mathrm{Max}}\ B_\delta v$$

where δ_v is the policy that maximizes the right hand side of (4) for each i at v. This maximum is attained at each $v \varepsilon V$ since Δ is finite.

The problem defined by (1) is equivalent to finding a $v^* \in V$ and a $\delta^* \in \Delta$ that satisfy

$$(7) \qquad B_{\delta^*} v^* = Hv^* = 0.$$

This is the usual functional equation of dynamic programming written in the form of Puterman and Brumelle [25].

For clarity we will use δ_n, r_n, P_n, B_n and T_n instead of δ_{v_n}, $r^{\delta_{v_n}}$, $P_{\delta_{v_n}}$, $B_{\delta_{v_n}}$ and $T_{\delta_{v_n}}$. The symbol $\tilde{1}$ will denote a column vector of ones and for $u \in V$ we adopt the notation

$$sp[u] \equiv \max_{i \in I} u(i) - \min_{i \in I} u(i).$$

Our analysis will be based on the following modified policy iteration algorithm.

I. Initialization

Select $\varepsilon > 0$, to be used as a stopping criterion. Set $n = 0$ and define $\delta_0 \in \Delta$ by

$$r_0(i) = r(i, \delta_0(i)) = \mathrm{Max}_{k \in K_i}\ r(i,k), \qquad i \in I.$$

Set

$$(8) \qquad v_0 = \frac{1}{1-\lambda} \left(\min_{i \in I} r_0(i)\right) \cdot \tilde{1}.$$

II. Evaluation Phase:

Calculate $v_{n+1} = T_n^{m+1} v_n$ iteratively by successive approximations using T_n. (The order of the algorithm, m, can be predetermined or else chosen to be

$$\text{(9)} \qquad \operatorname{Min}\{s : sp[T_n^{s+1} v_n - T_n^s v_n] \leq \varepsilon\}.)$$

Increment n by one.

III. Improvement Phase:

Find δ_n such that

$$B_n v_n = B_{\delta_n} v_n = \operatorname{Max}_{\delta \varepsilon \Delta} B_\delta v_n = H v_n .$$

If $sp[B_n v_n] \leq \varepsilon$, go to IV. Otherwise return to II.

IV. Final Extrapolation:

Set $\hat{v} = v_n + B_n v_n + \frac{\lambda}{1-\lambda} \underline{B_n v_n} \cdot \tilde{1}$ where $\underline{B_n v_n} = \operatorname{Min}_{i \varepsilon I} B_n v_n(i)$.

Some comments about this algorithm are in order. The selection of v_0 in the initialization phase is to ensure that $Hv_n \geq 0$ for all n. As demonstrated by Puterman and Shin [26, Thm. 1] this ensures monotone convergence of the sequence of iterates $\{v_n\}$. In practice the order of the algorithm, m, will usually be determined by (9). However this criteria might be too stringent and require excessive calculations. If the order is prespecified, though not necessarily fixed from iteration to iteration, this algorithm is similar to that presented in [26]. In that setting we observed that when $m = +\infty$, the algorithm was policy iteration and that $m = 0$ gave value iteration.

As a consequence of the stopping criteria, and the extrapolation in IV the sequence $\{v_n\}$ will terminate with a value function $\hat{v}$ that is $\varepsilon\lambda(1-\lambda)^{-1}$ optimal. This is shown in Proposition 2.4.

The following propositions summarize some results that will be needed in the sequel. The proof of the first appears in Puterman and Shin [26] and the proof of the second in Puterman and Brumelle [25].

Proposition 2.1.

Suppose $v_{n+1} = T_n^{m+1} v_n$. Then

$$v_{n+1} = v_n + \sum_{s=0}^{m} (\lambda P_n)^s B_n v_n = v_n + \sum_{s=0}^{m} (\lambda P_n)^s H v_n \ . \tag{10}$$

Proposition 2.2. (Support Inequality)

For any $u, v \varepsilon V$

$$Hu \geq Hv + (\lambda P_{\delta_v} - I)(u-v). \tag{11}$$

3. Scalar Extrapolations.

To reduce the computational effort to evaluate the expected total discounted return in Markov and semi-Markov chains Porteus and Totten [23] and Porteus [21] developed the extrapolation methods called trivial lower bound extrapolation, trivial scalar extrapolation, row sum extrapolation and delayed scalar extrapolation. In the Markov chains arising from discounted Markov decision problems all of the above extrapolation methods reduce to scalar extrapolation methods. For value iteration Porteus [20] mentioned ". . .When there are equal row sums, such an extrapolation would have an entirely predictable influence; that is, the results without such extrapolation would differ by a constant (pointwise) and the same policies would be 'optimal' at each iteration. . . ." and suggested using the extrapolation at each iteration. But in this section we show that the algorithm in Section 2 will give the same result as that using scalar extrapolations at each iteration and then applying the final scalar extrapolations for modified policy iteration. For semi-Markov chains or unequal row sums Markov chains these results do not hold.

In this section we discuss the incorporation of these scalar extrapolation schemes into the modified policy iteration procedure. Suppose in the evaluation phase of the algorithm in Section 2 we calculate v_{n+1} iteratively using one of these extrapolation schemes with v_n as the initial value. Then the resulting iterative extrapolation method can be represented by:

(12) $$\tilde{v}_n^s = T_n v_n^{s-1}$$

(13) $$v_n^s = \tilde{v}_n^s + c_n^s$$

where

$$v_n^o = v_n$$

(14) $$c_n^s = \frac{\lambda}{(1-\lambda)} \underline{B_n v_n^{s-1}} \ \tilde{1}$$ for the trivial lower bound extrapolation [TLBE],

(15) $$c_n^s = \frac{\lambda}{2(1-\lambda)} [\overline{B_n v_n^{s-1}} + \underline{B_n v_n^{s-1}}] \ \tilde{1}$$ for the trivial scalar extrapolation and other extrapolations [TSE],

$$\underline{B_n v_n^{\ell}} = \underset{i \in I}{\text{Min}} \ B_n v_n^{\ell} (i) ,$$

and

$$\overline{B_n v_n^{\ell}} = \underset{i \in I}{\text{Max}} \ B_n v_n^{\ell} (i) .$$

Note that the quantity c_n^s defined in (14) and (15) is a scalar times a column vector of ones. Hence these are called scalar extrapolations.

The following lemma gives the value of v_n^s and $B_n v_n^s$ after the sth iteration of these iterative extrapolation methods.

Lemma 2.1.

At iteration s of the above iterative extrapolation methods

(16) $$v_n^s = T_n^s v_n + \sum_{\ell=1}^{s} \lambda^{s-\ell} c_n^{\ell}$$

(17) $$B_n v_n^s = (\lambda P_n)^s B_n v_n + (\lambda - 1) [\sum_{\ell=1}^{s} \lambda^{s-\ell} c_n^{\ell}].$$

Proof. For s = 1, from (12) and (13)

$$v_n^1 = T_n v_n + c_n^1$$

Assuming (16) is true for s = p, from (3), (12) and (13)

$$v_n^{p+1} = T_n v_n^p + c_n^{p+1}$$

$$= T_n(T_n^p v_n + \sum_{\ell=1}^{p} \lambda^{p-\ell} c_n^\ell) + c_n^{p+1}$$

$$= T_n^{p+1} v_n + \lambda \sum_{\ell=1}^{p} \lambda^{p-\ell} c_n^\ell + c_n^{p+1}$$

$$= T_n^{p+1} v_n + \sum_{\ell=1}^{p+1} \lambda^{p+1-\ell} c_n^\ell .$$

Therefore (16) is true for any s.

From (10) and (16)

$$B_n v_n^s = B_n[T_n^s v_n + \sum_{\ell=1}^{s} \lambda^{s-\ell} c_n^\ell]$$

$$= B_n[v_n + \sum_{\ell=0}^{s-1} (\lambda P_n)^\ell B_n v_n + \sum_{\ell=1}^{s} \lambda^{s-\ell} c_n^\ell]$$

$$= r_n + \lambda P_n[v_n + \sum_{\ell=0}^{s-1} (\lambda P_n)^\ell B_n v_n + \sum_{\ell=1}^{s} \lambda^{s-\ell} c_n^\ell]$$

$$- [v_n + \sum_{\ell=0}^{s-1} (\lambda P_n)^\ell B_n v_n + \sum_{\ell=1}^{s} \lambda^{s-\ell} c_n]$$

$$= r_n + \lambda P_n v_n - v_n + \sum_{\ell=1}^{s} (\lambda P_n)^\ell B_n v_n - \sum_{\ell=0}^{s-1} (\lambda P_n)^\ell B_n v_n$$

$$+ (\lambda-1) \sum_{\ell=1}^{s} \lambda^{s-\ell} c_n^\ell$$

$$= B_n v_n + \sum_{\ell=1}^{s} (\lambda P_n)^\ell B_n v_n - \sum_{\ell=0}^{s-1} (\lambda P_n)^\ell B_n v_n + (\lambda-1) \sum_{\ell=1}^{s} \lambda^{s-\ell} c_n^\ell$$

$$= (\lambda P_n)^s B_n v_n + (\lambda-1) \sum_{\ell=1}^{s} \lambda^{s-\ell} c_n^\ell .$$

Define

$$\underline{(\lambda P_n)^s B_n v_n} = \operatorname*{Min}_{i \in I} (\lambda P_n)^s B_n v_n(i)$$

$$\overline{(\lambda P_n)^s B_n v_n} = \operatorname*{Max}_{i \in I} (\lambda P_n)^s B_n v_n(i) .$$

From Lemma 2.1 we obtain the following result which exhibits the relationship between $T_n^{s+1} v_n$ and v_n^{s+1}.

Lemma 2.2.

At iteration s of the iterative extrapolation method

(18) $$v_n^{s+1} = T_n^{s+1} v_n + c_{n+1}$$

where

$$c_{n+1} = \frac{\lambda}{(1-\lambda)} \underline{(\lambda P_n)^s B_n v_n} \; \tilde{1} \text{ for TLBE}$$

$$c_{n+1} = \frac{\lambda}{2(1-\lambda)} [\overline{(\lambda P_n)^s B_n v_n} + \underline{(\lambda P_n)^s B_n v_n}] \; \tilde{1} \text{ for TSE.}$$

Proof. From (16)

$$v_n^{s+1} = T_n^{s+1} v_n + \sum_{\ell=1}^{s+1} \lambda^{s+1-\ell} c_n^{\ell}$$

(19) $$= T_n^{s+1} v_n + \lambda \sum_{\ell=1}^{s} \lambda^{s-\ell} c_n^{\ell} + c_n^{s+1}$$

First we consider TLBE. From (14) and (17)

$$c_n^{s+1} = \frac{\lambda}{(1-\lambda)} \underline{Bv_n^s} \; \tilde{1}$$

$$= \frac{\lambda}{(1-\lambda)} [\underline{(\lambda P_n)^s B_n v_n} \tilde{1} + (\lambda-1) \sum_{\ell=1}^{s} \lambda^{s-\ell} c_n^\ell]$$

$$= \frac{\lambda}{(1-\lambda)} \underline{(\lambda P_n)^s B_n v_n} \tilde{1} - \lambda \sum_{\ell=1}^{s} \lambda^{s-\ell} c_n^\ell$$

(20) $$= c_{n+1} - \lambda \sum_{\ell=1}^{s} \lambda^{s-\ell} c_n^\ell .$$

For TSE it follows from (15) and (17) that

$$c_n^{s+1} = \frac{\lambda}{2(1-\lambda)} [\overline{B_n v_n^s} + \underline{B_n v_n^s}] \tilde{1}$$

$$= \frac{\lambda}{2(1-\lambda)} [\overline{(\lambda P_n)^s B_n v_n} \tilde{1} + (\lambda-1) (\sum_{\ell=1}^{s} \lambda^{s-\ell} c_n^\ell)$$

$$+ \underline{(\lambda P_n)^s B_n v_n} \tilde{1} + (\lambda-1) (\sum_{\ell=1}^{s} \lambda^{s-\ell} c_n^\ell)$$

$$= \frac{\lambda}{2(1-\lambda)} [\overline{(\lambda P_n)^s B_n v_n} + \underline{(\lambda P_n)^s B_n v_n}] \tilde{1} - \lambda \sum_{\ell=1}^{s} \lambda^{s-\ell} c_n^\ell$$

(21) $$= c_{n+1} - \lambda \sum_{\ell=1}^{s} \lambda^{s-\ell} c_n^\ell .$$

Substituting (20) or (21) into (19)

$$v_n^{s+1} = T_n^{s+1} v_n + \lambda \sum_{\ell=1}^{s} \lambda^{s-\ell} c_n^\ell + c_{n+1} - \lambda \sum_{\ell=1}^{s} \lambda^{s-\ell} c_n^\ell$$

$$= T_n^{s+1} v_n + c_{n+1} .$$

We show in the following theorem that for the stopping critierion (9) in the evaluation phase we will have the same order of the algorithm, m_n, at iteration n whether or not we use any of the iterative extrapolation methods.

Theorem 2.1.

At iteration s in the evaluation phase

$$\text{(22)} \qquad sp(v_n^{s+1} - v_n^s) = sp(T_n^{s+1} v_n - T_n^s v_n) \ .$$

Proof. From (16)

$$sp(v_n^{s+1} - v_n^s) = sp(T_n^{s+1} v_n + \sum_{\ell=1}^{s+1} \lambda^{s+1-\ell} c_n^\ell - T_n^s v_n - \sum_{\ell=1}^{s} \lambda^{s-\ell} c_n^\ell)$$

$$= sp(T_n^{s+1} v_n - T_n^s v_n)$$

since adding a constant to each of the vectors does not change sp[u].

Lemma 2.2 and Theorem 2.1 imply that using these scalar extrapolation schemes at each iteration in the evaluation phase of the algorithm will give the same result as using these scalar extrapolations only at the end of the iteration in the evaluation phase. This result appears in Theorem 2.2 below.

Suppose we use extrapolations only at the end of iterating T_n in the evaluation phase. Then we exit the evaluation phase of iteration n+1 with value v'_{n+1} defined by

(23) $$\tilde{v}'_{n+1} = T_n^{m+1} \quad v'_n$$

(24) $$v'_{n+1} = \tilde{v}'_{n+1} + c'_{n+1}$$

and

$$v'_o = v_o,$$

where

(25) $$c'_{n+1} = \frac{\lambda}{(1-\lambda)} \underline{(\lambda P_n)^m B_n v'_n} \; \tilde{1} \text{ for TLBE },$$

(26) $$c'_{n+1} = \frac{\lambda}{2(1-\lambda)} [\overline{(\lambda P_n)^m B_n v'_n} + \underline{(\lambda P_n)^m B_n v'_n}] \; \tilde{1} \text{ for TSE}$$

and m is the order of algorithm in the evaluation phase at iteration n of the algorithm. For clarity m is used instead of m_n.

Let v_n be the value obtained in the evaluation step without the above extrapolation scheme. Then the following theorem gives the relationship between v_{n+1} and v'_{n+1}.

<u>Theorem 2.2</u>

At iteration n of the algorithm

(27) $$v'_{n+1} = v_{n+1} + c_{n+1}$$

where

$$c_{n+1} = \frac{\lambda}{(1-\lambda)} \underline{(\lambda P_n)^m B_n v_n} \; \tilde{1} \quad \text{for TLBE}$$

and

$$c_{n+1} = \frac{\lambda}{2(1-\lambda)} [\overline{(\lambda P_n{}^m B_n v_n} + \underline{(\lambda P_n)^m B_n v_n}] \; \tilde{1} \quad \text{for TSE} .$$

<u>Proof.</u> Note $v'_o = v_o$.

For TLBE, at n = 0 from (18) with s = m

$$v_1' = v_1 + c_1$$

Suppose (27) is true at iteration (n-1), i.e.,

$$v_n' = v_n + c_n. \tag{28}$$

Then at iteration n+1 from (10), (23) and (24)

$$v_{n+1}' = T_n^{m+1} v_n' + c_{n+1}'$$

$$= v_n' + \sum_{\ell=0}^{m} (\lambda P_n)^{\ell} B_n v_n' + c_{n+1}' . \tag{29}$$

From (28)

$$B_n v_n' = B_n(v_n + c_n)$$

$$= r_n + \lambda P_n v_n + \lambda P_n c_n - v_n - c_n$$

$$= r_n + \lambda P_n v_n - v_n + \lambda c_n - c_n$$

$$= B_n v_n + (\lambda-1)\, c_n . \tag{30}$$

From (30)

$$c_{n+1}' = \frac{\lambda}{(1-\lambda)} \underline{(\lambda P_n)^m B_n v_n'}$$

$$= \frac{\lambda}{(1-\lambda)} [\underline{(\lambda P_n)^m B_n v_n}\ \tilde{1} + \lambda^m(\lambda-1)c_n]$$

$$= c_{n+1} - \lambda^{m+1} c_n \tag{31}$$

Substituting (10), (28), (30) and (31) in (29) we obtain

$$v'_{n+1} = v_n + c_n + \sum_{\ell=0}^{m} (\lambda P_n)^{\ell}[B_n v_n + (\lambda-1)c_n] + c_{n+1} - \lambda^{m+1} c_n$$

$$= v_n + \sum_{\ell=0}^{m} (\lambda P_n)^{\ell} B_n v_n + c_{n+1} + c_n + \lambda\sum_{\ell=0}^{m} (\lambda)^{\ell} c_n - \sum_{\ell=0}^{m} (\lambda)^{\ell} c_n$$

$$- \lambda^{m+1} c_n$$

$$= v_n + \sum_{\ell=0}^{m} (\lambda P_n)^{\ell} B_n v_n + c_{n+1}$$

$$= v_{n+1} + c_{n+1} \; .$$

For TSE all of the results are the same as for TLBE except that

$$c'_{n+1} = \frac{\lambda}{2(1-\lambda)} [\overline{(\lambda P_n)^m B_n v'_n} + \underline{(\lambda P_n)^m B_n v'_n}] \tilde{1}$$

$$= \frac{\lambda}{2(1-\lambda)} [\overline{(\lambda P_n)^m B_n v_n} \tilde{1} + \lambda^m(\lambda-1)c_n + \underline{(\lambda P_n)^m B_n v_n} \tilde{1}$$

$$+ \lambda^m(\lambda-1)c_n]$$

$$= \frac{\lambda}{2(1-\lambda)} [\overline{(\lambda P_n)^m B_n v_n} + \underline{(\lambda P_n)^m B_n v_n}] \tilde{1} + \lambda^{m+1} c_n$$

$$= c_{n+1} + \lambda^{m+1} c_n \; . \tag{32}$$

Therefore using (32) instead of (31) the result follows.

The following theorem shows that at each iteration we will select the same policy and have the same test value, $sp[B_n v_n]$, for the stopping criterion in the improvement phase of the algorithms whether we use scalar extrapolations or not.

Theorem 2.3.

At iteration n of the algorithm

(33) $$B_\delta v_n' = B_\delta v_n + (1-\lambda)c_n$$

(34) $$\{\delta_n \varepsilon \Delta | B_{\delta_n} v_n' = \operatorname*{Max}_{\delta\varepsilon\Delta} B_\delta v_n'\} = \{\delta_n \varepsilon\Delta | B_{\delta_n} v_n = \operatorname{Max} B_\delta v_n\}$$

(35) $$sp[B_n v_n'] = sp[B_n v_n] \ .$$

Proof. From (27)

$$B_\delta v_n' = B_\delta(v_n + c_n)$$

$$= r^\delta + \lambda P_\delta(v_n + c_n) - (v_n + c_n)$$

$$= r^\delta + \lambda P_\delta v_n - v_n + (\lambda-1)c_n$$

$$= B_\delta v_n + (\lambda-1)c_n \ .$$

(34) and (35) follow from (33).

Corollary 2.2.

In the final extrapolation step of the algorithm

$$\hat{v} = v_n' + B_n v_n' + \frac{\lambda}{(1-\lambda)} \underline{B_n v_n'} \, \tilde{1}$$

$$= v_n + B_n v_n + \frac{\lambda}{(1-\lambda)} \underline{B_n v_n} \, 1 \quad \text{for TLBE}$$

and

$$\hat{v} = v_n' + B_n v_n' + \frac{\lambda}{2(1-\lambda)} [\overline{B_n v_n'} + \underline{B_n v_n'}] \, \tilde{1}$$

$$= v_n + B_n v_n + \frac{\lambda}{2(1-\lambda)} [\overline{B_n v_n} + \underline{B_n v_n}] \, \tilde{1} \quad \text{for TSE.}$$

<u>Proof</u>. For TLBE from (27)

$$\hat{v} = v_n' + B_n v_n' + \frac{\lambda}{(1-\lambda)} \underline{B_n v_n'} \, \tilde{1}$$

$$= v_n + c_n + B_n(v_n + c_n) + \frac{\lambda}{(1-\lambda)} \underline{B_n(v_n + c_n)} \, \tilde{1}$$

$$= v_n + c_n + B_n v_n + (\lambda-1)c_n + \frac{\lambda}{(1-\lambda)} \underline{B_n v_n} \, \tilde{1} - \lambda c_n$$

$$= v_n + B_n v_n + \frac{\lambda}{(1-\lambda)} \underline{B_n v_n} \, \tilde{1} \, .$$

For TSE from (27)

$$\hat{v} = v_n' + B_n v_n' + \frac{\lambda}{2(1-\lambda)} [\underline{B_n v_n'} + \underline{B_n v_n'}] \, \tilde{1}$$

$$= v_n + c_n + B_n(v_n + c_n) + \frac{\lambda}{2(1-\lambda)} [\overline{B_n(v_n + c_n)} + \underline{B_n(v_n + c_n)}] \, \tilde{1}$$

$$= v_n + c_n + B_n v_n + (\lambda-1)c_n + \frac{\lambda}{2(1-\lambda)} [\overline{B_n v_n}\,\tilde{1} + (\lambda-1)c_n$$

$$+ \underline{B_n v_n}\,\tilde{1} + (\lambda-1)c_n]$$

$$= v_n + B_n v_n + \frac{\lambda}{2(1-\lambda)} [\overline{B_n v_n} + \underline{B_n v_n}]\,\tilde{1} + c_n + (\lambda-1)c_n - \lambda c_n$$

$$= v_n + B_n v_n + \frac{\lambda}{2(1-\lambda)} [\overline{B_n v_n} + \underline{B_n v_n}]\,\tilde{1} .$$

Hence Corollary 2.2 implies that the value obtained by the algorithm using only the final scalar extrapolations will give exactly the same result as that obtained by using scalar extrapolations at each iteration in the evaluation phase and then applying the final scalar extrapolation.

4. Elimination of Non-optimal Actions.

We will use the following notation in this section

$$Bv(i,k) = r(i,k) + \lambda \sum_{j \in I} P(i,j,k)v(j) - v(i) .$$

Throughout this section except where noted we will assume that $\{v_n\}$ is generated by the modified policy iteration algorithm described in Section 2.

Proposition 2.3.

For each n

(36) $$v_n + \left(\frac{1}{1-\lambda}\,\underline{B_n v_n}\right)\tilde{1} \le v_n + B_n v_n + \left(\frac{\lambda}{1-\lambda}\,\underline{B_n v_n}\right)\tilde{1} \le v_{n+1} + \left(\frac{\lambda^{m+1}}{1-\lambda}\,\underline{B_n v_n}\right)\tilde{1}$$

$$\le v^* \le v_n + B_n v_n + \left(\frac{\lambda}{1-\lambda}\,\overline{B_n v_n}\right)\tilde{1}$$

$$\le v_n + \left(\frac{1}{1-\lambda}\,\overline{B_n v_n}\right)\tilde{1} .$$

where $v_{n+1} = T_n^{m+1} v_n$.

Proof. Applying the support inequality (11) at v^* gives

$$B_n v_n \ge Hv^* + (\lambda P_{\delta_{v^*}} - I)(v_n - v^*)$$

Rearranging terms and noting $Hv^* = 0$ implies that

$$v^* \le v_n + (I-\lambda P_{v^*})^{-1} B_n v_n \le v_n + B_n v_n + \left(\frac{\lambda}{1-\lambda}\overline{B_n v_n}\right)\tilde{1}$$

$$\le v_n + \left(\frac{1}{1-\lambda}\,\overline{B_n v_n}\right)\tilde{1} .$$

This establishes the upper bounds for v^*.

To derive the lower bounds, use the support inequality at v_n to obtain

$$Hv^* \ge B_n v_n + (\lambda P_n - I)[v^* - v_n].$$

Rearranging terms we find that

$$v^* \geq v_n + (I-\lambda P_n)^{-1} B_n v_n + \sum_{s=0}^{\infty} (\lambda P_n)^s B_n v_n$$

$$= v_n + \sum_{s=0}^{m} (\lambda P_n)^s B_n v_n + \sum_{s=m+1}^{\infty} (\lambda P_n)^s B_n v_n$$

$$\geq v_{n+1} + \left(\frac{\lambda^{m+1}}{1-\lambda} \underline{B_n v_n}\right)\tilde{1} .$$

The last inequality follows from the definition of v_{n+1}. Next observe that from (10) it follows that

$$v_{n+1} + \left(\frac{\lambda^{m+1}}{1-\lambda} \underline{B_n v_n}\right)\tilde{1} = v_n + B_n v_n + \sum_{s=1}^{m} (\lambda P_n)^s B_n v_n + \left(\frac{\lambda^{m+1}}{1-\lambda} \underline{B_n v_n}\right)\tilde{1}$$

$$\geq v_n + B_n v_n + \left(\frac{\lambda}{1-\lambda} \underline{B_n v_n}\right)\tilde{1} \geq v_n + \left(\frac{1}{1-\lambda} \underline{B_n v_n}\right)\tilde{1}$$

which establishes the result.

When $\{v_n\}$ is generated by policy iteration, in which case $m = +\infty$, then the first lower bound for v^* in (36) becomes $v_{n+1} \leq v^*$.

The extreme lower bound and upper bound for v^* in (36) were established by MacQueen [16] for $\{v_n\}$ generated by value iteration. For policy iteration, Grinold [9] established the second lower bound for v^*. Our results for modified policy iteration are obvious extensions of these earlier results; however, the tightest lower bound for v^* is new.

As a consequence of the stopping criteria and the final extrapolation of the algorithm in Section 2 the sequence $\{v_n\}$ will terminate with a value function $\hat{v}$ that is $\varepsilon\lambda(1-\lambda)^{-1}$ optimal. This is shown in the following proposition.

Proposition 2.4.

$\hat{v}$ is $\varepsilon\lambda(1-\lambda)^{-1}$ optimal, i.e.,

$$\hat{v}(i) \le v^*(i) \le \hat{v}(i) + \varepsilon\lambda(1-\lambda)^{-1} \text{ for all } i \in I.$$

Proof. From (36)

$$\hat{v}(i) = v_n(i) + B_n v_n(i) + \frac{\lambda}{(1-\lambda)} \underline{B_n v_n} \le v^*(i)$$

and

$$v^*(i) \le v_n(i) + B_n v_n(i) + \frac{\lambda}{(1-\lambda)} \overline{B_n v_n}$$

$$= v_n(i) + B_n v_n(i) + \frac{\lambda}{(1-\lambda)} \underline{B_n v_n} + \frac{\lambda}{(1-\lambda)} (\overline{B_n v_n} - \underline{B_n v_n})$$

$$= \hat{v}(i) + \varepsilon\lambda(1-\lambda)^{-1} .$$

Action elimination is based on the following result of MacQueen [16]. The short proof is included for completeness.

Proposition 2.5.

If $Bv^*(i,k) < 0$ then k is a non-optimal action in state i.

Proof. Suppose k^* is an optimal action in state i. Then $Bv^*(i,k^*) = 0 > Bv^*(i,k)$. Therefore k is non-optimal.

Combining these two propositions we obtain an action elimination procedure for modified policy iteration.

Theorem 2.4.

Suppose at iteration n that

$$r(i,k) + \lambda \sum_{j\varepsilon I} P(i,j,k)v_n(j) + (\frac{\lambda}{1-\lambda} \overline{B_n v_n}) < v_{n+1}(i) + (\frac{\lambda^{m+1}}{1-\lambda} \underline{B_n v_n}) . \tag{37}$$

Then action $k \varepsilon K_i$ is non-optimal in state i.

Proof. From (36) and (37)

$$\begin{aligned} r(i,k) + \lambda \sum_{j\varepsilon I} P(i,j,k)v^*(j) &\le r(i,k) + \lambda \sum_{j\varepsilon I} P(i,j,k)[v_n(j) \\ &\quad + (\frac{1}{1-\lambda} \overline{B_n v_n})\tilde{1}] \\ &= r(i,k) + \lambda \sum_{j\varepsilon I} P(i,j,k)v_n(j) \\ &\quad + (\frac{\lambda}{1-\lambda} \overline{B_n v_n}) \\ &< v_{n+1}(i) + (\frac{\lambda^{m+1}}{1-\lambda} \underline{B_n v_n}) \le v^*(i). \end{aligned}$$

Hence $Bv^*(i,k) < 0$ and by Proposition 2.5 it follows that $k \varepsilon K_i$ is non-optimal.

In the case of policy iteration

Corollary 2.3.

Suppose $\{v_n\}$ is generated by policy iteration and that

$$r(i,k) + \lambda \sum_{j\varepsilon I} P(i,j,k)v_n(j) + (\frac{\lambda}{1-\lambda} \overline{B_n v_n}) < v_{n+1}(i) . \tag{38}$$

Then $k \varepsilon K_i$ is non-optimal in state i.

Grinold [7] replaced the term v_{n+1} on the righthand side of (38) by either $v_n(i) + (\frac{1}{1-\lambda} \underline{B_n v_n})$ or $v_n(i) + B_n v_n(i) + (\frac{\lambda}{1-\lambda} \underline{B_n v_n})$ in his action elimination algorithms. Since v_{n+1} is a tighter lower bound for v^* than either of these two bounds and is calculated prior to an action elimination step (see Section 6), it is surprising that he did not use the stronger test, (38), for action elimination.

5. Elimination of Actions for One Iteration.

The purpose of this section is to develop procedures to eliminate actions for a single iteration of a modified policy iteration algorithm. We use the following notation.

$$Dv_{n,\ell} = v_{n+\ell} - v_n \qquad\qquad \ell \geq 1$$

$$\overline{Dv}_{n,\ell} = \max_{i \varepsilon I} Dv_{n,\ell}(i) .$$

The following lemma is of a technical nature and important in our development.

Lemma 2.3.

At iteration n

(39) $$B_n v_{n+1} = (\lambda P_n)^{m+1} B_n v_n$$

where

$$v_{n+1} = T_n^{m+1} v_n .$$

Proof. From the definition of $B_n v$ and (10)

$$B_n v_{n+1} = r_n + (\lambda P_n - I) v_{n+1}$$

$$= r_n + (\lambda P_n - I)(v_n + \sum_{s=0}^{m} (\lambda P_n)^s B_n v_n)$$

$$= B_n v_n + (\lambda P_n)^{m+1} B_n v_n - B_n v_n = (\lambda P_n)^{m+1} B_n v_n .$$

The following proposition is the basis for a single iteration elimination procedure.

Proposition 2.6.

Suppose

$$(40) \qquad Bv_{n+1}(i,k) < [(\lambda P_n)^{m+1} B_n v_n](i) .$$

Then $k \in K_i$ cannot attain the maximum in state i in the improvement phase of iteration n+1.

Proof. From (39) and (40) we obtain

$$Bv_{n+1}(i,k) < [(\lambda P_n)^{m+1} B_n v_n](i) = B_n v_{n+1}(i) \leq Hv_{n+1}(i) .$$

From which we conclude the result.

Figure 2.1 shows schematically the implication of Proposition 2.6. If $Bv_{n+1}(i,k) < B_n v_{n+1}(i) = [(\lambda P_n)^{m+1} B_n v_n](i)$ then $Bv_{n+1}(i,k) <$

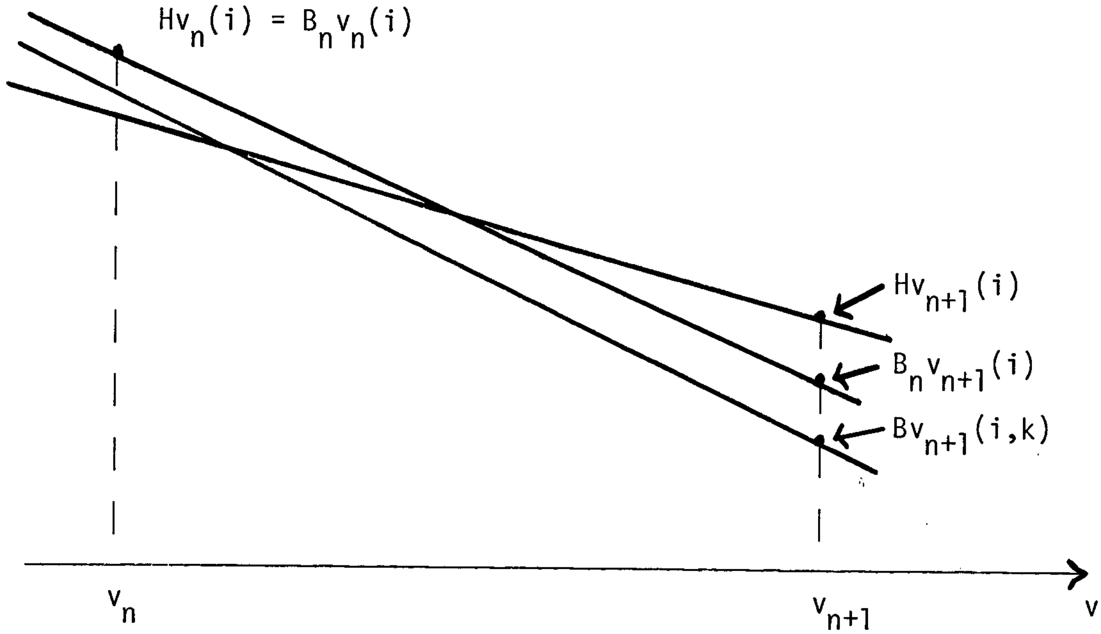

Figure 2.1. Implication of Proposition 2.6.

$B_n v_{n+1}(i) \leq Hv_{n+1}(i)$ as shown in Figure 2.1. Therefore $k \in K_i$ can be eliminated at iteration n+1.

Noting that for policy iteration, $m = +\infty$, we conclude the following:

<u>Corollary 2.4</u>.

Suppose $\{v_n\}$ is generated by policy iteration and

$$Bv_{n+1}(i,k) < 0.$$

Then action $k \in K_i$ cannot be optimal in state i at iteration n+1.

To make use of Proposition 2.4 in action elimination we must obtain bounds for both expressions in (40) that are easily computable from the quantities available at the completion of an evaluation step. These bounds are obtained in the course of proving the following theorem which gives an action elimination procedure.

<u>Theorem 2.5</u>.

Suppose at iteration (n+1) that for some $\ell \leq n$

(41) $$r(i,k) + \lambda \sum_{j \in I} P(i,j,k) v_\ell(j) + \sum_{p=\ell}^{n} \overline{Dv}_{p,1} - v_{n+1}(i) < \underline{\lambda(\lambda P_n)^m B_n v_n}$$

Then action $k \in K_i$ is non-optimal in state i at iteration (n+1)

<u>Proof</u>. From (40) and (41)

$$Bv_{n+1}(i,k) = r(i,k) + \lambda \sum_{j \in I} P(i,j,k) v_{n+1}(j) - v_{n+1}(i)$$

$$= r(i,k) + \lambda \sum_{j\varepsilon I} P(i,j,k)[v_{n+1}(j)-v_\ell(j)+v_\ell(j)]-v_{n+1}(i)$$

$$= r(i,k) + \lambda \sum_{j\varepsilon I} P(i,j,k)v_\ell(j) + \lambda \sum_{j\varepsilon I} P(i,j,k)Dv_{\ell,n-\ell+1}(j) - v_{n+1}(i)$$

$$\leq r(i,k) + \lambda \sum_{j\varepsilon I} P(i,j,k)v_\ell(j) + \lambda \sum_{p=\ell}^{n} \overline{Dv}_{p,1} - v_{n+1}(j) \tag{42}$$

$$< \underline{\lambda(\lambda P_n)^m B_n v_n} = \lambda P_n[\underline{(\lambda P_n)^m B_n v_n} \cdot \tilde{1}](i)$$

$$\leq [(\lambda P_n)^{m+1} B_n v_n](i) .$$

Therefore by Proposition 2.4, $k \varepsilon K_i$ is non-optimal in state i of iteration (n+1).

An alternative action elimination algorithm can be based on the following corollary.

<u>Corollary 2.5</u>.

Suppose at iteration (n+1) that for some $\ell \leq n$

$$r(i,k) + \lambda \sum_{j\varepsilon I} P(i,j,k)v_\ell(j) + \lambda\overline{Dv}_{\ell,n-\ell+1} - v_{n+1}(i) < \underline{\lambda(\lambda P_n)^m B_n v_n}. \tag{43}$$

Then action $k \varepsilon K_i$ is non-optimal in state i at iteration (n+1).

<u>Proof</u>. Using $\overline{Dv}_{\ell,n-\ell+1}$ instead of $\sum_{p=\ell}^{n} \overline{Dv}_{p,1}$ in (42) gives the result.

Single iteration action elimination algorithms for policy iteration can be obtained by replacing the bound on the right hand side of (41) or (43) by 0.

We discuss the relation of the results in this section to those of Hastings and van Nunen [12] and Hübner [15]. Hastings and van Nunen have proposed the following test for detecting non-optimal actions at iteration $n + \ell$ of value iteration; an action $k \in K_i$ is non-optimal in state i at iteration $n + \ell$ if

$$(44) \qquad v_{n+1}(i) - [r(i,k) + \lambda \sum_{j\in I} P(i,j,k)v_n(j)] - \lambda \sum_{p=n}^{n+\ell-1} sp[v_{p+1}-v_p] > 0$$

To investigate the equivalence of this test and the test based on (41) we set $m=0$ and $\ell=n$ in (41) and $\ell=1$ in (44). This is because with $m=0$, modified policy iteration is value iteration and with $\ell=1$ Hastings and van Nunen's algorithm eliminates actions at the next iteration. We find in this case that the two procedures are equivalent.

Hübner's elimination procedure [15] is based on replacing the last term on the left hand side of (44) by

$$(45) \qquad \delta_{ik} \sum_{p=n}^{n+\ell-1} \lambda^{p-n} sp[v_{p+1} - v_p]$$

where

$$\delta_{ik} = \underset{k'\in K_i}{\mathrm{Max}} \lambda(1 - \sum_{j\in I} \mathrm{Min}(P(i,j,k),P(i,j,k')))$$

and

$$\delta = \max_{(i,k),(i',k')} \lambda\Big(1 - \sum_{j\varepsilon I} \mathrm{Min}(P(i,j,k),P(i',j,k'))\Big).$$

He showed that $\delta_{ik} \le \delta \le \lambda$ and that a test based on substituting (45) into (44) is more efficient than using (44) directly. In applications δ_{ik} and δ are usually approximated with more easily computable quantities.

6. Action Elimination Algorithms

In this section we apply results from the previous sections to develop computational procedures. Policy iteration and modified policy iteration algorithms including action elimination are represented by the flow chart in Figure 2.2.

For the action elimination step we select one of the procedures below. The first, which is due to Grinold [9] applies to policy iteration only, while the remainder have versions for both policy iteration and modified policy iteration. Except in the first case we state the modified policy iteration versions of these procedure.

I. (Policy Iteration) Suppose that at iteration n+1

$$r(i,k) + \lambda \sum_{j\varepsilon I} P(i,j,k)\, v_n(j) < v_n(i) + \frac{1}{1-\lambda}\underline{B_n v_n} - \frac{\lambda}{1-\lambda}\overline{B_n v_n}\,. \tag{46}$$

Then action $k \varepsilon K_i$ in state i is non-optimal and need not be considered in the improvement phase of iterations n+1, n+2,$\cdots$.

II. Suppose that at iteration n+1.

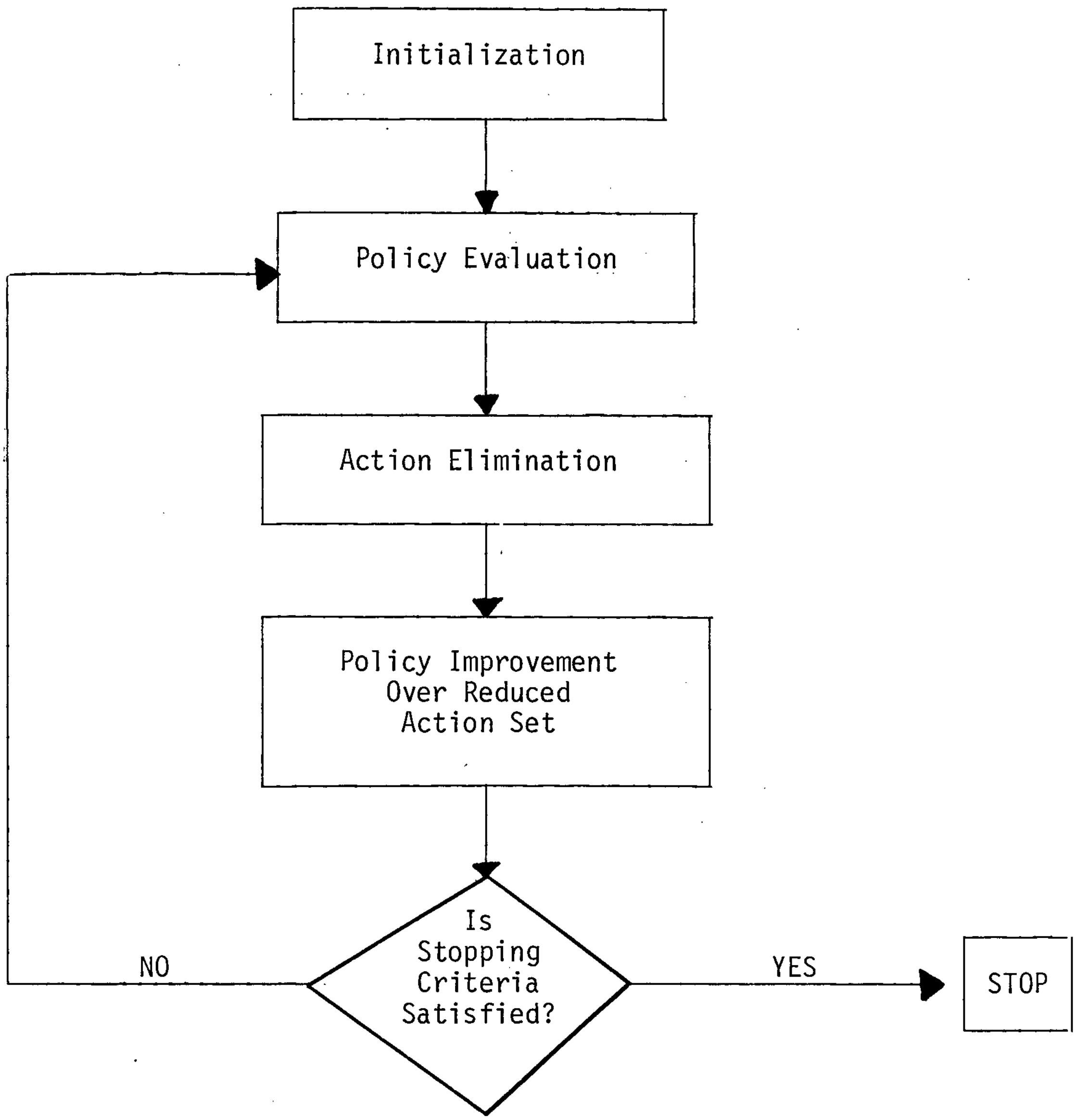

Figure 2.2. Flow Chart of an Action Elimination Algorithm.

$$(47) \qquad r(i,k) + \lambda \sum_{j\varepsilon I} P(i,j,k)\ v_n(j) < v_{n+1}(i) + \frac{\lambda^{m+1}}{1-\lambda}\underline{B_n v_n} - \frac{\lambda}{1-\lambda}\overline{B_n v_n}$$

Then action $k \varepsilon K_i$ in state i is non-optimal and need not be considered in the improvement phase of iterations n+1, n+2,$\cdots$.

In the case of policy iteration set $m = +\infty$ in (46) to obtain

$$(48) \qquad r(i,k) + \lambda \sum_{j\varepsilon I} P(i,j,k)\ v_n(j) < v_{n+1}(i) - \frac{\lambda}{1-\lambda}\overline{B_n v_n}\ .$$

The above two procedures detect non-optimal actions and eliminate them from all subsequent iterations. The following two procedures eliminate actions at the subsequent iteration only. Consequently, if action $k \varepsilon K_i$ has been eliminated in state i at iteration n, the test quantity $r(i,k) + \lambda \sum_{j\varepsilon I} P(i,j,k)v_n(j)$ has not been computed and hence is not available for action elimination at iteration n+1. Because of this fact, we introduce the following.

Define $E_n(i)$ to be the set of actions that have been eliminated in state i at iteration n and set $E_o(i) = \emptyset$. Define $F_{\ell,n}(i)$ to be the set of actions that have been eliminated for iterations $\ell+1, \ell+2, \cdots, n$ and last evaluated at iteration ℓ, i.e.,

$$F_{\ell,n}(i) = \left[\bigcap_{p=\ell+1}^{n} E_p(i)\right] \cap (E_\ell(i))^c$$

where the superscript c indicates complement.

The elimination procedures are as follows:

III. Suppose that at iteration n+1 and for $k \in K_i - E_n(i)$

(49) $$r(i,k) + \lambda \sum_{j \in I} P(i,j,k)\, v_n(j) + \lambda\, \overline{Dv}_{n,1} < v_{n+1}(i) + \underline{\lambda(\lambda P_n)^m B_n v_n}\,.$$

Then $k \in E_{n+1}(i)$.

Further, if for $\ell = n-1, n-2, \cdots, 1$ and $k \in F_{\ell,n}(i)$ the following holds:

(50) $$r(i,k) + \lambda \sum_{j \in I} P(i,j,k)\, v_\ell(j) + \lambda \sum_{p=\ell}^{n} \overline{Dv}_{p,1} < v_{n+1}(i) + \underline{\lambda(\lambda P_n)^m B_n v_n}.$$

Then $k \in E_{n+1}(i)$.

IV. Replace (50) by

(51) $$r(i,k) + \lambda \sum_{j \in I} P(i,j,k)\, v_\ell(j) + \lambda \overline{Dv}_{\ell,n-\ell+1} < v_{n+1}(i) + \underline{\lambda(\lambda P_n)^m B_n v_n}.$$

For policy iteration, the quantity $\underline{\lambda(\lambda P_n)^m B_n v_n}$ is replaced by 0 in (49)-(51).

We now discuss the implementation of procedures III or IV in the modified policy iteration algorithm. We initialize by selecting v_0

according to (8). We next evaluate $v_1 = T_o^{m+1} v_o$. It may be computationally efficient to perform action elimination before the first improvement. That is because for action elimination we compute

$$(52) \qquad r(i,k) + \lambda \sum_{j\varepsilon I} P(i,j,k)\, v_o(j) + \lambda\overline{Dv}_{0,1} - v_1(i).$$

Since v_o is a constant vector we see that (52) is equal to

$$r(i,k) + \frac{\lambda}{1-\lambda}(\operatorname*{Min}_{i\varepsilon I} r_o(i)) + \lambda\overline{Dv}_{0,1} - v_1(i)$$

which can be evaluated by fewer calculations than the test quantity

$$r(i,k) + \lambda \sum_j P(i,j,k)\, v_1(j)$$

in the subsequent improvement step. But it will not eliminate many actions since $\frac{\lambda}{1-\lambda}(\operatorname*{Min}_{i\varepsilon I} r_o(i)) + \overline{Dv}_{0,1}$ is usually larger than $\lambda\sum_j P(i,j,k)v_1(j)$. Therefore action elimination procedures will start from iteration 1.

The difference between III and IV is the quantities $\sum_{p=\ell}^{n} \overline{Dv}_{p,1}$ and $\overline{Dv}_{\ell,n+1-\ell}$. Note that

$$(53) \qquad \overline{Dv}_{\ell,n+1-\ell} = \operatorname*{Max}_{i\varepsilon I}[v_{n+1}(i) - v_\ell(i)] = \operatorname*{Max}_{i\varepsilon I}[\sum_{p=\ell}^{n}(v_{p+1}(i) - v_p(i))]$$

$$\leq \sum_{p=\ell}^{n} \operatorname*{Max}_{i\varepsilon I}[v_{p+1}(i) - v_p(i)] = \sum_{p=\ell}^{n} \overline{Dv}_{p,1} .$$

Hence IV will eliminate more actions than III but require storing the vectors $v_1,\cdots,v_n$. In practice this algorithm converges quite quickly and this additional storage will be fairly insignificant.

Note also that the bound used on the right hand side of (49), (50) and (51) is

$$\underline{\lambda(\lambda P_n)^m B_n v_n} = \lambda \underset{i\varepsilon I}{\mathrm{Min}} \; ([(\lambda P_n)^m B_n v_n](i)).$$

It is evident from the proof of Theorem 2.4 that a better lower bound would be

$$\underset{i\varepsilon I}{\mathrm{Min}} \; ([(\lambda P_n)^{m+1} B_n v_n](i)).$$

However this would require an additional calculation after evaluating v_{n+1} while $\underline{\lambda(\lambda P_n)^m B_n v_n}$ is readily available. This is because

$$(54) \qquad (\lambda P_n)^m B_n v_n = T_n^{m+1} v_n - T_n^m v_n .$$

Both terms on the right hand side of (54) are compuated in the evaluation step.

We present the detailed algorithm incorporating action elimination procedure II, III and IV into the modified policy iteration procedure. Minor changes will give the corresponding algorithms for policy iteration. We will initialize an additional array, $\mathrm{Flag}_n(i,k)$, to indicate whether an action has been eliminated. It is subscripted as n only for the purpose of stating the algorithm.

STEP 1) Initialization:

Select $\varepsilon > 0$ to be used as a stopping criterion. Set $n = 0$ and find $\delta_o \in \Delta$ such that

$$r_o(i) = r(i, \delta_o(i)) = \underset{k \varepsilon K_i}{\text{Max}} \; r(i,k), \quad i \in I.$$

Set

$$v_o = \frac{1}{(1-\lambda)} \; (\underset{i \varepsilon I}{\text{Min}} \; r_o(i)) \cdot \tilde{1} \; .$$

Calculate $v_1 = T_o^{m+1} \; v_o$ where m is determined by

$$m = \min\{s: sp[T_n^{s+1} v_n - T_n^s v_n] \leq \varepsilon\} \; .$$

Set $n = n+1$ and $\text{Flag}_n(i,k) = 0$ for all $i \in I$ and $k \in K_i$. Go to step 2.

STEP 2) Improvement Phase:

For each $i \in I$, $k \in K_i$ if $\text{Flag}_n(i,k) \leq 0$ calculate $Tv_n(i,k)$. Otherwise do not calculate $Tv_n(i,k)$. Find δ_n such that

$$Tv_n(i,\delta_n(i)) = \underset{k \varepsilon K_i}{\text{Max}} \; \{Tv_n(i,k) | \text{Flag}_n(i,k) \leq 0\} \; .$$

Compute $\overline{B_n v_n}$ and $\underline{B_n v_n}$. If $sp(B_n v_n) \leq \varepsilon$ go to step 5. Otherwise go to step 3.

STEP 3) <u>Evaluation Phase:</u>

$v_{n+1} = T_n^{m+1} v_n$ where m is determined by

$$m = \text{Min}\ \{s: sp[T_n^{s+1} v_n - T_n^s v_n] \leq \varepsilon\}\ .$$

Go to step 4.

STEP 4) <u>Elimination Phase:</u>

In procedure II;

If $\text{Flag}_n(i,k) \leq 0$ and $T_n v_n(i,k) - v_{n+1}(i) < \underline{\frac{\lambda^{m+1}}{(1-\lambda)} B_n v_n} - \frac{\lambda}{(1-\lambda)} \overline{B_n v_n}$, then $\text{Flag}_{n+1}(i,k) = 1$. Otherwise $\text{Flag}_{n+1}(i,k) = \text{Flag}_n(i,k)$. Set $n = n+1$ and go to step 2.

In procedure III;

Set $Tv_{n+1}(i,k) = Tv_n(i,k) + \overline{Dv}_{n,1}$.

If $Tv_{n+1}(i,k) < v_{n+1}(i) + \underline{\lambda(\lambda P_n)^m B_n v_n}$ then $\text{Flag}_{n+1}(i,k) = 1$. Otherwise $\text{Flag}_{n+1}(i,k) = 0$. Set $n = n+1$ and go to step 2.

In procedure IV;

If $\text{Flag}_n(i,k) \leq 0$ and $Tv_n(i,k) + \lambda\overline{Dv}_{n,1} < v_{n+1}(i) + \underline{\lambda(\lambda P_n)^m B_n v_n}$ then $\text{Flag}_{n+1}(i,k) = n$.

If $\text{Flag}_n(k,k) \leq 0$ and $Tv_n(i,k) + \lambda\overline{Dv}_{n,1} \geq v_{n+1}(i) + \underline{\lambda(\lambda P_n)^m B_n v_n}$ then $\text{Flag}_{n+1}(i,k) = \text{Flag}_n(i,k) = 0$.

If $\text{Flag}_n(i,k) = \ell > 0$ and $Tv_\ell(i,k) + \overline{Dv}_{\ell,n-\ell+1} < v_{n+1}(i) + \underline{\lambda(\lambda P_n)^m B_n v_n}$ then $\text{Flag}_{n+1}(i,k) = \ell$.

If $\text{Flag}_n(i,k) = \ell > 0$ and $Tv_\ell(i,k) + \lambda\overline{Dv}_{\ell,n=\ell+1} \geq v_{n+1}(i) + \underline{\lambda(\lambda P_n)^m B_n v_n}$ then $\text{Flag}_{n+1}(i,k) = 0$.

Set $n = n+1$ and go to step 2.

STEP 5) Final Extrapolation:

Set $v = v_n + B_n v_n + \frac{\lambda}{(1-\lambda)} \underline{B_n v_n}\, \tilde{1}$ and stop.

Some comments about this algorithm are in order. For procedure III we do not have to store the indicator of the last evaluated iteration, ℓ, because the only term added to the left hand side of (50) at each iteration is always $\overline{Dv}_{n,1}$. On the other hand, the last evaluated iteration, ℓ, must be stored for procedure IV because the value of $\lambda\overline{Dv}_{\ell,n-\ell+1}$ must be stored for procedure IV because the value of $\lambda\overline{Dv}_{\ell,n-\ell+1}$ in (56) depends on v_{n+1} and v_ℓ. Therefore in procedure IV, if $k \in F_{\ell,n}(i)$ then $\text{Flag}_n(i,k) = \ell$.

In the above algorithms with action elimination procedures we must store the $N \times M$ array, $Tv(i,k)$, and the $N \times M$ array, $\text{Flag}(i,k)$, in addition to the storage space required for the algorithm without an action elimination procedure. Here N is the number of states and M is the number of actions in each state assuming each state has an equal number of actions. Furthermore in procedure IV we require additional space to store the vectors $v_1,\cdots,v_n$ and in procedure III additional space to store the vector v_n. Therefore at least $2N \times M$ additional storage space is required for the algorithms above. But compared to the space to store the $N^2 \times M$ array for the transition probabilities, $P(i,j,k)$, this additional storage requirement is not too significant.

In the cases where even 2N x M additional words cannot be put into core storage, we can utilize another method to decrease the 2N x M spaces to N x M spaces by using P(i,1,k) as Flag(i,k). Changes to be made in the algorithms above are as follows:

(a) We initialize $P(i,j,k) = \lambda P(i,j,k) < 1$ for $i\varepsilon I$, $j\varepsilon I$, $k\varepsilon K_i$. Drop λ in the calculation of $Tv_n(i,k)$ and $Bv_n(i,k)$. $\underline{(\lambda P_n)^m B_n v_n}$ becomes $\underline{(P_n)^m B_n v_n}$.

(b) In the improvement phase: If $P(i,1,k) < 1$ calculate $Tv_n(i,k)$. Otherwise do not calculate $Tv_n(i,k)$. Find δ_n such that

$$Tv_n(i,\delta_n(i)) = \underset{k\varepsilon K_i}{\text{Max}} \{Tv_n(i,k) | P(i,1,k) < 1\}$$

(c) In the elimination phase:

For procedure II; If $P(i,j,k) < 1$ and (47) is satisfied then $P(i,1,k) = 1 + P(i,1,k)$. Otherwise do not change $P(i,1,k)$.

For procedure III; If $Tv_{n+1}(i,k) < v_{n+1}(i)\ \lambda\underline{(P_n)^m B_n v_n}$ then $P(i,1,k) = 1 + P(i,1,k)$. Otherwise, $P(i,1,k) = P(i,1,k) - 1$ if $P(i,1,k) \geq 1$ and $P(i,1,k) = P(i,1,k)$ if $P(i,1,k) < 1$.

For procedure IV; If $P(i,1,k) < 1$ and (49) is satisfied then $P(i,1,k) = n + P(i,1,k)$. If $P(i,1,k) < 1$ and (49) is not satisfied then $P(i,1,k) = P(i,1,k)$. If $P(i,1,k) \geq 1$ and (51) with ℓ = greatest integer in $P(i,1,k)$ is satisfied then $P(i,1,k) = P(i,1,k)$. If $P(i,1,k) \geq 1$ and (51) with ℓ = greatest integer in $P(i,1,k)$ is not satisfied then $P(i,1,k) = P(i,1,k) - \ell$.

For a problem with large state and action spaces, the transition probabilities may be stored in a file or on a tape and read at each improvement phase. In this case, if we use $P(i,1,k)$ as $Flag(i,k)$ we have to read $P(i,1,k)$ from a file or a tape to check whether an action is eliminated or not. The operation of reading from a file or tape into core storage is quite inefficient and requires considerable CPU time. This can be avoided by storing $Flag(i,k)$ and $Tv(i,k)$ in the same array for procedures II and III. Storing $Flag(i,k)$ and $Tv(i,k)$ in the same array may not be done for procedure IV since we have to identify the last evaluated iteration, ℓ, of each action eliminated. To store $Flag(i,k)$ and $Tv(i,k)$ in the same array, we use a scalar constant $\bar{v}$ that is greater than $\underset{I \varepsilon I}{\text{Max}}\ v^*(i)$. This scalar value $\bar{v}$ can be calculated easily in the initialization phase as follows:

$$r_0(i) = r(i, \delta_0(i)) = \underset{k \varepsilon K_i}{\text{Max}}\ r(i,k)\ ,\ i\ \varepsilon\ I.$$

$$\bar{v} = \frac{1}{(1-\lambda)} \underset{i \varepsilon I}{\text{Max}}\ r_0(i) > \underset{i \varepsilon I}{\text{Max}}\ v^*(i)\ .$$

Using $\bar{v}$ above the changes necessary in the algorithm for procedure II and III are as follows:

(a) In improvement phase: If $Flag_n(i,k) \leq 0$ calculate $Tv_n(i,k)$ and set $Flag_n(i,k) = Tv_n(i,k) - \bar{v} < 0$.

(b) In elimination phase:

For procedure II; If $Flag_n(i,k) \leq 0$ and $Flag_n(i,k) + \bar{v} < v_{n+1}(i) + \frac{\lambda^{m+1}}{(1-\lambda)} \underline{B_n v_n} - \frac{\lambda}{(1-\lambda)} \overline{B_n v_n}$, then $Flag_{n+1}(i,k) = 1$.

Otherwise $\mathrm{Flag}_{n+1}(i,k) = \mathrm{Flag}_n(i,k)$.

For procedure III; If $\mathrm{Flag}_n(i,k) \leq 0$ and $A_n(i,k) = \mathrm{Flag}_n(i,k) + \bar{v} + \lambda\overline{Dv}_{n,1} < v_{n+1}(i) + \underline{\lambda(\lambda P_n)^m B_n v_n}$ then $\mathrm{Flag}_{n+1}(i,k) = v_{n+1}(i) + \underline{\lambda(\lambda P_n)^m B_n v_n} - A_n(i,k) > 0$. If $\mathrm{Flag}_n(i,k) \leq 0$ and $A_n(i,k) \geq v_{n+1}(i) + \lambda(\lambda P_n)^m B_n v_n$ then $\mathrm{Flag}_{n+1}(i,k) = \mathrm{Flag}_n(i,k)$. If $\mathrm{Flag}_n(i,k) > 0$ and $C_n(i,k) = -\mathrm{Flag}_n(i,k) + v_n(i) + \underline{\lambda(\lambda P_{n-1})^m B_{n-1} v_{n-1}} + \lambda\overline{Dv}_{n,1} < v_{n+1}(i) + \underline{\lambda(\lambda P_n)^m B_n v_n}$ then $\mathrm{Flag}_{n+1}(i,k) = v_{n+1}(i) + \underline{\lambda(\lambda P_n)^m B_n v_n} - C_n(i,k) > 0$. If $\mathrm{Flag}_n(i,k) > 0$ and $C_n(i,k) \geq v_{n+1}(i) + \underline{\lambda(\lambda P_n)^m B_n v_n}$ then $\mathrm{Flag}_{n+1}(i,k) = 0$.

Some comments about the two storage saving methods are in order. $A_n(i,k)$ and $C_n(i,k)$ are just the terms on the left hand side of (49) and (50) respectively. $A_n(i,k)$ and $C_n(i,k)$ are used for clarity but are not required to be stored. For the value iteration algorithm an alternative approach to reduce storage has been developed by Hastings and Mello [10] and Porteus [20]. Their methods eliminate fewer actions than those based on the best available bounds because they use less tight bounds to reduce storage. Using P(i,1,k) as Flag (i,k) their methods require no additional storage space. While in the case where the P(i,j,k) array is stored in a file or tape and a Flag(i,k)-type array is used to indicate whether or not an action has been eliminated, a modification similar to that described above using Flag(i,k) and Tv(i,k) in the same array with tighter bounds than in Hastings and Mello [10] and Porteus [20] will eliminate more actions without increasing storage. For modified policy iteration a similar approach to that in [10] and [20] would eliminate far fewer actions than the methods herein because v_{n+1} is much greater than v_n.

7. Computational Results

To determine the efficiency of the algorithm described in the previous section we solved Howard's [14] automobile replacement problem and a randomly generated problem with discount rates .8333, .86956, .909 and .9532. These discount rates were chosen because they were also used by Grinold [9] in testing his procedure. We first describe the automobile replacement problem in detail.

Automobile Replacement Problem. (Howard [14, p. 54].)

Consider the problem of automobile replacement over a time interval of ten years. The state of the system, i, is described by the age of the car in three-month periods; i running from 1 to 40. Every three months we decide whether to keep our present car (k=1) or to trade it in for a car of age k-2; k running from 2 to 41. In order to keep the number of states finite, a car of age 40 remains a car of age 40 forever (it is considered to be essentially worn out). The actions available in each state are: to keep the present car for another quarter (k=1) or to buy a car of age k-2, k=2,3,•••,41. The problem has 40 states with 41 actions in each state. Hence there are 41^{40} possible stationary strategies.

The data supplied are the following: C_i is the cost of buying a car of age i: T_i is the trade-in value of a car of age i; E_i is the expected cost of operating a car of age i until it reaches age i+1; and P_i is the probability that a car of age i will survive to be i+1 without incurring a prohibitively expensive repair.

The probability defined here is necessary to limit the number of states. A car of any age that has a hopeless breakdown is immediately sent to state 40. Naturally, $P_{40} = 0$.

Using our earlier notation we have

$$r(i,k) = -E_i \quad \text{for} \quad k = 1$$

$$r(i,k) = T_i - C_{k-2} - E_{k-2} \quad \text{for} \quad k > 1$$

$$P(i,j,k) = \begin{cases} P_i & j = i+1 \\ 1 - P_i & j = 40 \\ 0 & \text{others} \end{cases} \qquad \text{for } k = 1$$

$$P(i,j,k) = \begin{cases} P_{k-2} & j = i+1 \\ 1 - P_{k-2} & j = 40 \\ 0 & \text{others} \end{cases} \qquad \text{for } k > 1 \; .$$

The numerical values for these parameters are listed in Table 2.1.

The Randomly Generated Problem.

The randomly generated problem had 40 states and 100 actions in each. It was generated as follows: the data for the expected one period rewards, $r(i,k)$, were generated from a truncated normal distribution, the transition probabilities, $P(i,j,k)$, were selected so that there were non-zero entries at three random locations in each state for each action and the non-zero probabilities were generated from a uniform distribution.

Table 2.1

Data for the Automobile Replacement Problem

Age in Periods i	Cost C_i	Trade-in Value T_i	Operating Expense E_i	Survival Probability P_i	Age in Periods i	Cost C_i	Trade-in Value T_i	Operating Expense E_i	Survival Probability P_i
0	\$2000	\$1600	\$50	1.000					
1	1840	1460	53	0.999	21	\$345	\$240	\$115	0.925
2	1680	1340	56	0.998	22	330	225	118	0.919
3	1560	1230	59	0.997	23	315	210	121	0.910
4	1300	1050	62	0.996	24	300	200	125	0.900
5	1220	980	65	0.994	25	290	190	129	0.890
6	1150	910	68	0.991	26	280	180	133	0.880
7	1080	840	71	0.988	27	265	170	137	0.865
8	900	710	75	0.985	28	250	160	141	0.850
9	840	650	78	0.983	29	240	150	145	0.820
10	780	600	81	0.980	30	230	145	150	0.790
11	730	550	84	0.975	31	220	140	155	0.760
12	600	480	87	0.970	32	210	135	160	0.730
13	560	430	90	0.965	33	200	130	167	0.660
14	520	390	93	0.960	34	190	120	175	0.590
15	480	360	96	0.955	35	180	115	182	0.510
16	440	330	100	0.950	36	170	110	190	0.430
17	420	310	103	0.945	37	160	105	205	0.300
18	400	290	106	0.940	38	150	95	220	0.200
19	380	270	109	0.935	39	140	87	235	0.100
20	360	255	112	0.930	40	130	80	250	0

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

PAGE 49 HAS BEEN OMITTED

The computer program for the randomly generated problem is in the Appendix. The randomly generated problems in Puterman and Shin [26] were not used since the optimal policies for these problems were found in one or two iterations. In the problem used here the structure varied considerably from policy to policy and consequently required more effort to solve. This was advantageous for investigating the properties of our algorithm.

We solved these problems using policy iteration and modified policy iteration alone and with each of the action elimination procedures. The evaluation phase was terminated by using (9) with $\varepsilon = 0.1$ This value of ε was also used for stopping the algorithm.

All calculations were performed on the University of British Columbia AMDAHL 470 computer using the codes in the Appendix. Results reported in Tables 2.2. and 2.3 are the fraction of actions eliminated at each iteration. These results are based on using policy iteration. We found that when modified policy iteration was used results did not differ in every case. In the cases where different results were obtained the modified policy iteration results are included in parentheses. For modified policy iteration, we did not use procedure I which was clearly dominated by procedure II.

It is interesting to note that in all cases except those indicated by * in Table 2.2 the number of actions eliminated by procedures III and IV increased at each iteration. The reason for the decrease in those cases marked by * is that $\sum_{p=\ell}^{n} \overline{Dv}_{p,1}$ is not a very good upper bound for $v_{n+1} - v_\ell$. When the better upper bound, $\overline{Dv}_{\ell,n+1-\ell}$, was employed this monotonicity property was preserved (see (53)). It was interesting to note that although

Table 2.2

Comparison of Action Elimination Procedures with Policy Iteration -

Automobile Replacement Problem

(Numbers are fraction of actions eliminated at iteration n, asterisked cells are described in text.)

λ	.8333				.86956				.909				.9532			
n \ Procedure	I	II	III	IV	I	II	III	IV	I	II	III	IV	I	II	III	IV
1	.0	.0	.4573	.4573	.0	.0	.4707	.4707	.0	.0	.4152	.4152	.0	.0	.3409	.3409
2	.0561	.0707	.5518	.6659	.0	.0	.5037	.6311	.0	.0	.4183	.5530	.0	.0	.3177* (.3171)	.4470
3	.2549	.2683	.7152	.8561	.0982	.1000 (.0994)	.5829	.8238	.0	.0	.5366	.7726	.0	.0	.3128* (.3134)	.5311
4	opt.	opt.	opt.	opt.	.4585	.4604 (.4616)	.8183	.9110	.0823	.0835	.5671	.8695	.0	.0	.6628 (.6622)	.7927
5					opt.	opt.	opt.	opt.	.1628	.1652 (.1640)	.7732	.8921	.0	.0	.4354*	.8195
6									.3738	.3744 (.3738)	.8512 (.8506)	.9372	opt.	opt.	opt.	opt.
									opt.	opt.	opt.	opt.				

Table 2.3

Comparison of Action Elimination Procedures with Policy Iteration - Randomly Generated Problem

(Numbers are fractions of actions eliminated at iterations n)

λ	.8333				.86956				.909				.9532			
n \ Procedure	I	II	III	IV	I	II	III	IV	I	II	III	IV	I	II	III	IV
1	.0	.0	.6972	.6972	.0	.0	.6525	.6525 (.6520)	.0	.0	.6255	.6255	.0	.0	.5667	.5667
2	.0100	.0445 (.0345)	.7620 (.7617)	.9163 (.9160)	.0	.0010 (.0)	.7592	.9150 (.9152)	.0	.0	.7445 (.7438)	.9122 (.9117)	.0	.0	.5800 (.5797)	.8530
3	.6005	.6915 (.6797)	.8920 (.8917)	.9765	.3833	.5427 (.4947)	.8872 (.8877)	.9765 (.9763)	.0510	.1513 (.0963)	.8817 (.8820)	.9752 (.9750)	.0	.0	.8547	.9710
4	.9602	.9663 (.9645)	.9540 (.9545)	.9877	.9027	.9065 (.9055)	.9338	.9858 (.9860)	.9197	.9238 (.9217)	.9575 (.9580)	.9877 (.9880)	.6005	.6542 (.6130)	.9433 (.9422)	.9877
5	.9890	.9890	.9865	.9900	opt.	opt.	opt.	opt.	opt.	opt.	opt.	opt.	opt.	opt.	opt.	opt.
6	opt.	opt.	opt.	opt.												

the number of actions eliminated at each iteration increased in the remaining cases, we could not conclude that an action eliminated at a particular iteration would remain eliminated at subsequent iterations. This is the reason for defining the sets $F_{\ell,n}(i)$ for procedures III and IV.

For the automobile replacement problem, we also combined procedure II, with III and IV. In each case we obtained the same results as using procedures III and IV alone.

The CPU times to solve problems using various procedures are shown in Table 2.4. Reduction of CPU times using procedures III and IV were significant in each problem. For instance, in the randomly generated problem with $\lambda = .8333$, policy iteration with procedure IV took only 34.8 per cent of the CPU time taken by policy iteration without any action elimination procedure. Modified policy iteration was slower than policy iteration in every case. This is due to the small state spaces of the problems. This issue is discussed in detail in Puterman and Shin [26, p. 1134].

8. Conclusions.

We were very encouraged by our computational results. As indicated in Table 2.2 and 2.3 procedures III and IV were very effective in decreasing the size of the action space to be searched during an improvement step. In problems with a large number of actions, this will decrease computational requirements considerably, i.e., if there are N states and M actions then without action elimination each improvement step will require MN^2 multiplications while if 100α per cent of the actions have been eliminated, only $(1-\alpha)MN^2$ multiplications would be required. Also as indicated

Table 2.4

Computational Time

λ	Procedure	CPU Time (Secs.) for the automobile replacement problem: Policy Iteration	CPU Time (Secs.) for the automobile replacement problem: Modified Policy Iteration	CPU Time (Secs.) for the randomly generated problem: Policy Iteration	CPU Time (Secs.) for the randomly generated problem: Modified Policy Iteration
.8333	*	.683	.812	2.422	2.457
	I	.682	-	1.549	-
	II	.652	.789	1.457	1.538
	III	.432	.564	.861	.919
	IV	.429	.534	.844	.884
.86956	*	.856	1.061	2.015	2.076
	I	.820	-	1.629	-
	II	.796	1.025	1.531	1.622
	III	.501	.717	.852	.920
	IV	.476	.678	.815	.860
.909	*	1.197	1.492	2.024	2.077
	I	1.171	-	1.759	-
	II	1.134	1.420	1.666	1.758
	III	.660	.944	.863	.935
	IV	.586	.840	.823	.874
.9532	*	1.040	1.494	2.013	2.103
	I	1.090	-	1.910	-
	II	1.063	1.542	1.830	1.926
	III	.717	1.224	.968	1.045
	IV	.648	1.099	.872	.927

(* indicates results without action elimination procedures.)

in Table 2.4 we obtained significant reduction in computation time by including action action elimination procedures III and IV.

For problems with large state spaces, Puterman and Shin [26] found modified policy iteration to be considerably more effective than policy iteration and value iteration with all its variants. We recommend using modified policy iteration together with procedure IV for solving discounted Markovian decision problems with large state spaces and policy iteration together with procedure IV for problems with small state spaces and large action spaces.

CHAPTER III

COMPUTATIONAL METHODS FOR PARAMETRIC MARKOV DECISION PROBLEMS

1. Introduction.

In this chapter we study finite discounted Markov decision problems (MDP) in which the reward vector is parametrized by a scalar and develop algorithms to solve them. The algorithms use dynamic programming methods based on properties of the optimality equation and results in the previous chapter on action elimination. Without action elimination the method is similar to the parametric simplex algorithm of linear programming (c.f. Dantzig [5]). Because of the sensitivity of results to roundoff error a modification based on approximations to the expected total discounted returns is also presented.

These algorithms are of interest because bi-criterion MDP and MDP subject to a single constraint can be formulated as problems with the reward vector including a single parameter.

Henig [13] investigated a general class of dynamic programs with vector criterion and presented conditions which imply that the set of policies with nondominated total returns can be characterized and approximated by the set of all nondominated stationary policies. Viswanathan, Aggarwal and Nair [33] and Henig [13] have suggested solution procedures for MDP with vector criterion based on solution procedures for vector criterion linear programming methods. Later, White and Kim [34] reformulated the MDP with vector criterion as a specially structured partially observed MDP and introduced two procedures for solving them based on successive approximations and policy iterations. These methods in [34] were developed by Smallwood and Sondik [28] and Sondik [29] for partially observed problems. However, a shortcoming of these methods in [34] is that they require more computational time and storage than the method proposed here simply because they require more evaluation and improvement steps.

In Section 2 the problem and notation are defined. Algorithms based on parametric linear programming with simplex and block pivoting are developed using dynamic programming terminology in Section 3. The algorithm in Section 3 is motivational and the actual computational method is discussed in Section 4 using approximations of values to resolve the difficulties caused by roundoff error.

In Section 5 an action elimination procedure for this problem is developed and the detailed algorithms are presented in Section 6 to reduce computational efforts.

Bi-criterion MDP and one-constrained MDP are formulated as parametric MDP in Section 7. Numerical results appear in Section 8 where a comparison of algorithms with and without action elimination procedures on Howard's [14] automobile replacement problem with another criterion are presented.

2. Preliminaries.

In this chapter we use the same notation as in Chapter 2 except for the additional quantities defined below. Let α denote the scalar parameter of variation in the one-stage rewards and define constants $\underline{\alpha}$ and $\bar{\alpha}$ such that $\underline{\alpha} \leq \alpha \leq \bar{\alpha}$. When the system is in state $i \in I$ and an action $k \in K_i$ is selected, $r(i,k)$ and $d(i,k)$ are the one-stage rewards. Let $r^\delta(i)$ and $d^\delta(i)$ be $r(i,\delta(i))$ and $d(i,\delta(i))$. Define r_α^δ by $r_\alpha^\delta = r^\delta + \alpha d^\delta$. For each $\delta \in \Delta$ let v_α^δ and u^δ be the expected total discounted rewards with respect to r_α^δ and d^δ. Then

(1) $$v_\alpha^\delta = v^\delta + \alpha u^\delta.$$

For each α, the problem we study is that of finding a $\delta_\alpha^* \in \Delta$ and $v_\alpha^* \in V$ such that

(2) $$v_\alpha^* = v_\alpha^{\delta_\alpha^*} = \underset{\delta \in \Delta}{\operatorname{Max}}\, v_\alpha^\delta .$$

For a given $v, u \in V$ and $\delta \in \Delta$, define the linear operators T_δ^α, S_δ, B_δ^α and G_δ mapping V to V by

(3) $$T_\delta^\alpha v \equiv r_\alpha^\delta + \lambda P_\delta v$$

(4) $$S_\delta u \equiv d^\delta + \lambda P_\delta u$$

(5) $$B_\delta^\alpha v \equiv T_\delta^\alpha v - v$$

(6) $$G_\delta u \equiv S_\delta u - u$$

and define the policy δ_{v_α} and the operator $H^\alpha: V \to V$ by

(7) $$H^\alpha v_\alpha \equiv B^\alpha_{\delta_{v_\alpha}} v_\alpha = \operatorname*{Max}_{\delta\varepsilon\Delta} B^\alpha_\delta v_\alpha \ .$$

For $k = \delta(i) \varepsilon K_i$ we use the following notation:

$$B^\alpha_\delta v_\alpha(i) = B^\alpha v_\alpha(i,\delta(i)) = B^\alpha v_\alpha(i,k)$$

$$G_\delta u(i) = Gu(i,\delta(i)) = Gu(i,k) \ .$$

The problem defined by (2) is equivalent to finding v^*_α and δ^*_α that satisfy

(8) $$B^\alpha_{\delta^*_\alpha} v^*_\alpha = H^\alpha v^*_\alpha = 0 \ .$$

For clarity we will use r^δ_n, δ_n, v_n, u_n, B^n, T^n and H^n instead of $r^\delta_{\alpha_n}$, $\delta^*_{\alpha_n}$, $v^*_{\alpha_n}$, u^{δ_n}, B^{α_n}, T^{α_n} and H^{α_n} .

3. An Algorithm Based on Parametric Linear Programming.

The linear programming formulation of the discounted MDP was first given by D'Epenoux [7]. Mine and Osaki [17] studied the relationship between policy iteration and linear programming and showed that the policy iteration is a modification of the simplex algorithm of linear programming in which the pivot operations for many variables are performed simultaneously. Here we develop an algorithm based on parametric linear programming to solve the parametric MDP. In this algorithm pivoting is done on a block of variables or states. We use dynamic programming terminology throughout.

The following proposition is of a technical nature and is needed in the sequel.

Proposition 3.1.

The following equalities hold for all $\alpha \in [\underline{\alpha},\bar{\alpha}]$ and $\delta \in \Delta$;

(9) $$v_\alpha^\delta = v + (I - \lambda P_\delta)^{-1} B_\delta^\alpha v$$

(10) $$u^\delta = u + (I - \lambda P_\delta)^{-1} G_\delta u \quad .$$

Proof. We have

$$v + (I - \lambda P_\delta)^{-1} B_\delta^\alpha u = v + (I - \lambda P_\delta)^{-1}[r_\alpha^\delta + (\lambda P_\delta - I)v]$$

$$= (I - \lambda P_\delta)^{-1} r_\alpha^\delta = v_\alpha^\delta$$

$$u + (I - \lambda P_\delta)^{-1} G_\delta u = u + (I - \lambda P_\delta)^{-1}[d^\delta + (\lambda P_\delta - I)u]$$

$$= (I - \lambda P_\delta)^{-1} d^\delta = u^\delta.$$

Proposition 3.2.

Suppose δ_n is optimal at α_n. Then

(11) $B^n v_n(i,k) \leq 0,$ $\quad i \in I$ and $k \in K_i$, and

(12) $v_\alpha^{\delta_n} = v_n + (\alpha - \alpha_n)u_n$.

Proof. Inequality (11) is an immediate consequence of (7) and the definition of v_n.

From (1) and the definition of v_n and u_n,

$$v_\alpha^{\delta_n} = v^{\delta_n} + \alpha u^{\delta_n} = v^{\delta_n} + \alpha_n u^{\delta_n} + (\alpha - \alpha_n)u^{\delta_n}$$

$$= v_n + (\alpha - \alpha_n)u_n .$$

As a consequence of (12) the following lemmas hold.

Lemma 3.1.

Suppose δ_n is optimal at α_n. Then

(13) $B^\alpha v_\alpha^{\delta_n}(i,k) = B^n v_n(i,k) + (\alpha-\alpha_n)Gu_n(i,k),$ $\quad i \in I$ and $k \in K_i$ and

(14) $B_{\delta_n}^\alpha v_\alpha^{\delta_n} = 0$.

Proof. Equation (13) follows from substituting (12) into $B^{\alpha}v_{\alpha}^{\delta_n}(i,k)$ and rearranging terms.

From (13) we have

$$B_{\delta_n}^{\alpha} v_{\alpha}^{\delta_n} = B_{\delta_n}^{n} v_n + (\alpha - \alpha_n) G_{\delta_n} u_n .$$

$B_{\delta_n}^{n} v_n$ and $G_{\delta_n} u_n = 0$ since $v_n = (I - \lambda P_{\delta_n})^{-1} r^{\delta_n}$ and $u_n = (I - \lambda P_{\delta_n})^{-1} d^{\delta_n}$. Therefore (14) follows.

Lemma 3.2.

Suppose

(15) $\quad B^{\alpha}v_{\alpha}^{\delta_n}(i,k) \leq 0 , \qquad i \in I \text{ and } k \in K_i \quad .$

Then δ_n is optimal at α.

Proof. The result follows immediately from (14).

Define $A_n \equiv \{(i,k) | Gu_n(i,k) > 0\}$, $R_n(i,k) \equiv \dfrac{-B^n v_n(i,k)}{Gu_n(i,k)}$ and let Φ denote the null set. Then the following theorem is the basis for the algorithm to solve the parametric MDP.

Theorem 3.1.

The optimal policy δ_n at α_n is optimal for α satisfying $\alpha_n \leq \alpha \leq \alpha_{n+1}$ where α_{n+1} is defined by

$$\alpha_{n+1} = \alpha_n + \hat{\alpha} \tag{16}$$

and

$$\hat{\alpha} = \begin{cases} \underset{(i,k)\varepsilon A_n}{\text{Min}} \{R_n(i,k)\} & \text{if } A_n \neq \Phi \\ \\ +\infty & \text{if } A_n = \Phi . \end{cases}$$

Proof. If $A_n = \Phi$, i.e., $Gu_n(i,k) \leq 0$ for all $i \varepsilon I$ and $k \varepsilon K_i$ then for all $\alpha \geq \alpha_n$, $B^\alpha v_\alpha^{\delta_n}(i,k) \leq 0$, $i \varepsilon I$ and $k \varepsilon K_i$ from (13) and (11). Therefore from (15) δ_n is optimal for all $\alpha \geq \alpha_n$.

Otherwise at α satisfying $\alpha_n \leq \alpha \leq \alpha_n + \hat{\alpha}$

$$B^\alpha v_\alpha^{\delta_n}(i,k) \leq 0 \quad \text{for } (i,k) \notin A_n \text{ from (13) and (11)}$$

and

$$(\alpha - \alpha_n) \leq \hat{\alpha} \leq R_n(i,k) \quad \text{for} \quad (i,k) \varepsilon A_n .$$

Thus

$$B^\alpha v_\alpha^{\delta_n}(i,k) = B^n v_n(i,k) + (\alpha-\alpha_n)\, Gu_n(i,k) \leq 0 \text{ for } (i,k) \varepsilon A_n .$$

Therefore from (15) δ_n is optimal for α satisfying $\alpha_n \leq \alpha \leq \alpha_n + \hat{\alpha}$.

At $\alpha_{n+1} < \bar{\alpha}$ define δ_{n+1} as follows:

$$(A)\quad \begin{cases} \delta_{n+1}(i) = \delta_n(i) & \text{for} \quad i \notin I_n \\ \delta_{n+1}(i) = k_i^* & \text{for} \quad i \in I_n \end{cases}$$

where

$$I_n = \{i \in I | R_n(i,k) = \hat{\alpha} \ / \ (i,k) \in A_n\}$$

$$K_i^n = \{k \in K_i | R_n(i,k) = \hat{\alpha} \ / \ (i,k) \in A_n, \ i \in I_n\}$$

and

$k_i^* \in K_i^n$ is determined by

$$Gu_n(i,k_i^*) = \operatorname*{Max}_{k \in K_i^n} Gu_n(i,k) \ .$$

When there exist ties in determining $k_i^* \in K_i^n$ we choose any $k_i^* \in K_i^n$.

This selection of a new optimal policy at α_{n+1} is different from the usual selection of a new optimal policy if an approach similar to parametric linear programming was used. In that case only one action would be changed and the selection rule is

$$\delta_{n+1}(i) = \delta_n(i) \qquad \text{for} \qquad i \neq i^*$$

$$\delta_{n+1}(i) = k^* \qquad \text{for} \qquad i = i^*$$

where i^* and k^* are selected by

$$Gu_n(i^*,k^*) = \operatorname*{Max}_{\substack{i \in I_n \\ k \in K_i^n}} \{Gu_n(i,k)\}.$$

Each of these selection rules will give a new optimal policy at α_{n+1} and a better policy than the current policy, δ_n, over the region of α, $\alpha > \alpha_{n+1}$. But the selection rule (A) may save computation if there is more than one state in I_n. Therefore we will use the selection rule (A) in the sequel even though the same results hold with the other selection rule.

Corollary 3.1.

Suppose δ_{n+1} is selected using the rule (A) and $\alpha_{n+1} \leq \bar{\alpha}$. Then at α_{n+1} both δ_n and δ_{n+1} are optimal and $v_\alpha^{\delta_{n+1}} > v_\alpha^{\delta_n}$ for all $\alpha > \alpha_{n+1}$.

Proof. Using (1), (9), (10) and the selection rule (A),

$$v_{\alpha_{n+1}}^{\delta_{n+1}} = v_{\alpha_n}^{\delta_{n+1}} + (\alpha_{n+1} - \alpha_n)\, u^{\delta_{n+1}}$$

$$= v_n + (I-\lambda P_{\delta_{n+1}})^{-1} B^n_{\delta_{n+1}} v_n + (\alpha_{n+1}-\alpha_n)[u_n + (I-\lambda P_{\delta_{n+1}})^{-1} G_{\delta_{n+1}} u_n]$$

$$= v_n + (\alpha_{n+1}-\alpha_n)u_n + (I-\lambda P_{\delta_{n+1}})^{-1}[B^n_{\delta_{n+1}} v_n + (\alpha_{n+1}-\alpha_n)G_{\delta_{n+1}} u_n]$$

$$= v_{\alpha_{n+1}}^{\delta_n} .$$

Because δ_n is optimal at α_{n+1} from Theorem 3.1, δ_{n+1} is also optimal at α_{n+1}.

For $\alpha > \alpha_{n+1}$ from the selection rule (A)

$$(17) \qquad B^{\alpha}_{\delta_{n+1}} v_{\alpha}^{\delta_n} = B^{n+1}_{\delta_{n+1}} v_{n+1} + (\alpha-\alpha_{n+1}) G_{\delta_{n+1}} u_n = (\alpha-\alpha_{n+1}) G_{\delta_{n+1}} u_n > 0 \ .$$

Therefore from (9) and (17)

$$v^{\delta_{n+1}} = v^{\delta_n} + (I - \lambda P_{\delta_{n+1}})^{-1} B^{\alpha}_{\delta_{n+1}} v^{\delta_n}$$

$$= v_{\alpha}^{\delta_n} + \sum_{s=0}^{\infty} (\lambda P_{\delta_{n+1}})^s B^{\alpha}_{\delta_{n+1}} v_{\alpha}^{\delta_n} \geq v_{\alpha}^{\delta_n} + B^{\alpha}_{\delta_{n+1}} v_{\alpha}^{\delta_n} > v_{\alpha}^{\delta_n} \ .$$

In the following corollary we show that at α_{n+1}, v_{n+1} and $B^{n+1}_{\delta} v_{n+1}$ can be calculated directly from the values available at α_n.

<u>Corollary 3.2</u>.

At α_{n+1} we have

$$(18) \qquad v_{n+1} = v_n + (\alpha_{n+1} - \alpha_n) u_n ,$$

$$(19) \qquad B^{n+1}_{\delta} v_{n+1} = B^{n}_{\delta} v_n + (\alpha_{n+1} - \alpha_n) G_{\delta} u_n \ .$$

<u>Proof</u>. (18) follows from the fact that δ_n is still optimal at α_{n+1}. Equation (13) evaluated at α_{n+1} gives (19) .

From the above we develop the following algorithm to solve the problem for all $\alpha \in [\underline{\alpha}, \bar{\alpha}]$.

STEP 1) Set $\alpha_o = \underline{\alpha}$ and $n = 0$. Find δ_o, v_o, $B^o v_o(i,k)$ and $u_o = (I - \lambda P_{\delta_o})^{-1} d^{\delta_o}$. Go to step 2.

STEP 2) Calculate $Gu_n(i,k) = d(i,k) + \lambda \sum_{u \varepsilon I} P(i,j,k) u_n(j) - u_n(i)$ for all $i \varepsilon I$ and $k \varepsilon K_i$. Go to step 3.

STEP 3) If $Gu_n(i,k) \leq 0$ for all $i \varepsilon I$ and $k \varepsilon K_i$, stop. In this case δ_n remains optimal for all $\alpha \geq \alpha_n$. Otherwise, set $\alpha_{n+1} = \alpha_n + \hat{\alpha}$ where $\hat{\alpha}$ is defined by (16). Choose δ_{n+1} using the selection rule (A). Go to step 4.

STEP 4) If $\alpha_{n+1} \geq \bar{\alpha}$, stop. Otherwise set

$$v_{n+1} = v_n + (\alpha_{n+1} - \alpha_n) u_n$$

$$B^{n+1} v_{n+1}(i,k) = B^n v_n(i,k) + (\alpha_{n+1} - \alpha_n) Gu_n(i,k)$$

$$u_{n+1} = (I - \lambda P_{\delta_{n+1}})^{-1} d^{\delta_{n+1}} = u_n + (I - \lambda P_{\delta_{n+1}})^{-1} G_{\delta_{n+1}} u_n.$$

Set $n = n+1$ and go to step 2.

Note that the calculation of $(I-\lambda P_{\delta_{n+1}})^{-1}$ in step 4 can be implemented by block pivoting.

One might conjecture that for some m $\alpha_n = \alpha_m < \bar{\alpha}$ for all $n \geq m$, i.e. we do not find an optimal policy for α, $\alpha_n < \alpha \leq \bar{\alpha}$. But this is not the case because even if $\alpha_{m+1} = \alpha_m$, $u_{m+1} > u_m$. Hence the finiteness of the policy set implies that for some $k > m$, $\alpha_k > \alpha_m$.

However difficulties might arise because of roundoff. In particular A_n might not include potentially optimal actions because $Gu_n(i,k)$ might have the wrong sign or α_{n+1} might be less than α_n because for some $(i,k) \in A_n$, $B^n v_n(i,k)$ might be positive instead of negative. In the first case, a nonoptimal policy might be selected while in the later case the algorithm would cycle. A method to alleviate these difficulties is presented in the next section.

4. An Algorithm for Finding ε_n-Optimal Policies.

In this section we develop an algorithm based on approximations of the values of v_n and u_n to resolve the problems caused by roundoff when we implement the algorithm of the previous section. This algorithm is based on redefining $R_n(i,k)$ and A_n to eliminate cycling and ensure selection of optimal actions.

Let $\hat{v}_n$ and $\hat{u}_n$ be approximations of v_n and $u^{\hat{\delta}_n}$ where $\hat{\delta}_n$ is defined such that $H^n\hat{v}_n = B^n_{\hat{\delta}_n}\hat{v}_n$, $\varepsilon_n \equiv \varepsilon_o + \alpha_n\varepsilon'$ for $\varepsilon_o > 0$ and $\varepsilon' > 0$, $sp(H^n\hat{v}_n) \leq (1-\lambda)\varepsilon_n$ and $sp(G_{\hat{\delta}_n}\hat{u}_n) \leq (1-\lambda)\varepsilon'$. Then we have the following proposition.

Proposition 3.3.

Suppose $sp(H^n\hat{v}_n) = sp(B^n_{\hat{\delta}_n}\hat{v}_n) \leq (1-\lambda)\varepsilon_n$. Then $\hat{\delta}_n$ is ε_n-optimal, i.e.,

$$v_{\alpha_n}^{\hat{\delta}_n} \leq v_n \leq v_{\alpha_n}^{\hat{\delta}_n} + \varepsilon_n \ . \tag{20}$$

Proof. The left inequality of (20) is true because $v_{\alpha_n}^{\hat{\delta}_n} \leq v_{\alpha_n}^{\delta_n} = v^*_{\alpha_n} = v_n$.

From (9)

$$v_n - v_{\alpha_n}^{\hat{\delta}_n} = \hat{v}_n + (I-\lambda P_{\delta_n})^{-1}B^n_{\delta_n}\hat{v}_n - [\hat{v}_n + (I-\lambda P_{\hat{\delta}_n})^{-1}B^n_{\hat{\delta}_n}\hat{v}_n]$$

$$= (I-\lambda P_{\delta_n})^{-1}B^n_{\delta_n}\hat{v}_n - (I-\lambda P_{\hat{\delta}_n})^{-1}B^n_{\hat{\delta}_n}\hat{v}_n$$

$$\leq (I-\lambda P_{\delta_n})^{-1}H^n\hat{v}_n - (I-\lambda P_{\hat{\delta}_n})^{-1}B^n_{\hat{\delta}_n}\hat{v}_n$$

$$= (I-\lambda P_{\delta_n})^{-1}B^n_{\hat{\delta}_n}\hat{v}_n - (I-\lambda P_{\hat{\delta}_n})^{-1}B^n_{\hat{\delta}_n}\hat{v}_n$$

$$\leq (I-\lambda P_{\delta_n})^{-1}\overline{B^n_{\hat{\delta}_n}\hat{v}_n}\ \hat{1} - (I-\lambda P_{\hat{\delta}_n})^{-1}\ \underline{B^n_{\hat{\delta}_n}\hat{v}_n}\ \hat{1}$$

$$= \frac{1}{(1-\lambda)}\, sp(B^n_{\hat{\delta}_n}\hat{v}_n) \leq \varepsilon_n$$

where $\overline{B^n_{\hat{\delta}_n}\hat{v}_n} = \underset{i\varepsilon I}{\text{Max}}\ B^n_{\hat{\delta}_n}\hat{v}_n(i)$ and $\underline{B^n_{\hat{\delta}_n}\hat{v}_n} = \underset{i\varepsilon I}{\text{Min}}\ B^n_{\hat{\delta}_n}\hat{v}_n(i)$.

Therefore $v_n \leq v_{\alpha_n}^{\delta_n} + \varepsilon_n$ and (20) holds.

Define $v_\alpha^{\hat{\delta}_n}$ by

$$\hat{v}_\alpha^{\hat{\delta}_n} = \hat{v}_n + (\alpha - \alpha_n)\,\hat{u}_n \;. \tag{21}$$

We drop hats to facilitate typing and use the same notation as in Section 3. But all v_n, u_n and δ_n are $\hat{v}_n$, $\hat{u}_n$ and $\hat{\delta}_n$. Then the following lemma is immediate from (21).

Lemma 3.3.

For $\alpha \geq \alpha_n$

$$B_\delta^\alpha\, v_\alpha^{\delta_n} = B_\delta^n\, v_n + (\alpha - \alpha_n) G_\delta u_n. \tag{22}$$

Lemma 3.4.

Suppose for $\alpha \geq \alpha_n$, $B_\delta^\alpha v_\alpha^{\delta_n} \leq B_{\delta_n}^\alpha\, v_\alpha^{\delta_n}$ for all $\delta \in \Delta$. Then δ_n is ε_α-optimal at α where $\varepsilon_\alpha = \varepsilon_n + (\alpha - \alpha_n)\,\varepsilon'$.

Proof. Note $H^\alpha v_\alpha^{\delta_n} = B_{\delta_n}^\alpha\, v_\alpha^{\delta_n}$.

$$sp(H^\alpha v_\alpha^{\delta_n}) = sp(B_{\delta_n}^\alpha\, v_\alpha^{\delta_n}) = sp(B_{\delta_n}^n\, v_n + (\alpha-\alpha_n) G_{\delta_n} u_n)$$

$$\leq sp(B_{\delta_n}^n\, v_n) + (\alpha-\alpha_n)\, sp(G_{\delta_n} u_n) \leq (1-\lambda)\varepsilon_n + (\alpha-\alpha_n)(1-\lambda)\varepsilon'$$

$$= (1-\lambda)[\varepsilon_n + (\alpha-\alpha_n)\,\varepsilon'] = (1-\lambda)\,\varepsilon_\alpha.$$

Therefore δ_n is ε_α-optimal at α from Proposition 3.3.

Lemma 3.4 indicates that we can find the region where δ_n is ε_α-optimal by identifying α for which

$$H^\alpha v_\alpha^{\delta_n} = B_{\delta_n}^\alpha v_\alpha^{\delta_n} = B_{\delta_n}^n v_n + (\alpha-\alpha_n) G_{\delta_n} u_n .$$

Redefine $A_n \equiv \{(i,k) | Gu_n(i,k) - Gu_n(i,\delta_n(i)) > 0\}$ and $R_n(i,k)$ by

$$R_n(i,k) = \frac{B^n v_n(i,\delta_n(i)) - B^n v_n(i,k)}{Gu_n(i,k) - Gu_n(i,\delta_n(i))} .$$

Theorem 3.2.

The ε_n-optimal policy δ_n at α_n is ε_α-optimal for α satisfying $\alpha_n \leq \alpha \leq \alpha_{n+1}$ where α_{n+1} is defined by

$$\alpha_{n+1} = \alpha_n + \hat{\alpha} \tag{23}$$

$$\hat{\alpha} = \begin{cases} \min\limits_{(i,k)\varepsilon A_n} \{R_n(i,k)\} & \text{if } A_n \neq \Phi \\ +\infty & \text{if } A_n = \Phi . \end{cases}$$

Proof. Using Lemma 3.4 and (22) instead of (11) and (15), and (13) in the proof Theorem 3.1 gives the result.

At $\alpha_{n+1} < \bar{\alpha}$ we will have the same rule for selecting δ_{n+1} as the selection rule (A) in Section 3 except using $R_n(i,k)$ and A_n redefined here. Let us call it the selection rule (B). Also we can have the same selection rule based on the change of a single action as done in Section 3 by using $R_n(i,k)$ and A_n redefined here. Then each of these selection rules gives an ε_{n+1}-optimal policy at α_{n+1} by defining $v_{\alpha_{n+1}}^{\delta_n} \equiv v_{n+1} \equiv v_{\alpha_{n+1}}^{\delta_{n+1}}$.

Corollary 3.3.

Suppose δ_{n+1} is selected using the rule (B) and $\alpha_{n+1} < \bar{\alpha}$. Then at α_{n+1} both δ_n and δ_{n+1} are ε_{n+1}-optimal.

Proof. From (23)

$$B_{\delta_{n+1}}^{n+1} v_{\alpha_{n+1}}^{\delta_n} = B_{\delta_{n+1}}^{n+1} v_{n+1} = B_{\delta_n}^{n+1} v_{n+1} = H^{n+1} v_{n+1}.$$

I.e., δ_n and δ_{n+1} are ε_{n+1}-optimal at α_{n+1} from Lemma 3.4.

At α_{n+1}, v_{n+1} and $B_\delta^{n+1} v_{n+1}$ can be calculated directly from the values available at α_n in the following way:

$$v_{n+1} = v_n + (\alpha_{n+1} - \alpha_n)u_n \tag{24}$$

$$B_\delta^{n+1} v_{n+1} = B_\delta^n v_n + (\alpha_{n+1} - \alpha_n)G_\delta u_n \tag{25}$$

The changes to be made in the algorithm of the previous section are as follows:

(a) In step 1, find v_0 with $H^o v_0 = B^o_{\delta_0} v_0$ and $sp(H^o v_0) \leq (1-\lambda)\ \varepsilon$.

(b) In step 3, $\hat{\alpha}$ is defined by (24) and choose δ_{n+1} using the selection rule (B).

(c) In step 4, find u_{n+1} with $sp(G_{\delta_{n+1}} u_{n+1}) \leq (1-\lambda)\varepsilon'$.

5. Action Elimination.

The purpose of this section is to develop action elimination procedures for the algorithm of the previous section. To eliminate a certain action at step 2 and 3 of the algorithm we need upper bounds on $B^n v_n(i,k)$, $Gu_n(i,k)$ and α_{n+1}. A special feature of this procedure is eliminating actions at the subsequent iteration only by using bounds for α_{n+1}. We use the following notation.

(26) $$UGu^s_p(i,k) \equiv Gu_s(i,k) + \sum_{q=s}^{p-1} [\lambda\overline{Du}_{q,1} - Du_{q,1}(i)] \text{ for } p \geq s+1$$

or

(27) $$\equiv Gu_s(i,k) + \lambda\overline{Du}_{s,p-s} - Du_{s,p-s}(i) \text{ for } p \geq s+1$$

(28) $$UBv^{\ell}_n(i,k) \equiv B^{\ell}v_{\ell}(i,k) + \sum_{p=\ell}^{n-1} (\alpha_{p+1} - \alpha_p)UGu^{\ell-1}_p(i,k) \text{ for } n \geq \ell+1$$

where $\overline{Du}_{q,1}$, $Du_{q,1}$, $\overline{Du}_{s,p-s}$ and $Du_{s,p-s}$ are defined in Section 5 of Chapter 2.

Lemma 3.5.

$UGu^s_p(i,k)$ and $UBv^{\ell}_n(i,k)$ are the upper bounds of $Gu_p(i,k)$ and $B^n v_n(i,k)$ respectively.

Proof. $Gu_p(i,k) \leq UGu_p^s(i,k)$ follows from the results in the proof of Theorem 2.5 and Corollary 2.5.

From (25) and $Gu_p(i,k) \leq UGu_p(i,k)$

$$B^n v_n(i,k) = B^{n-1}v_{n-1}(i,k) + (\alpha_n - \alpha_{n-1})Gu_{n-1}(i,k)$$

$$= B^{n-2}v_{n-2}(i,k) + (\alpha_{n-1} - \alpha_{n-2})Gu_{n-2}(i,k) + (\alpha_n - \alpha_{n-1}) Gu_{n-1}(i,k)$$

$$\vdots$$

$$= B^{\ell}v_{\ell}(i,k) + \sum_{p=\ell}^{n-1} (\alpha_{p+1} - \alpha_p)Gu_p(i,k)$$

$$\leq B^{\ell}v_{\ell}(i,k) + \sum_{p=\ell}^{n-1} (\alpha_{p+1} - \alpha_p)UGu_p^{\ell-1}(i,k) = UBv_n^{\ell}(i,k).$$

Assuming $\alpha'_{n+1} \geq \alpha_{n+1}$ is known at iteration n, the following lemma is the basis for an action elimination procedure.

Lemma 3.6.

Suppose at iteration n

$$(29) \qquad B^{\alpha'_{n+1}} v^{\delta_n}_{\alpha'_{n+1}}(i,k) < B^{\alpha'_{n+1}}_{\delta_n} v^{\delta_n}_{\alpha'_{n+1}}(i) .$$

Then $k \in K_i$ is not ε_{n+1}-optimal in state i at α_{n+1}.

Proof. If $Gu_n(i,k) - Gu_n(i,\delta_n(i)) \leq 0$ then $(i,k) \notin A_n$ and so $k \in K_i$ is not ε_{n+1}-optimal at α_{n+1}. Otherwise, from (29) and (25)

$B^n v_n(i,k) + (\alpha'_{n+1}-\alpha_n)Gu_n(i,k) < B^n v_n(i, \delta_n(i)) + (\alpha'_{n+1}-\alpha_n)Gu_n(i,\delta_n(i))$.

Hence $(\alpha'_{n+1}-\alpha_n)[Gu_n(i,k) - Gu_n(i,\delta_n(i))] < [B^n v_n(i,\delta_n(i)) - B^n v_n(i,k)]$.

Because $Gu_n(i,k) - Gu_n(i,\delta_n(i)) > 0$, $R_n(i,k) > (\alpha'_{n+1}-\alpha_n) \geq (\alpha_{n+1}-\alpha_n)$.

Therefore $k\epsilon K_i$ is not ε_{n+1}-optimal in state i at α_{n+1}.

To use Lemma 3.6 in action elimination we need bounds for the terms in (26), (27) and (28).

Theorem 3.3.

Suppose at iteration n that for some $\ell \leq n-1$

(30) $$UBv_n^{\ell}(i,k) + (\alpha'_{n+1}-\alpha_n)UGu_n^{\ell-1}(i,k) < B^n_{\delta_n} v_n(i) + (\alpha'_{n+1}-\alpha_n)G_{\delta_n} u_n(i).$$

Then $k\epsilon K_i$ is not ε_{n+1}-optimal in state i at α_{n+1}.

Proof. From (22), Lemma 3.5 and (30)

(31) $$\begin{aligned} B^{\alpha'_{n+1}} v^{\delta_n}_{\alpha'_{n+1}}(i,k) &= B^n v_n(i,k) + (\alpha'_{n+1}-\alpha_n)Gu_n(i,k) \\ &\leq UBv_n^{\ell}(i,k) + (\alpha'_{n+1}-\alpha_n)UGu_n^{\ell-1}(i,k) \\ &< B^n_{\delta_n} v_n(i) + (\alpha'_{n+1}-\alpha_n)G_{\delta_n} u_n(i) = B^{\alpha'_{n+1}}_{\delta_n} v^{\delta_n}_{\alpha'_{n+1}}(i) \end{aligned}$$

The result follows from Lemma 3.6.

For $\ell = n$ we obtain the following corollary.

Corollary 3.4.

Suppose at iteration n

$$B^n v_n(i,k) + (\alpha'_{n+1}-\alpha_n)UGu_n^{n-1}(i,k) < B^n_{\delta_n} v_n(i) + (\alpha'_{n+1}-\alpha_n)G_{\delta_n} u_n(i).$$

Then $k \in K_i$ is not ε_{n+1}-optimal in state i at α_{n+1}.

Proof. Using $B^n v_n(i,k)$ instead of $UBv_n^{\ell}(i,k)$ in (31) gives the result.

Finding appropriate α'_{n+1} and using $UGu_p^{\ell-1}(i,k)$ in (26) or (27) we can develop several action elimination algorithms.

6. Action Elimination Algorithms.

In this section we develop specific computational procedure based on choosing appropriate values for α'_{n+1} and $UGu_n^{\ell-1}(i,k)$ in Theorem 3.3 and Corollary 3.4. These action elimination procedures will be included at the second step of the algorithm presented in Section 4. This is a little different from the action elimination procedures in Chapter 2. The reason is that we have to calculate $G_{\delta_n} u_n$ at step 3 even for the algorithm without action elimination. We use $E_n(i)$ defined in Section 6 of Chapter 2 but redefine $F_{\ell,n}(i)$ to be the set of actions that have been eliminated for iterations ℓ, $\ell+1,\cdots,n$, i.e.,

$$F_{\ell,n}(i) = [\bigcap_{p=\ell}^{n} E_p(i)] \cap (E_{\ell-1}(i))^c.$$

Action Elimination Procedure (I); $\alpha'_{n+1} = \bar{\alpha}$.

Suppose at step 2 of iteration n in the algorithm of Section 4 and for $k \in K_i - E_{n-1}(i)$

(32) $$B^n v_n(i,k) + (\bar{\alpha}-\alpha_n)UGu_n^{n-1}(i,k) < B^n v_n(i,\delta_n(i)) + (\bar{\alpha}-\alpha_n)Gu_n(i,\delta_n(i)).$$

Then $k \in E_n(i)$:

Furthermore, if for $\ell = n-1, n-2,\cdots,1$ and $k \in F_{\ell,n-1}(i)$ the following holds:

(33) $$UBv_n^{\ell}(i,k) + (\bar{\alpha}-\alpha_n)UGu_n^{\ell-1}(i,k) < B^n v_n(i,\delta_n(i)) + (\bar{\alpha}-\alpha_n)Gu_n(i,\delta_n(i)).$$

Then $k \in E_n(i)$.

Let procedure (I-1) and (I-2) denote the action elimination procedure (I) with $UGu_p^s(i,k)$ of (26) and (27) respectively. Then $UBv_n^{\ell}(i,k)$ and $UGu_n^{\ell-1}(i,k)$ in (32) and (33) can be calculated as follows:

(34) $$UBv_n^{\ell}(i,k) = UBv_{n-1}^{\ell}(i,k) + (\alpha_n-\alpha_{n-1})UGu_{n-1}^{\ell-1}(i,k) \text{ for } \ell \leq n-2.$$

or

(35) $$= B^{n-1}v_{n-1}(i,k) + (\alpha_n-\alpha_{n-1})UGu_{n-1}^{\ell-1}(i,k) \text{ for } \ell = n-1.$$

(36) $$UGu_n^{n-1}(i,k) = Gu_{n-1}(i,k) + \overline{Du}_{n-1,1} - Du_{n-1,1}(i).$$

In procedure (I-1);

(37) $$UGu_n^{\ell-1}(i,k) = UGu_{n-1}^{\ell-1}(i,k) + \lambda\overline{Du}_{n-1,1} - \overline{Du}_{n-1,1}(i) \text{ for } \ell \leq n-1.$$

In procedure (I-2);

(38) $$UGu_n^{\ell-1}(i,k) = UGu_{n-1}^{\ell-1}(i,k) + \lambda(\overline{Du}_{\ell-1,n+1-\ell} - \overline{Du}_{\ell-1,n-\ell}) - Du_{n-1,1}(i)$$
$$\text{for } \ell \leq n-1.$$

$Flag_n(i,k)$ indicates whether an action $k \varepsilon K_i$ is eliminated or not. In procedure (I-2) $Flag_n(i,k) = \ell$ if $k \varepsilon F_{\ell,n}(i)$. But in procedure (I-1) we do not have to store ℓ for $k \varepsilon F_{\ell,n}(i)$ because the term added to $UGu_n^{\ell-1}(i,k)$ for $UGu_{n-1}^{\ell-1}(i,k)$ in (37) is $\lambda\overline{Du}_{n-1,1} - Du_{n-1,1}(i)$. Redefine A_n by $A_n \equiv \{(i,k) \mid Gu_n(i,k) - Gu_n(i,\delta_n(i)) > 0 \text{ and } Flag_n(i,k) < 0\}$. When there is a difference between procedure (I-1) and (I-2), procedure (I-2) is shown in parenthesis in the following algorithm.

STEP 1) Set $\alpha_o = \underline{\alpha}$ and $n = 0$. Find δ_o, v_o, $B_\delta^o v_o$ and u_o with $H^\rho v_o = B_{\delta_o}^o v_o$, $sp(H^\rho v_o \leq \varepsilon_o$ and $sp(G_{\delta_o} u_o) \leq \varepsilon'$. Initialize $Flag_o(i,k) = -1$ for all $i \varepsilon I$ and $k \varepsilon K_i$ and calculate $Gu_o(i,k)$ for all $i \varepsilon I$ and $k \varepsilon K_i$ and go to step 3.

STEP 2) Calculate $G_{\delta_n} u_n$.

If $Flag_{n-1}(i,k) < 0$ and (32) is satisfied then $Flag_n(i,k) = 1$ [$Flag_n(i,k) = n$] and do not calculate $Gu_n(i,k)$.

If $Flag_{n-1}(i,k) < 0$ and (32) is not satisfied then $Flag_n(i,k) = Flag_{n-1}(i,k)$ and calculate $Gu_n(i,k)$.

If $Flag_{n-1}(i,k) \geq 0$ and (33) is satisfied then $Flag_n(i,k) = Flag_{n-1}(i,k)$ and do not calculate $Gu_n(i,k)$.

If $Flag_{n-1}(i,k) \geq 0$ and (33) is not satisfied then $Flag_n(i,k) = -1$ and calculate $B^n v_n(i,k)$ and $Gu_n(i,k)$.

Go to step 3.

STEP 3) If $Gu_n(i,k) - Gu_n(i, {}_n(i)) \leq 0$ for all $i \varepsilon I$ and $k \varepsilon K_i$ where $Flag_n(i,k) < 0$, then δ_n remains α-optimal for all α satisfying $\alpha \geq \alpha_n$ and stop. Otherwise, set α_{n+1} as in (23) and choose δ_{n+1} by the selection rule (B).

STEP 4) If $\alpha_{n+1} \geq \bar{\alpha}$ stop. Otherwise set $v_{n+1} = v_n + (\alpha_{n+1}-\alpha_n)u_n$ and calculate u_{n+1} with $sp(G_{\delta_{n+1}} u_{n+1}) \leq \varepsilon'$. Set $B^{n+1}v_{n+1}(i,k) = B^n v_n(i,k) + (\alpha_{n+1}-\alpha_n)Gu_n(i,k)$ for $k \varepsilon K_i$ where $Flag_n(i,k) < 0$. For $k \varepsilon K_i$ where $Flag_n(i,k) \geq 0$, calcualte $UBv^{\ell}_{n+1}(i,k)$ and $UGu^{\ell-1}_{n+1}(i,k)$ using (34) or (35) and (36) or (37) [using (34) or (35) and (36) or (38) depending on $\ell = Flag_n(i,k)$]. Set $n = n+1$ and return to step 2.

Some comments about this algorithm are in order. $UBv_n^{\ell}(i,k)$ and $UGu_n^{\ell-1}(i,k)$ can be stored in the same arrays as $B^n v_n$ and $Gu_n(i,k)$ respectively. $UGu_p^s(i,k)$ of (27) is a tighter upper bound for $Gu_p(i,k)$ than $UGu_p^s(i,k)$ of (26). Hence procedure (I-2) will eliminate more actions than procedure (I-1), but requires storage of the vectors $u_o, u_1, \cdots, u_n$, whereas procedure (I-1) only requires storage of the vector u_n. Therefore the algorithm above with action elimination procedure (I-1) or (I-2) requires at least an N x M array, $Flag_n(i,k)$, in addition to the storage space required for the algorithm without the action elimination procedure. Here N is the number of states and M is the number of actions in each state assuming each state has an equal number of actions.

Using P(i,1,k) for $Flag_n(i,k)$ as in Section 6 of Chapter 2 we can reduce the additional storage requirements. Furthermore, for a problem with a large state and action space, the transition probabilities may be stored in a file or on a tape and read whenever needed. Denote $a_n(i)$ by $a_n(i) = B^n v_n(i, {}_n(i)) + (\bar{\alpha}-\alpha_n)Gu_n(i,\delta_n(i))$. In this case a method for storing only $Flag_n(i,k)$ and $a_n(i)$ instead of $B^n v_n(i,k)$ and $Flag_n(i,k)$ is based on rearranging terms in (33) to obtain;

$$(39) \qquad UBv_{n-1}^{\ell}(i,k) + (\bar{\alpha}-\alpha_{n-1})UGu_{n-1}^{\ell-1}(i,k) + (\bar{\alpha}-\alpha_n)(\lambda\overline{Du}_{n-1,1} - Du_{n-1,1}(i)] < a_n(i)$$

where $\quad UBv_{n-1}^{\ell}(i,k) = B^{n-1}v_{n-1}(i,k)$ for $\ell = n-1$.

The quantity $Flag_{n-1}(i,k)$ is redefined as either $B^n v_n(i,\delta_n(i)) - B^n v_n(i,k)$ or $UBv_{n-1}^{\ell}(i,k) + (\bar{\alpha}-\alpha_{n-1})UGu_{n-1}^{\ell-1}(i,k) - a_{n-1}(i)$ with $UBv_{n-1}^{\ell}(i,k) =$

$B^{n-1}v_{n-1}(i,k)$ for $\ell = n-1$. If an action $k \in K_i$ is not eliminated at iteration $(n-1)$, $Flag_{n-1}(i,k)$ takes on the former value which is nonnegative while if the action is eliminated it takes on the latter value which is negative. Then (32) becomes for $k \in K_i - E_{n-1}(i)$

$$(40) \qquad b_n(i,k) = - Flag_{n-1}(i,k) + (\bar{\alpha}-\alpha_n)[UGu_n^{n-1}(i,k) - Gu_n(i,\delta_n(i))] < 0$$

and (33) or (39) becomes for $k \in F_{\ell,n-1}(i)$

$$(41) \qquad c_n(i,k) = Flag_{n-1}(i,k) + a_{n-1}(i) + (\bar{\alpha}-\alpha_n)[\lambda\overline{Du}_{n-1,1} - \overline{Du}_{n-1,1}(i)] - a_n(i) < 0.$$

In the above procedure we require an array of $a_n(i)$ instead of $B^n v_n(i,k)$. Therefore there will be significant storage saving if there is large action space in each state. Let (I-1') denote this storage saving procedure. Then the changes to be made in the algorithm with procedure (I-1) to incorporate procedure (I-1') are as follows:

In step 1; Initialize $Flag_0(i,k) = B^0v_0(i, \delta_0(i)) - B^0v_0(i,k) \geq 0$ for all $i \in I$ and $k \in K_i$.

In step 2; If $Flag_{n-1}(i,k) \geq 0$ and (40) is satisfied then $Flag_n(i,k) = b_n(i,k) < 0$ and do not calculate $Gu_n(i,k)$. If $Flag_{n-1}(i,k) \geq 0$ and (40) is not satisfied then $Flag_n(i,k) = Flag_{n-1}(i,k)$ and calculate $Gu_n(i,k)$. If $Flag_{n-1}(i,k) > 0$ and (41) is satisfied then $Flag_n(i,k) =$

$c_n(i,k)$ and do not calculate $Gu_n(i,k)$. If $Flag_{n-1}(i,k) > 0$ and (41) is not satisfied then calculate $B^n v_n(i,k)$ and $Gu_n(i,k)$ and set $Flag_n(i,k) = a_n(i) - (\bar{\alpha}-\alpha_n)Gu_n(i,\delta_n(i)) - B^n v_n(i,k) = B^n v_n(i,\delta_n(i)) - B^n v_n(i,k) \geq 0$.

In step 3; A_n is defined by $A_n = \{(i,k)|Gu_n(i,k) - Gu_n(i,\delta_n(i)) > 0$ and $Flag_n(i,k) \geq 0\}$. If $Gu_n(i,k) - Gu_n(i,\delta_n(i)) \leq 0$ for all $i \in I$ and $k \in K_i$ where $Flag_n(i,k) \geq 0$, then δ_n remains α-optimal for all α, $\alpha > \alpha_n$ and stop.

In step 4; (Add the following calculations.) If $Flag_n(i,k) \geq 0$, reset $Flag_n(i,k) = Flag_n(i,k) + (\alpha_{n+1}-\alpha_n)[Gu_n(i,\delta_n(i)) - Gu_n(i,k)] = B^{n+1}v_{n+1}(i,\delta_{n+1}(i)) - B^{n+1}v_{n+1}(i,k) > 0$.

This kind of storage saving procedure may not be used for procedure (I-2) since there, we have to store ℓ, i.e., the last evaluated iteration for $B^\ell v_\ell(i,k)$ for each $k \in F_{\ell,n-1}(i)$.

The following action elimination procedures are based on a tighter upper bound for α_{n+1}, than $\bar{\alpha}$.

<u>Action Elimination Procedure (II)</u>; $\alpha'_{n+1} = \bar{\alpha}_{n+1}$.

Here all the results are exactly the same as in procedure (I) except we use $\bar{\alpha}_{n+1}$ instead of $\bar{\alpha}$. $\bar{\alpha}_{n+1}$ satisfying $\alpha_{n+1} \leq \bar{\alpha}_{n+1} \leq \bar{\alpha}$ can be calculated at the beginning of step 2 of iteration n using the following method.

First find (i',k') at the end of step 3 of <u>iteration n-1</u> so that

$$Gu_{n-1}(i',k') - Gu_{n-1}(i',\delta_{n-1}(i')) = \underset{\substack{i\varepsilon\bar{I}_{n-1} \\ k\varepsilon\bar{K}_i^{n-1}}}{\text{Max}} \{Gu_{n-1}(i,k)-Gu_{n-1}(i,\delta_{n-1}(i))\}$$

where

$$\bar{A}_{n-1} \equiv \{(i,k)|Gu_{n-1}(i,k) > Gu_{n-1}(i,\delta_n(i)) \text{ and } Flag_{n-1}(i,k) < 0\},$$

$$\alpha' = \underset{(i,k)\varepsilon\bar{A}_{n-1}}{\text{Min}} \{R_{n-1}(i,k)\},$$

$$\bar{I}_{n-1} = \{i \mid R_{n-1}(i,k) = \alpha' \,/\, (i,k) \in \bar{A}_{n-1}\}$$

and $$\bar{K}_i^{n-1} = \{k \in K_i \mid R_{n-1}(i,k) = \alpha' \,/\, (i,k) \in \bar{A}_{n-1}\} \text{ for } i \in \bar{I}_{n-1}.$$

Note $R_{n-1}(i,k)$ is defined in Section 4.

Then at the beginning of step 2 calculate $\bar{\alpha}_{n+1}$ as follows; define

$$\tilde{\alpha}_{n+1} = \begin{cases} \alpha_n + R_n(i',k') & \text{if } Gu_n(i',k') > Gu_n(i,\delta_n(i)) \\ \bar{\alpha} & \text{otherwise} \end{cases}$$

and set $\bar{\alpha}_{n+1} = \min(\bar{\alpha}, \tilde{\alpha}_{n+1})$.

Then $\bar{\alpha}_{n+1} \geq \alpha_{n+1}$ because $\alpha_{n+1} = \alpha_n + \underset{(i,k)\varepsilon A_n}{\text{Min}} \{R_n(i,k)\}$ and if $Gu_n(i',k') >$ $Gu_n(i',\delta_n(i'))$ then $(i',k') \in A_n$.

Denote by (II-1) and (II-2) the above procedure with $UGu_p^s(i,k)$ of (26) and (27) respectively. Then all the algorithms with procedure (II-1) and (II-2) are exactly the same as the algorithms with procedure (I-1) and (I-2) except we replace $\bar{\alpha}$ by $\bar{\alpha}_{n+1}$. All of the above results concerning storage saving also apply.

7. Application of Parametric Markov Decision Problems.

The most simple application of a MDP with one parameter is sensitivity analysis of the one-stage reward when it is in the form, $r_\alpha^\delta = r^\delta + \alpha d^\delta$. Another application is MDP with two criteria. The objective of the discounted MDP with vector criterion is to determine strategies with nondominated expected total discounted rewards. In this setting v(i) is an S-vector of $\{v_1(i), v_2(i), \cdots, v_S(i)\}$ where S is the number of criteria, i.e., v is a N x S matrix where N is the number of states. Then the problem is to find the set of $\pi \in \Pi$, the nondominated set, such that there does not exist a $\rho \in \Pi$ with the properties: $v^\rho \geq v^\pi$ and for some $i \in I$, $v^\rho(i) \neq v^\pi(i)$. Henig [13] has shown that it is sufficient to restrict our attention to stationary strategies in determining the nondominated v^π since any nondominated v^π generated by a nonstationary strategy will be contained in the convex hull of the set of all nondominated v^δ. It has been shown in Henig [13] and Viswanathan, Aggarwal and Nair [33] that the set of all $\delta \in \Delta$ which generate nondominated v^δ can be determined by considering the following scalar criterion problem.

$$(42) \qquad v_a^* = v_a^{\delta_a^*} = \underset{\delta\varepsilon\Delta}{\text{Max}} \{v^\delta \cdot a\}$$

Where $v^\delta \cdot a$ is an inner product of v^δ and a, a is a column vector of $(a_1, a_2, \cdots, a_S)$, $a_s \geq 0$ for $s = 1,2,\cdots,S$ and $\sum_{s=1}^{S} a_s = 1$.

For the MDP with two criteria the problem can be written as follows:

$$(43) \qquad v_a^* = v_a^{\delta_a^*} = \underset{\delta\varepsilon\Delta}{\text{Max}} \{a_1 v_1^\delta + a_2 v_2^\delta\}$$

where $a_1 + a_2 = 1$, $a_1 \geq 0$, $a_2 \geq 0$ and $a = (a_1, a_2)^t$.

Setting $a_1 = 1 - a_2$, (43) becomes

$$(44) \qquad v_{a_2}^* = v_{a_2}^{\delta_{a_2}^*} = \underset{\delta\varepsilon\Delta}{\text{Max}} \{v_1^\delta + a_2(v_2^\delta - v_1^\delta)\}$$

where $0 \leq a_2 \leq 1$.

Then setting $\alpha = a_2$, $u^\delta = v_2^\delta - v_1^\delta$ and $v^\delta = v_1^\delta$ the problem (44) is exactly the same as MDP with one parameter.

Another interesting application of an MDP with one parameter is a problem with one constraint. From now on we extend the range of decisions to include randomized (or mixed) strategies. Let Π denote the set of all randomized strategies. Nonrandomized stationary strategies will be called pure strategies. The problem to be considered here is to find a $\pi^* \varepsilon \Pi$ such that

(45) $$av^{\pi^*} = \underset{\pi\varepsilon\Pi}{\text{Max}}\ av^{\pi}$$

$$\text{s.t.} \qquad au^{\pi} \geq b$$

where $a(i)$ is the probability that the system is in state $i\varepsilon I$ at stage 0 with $\sum_{I\varepsilon I} a(i) = 1$ and $a(i) \geq 0$ for each $i\varepsilon I$, b is a scalar and v^{π} and u^{π} are the expected total discounted reward vectors corresponding to one stage reward vectors r and d respectively for strategy $\pi\varepsilon\Pi$. Note the constraint $a\tilde{u}^{\pi} \leq \tilde{b}$ can be transformed to $au^{\pi} \geq b$ by setting $u^{\pi} = -\tilde{u}^{\pi}$ and $b = -\tilde{b}$.

To solve the above problem we introduce the following problem:

(46) $$av_{\alpha}^{\pi_{\alpha}^*} = \underset{\pi\varepsilon\Pi}{\text{Max}}\ av_{\alpha}^{\pi}$$

where $v_{\alpha}^{\pi} = v^{\pi} + \alpha u^{\pi}$ and $\alpha \geq 0$. Then we have the following lemma describing the relationship of these two problems above.

<u>Lemma 3.7.</u>

Suppose π_{α}^* is optimal in the problem (46) for some α and $au^{\pi_{\alpha}^*} = b$. Then π_{α}^* is optimal in the problem (45).

<u>Proof.</u> Suppose π_{α}^* were not optimal in the problem (45), then there would exist a strategy π' such that $av^{\pi'} > av^{\pi_{\alpha}^*}$ and $au^{\pi'} \geq b = au^{\pi_{\alpha}^*}$. Then $av^{\pi'} + \alpha au^{\pi'} > av^{\pi_{\alpha}^*} + \alpha au^{\pi_{\alpha}^*}$. This contradicts the fact that π_{α}^* is optimal in the problem (46). Hence π_{α}^* is optimal in the problem (45).

The optimal strategy π_α^* of (45) given α will have an optimal pure strategy independent of a since the problem (46) given α is an ordinary discounted MDP. Therefore the problem reduces to that of finding δ_α^* such that

$$(47) \qquad v_\alpha^{\delta_\alpha^*} = \underset{\delta\varepsilon\Delta}{\text{Max}}\ v_\alpha^\delta$$

where $v_\alpha^\delta = v^\delta + \alpha u^\delta$ and $\alpha \geq 0$. Recall Δ is the set of nonrandomized policies.

The problem (47) can be solved using the algorithm developed in Section 3 with $\underline{\alpha} = 0$ over the region of α. But an upper bound for α, i.e. $\bar{\alpha}$, is not given and so the region of α may be unbounded. We will show the boundedness of α in Proposition 3.5 and the stopping rules of the algorithm that result in an optimal strategy to the problem (45). We will use the notation of Section 3 for α_n, δ_n, v_n and u_n where $v_n = v^{\delta_n} + \alpha_n u^{\delta_n}$ and $u_n = u^{\delta_n}$. The following proposition motivates the stopping rules of the algorithm for solving the problem (45).

<u>Proposition 3.4</u>.

At iteration n+1, $u^{\delta_{n+1}} = u_{n+1} > u_n = u^{\delta_n}$ and if $\alpha_{n+1} > 0$, $v^{\delta_n} > v^{\delta_{n+1}}$.

<u>Proof</u>. From Proposition 3.1 at α_{n+1} and the selection rule (A)
$u^{\delta_{n+1}} = u_{n+1} = u_n + (I - \lambda P_{\delta_{n+1}})^{-1}\, G_{\delta_{n+1}}\, u_n > u_n = u^{\delta_n}$.

From Corollary 3.1

$$v^{\delta_{n+1}} + \alpha_{n+1} u^{\delta_{n+1}} = v_{\alpha_{n+1}}^{\delta_{n+1}} = v_{\alpha_{n+1}}^{\delta_n} = v^{\delta_n} + \alpha_{n+1} u^{\delta_n}.$$

Therefore $v^{\delta_{n+1}} < v^{\delta_n}$ since $u^{\delta_{n+1}} > u^{\delta_n}$ and $\alpha_{n+1} > 0$.

The following proposition implies that the region of α to be considered is actually bounded.

Proposition 3.5.

There exists α_N such that for all $\alpha \geq \alpha_N$, δ_N is optimal in the problem (47) and δ_N is also optimal for the problem of $\underset{\delta\varepsilon\Delta}{\text{Max}}\ u^{\delta}$.

Proof. The fact that $u_{n+1} > u_n$ in Proposition 3.4 and the finiteness of policy space implies that there exists α_N such that for all $\alpha \geq \alpha_N$, δ_N is optimal in the problem (47).

Suppose δ_N is not optimal for the problem of maximizing u^{δ}. Then choose δ' such that $u^{\delta'} = \underset{\delta\varepsilon\Delta}{\text{Max}}\ u^{\delta}$. There exists $i'\varepsilon I$ such that $u^{\delta'}(i') > u^{\delta_N}(i')$ since $u^{\delta'} \geq u^{\delta_N}$ and $u^{\delta'} \neq u^{\delta_N}$. Then there exists α' such that for all $\alpha > \alpha' \geq \alpha_N$, $\alpha(u^{\delta'}(i') - u^{\delta_N}(i')) > v^{\delta_N}(i') - v^{\delta'}(i')$, i.e., $v^{\delta'}(i') + \alpha u^{\delta'}(i') > v^{\delta_N}(i') + \alpha u^{\delta_N}(i')$. This contradicts the fact that for all $\alpha \geq \alpha_N$, δ_N is optimal in the problem (47). Hence δ_N is optimal for the problem of $\underset{\delta\varepsilon\Delta}{\text{Max}}\ u^{\delta}$.

Therefore using the algorithm in Section 3 δ_α^* can be found for each α, $0 \leq \alpha \leq \alpha_N$ where α_N will be determined in the algorithm. The following theorems indicate how to modify the algorithm of Section 3 to find the optimal strategy, π^*, of the problem (45).

Theorem 3.4.

Suppose $au_0 \geq b$. Then $\pi^* = (\delta_0, \delta_0, \cdots)$ is an optimal pure strategy to the problem (45).

Proof. Since $\alpha_0 = \underline{\alpha} = 0$, δ_0 is optimal in the problem of $\underset{\pi\varepsilon\Pi}{\text{Max}}\ av^\pi$ without the constraint and δ_0 satisfies the constraint, i.e., $au^{\delta_0} = au_0 \geq b$. Therefore δ_0 is optimal in the problem (45).

Theorem 3.5.

i) Suppose $au_0 < b$ and $au_n = b$ for $n \geq 1$. Then $\pi^* = (\delta_n, \delta_n, \cdots)$ is an optimal pure strategy to the problem (45).

ii) Suppose $au_0 < b$ and $au_n < b < au_{n+1}$ for $n \geq 0$. Then the strategy, π^*, of using $(\delta_n, \delta_n, \cdots)$ with probability p^* and $(\delta_{n+1}, \delta_{n+1}, \cdots)$ with probability $(1-p^*)$ at stage 0 where $p^* = (au_{n+1}-b)/(au_{n+1}-au_n)$ is optimal in the problem (45).

Proof. First observe in (ii) $au^{\pi^*} = p^* au_n + (1-p^*)au_{n+1} = [(au_{n+1}-b)/(au_{n+1} - au_n)]au_n + [(b - au_n)/(au_{n+1} - au_n)]au_{n+1} = b$. And in (ii) π^* is also optimal in the problem (46) for $\alpha = \alpha_{n+1}$ since δ_n and δ_{n+1} are both optimal in the problem (46) at α_{n+1}. Therefore for (i) and (ii) $au^{\pi^*} = b$ and π^* is optimal in the problem (46). Hence by Lemma 3.7 π^* is optimal in the problem (45).

Theorem 3.6.

Suppose $au_N < b$. Then there is no feasible strategy to the problem (45).

Proof. Note $au_N = au^{\delta_N}$ and δ_N is optimal at α_N. Then by Proposition 3.5 δ_N is optimal for the problem of $\underset{\delta\varepsilon\Delta}{\text{Max}}\ u^{\delta}$, i.e., $(\delta_N, \delta_N, \cdots)$ is an optimal strategy for the problem of $\underset{\pi\varepsilon\Pi}{\text{Max}}\ u^{\pi}$. Because $au^{\delta_N} < b$ there is no feasible strategy.

Note (ii) of Theorem 3.5 leads to choosing an infeasible strategy with probability p*. To resolve this problem we can consider two alternatives — one of using a stationary randomized strategy and the other of finding the best pure strategy.

Proposition 3.6.

Suppose $au_o < b$ and $au_n < b < au_{n+1}$ for $n \geq 0$. Then there exists an optimal stationary randomized strategy.

Proof. Assuming $\alpha_{n+1} > 0$ [otherwise Theorem 3.4 holds at $\alpha_{n+1} = 0$ and we have an optimal pure strategy] $v^{\delta_n} > v^{\delta_{n+1}}$ by Proposition 3.4. Let π^* be the optimal strategy defined in (ii) of Theorem 3.5. Then there exists $\hat{p}$ such that $v^{\pi^*} = (I - \lambda\hat{P})^{-1}\hat{r}$ where $\hat{P} = \hat{p}P_{\delta_n} + (1-\hat{p})P_{\delta_{n+1}}$ and $\hat{r} = \hat{p}r^{\delta_n} + (1-\hat{p})r^{\delta_{n+1}}$ because $(I-\lambda\hat{P})^{-1}\hat{r}$ is continuous in $\hat{p}$ with $(I-\lambda\hat{P})^{-1}\hat{r} = v^{\delta_n}$ at $\hat{p} = 1$ and $(I-\lambda\hat{P})^{-1}\hat{r} = v^{\delta_{n+1}}$ at $\hat{p} = 0$ and $v^{\delta_n} > v^{\pi^*} > v^{\delta_{n+1}}$. I.e., there exists an optimal stationary randomized strategy.

Theorem 3.7.

Suppose $au_0 < b$ and $au_n < b < au_{n+1}$ for $n \geq 0$. Then $(\delta_{n+1}, \delta_{n+1}, \cdots)$ is an optimal pure strategy among all pure strategies feasible to the problem (45).

Proof. Suppose there were a better pure feasible strategy, say $(\delta', \delta', \cdots)$, than $(\delta_{n+1}, \delta_{n+1}, \cdots)$, i.e., $av^{\delta'} > av^{\delta_{n+1}}$ and $au^{\delta'} \geq b$. Setting $p' = (au^{\delta'} - b)/(au^{\delta'} - au_n)$ we have $p'au_n + (1-p')au^{\delta'} = b$.

If $p' \geq p^*$ then $p'av^{\delta_n} + (1-p')av^{\delta'} > p'av^{\delta_n} + (1-p')av^{\delta_{n+1}} \geq p^*av^{\delta_n} + (1-p^*)av^{\delta_{n+1}}$. The last inequality follows from Proposition 3.4. Therefore by denoting π' as a strategy using $(\delta_n, \delta_n \cdots)$ with probability p' and $(\delta', \delta', \cdots)$ with probability $(1-p')$ at stage 0, $av^{\pi'} = p'av^{\delta_n} + (1-p')av^{\delta'} > p^*av^{\delta_n} + (1-p^*)av^{\delta_{n+1}} = av^{\pi^*}$ and $au^{\pi'} = p'au_n + (1-p')au^{\delta'} = b$. Thus π' is a better strategy than π^* contradicting Theorem 3.4.

If $p' < p^*$ then $p'av^{\delta_n} + (1-p')av^{\delta'} > p'av^{\delta_n} + (1-p')av^{\delta_{n+1}}$ and $p'au_n + (1-p')au^{\delta'} = b = p^*au_n + (1-p^*)au_{n+1} > p'au_n + (1-p')au_{n+1}$ because $u_{n+1} > u_n$ from Proposition 3.4. and $p' < p^*$. Therefore $p'av^{\delta_n} + (1-p')av^{\delta'} + \alpha_{n+1}[p'au_n + (1-p')au^{\delta'}] > p'av^{\delta_n} + (1-p')av^{\delta_{n+1}} + \alpha_{n+1}[p'au_n + (1-p')au_{n+1}]$. I.e., $p'[av^{\delta_n} + \alpha_{n+1}au_n] + (1-p')[av^{\delta'} + \alpha_{n+1}au^{\delta'}] > p'[av^{\delta_n} + \alpha_{n+1}au_n] + (1-p')[av^{\delta_{n+1}} + \alpha_{n+1}au_{n+1}]$. Hence $av^{\delta'} + \alpha_{n+1}au^{\delta'} > av^{\delta_{n+1}} + \alpha_{n+1}au_{n+1}$ since $p' < 1$ by the assumption of $au_n < b$. This contradicts the fact that δ_{n+1} is optimal in the problem (47). Therefore $(\delta_{n+1}, \delta_{n+1} \cdots)$ is an optimal pure strategy among all pure strategies feasible to the problem (45).

The stopping rules to be changed or added in the algorithm of Section 3 are as follows:

(a) At the end of step 1; If $au_o \geq b$, stop. Then δ_o is optimal in the problem (45).

(b) At the beginning of step 3; If $Bu_n(i,k) \leq 0$ for all $i \varepsilon I$ and $k \varepsilon K_i$ and $au_n < b$, stop. There is no feasible strategy to the problem (45). If $Bu_n(i,k) \leq 0$ for all $i \varepsilon I$ and $k \varepsilon K_i$ and $au_n \geq b$, stop. Then π^* defined in Theorem 3.5 is optimal.

(c) At the beginning of step 4; If $au_n \geq 0$ [instead of "if $\alpha_{n+1} \geq \bar{\alpha}$"], stop. Then π^* defined in Theorem 3.5 is optimal.

Furthermore, α_0 in Theorem 3.4, α_n in (i) of Theorem 3.5 and α_{n+1} in (ii) of Theorem 3.5 give the shadow price of the constraint in the problem (45).

Another application of MDP with one parameter is the MDP with one ratio constraint. Solution procedures for MDP with a ratio criterion were studied by Derman [8] for the undiscounted case and by Aggarwal, Chandrasekaran and Nair [1] and Brosh, Schlifer and Schweitzer [4] for the discounted case. Here we consider the following ratio constrained MDP;

(48) $$\max_{\pi \in \Pi} a\tilde{v}^{\pi} - a\tilde{u}^{\pi}$$

$$\text{s.t.} \quad \frac{a\tilde{v}^{\pi}}{a\tilde{u}^{\pi}} \geq R$$

where $\tilde{v}^{\pi}(i)$, $-\infty < \tilde{v}^{\pi}(i) < +\infty$ and $\tilde{u}^{\pi}(i)$, $0 < \tilde{u}^{\pi}(i) < +\infty$ are the expected total discounted returns and costs respectively in state i if we take a strategy π. This problem can be rewritten as

(49) $$\max_{\pi \in \Pi} a\tilde{v}^{\pi} - a\tilde{u}^{\pi}$$

$$\text{s.t.} \qquad a\tilde{v}^{\pi} - R\, a\tilde{u}^{\pi} \geq 0.$$

Then setting $v^{\pi} = \tilde{v}^{\pi} - \tilde{u}^{\pi}$, $u^{\pi} = \tilde{v}^{\pi} - R\tilde{u}^{\pi}$ and $b = 0$, the problem (49) is exactly the same as the problem (45). All the results for the problem (45) hold for this problem.

8. Computational Results.

In this section we discuss the results of applying the algorithms developed in Section 4 to a two-criterion version of Howard's [14] automobile replacement problem. Included are a comparison of the algorithms under the various selection rules and action elimination procedures.

Let r_1^δ and r_2^δ be the expected one-stage rewards corresponding to v_1^δ and v_2^δ in (43) and defined as follows:

$r_1(i,k) = r(i,k)$ defined for the automobile replacement problem in Section 8 of Chapter 2 and

$$r_2(i,k) = \begin{cases} \dfrac{400}{\sqrt{k-1}} & \text{if } k \neq 1 \\[2ex] \dfrac{400}{\sqrt{i+1}} & \text{if } k = 1 \end{cases}$$

where $r_2(i,k)$ is an additive expected one-stage utility function for time preference if we take action $k \varepsilon K_i$ in state $i \varepsilon I$. Therefore if we keep the present car of age i (k=1) then the time preference utility for one period will be $400/\sqrt{i+1}$. If we trade it in for a car of age k-2 the one-period time preference utility will be $400/\sqrt{k-1}$.

Resetting $r(i,k) = r_1(i,k)$ and $d(i,k) = r_2(i,k) - r_1(i,k)$ the nondominated strategies of the automobile replacement problem with two criteria can be generated by solving the following problem;

$$\max_{\delta\varepsilon\Delta} \{v^{\delta} + \alpha u^{\delta}\}$$

$$0 \leq \alpha \leq 1,$$

where v^{δ} and u^{δ} are the expected total discounted reward corresponding to r^{δ} and d^{δ} respectively. In the notation of this chapter this means we take $\underline{\alpha} = 0$ and $\bar{\alpha} = 1$.

We solved this problem with $\lambda = .96$ and $\lambda = .97$, $(1-\lambda)\varepsilon_0 = 0{,}001$ and $(1-\lambda)\varepsilon' = 0.001$ using the algorithm with and without action elimination procedures. All calculations were performed on the University of British Columbia AMDAHL 470 computer using the codes in the Appendix. For the algorithms without action elimination we used the selection rule (B) and the other selection rule (denoted from now on as (C)) of changing only a single action as described in Section 4. The selection rule (B) leads to block pivoting while the other selection rule leads to simplex pivoting. Block pivoting always gives a value of u_{n+1} greater than or equal to that of simplex pivoting, which implies that block pivoting will generate optimal policies over the region of α in fewer iterations than simplex pivoting. The initial optimal actions, i.e. at $\alpha = 0$, with $\lambda = 0.96$ are $k = 18$ (trade in for the car of age 16, i.e., the 4 year-old car) for i from 1 to 7 and from 28 to 40 and $k = 1$ (keep the current car) for i from 8 (2 year-old car) to 27 (6 3/4 year-old car). The action $k = 2$ (trade in for a new car) is optimal in all states for $\alpha \geq 0.782133444$ and $\lambda = 0.97$ are $k = 14$ (trade in for thecar of age 12, i.e., the 3 year-old car) for i from 1 to 3 and from 27 to 40 and $k = 1$ for i from 4 (1 year-old car) to 26 (6 1/2 year-old car). All optimal actions for $\alpha \geq 0.781641042$ are $k = 2$ in all states. In between optimal actions for $\lambda = 0.96$ and $\lambda = 0.97$ are presented in the Appendix.

The CPU times required after the initialization and the number of iterations (r in the algorithm of Section 4) required are shown in Table 3.1 for the algorithm with the selection rule (B) and (C) and without action elimination. Clearly the algorithm using the rule (B) is much faster than the rule (C). This is because for $\lambda = .97$, 17 actions changed together at the one particular value of α and for $\lambda = .96$ there were values of α at which 17 actions and 16 actions changed together. Thus for the algorithms with action elimination procedures we used only the selection rule (B). The CPU times required after the initialization and the average fraction of actions eliminated for the algorithm with each action elimination procedure are shown in Table 3.2. Reduction of CPU times using action elimination procedure (II-1) and (II-2) were significant. For instance, with $\lambda = .96$ the algorithm with the action elimination procedure (II-2) took only 47.81 per cent of the CPU time taken by the algorithm with the rule (B) and without action elimination and only 28.78 per cent of the CPU time taken by the algorithm only with the rule (C).

9. Conclusions and Extensions.

The algorithm developed in Section 4 can be implemented to solve a problem with a large state and action space. As shown by the results in Table 3.1 use of an algorithm with the selection rule (B) is recommended. Even though we used codes based on policy iteration, similar codes based

Table 3.1

Comparison of Selection Rules
[CPU Times (Secs.)/Number of Iterations Required]

(λ) Discount Rate	Algorithm with the Selection Rule (B)	Algorithm with the Selection Rule (C)
0.96	8.623/37	14.326/68
0.97	7.121/31	9.875/47

Table 3.2

Comparison of Action Elimination Procedures
[CPU Times (Secs.)/Average Fraction of Actions Eliminated]

(λ) Discount Rate	Procedure (I-1)	Procedure (I-2)	Procedure (II-1)	Procedure (II-2)
0.96	7.158/.2939	6.104/.4553	4.792/.6681	4.123/.7778
0.97	5.829/.3193	4.645/.5065	4.530/.5925	3.880/7127

on modified policy iteration can be developed and will be more efficient for problems with very large state and action spaces. Results based on using the action elimination procedure (II-1) and (II-2) are also very encouraging and we recommend incorporating them in an algorithm.

An interesting extension would be to develop an algorithm for the problem with a vector of parameters. This would have application to more than two criterion MDP and MDP with several constraints. The methods developed here would be applicable but modifications to efficieintly search the parameter set would be required.

REFERENCES

1. Aggarwal, V., Chandrasekaran, R. and Nair, K.P.K., "Markov Ratio Decision Processes", JOTA 21, 1977, 27-37.

2. Blackwell, D., "Discrete Dynamic Programming", Ann. Math. Statist. 33, 1962, 719-726.

3. Blackwell, D., "Discounted Dynamic Programming", Ann. Math. Statist. 36, 1965, 226-235.

4. Brosh, I., Schlifer, E., Schweitzer, P.J., "Generalized Markovian Decision Processes", Zeitschrift für Operations Research 21, 1977, 173-186.

5. Dantzig, G.B., Linear Programming and Extensions, Princeton Univ. Press, Princeton, New Jersey, 1963.

6. Denardo, E., "Contraction Mappings in the Theory Underlying Dynamic Programming", SIAM Rev. 9, 1967, 165-177.

7. D'Epenoux, F., "Sur un Problème de Production et de Stockage dans l'Aetoire", Revue Franç. Rech. Opérationelle 14, 1960, 3-16; (translation in Man. Sci. 10, 1963, 98-108).

8. Derman, C., "On Sequential Decision and Markov Chains", Man. Sci. 9, 1962, 16-24.

9. Grinold, R., "Elimination of Suboptimal Actions in Markov Decision Problems", Oper. Res. 21, 1973, 848-85.

10. Hastings, N.A.J. and Mello, J.M.C., "Tests for Suboptimal Actions in Discounted Markov Programming", Man. Sci. 19, 1973, 1019-1022.

11. Hastings, N.A.J., "A Test for Nonoptimal Actions in Undiscounted Finite Markov Decision Chains", Man. Sci. 23, 1976, 87-92.

12. Hastings, N.A.J., and van Nunen, J.A.E.E., "The Action Elimination Algorithm for Markov Decision Processes", in Markov Decision Theory, Eds. H.C. Tijms and J. Wessels, Mathematical Centre Tract 93, Amsterdam, 1977.

13. Henig, M.I., "Multicriteria Dynamic Programming", Ph.D. Thesis, Yale Univ., New Haven, Conn., 1978.

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

15. Hübner, G., "Improved Procedures for Eliminating Suboptimal Actions in Markov Programming by the Use of Contraction Properties", Transactions of the Seventh Prague Conference, Volume A, 1977, 257-263.

16. MacQueen, J., "A Test for Suboptimal Actions in Markov Decision Problems", Oper. Res. 15, 1967, 559-561.

17. Mine, H. and Osaki, S., Markovian Decision Process, Amer. Elsevier Pub. Co., Inc., New York, 1970.

18. Morton, T., "Undiscounted Markov Renewal Programming via Modified Successive Approximations", Oper. Res. 19, 1971, 1081-1089.

19. Porteus, E., "Some Bounds for Discounted Sequential Decision Processes", Man. Sci. 18, 1971, 7-11

20. Porteus, E., "Bounds and Transformations for Finite Markov Decision Chains", Oper. Res. 23, 1975, 761-784.

21. Porteus, E., "Improved Iterative Computation of the Expected Discounted Return in a Semi-Markov Chain", Working Paper, Stanford University, 1978.

22. Porteus, E., "Overview of Iterative Methods for Discounted Finite Markov and Semi-Markov Decision Chains", to appear in: D. White Ed.: Pro. of Intl. Conf. on Markov Decision Processes, Manchester, 1979.

23. Porteus, E. and Toten, J., "Accelerated Computation of the Expected Discounted Return in a Markov Chain", Oper. Res. 26, 1978, 350-358.

24. Puterman, M.L. and Brumelle, S.L., "The Analytic Theory of Policy Iteration", in Dynamic Programming and Its Applications, Pro. of Intl. Conf. on Dynamic Programming, Ed., M.L. Puterman, Academic Press, New York, 1978.

25. Puterman, M.L. and Brumelle, S.L., "On the Convergence of Policy Iteration in Stationary Dynamic Programming", Math. of Oper. Res. 4, 1979, 60-69.

26. Puterman, M.L. and Shin, M.C., "Modified Policy Iteration Algorithms for Discounted Markov Decision Problems", Man. Sci. 24, 1978, 1127-1137.

27. Rothblum, U., "Iterated Successive Approximation for Sequential Decision Processes", Working Paper, Yale Univ., New Haven, 1979.

28. Smallwood, R.D. and Sondik, E.J., "The Optimal Control of Partially Observable Markov Processes over a Finite Horizon", Oper. Res. 21, 1973, 1300-1322.

29. Sondik, E.J., "The Optimal Control of Partially Observable Markov Processes over the Infinite Horizon: Discounted Costs", Oper. Res. 26, 1978, 282-304.

30. van der Wal, J., "Discounted Markov Games: Generalized Policy Iteration Method", JOTA 25, 1978, 125-138.

31. van Nunen, J.A.E.E., "A Set of Successive Approximation Methods for Discounted Markovian Decision Problems", Zeitschrift für Oper. Res. 20, 1976, 203-208.

32. van Nunen, J.A.E.E., Contracting Markov Decision Processes, Mathematical Center Tract 71, Amsterdam, 1976.

33. Viswanathan, B., Aggarwal, V.V. and Nair, K.P.K., "Multiple Criteria Markov Decision Processes", North Holland TIMS Studies in Management Science, 6, 1977, 263-272.

34. White, C.C. and Kim, K.W., "Two Solution Procedures for Solving Vector Criterion Markov Decision Processes", Working Paper, Univ. of Virginia, Charlottesville, Virginia, 1979.

35. White, D.J., "Elimination of Non-optimal Actions in Markov Decision Processes", in Dynamic Programming and Its Applications, Pro. of Intl. Conf. on Dynamic Programming, Ed., M.L. Puterman, Academic Press, New York, 1978.

APPENDIX

```
C***
C*** A) CODE FOR POLICY ITERATION WITHOUT ACTION ELIMINATION
C***
      DIMENSION V(40),C(40,100),CPV(40,100),P(40,100,40),IIPOL(40),
     1A(40,40),T(40,40),B(40,1),IPERM(100),KKPOL(40)
      REWIND8
      REWIND9
      NDIMA=40
      NDIMBX=40
      NDIMT=40
      NSOL=1
      READ(5,1) NST,NAC,DISCR,DERR
    1 FORMAT(I5,I5,2F10.5)
      READ(8)((C(I,K),I=1,NST),K=1,NAC)
      DO 2 I=1,NST
      READ(9)((P(I,K,J),K=1,NAC),J=1,NST)
    2 CONTINUE
      CALL TIME(0)
      ITER=0
      ELM=0.
      TNM=NST*NAC
      KN=0
      DO 50 I=1,NST
      DO 20 K=1,NAC
      IF(K.EQ.1) GO TO 10
      IF(OCI.GE.C(I,K)) GO TO 20
   10 OCI=C(I,K)
      IIP=K
   20 CONTINUE
      B(I,1)=OCI
      KKPOL(I)=IIP
      DO 40 J=1,NST
      IF(I.EQ.J) GO TO 30
      A(I,J)=-DISCR*P(I,IIP,J)
      GO TO 40
   30 A(I,J)=1.-DISCR*P(I,IIP,J)
   40 CONTINUE
   50 CONTINUE
      GO TO 130
   60 ITER=ITER+1
      DO 120 I=1,NST
      DO 110 K=1,NAC
      IF(FLAG(I,K).GE.1.0) GO TO 110
      KN=KN+1
      PV=0.
      DO 100 J=1,NST
      PV=PV+P(I,K,J)*V(J)
  100 CONTINUE
      CPV(I,K)=C(I,K)+DISCR*PV
      IF(KN.EQ.1) GO TO 105
      IF(CVI.GE.CPV(I,K)) GO TO 110
  105 CVI=CPV(I,K)
      IIP=K
  110 CONTINUE
      KN=0
      DO 115 J=1,NST
      IF(I.EQ.J) GO TO 113
      A(I,J)=-DISCR*P(I,IIP,J)
      GO TO 115
  113 A(I,J)=1.-DISCR*P(I,IIP,J)
```

```
  115 CONTINUE
      B(I,1)=C(I,IIP)
      BVI=CVI-V(I)
      IIPOL(I)=IIP
      IF(I.EQ.1) GO TO 116
      IF(BVMAX.GE.BVI) GO TO 117
      BVMAX=BVI
      GO TO 120
  116 BVMAX=BVI
      BVMIN=BVI
      GO TO 120
  117 IF(BVMIN.LE.BVI) GO TO 120
      BVMIN=BVI
  120 CONTINUE
      DO 125 I=1,NST
      IF(IIPOL(I).NE.KKPOL(I)) GO TO 127
  125 CONTINUE
      GO TO 200
  127 DO 129 I=1,NST
      KKPOL(I)=IIPOL(I)
  129 CONTINUE
C***     SOLVE A SYSTEM OF EQUATIONS A*V=B BY CALLING SUBROUTINE FSLE
  130 CALL FSLE(NST,NDIMA,A,NSOL,NDIMBX,B,V,IPERM,NDIMT,T,DET,JEXP)
      IF(DET)135,175,135
  135 GO TO 60
  175 WRITE(6,180)
  180 FORMAT('SOLUTION FAILED')
  200 CALL TIME(1,1)
      WRITE (6,210)
  210 FORMAT (10X,'STATE',10X,'POLICY',15X,'RETURN',10X,'OPT. POLICY')
      WRITE (6,250) (I,NAC,V(I),IIPOL(I),I=1,NST)
  250 FORMAT (10X,I3,12X,I3,10X,E16.7,10X,I3/)
      WRITE (6,260)
  260 FORMAT(10X,'DISCR',10X,'DERR',10X,'ITERATION')
      WRITE (6,300) DISCR,DERR,ITER
  300 FORMAT(10X,F5.3,8X,F8.5,8X,I5)
      STOP
      END
```

```
C***
C*** B) CODE FOR POLICY ITERATION WITH PROCEDURE (I)
C***
      DIMENSION V(40),C(40,100),CPV(40,100),P(40,100,40),IIPOL(40),
     1A(40,40),T(40,40),B(40,1),IPERM(100),KKPOL(40),FLAG(40,100),
     2FV(40)
      REWIND8
      REWIND9
      NDIMA=40
      NDIMBX=40
      NDIMT=40
      NSOL=1
      READ(5,1) NST,NAC,DISCR,DERR
    1 FORMAT(I5,I5,2F10.5)
      READ(8)((C(I,K),I=1,NST),K=1,NAC)
      DO 2 I=1,NST
      READ(9)((P(I,K,J),K=1,NAC),J=1,NST)
    2 CONTINUE
      CALL TIME(0)
      ITER=0
      ELM=0.
      TNM=NST*NAC
      KN=0
      DO 50 I=1,NST
      DO 20 K=1,NAC
      FLAG(I,K)=0.
      IF(K.EQ.1) GO TO 10
      IF(OCI.GE.C(I,K)) GO TO 20
   10 OCI=C(I,K)
      IIP=K
   20 CONTINUE
      B(I,1)=OCI
      KKPOL(I)=IIP
      DO 40 J=1,NST
      IF(I.EQ.J) GO TO 30
      A(I,J)=-DISCR*P(I,IIP,J)
      GO TO 40
   30 A(I,J)=1.-DISCR*P(I,IIP,J)
   40 CONTINUE
   50 CONTINUE
      GO TO 130
   60 ITER=ITER+1
      DO 120 I=1,NST
      DO 110 K=1,NAC
      IF(FLAG(I,K).GE.1.0) GO TO 110
      KN=KN+1
      PV=0.
      DO 100 J=1,NST
      PV=PV+P(I,K,J)*V(J)
  100 CONTINUE
      CPV(I,K)=C(I,K)+DISCR*PV
      IF(KN.EQ.1) GO TO 105
      IF(CVI.GE.CPV(I,K)) GO TO 110
  105 CVI=CPV(I,K)
      IIP=K
  110 CONTINUE
      KN=0
      DO 115 J=1,NST
      IF(I.EQ.J) GO TO 113
      A(I,J)=-DISCR*P(I,IIP,J)
```

```
      GO TO 115
  113 A(I,J)=1.-DISCR*P(I,IIP,J)
  115 CONTINUE
      B(I,1)=C(I,IIP)
      BVI=CVI-V(I)
      FV(I)=V(I)
      IIPOL(I)=IIP
      IF(I.EQ.1) GO TO 116
      IF(BVMAX.GE.BVI) GO TO 117
      BVMAX=BVI
      GO TO 120
  116 BVMAX=BVI
      BVMIN=BVI
      GO TO 120
  117 IF(BVMIN.LE.BVI) GO TO 120
      BVMIN=BVI
  120 CONTINUE
      DO 125 I=1,NST
      IF(IIPOL(I).NE.KKPOL(I)) GO TO 127
  125 CONTINUE
      GO TO 200
  127 DO 129 I=1,NST
      KKPOL(I)=IIPOL(I)
  129 CONTINUE
C***    SOLVE A SYSTEM OF EQUATIONS A*V=B BY CALLING SUBROUTINE FSLE
  130 CALL FSLE(NST,NDIMA,A,NSOL,NDIMBX,B,V,IPERM,NDIMT,T,DET,JEXP)
      IF(DET)135,175,135
  135 IF(ITER.EQ.0) GO TO 60
      ADD3=BVMIN/(1.-DISCR)
      SUBT=(DISCR*BVMAX)/(1.-DISCR)
      DO 171 I=1,NST
      ADVI=FV(I)+ADD3-SUBT
      DO 170 K=1,NAC
      IF(FLAG(I,K).GE.1.0) GO TO 170
      IF(CPV(I,K).GE.ADVI) GO TO 170
      FLAG(I,K)=1.
      ELM=ELM+1.
  170 CONTINUE
  171 CONTINUE
      PRCNT=ELM*100./TNM
      WRITE(6,172) PRCNT
  172 FORMAT(5X,'PRCNT',5X,F10.5)
      GO TO 60
  175 WRITE(6,180)
  180 FORMAT('SOLUTION FAILED')
  200 CALL TIME(1,1)
      WRITE (6,210)
  210 FORMAT (10X,'STATE',10X,'POLICY',15X,'RETURN',10X,'OPT. POLICY')
      WRITE (6,250) (I,NAC,V(I),IIPOL(I),I=1,NST)
  250 FORMAT (10X,I3,12X,I3,10X,E16.7,10X,I3/)
      WRITE (6,260)
  260 FORMAT(10X,'DISCR',10X,'DERR',10X,'ITERATION')
      WRITE (6,300) DISCR,DERR,ITER
  300 FORMAT(10X,F5.3,8X,F8.5,8X,I5)
      STOP
      END
```

```
C***
C*** C) CODE FOR POLICY ITERATION WITH PROCEDURE (II)
C***
      DIMENSION V(40),C(40,100),CPV(40,100),P(40,100,40),IIPOL(40),
     1A(40,40),T(40,40),B(40,1),IPERM(100),KKPOL(40),FLAG(40,100),
      REWIND8
      REWIND9
      NDIMA=40
      NDIMBX=40
      NDIMT=40
      NSOL=1
      READ(5,1) NST,NAC,DISCR,DERR
    1 FORMAT(I5,I5,2F10.5)
      READ(8)((C(I,K),I=1,NST),K=1,NAC)
      DO 2 I=1,NST
      READ(9)((P(I,K,J),K=1,NAC),J=1,NST)
    2 CONTINUE
      CALL TIME(0)
      ITER=0
      ELM=0.
      TNM=NST*NAC
      KN=0
      DO 50 I=1,NST
      DO 20 K=1,NAC
      FLAG(I,K)=0.
      IF(K.EQ.1) GO TO 10
      IF(OCI.GE.C(I,K)) GO TO 20
   10 OCI=C(I,K)
      IIP=K
   20 CONTINUE
      B(I,1)=OCI
      KKPOL(I)=IIP
      DO 40 J=1,NST
      IF(I.EQ.J) GO TO 30
      A(I,J)=-DISCR*P(I,IIP,J)
      GO TO 40
   30 A(I,J)=1.-DISCR*P(I,IIP,J)
   40 CONTINUE
   50 CONTINUE
      GO TO 130
   60 ITER=ITER+1
      DO 120 I=1,NST
      DO 110 K=1,NAC
      IF(FLAG(I,K).GE.1.0) GO TO 110
      KN=KN+1
      PV=0.
      DO 100 J=1,NST
      PV=PV+P(I,K,J)*V(J)
  100 CONTINUE
      CPV(I,K)=C(I,K)+DISCR*PV
      IF(KN.EQ.1) GO TO 105
      IF(CVI.GE.CPV(I,K)) GO TO 110
  105 CVI=CPV(I,K)
      IIP=K
  110 CONTINUE
      KN=0
      DO 115 J=1,NST
      IF(I.EQ.J) GO TO 113
      A(I,J)=-DISCR*P(I,IIP,J)
      GO TO 115
```

```
 113 A(I,J)=1.-DISCR*P(I,IIP,J)
 115 CONTINUE
     B(I,1)=C(I,IIP)
     BVI=CVI-V(I)
     IIPOL(I)=IIP
     IF(I.EQ.1) GO TO 116
     IF(BVMAX.GE.BVI) GO TO 117
     BVMAX=BVI
     GO TO 120
 116 BVMAX=BVI
     BVMIN=BVI
     GO TO 120
 117 IF(BVMIN.LE.BVI) GO TO 120
     BVMIN=BVI
 120 CONTINUE
     DO 125 I=1,NST
     IF(IIPOL(I).NE.KKPOL(I)) GO TO 127
 125 CONTINUE
     GO TO 200
 127 DO 129 I=1,NST
     KKPOL(I)=IIPOL(I)
 129 CONTINUE
C***   SOLVE A SYSTEM OF EQUATIONS A*V=B BY CALLING SUBROUTINE FSLE
 130 CALL FSLE(NST,NDIMA,A,NSOL,NDIMBX,B,V,IPERM,NDIMT,T,DET,JEXP)
     IF(DET)135,175,135
 135 IF(ITER.EQ.0) GO TO 60
     SUBT=(DISCR*BVMAX)/(1.-DISCR)
     DO 171 I=1,NST
     ADVI=V(I)-SUBT
     DO 170 K=1,NAC
     IF(FLAG(I,K).GE.1.0) GO TO 170
     IF(CPV(I,K).GE.ADVI) GO TO 170
     FLAG(I,K)=1.
     ELM=ELM+1.
 170 CONTINUE
 171 CONTINUE
     PRCNT=ELM*100./TNM
     WRITE(6,172) PRCNT
 172 FORMAT(5X,'PRCNT',5X,F10.5)
     GO TO 60
 175 WRITE(6,180)
 180 FORMAT('SOLUTION FAILED')
 200 CALL TIME(1,1)
     WRITE (6,210)
 210 FORMAT (10X,'STATE',10X,'POLICY',15X,'RETURN',10X,'OPT. POLICY')
     WRITE (6,250) (I,NAC,V(I),IIPOL(I),I=1,NST)
 250 FORMAT (10X,I3,12X,I3,10X,E16.7,10X,I3/)
     WRITE (6,260)
 260 FORMAT(10X,'DISCR',10X,'DERR',10X,'ITERATION')
     WRITE (6,300) DISCR,DERR,ITER
 300 FORMAT(10X,F5.3,8X,F8.5,8X,I5)
     STOP
     END
```

```
C***
C*** D) CODE FOR POLICY ITERATION WITH PROCEDURE (III)
C***
      DIMENSION V(40),C(40,100),CPV(40,100),P(40,100,40),IIPOL(40),
     1A(40,40),T(40,40),B(40,1),IPERM(100),KKPOL(40),FLAG(40,100),
     2FV(40)
      REWIND8
      REWIND9
      NDIMA=40
      NDIMBX=40
      NDIMT=40
      NSOL=1
      READ(5,1) NST,NAC,DISCR,DERR
    1 FORMAT(I5,I5,2F10.5)
      READ(8)((C(I,K),I=1,NST),K=1,NAC)
      DO 2 I=1,NST
      READ(9)((P(I,K,J),K=1,NAC),J=1,NST)
    2 CONTINUE
      CALL TIME(0)
      ITER=0
      ELM=0.
      TNM=NST*NAC
      KN=0
      DO 50 I=1,NST
      DO 20 K=1,NAC
      FLAG(I,K)=0.
      IF(K.EQ.1) GO TO 10
      IF(OCI.GE.C(I,K)) GO TO 20
   10 OCI=C(I,K)
      IIP=K
   20 CONTINUE
      B(I,1)=OCI
      KKPOL(I)=IIP
      DO 40 J=1,NST
      IF(I.EQ.J) GO TO 30
      A(I,J)=-DISCR*P(I,IIP,J)
      GO TO 40
   30 A(I,J)=1.-DISCR*P(I,IIP,J)
   40 CONTINUE
   50 CONTINUE
      GO TO 130
   60 ITER=ITER+1
      DO 120 I=1,NST
      DO 110 K=1,NAC
      IF(FLAG(I,K).GE.1.0) GO TO 110
      KN=KN+1
      PV=0.
      DO 100 J=1,NST
      PV=PV+P(I,K,J)*V(J)
  100 CONTINUE
      CPV(I,K)=C(I,K)+DISCR*PV
      IF(KN.EQ.1) GO TO 105
      IF(CVI.GE.CPV(I,K)) GO TO 110
  105 CVI=CPV(I,K)
      IIP=K
  110 CONTINUE
      KN=0
      DO 115 J=1,NST
      IF(I.EQ.J) GO TO 113
      A(I,J)=-DISCR*P(I,IIP,J)
```

```
      GO TO 115
  113 A(I,J)=1.-DISCR*P(I,IIP,J)
  115 CONTINUE
      B(I,1)=C(I,IIP)
      BVI=CVI-V(I)
      FV(I)=V(I)
      IIPOL(I)=IIP
      IF(I.EQ.1) GO TO 116
      IF(BVMAX.GE.BVI) GO TO 117
      BVMAX=BVI
      GO TO 120
  116 BVMAX=BVI
      BVMIN=BVI
      GO TO 120
  117 IF(BVMIN.LE.BVI) GO TO 120
      BVMIN=BVI
  120 CONTINUE
      DO 125 I=1,NST
      IF(IIPOL(I).NE.KKPOL(I)) GO TO 127
  125 CONTINUE
      GO TO 200
  127 DO 129 I=1,NST
      KKPOL(I)=IIPOL(I)
  129 CONTINUE
C***   SOLVE A SYSTEM OF EQUATIONS A*V=B BY CALLING SUBROUTINE FSLE
  130 CALL FSLE(NST,NDIMA,A,NSOL,NDIMBX,B,V,IPERM,NDIMT,T,DET,JEXP)
      IF(DET)135,175,135
  135 IF(ITER.EQ.0) GO TO 60
      DO 163 I=1,NST
      DVI=V(I)-FV(I)
      IF(I.EQ.1) GO TO 161
      IF(DVMAX.GE.DVI) GO TO 163
  161 DVMAX=DVI
  163 CONTINUE
      ADD1=DISCR*DVMAX
      ELM=0.
      DO 171 I=1,NST
      ADVI=V(I)
      DO 170 K=1,NAC
      CPV(I,K)=CPV(I,K)+ADD1
      IF(CPV(I,K).LT.ADVI) GO TO 165
      FLAG(I,K)=0.
      GO TO 170
  165 FLAG(I,K)=1.
      ELM=ELM+1.
  170 CONTINUE
  171 CONTINUE
      PRCNT=ELM*100./TNM
      WRITE(6,172) PRCNT
  172 FORMAT(5X,'PRCNT',5X,F10.5)
      GO TO 60
  175 WRITE(6,180)
  180 FORMAT('SOLUTION FAILED')
  200 CALL TIME(1,1)
      WRITE (6,210)
  210 FORMAT (10X,'STATE',10X,'POLICY',15X,'RETURN',10X,'OPT. POLICY')
      WRITE (6,250) (I,NAC,V(I),IIPOL(I),I=1,NST)
  250 FORMAT (10X,I3,12X,I3,10X,E16.7,10X,I3/)
      WRITE (6,260)
  260 FORMAT(10X,'DISCR',10X,'DERR',10X,'ITERATION')
```

```
      WRITE (6,300) DISCR,DERR,ITER
  300 FORMAT(10X,F5.3,8X,F8.5,8X,I5)
      STOP
      END
```

```
C***
C*** E) CODE FOR POLICY ITERATION WITH PROCEDURE (IV)
C***
      DIMENSION V(40),C(40,100),CPV(40,100),P(40,100,40),IIPOL(40),
     1A(40,40),T(40,40),B(40,1),IPERM(100),KKPOL(40),FLAG(40,100),
     2FV(40,100),DVMX(100)
      REWIND8
      REWIND9
      NDIMA=40
      NDIMBX=40
      NDIMT=40
      NSOL=1
      READ(5,1) NST,NAC,DISCR,DERR
    1 FORMAT(I5,I5,2F10.5)
      READ(8)((C(I,K),I=1,NST),K=1,NAC)
      DO 2 I=1,NST
      READ(9)((P(I,K,J),K=1,NAC),J=1,NST)
    2 CONTINUE
      CALL TIME(0)
      ITER=0
      ELM=0.
      TNM=NST*NAC
      KN=0
      DO 50 I=1,NST
      DO 20 K=1,NAC
      FLAG(I,K)=0.
      IF(K.EQ.1) GO TO 10
      IF(OCI.GE.C(I,K)) GO TO 20
   10 OCI=C(I,K)
      IIP=K
   20 CONTINUE
      B(I,1)=OCI
      KKPOL(I)=IIP
      DO 40 J=1,NST
      IF(I.EQ.J) GO TO 30
      A(I,J)=-DISCR*P(I,IIP,J)
      GO TO 40
   30 A(I,J)=1.-DISCR*P(I,IIP,J)
   40 CONTINUE
   50 CONTINUE
      GO TO 130
   60 ITER=ITER+1
      DO 120 I=1,NST
      DO 110 K=1,NAC
      IF(FLAG(I,K).GE.1.0) GO TO 110
      KN=KN+1
      PV=0.
      DO 100 J=1,NST
      PV=PV+P(I,K,J)*V(J)
  100 CONTINUE
      CPV(I,K)=C(I,K)+DISCR*PV
      IF(KN.EQ.1) GO TO 105
      IF(CVI.GE.CPV(I,K)) GO TO 110
  105 CVI=CPV(I,K)
      IIP=K
  110 CONTINUE
      KN=0
      DO 115 J=1,NST
      IF(I.EQ.J) GO TO 113
      A(I,J)=-DISCR*P(I,IIP,J)
```

```
      GO TO 115
  113 A(I,J)=1.-DISCR*P(I,IIP,J)
  115 CONTINUE
      B(I,1)=C(I,IIP)
      BVI=CVI-V(I)
      FV(I,ITER)=V(I)
      IIPOL(I)=IIP
      IF(I.EQ.1) GO TO 116
      IF(BVMAX.GE.BVI) GO TO 117
      BVMAX=BVI
      GO TO 120
  116 BVMAX=BVI
      BVMIN=BVI
      GO TO 120
  117 IF(BVMIN.LE.BVI) GO TO 120
      BVMIN=BVI
  120 CONTINUE
      DO 125 I=1,NST
      IF(IIPOL(I).NE.KKPOL(I)) GO TO 127
  125 CONTINUE
      GO TO 200
  127 DO 129 I=1,NST
      KKPOL(I)=IIPOL(I)
  129 CONTINUE
C***    SOLVE A SYSTEM OF EQUATIONS A*V=B BY CALLING SUBROUTINE FSLE
  130 CALL FSLE(NST,NDIMA,A,NSOL,NDIMBX,B,V,IPERM,NDIMT,T,DET,JEXP)
      IF(DET)135,175,135
  135 IF(ITER.EQ.0) GO TO 60
      DO 163 K=1,ITER
      DO 162 I=1,NST
      DVIK=V(I)-FV(I,K)
      IF(I.EQ.1) GO TO 161
      IF(DVMAX.GE.DVIK) GO TO 162
  161 DVMAX=DVIK
  162 CONTINUE
      DVMX(K)=DVMAX
  163 CONTINUE
      ELM=0.
      DO 171 I=1,NST
      ADVI=V(I)
      DO 170 K=1,NAC
      L=FLAG(I,K)
      IF(L.LE.0) GO TO 165
      CPVAD=CPV(I,K)+DISCR*DVMX(L)
      IF(CPVAD.LT.ADVI) GO TO 167
      FLAG(I,K)=0.
      GO TO 170
  165 CPVAD=CPV(I,K)+DISCR*DVMX(ITER)
      IF(CPVAD.GE.ADVI) GO TO 170
      FLAG(I,K)=ITER
  167 ELM=ELM+1.
  170 CONTINUE
  171 CONTINUE
      PRCNT=ELM*100./TNM
      WRITE(6,172) PRCNT
  172 FORMAT(5X,'PRCNT',5X,F10.5)
      GO TO 60
  175 WRITE(6,180)
  180 FORMAT('SOLUTION FAILED')
  200 CALL TIME(1,1)
```

```
      WRITE (6,210)
  210 FORMAT (10X,'STATE',10X,'POLICY',15X,'RETURN',10X,'OPT. POLICY')
      WRITE (6,250) (I,NAC,V(I),IIPOL(I),I=1,NST)
  250 FORMAT (10X,I3,12X,I3,10X,E16.7,10X,I3/)
      WRITE (6,260)
  260 FORMAT(10X,'DISCR',10X,'DERR',10X,'ITERATION')
      WRITE (6,300) DISCR,DERR,ITER
  300 FORMAT(10X,F5.3,8X,F8.5,8X,I5)
      STOP
      END
```

```
C***
C*** F) CODE FOR MODIFIED POLICY ITERATION WITHOUT ACTION ELIMINATION
C***
      DIMENSION V(40),CV(40),IIPOL(40),CPV(40,100),C(40,100),BV(40),
     1P(40,100,40),OP(40,40),OC(40)
      REWIND8
      REWIND9
      READ(5,1) NST,NAC,DISCR,DERR
    1 FORMAT(I5,I5,2F10.5)
      READ(8)((C(I,K),I=1,NST),K=1,NAC)
      DO 2 I=1,NST
      READ(9)((P(I,K,J),K=1,NAC),J=1,NST)
    2 CONTINUE
      CALL TIME(0)
      ITER=0
      ELM=0.
      TNM=NST*NAC
      KN=0
      DO 50 I=1,NST
      DO 20 K=1,NAC
      IF(K.EQ.1) GO TO 10
      IF(OCI.GE.C(I,K)) GO TO 20
   10 OCI=C(I,K)
      IIP=K
   20 CONTINUE
      IF(I.EQ.1) GO TO 30
      IF(CMN.LE.OCI) GO TO 40
   30 CMN=OCI
   40 OC(I)=OCI
      IIPOL(I)=IIP
      DO 45 J=1,NST
      OP(I,J)=P(I,IIP,J)
   45 CONTINUE
   50 CONTINUE
      AD=(DISCR*CMN)/(1.-DISCR)
      DO 55 I=1,NST
      CV(I)=OC(I)+AD
   55 CONTINUE
      NORD=0
      GO TO 130
   60 ITER=ITER+1
      DO 120 I=1,NST
      DO 110 K=1,NAC
      IF(FLAG(I,K).GE.1.0) GO TO 110
      KN=KN+1
      PV=0.
      DO 100 J=1,NST
      PV=PV+P(I,K,J)*V(J)
  100 CONTINUE
      CPV(I,K)=C(I,K)+DISCR*PV
      IF(KN.EQ.1) GO TO 105
      IF(CVI.GE.CPV(I,K)) GO TO 110
  105 CVI=CPV(I,K)
      IIP=K
  110 CONTINUE
      KN=0
      DO 115 J=1,NST
      OP(I,J)=P(I,IIP,J)
  115 CONTINUE
      OC(I)=C(I,IIP)
```

```
      CV(I)=CVI
      IIPOL(I)=IIP
      BVI=CVI-V(I)
      IF(I.EQ.1) GO TO 116
      IF(BVMAX.GE.BVI) GO TO 117
      BVMAX=BVI
      GO TO 120
  116 BVMAX=BVI
      BVMIN=BVI
      GO TO 120
  117 IF(BVMIN.LE.BVI) GO TO 120
      BVMIN=BVI
  120 CONTINUE
      ERR=BVMAX-BVMIN
      IF(ERR.LE.DERR) GO TO 200
      NORD=0
  130 NORD=NORD+1
      DO 140 I=1,NST
      OPV=0.
      DO 135 J=1,NST
      OPV=OPV+OP(I,J)*CV(J)
  135 CONTINUE
      VI=OC(I)+DISCR*OPV
      V(I)=VI
      PBV=VI-CV(I)
      IF(I.EQ.1) GO TO 139
      IF(PBVMX.GE.PBV) GO TO 136
      PBVMX=PBV
      GO TO 140
  136 IF(PBVMN.LE.PBV) GO TO 140
      PBVMN=PBV
      GO TO 140
  139 PBVMX=PBV
      PBVMN=PBV
  140 CONTINUE
      ERR=PBVMX-PBVMN
      IF(ERR.LE.DERR) GO TO 151
      DO 141 I=1,NST
      CV(I)=V(I)
  141 CONTINUE
      GO TO 130
  151 WRITE (6,155) NORD
  155 FORMAT(10X,'NORD',5X,I4)
      GO TO 60
  200 CALL TIME(1,1)
      WRITE (6,210)
  210 FORMAT (10X,'STATE',10X,'POLICY',15X,'RETURN',10X,'OPT. POLICY')
      AD=DISCR*BVMIN/(1.-DISCR)
      DO 205 I=1,NST
      V(I)=CV(I)+AD
  205 CONTINUE
      WRITE (6,250) (I,NAC,V(I),IIPOL(I),I=1,NST)
  250 FORMAT (10X,I3,12X,I3,10X,E16.7,10X,I3/)
      WRITE (6,260)
  260 FORMAT(10X,'DISCR',10X,'DERR',10X,'ITERATION')
      WRITE (6,300) DISCR,DERR,ITER
  300 FORMAT(10X,F5.3,8X,F8.5,8X,I5)
      STOP
      END
```

```
C***
C*** G) CODE FOR MODIFIED POLICY ITERATION WITH PROCEDURE (II)
C***
      DIMENSION V(40),CV(40),IIPOL(40),CPV(40,100),C(40,100),BV(40),
     1P(40,100,40),OP(40,40),OC(40),FLAG(40,100)
      REWIND8
      REWIND9
      READ(5,1) NST,NAC,DISCR,DERR
    1 FORMAT(I5,I5,2F10.5)
      READ(8)((C(I,K),I=1,NST),K=1,NAC)
      DO 2 I=1,NST
      READ(9)((P(I,K,J),K=1,NAC),J=1,NST)
    2 CONTINUE
      CALL TIME(0)
      ITER=0
      ELM=0.
      TNM=NST*NAC
      KN=0
      DO 50 I=1,NST
      DO 20 K=1,NAC
      FLAG(I,K)=0.
      IF(K.EQ.1) GO TO 10
      IF(OCI.GE.C(I,K)) GO TO 20
   10 OCI=C(I,K)
      IIP=K
   20 CONTINUE
      IF(I.EQ.1) GO TO 30
      IF(CMN.LE.OCI) GO TO 40
   30 CMN=OCI
   40 OC(I)=OCI
      IIPOL(I)=IIP
      DO 45 J=1,NST
      OP(I,J)=P(I,IIP,J)
   45 CONTINUE
   50 CONTINUE
      AD=(DISCR*CMN)/(1.-DISCR)
      DO 55 I=1,NST
      CV(I)=OC(I)+AD
   55 CONTINUE
      NORD=0
      GO TO 130
   60 ITER=ITER+1
      DO 120 I=1,NST
      DO 110 K=1,NAC
      IF(FLAG(I,K).GE.1.0) GO TO 110
      KN=KN+1
      PV=0.
      DO 100 J=1,NST
      PV=PV+P(I,K,J)*V(J)
  100 CONTINUE
      CPV(I,K)=C(I,K)+DISCR*PV
      IF(KN.EQ.1) GO TO 105
      IF(CVI.GE.CPV(I,K)) GO TO 110
  105 CVI=CPV(I,K)
      IIP=K
  110 CONTINUE
      KN=0
      DO 115 J=1,NST
      OP(I,J)=P(I,IIP,J)
  115 CONTINUE
```

```
      OC(I)=C(I,IIP)
      CV(I)=CVI
      IIPOL(I)=IIP
      BVI=CVI-V(I)
      IF(I.EQ.1) GO TO 116
      IF(BVMAX.GE.BVI) GO TO 117
      BVMAX=BVI
      GO TO 120
  116 BVMAX=BVI
      BVMIN=BVI
      GO TO 120
  117 IF(BVMIN.LE.BVI) GO TO 120
      BVMIN=BVI
  120 CONTINUE
      ERR=BVMAX-BVMIN
      IF(ERR.LE.DERR) GO TO 200
      NORD=0
  130 NORD=NORD+1
      DO 140 I=1,NST
      OPV=0.
      DO 135 J=1,NST
      OPV=OPV+OP(I,J)*CV(J)
  135 CONTINUE
      VI=OC(I)+DISCR*OPV
      V(I)=VI
      PBV=VI-CV(I)
      IF(I.EQ.1) GO TO 139
      IF(PBVMX.GE.PBV) GO TO 136
      PBVMX=PBV
      GO TO 140
  136 IF(PBVMN.LE.PBV) GO TO 140
      PBVMN=PBV
      GO TO 140
  139 PBVMX=PBV
      PBVMN=PBV
  140 CONTINUE
      ERR=PBVMX-PBVMN
      IF(ERR.LE.DERR) GO TO 151
      DO 141 I=1,NST
      CV(I)=V(I)
  141 CONTINUE
      GO TO 130
  151 WRITE (6,155) NORD
  155 FORMAT(10X,'NORD',5X,I4)
      IF(ITER.EQ.0) GO TO 60
      SUBT=(DISCR*BVMAX-(DISCR**(NORD+1))*BVMIN)/(1.-DISCR)
      DO 171 I=1,NST
      ADVI=V(I)-SUBT
      DO 170 K=1,NAC
      IF(FLAG(I,K).GE.1.0) GO TO 170
      IF(CPV(I,K).GE.ADVI) GO TO 170
      FLAG(I,K)=1.
      ELM=ELM+1.
  170 CONTINUE
  171 CONTINUE
      PRCNT=ELM*100./TNM
      WRITE(6,172) PRCNT
  172 FORMAT(30X,'PRCNT',5X,F10.5)
      GO TO 60
  200 CALL TIME(1,1)
```

```
      WRITE (6,210)
  210 FORMAT (10X,'STATE',10X,'POLICY',15X,'RETURN',10X,'OPT. POLICY')
      AD=DISCR*BVMIN/(1.-DISCR)
      DO 205 I=1,NST
      V(I)=CV(I)+AD
  205 CONTINUE
      WRITE (6,250) (I,NAC,V(I),IIPOL(I),I=1,NST)
  250 FORMAT (10X,I3,12X,I3,10X,E16.7,10X,I3/)
      WRITE (6,260)
  260 FORMAT(10X,'DISCR',10X,'DERR',10X,'ITERATION')
      WRITE (6,300) DISCR,DERR,ITER
  300 FORMAT(10X,F5.3,8X,F8.5,8X,I5)
      STOP
      END
```

```
C***
C*** H) CODE FOR MODIFIED POLICY ITERATION WITH PROCEDURE (III)
C***
      DIMENSION V(40),CV(40),IIPOL(40),CPV(40,100),C(40,100),BV(40),
     1P(40,100,40),OP(40,40),OC(40),FLAG(40,100),FV(40)
      REWIND8
      REWIND9
      READ(5,1) NST,NAC,DISCR,DERR
    1 FORMAT(I5,I5,2F10.5)
      READ(8)((C(I,K),I=1,NST),K=1,NAC)
      DO 2 I=1,NST
      READ(9)((P(I,K,J),K=1,NAC),J=1,NST)
    2 CONTINUE
      CALL TIME(0)
      ITER=0
      ELM=0.
      TNM=NST*NAC
      KN=0
      DO 50 I=1,NST
      DO 20 K=1,NAC
      FLAG(I,K)=0.
      IF(K.EQ.1) GO TO 10
      IF(OCI.GE.C(I,K)) GO TO 20
   10 OCI=C(I,K)
      IIP=K
   20 CONTINUE
      IF(I.EQ.1) GO TO 30
      IF(CMN.LE.OCI) GO TO 40
   30 CMN=OCI
   40 OC(I)=OCI
      IIPOL(I)=IIP
      DO 45 J=1,NST
      OP(I,J)=P(I,IIP,J)
   45 CONTINUE
   50 CONTINUE
      AD=(DISCR*CMN)/(1.-DISCR)
      DO 55 I=1,NST
      CV(I)=OC(I)+AD
   55 CONTINUE
      NORD=0
      GO TO 130
   60 ITER=ITER+1
      DO 120 I=1,NST
      DO 110 K=1,NAC
      IF(FLAG(I,K).GE.1.0) GO TO 110
      KN=KN+1
      PV=0.
      DO 100 J=1,NST
      PV=PV+P(I,K,J)*V(J)
  100 CONTINUE
      CPV(I,K)=C(I,K)+DISCR*PV
      IF(KN.EQ.1) GO TO 105
      IF(CVI.GE.CPV(I,K)) GO TO 110
  105 CVI=CPV(I,K)
      IIP=K
  110 CONTINUE
      KN=0
      DO 115 J=1,NST
      OP(I,J)=P(I,IIP,J)
  115 CONTINUE
```

```
      OC(I)=C(I,IIP)
      CV(I)=CVI
      IIPOL(I)=IIP
      BVI=CVI-V(I)
      FV(I)=V(I)
      IF(I.EQ.1) GO TO 116
      IF(BVMAX.GE.BVI) GO TO 117
      BVMAX=BVI
      GO TO 120
  116 BVMAX=BVI
      BVMIN=BVI
      GO TO 120
  117 IF(BVMIN.LE.BVI) GO TO 120
      BVMIN=BVI
  120 CONTINUE
      ERR=BVMAX-BVMIN
      IF(ERR.LE.DERR) GO TO 200
      NORD=0
  130 NORD=NORD+1
      DO 140 I=1,NST
      OPV=0.
      DO 135 J=1,NST
      OPV=OPV+OP(I,J)*CV(J)
  135 CONTINUE
      VI=OC(I)+DISCR*OPV
      V(I)=VI
      PBV=VI-CV(I)
      IF(I.EQ.1) GO TO 139
      IF(PBVMX.GE.PBV) GO TO 136
      PBVMX=PBV
      GO TO 140
  136 IF(PBVMN.LE.PBV) GO TO 140
      PBVMN=PBV
      GO TO 140
  139 PBVMX=PBV
      PBVMN=PBV
  140 CONTINUE
      ERR=PBVMX-PBVMN
      IF(ERR.LE.DERR) GO TO 151
      DO 141 I=1,NST
      CV(I)=V(I)
  141 CONTINUE
      GO TO 130
  151 WRITE (6,155) NORD
  155 FORMAT(10X,'NORD',5X,I4)
      IF(ITER.EQ.0) GO TO 60
      DO 163 I=1,NST
      DVI=V(I)-FV(I)
      IF(I.EQ.1) GO TO 161
      IF(DVMAX.GE.DVI) GO TO 163
  161 DVMAX=DVI
  163 CONTINUE
      ADD1=DISCR*DVMAX
      ADD2=DISCR*PBVMN
      ELM=0.
      DO 171 I=1,NST
      ADVI=V(I)+ADD2
      DO 170 K=1,NAC
      CPV(I,K)=CPV(I,K)+ADD1
      IF(CPV(I,K).LT.ADVI) GO TO 165
```

```
      FLAG(I,K)=0.
      GO TO 170
  165 FLAG(I,K)=1.
      ELM=ELM+1.
  170 CONTINUE
  171 CONTINUE
      PRCNT=ELM*100./TNM
      WRITE(6,172) PRCNT
  172 FORMAT(30X,'PRCNT',5X,F10.5)
      GO TO 60
  200 CALL TIME(1,1)
      WRITE (6,210)
  210 FORMAT (10X,'STATE',10X,'POLICY',15X,'RETURN',10X,'OPT. POLICY')
      AD=DISCR*BVMIN/(1.-DISCR)
      DO 205 I=1,NST
      V(I)=CV(I)+AD
  205 CONTINUE
      WRITE (6,250) (I,NAC,V(I),IIPOL(I),I=1,NST)
  250 FORMAT (10X,I3,12X,I3,10X,E16.7,10X,I3/)
      WRITE (6,260)
  260 FORMAT(10X,'DISCR',10X,'DERR',10X,'ITERATION')
      WRITE (6,300) DISCR,DERR,ITER
  300 FORMAT(10X,F5.3,8X,F8.5,8X,I5)
      STOP
      END
```

```
C***
C*** I) CODE FOR MODIFIED POLICY ITERATION WITH PROCEDURE (IV)
C***
      DIMENSION V(40),CV(40),IIPOL(40),CPV(40,100),C(40,100),BV(40),
     1P(40,100,40),OP(40,40),OC(40),FLAG(40,100),FV(40,100),DVMX(100)
      REWIND8
      REWIND9
      READ(5,1) NST,NAC,DISCR,DERR
    1 FORMAT(I5,I5,2F10.5)
      READ(8)((C(I,K),I=1,NST),K=1,NAC)
      DO 2 I=1,NST
      READ(9)((P(I,K,J),K=1,NAC),J=1,NST)
    2 CONTINUE
      CALL TIME(0)
      ITER=0
      ELM=0.
      TNM=NST*NAC
      KN=0
      DO 50 I=1,NST
      DO 20 K=1,NAC
      FLAG(I,K)=0.
      IF(K.EQ.1) GO TO 10
      IF(OCI.GE.C(I,K)) GO TO 20
   10 OCI=C(I,K)
      IIP=K
   20 CONTINUE
      IF(I.EQ.1) GO TO 30
      IF(CMN.LE.OCI) GO TO 40
   30 CMN=OCI
   40 OC(I)=OCI
      IIPOL(I)=IIP
      DO 45 J=1,NST
      OP(I,J)=P(I,IIP,J)
   45 CONTINUE
   50 CONTINUE
      AD=(DISCR*CMN)/(1.-DISCR)
      DO 55 I=1,NST
      CV(I)=OC(I)+AD
   55 CONTINUE
      NORD=0
      GO TO 130
   60 ITER=ITER+1
      DO 120 I=1,NST
      DO 110 K=1,NAC
      IF(FLAG(I,K).GE.1.0) GO TO 110
      KN=KN+1
      PV=0.
      DO 100 J=1,NST
      PV=PV+P(I,K,J)*V(J)
  100 CONTINUE
      CPV(I,K)=C(I,K)+DISCR*PV
      IF(KN.EQ.1) GO TO 105
      IF(CVI.GE.CPV(I,K)) GO TO 110
  105 CVI=CPV(I,K)
      IIP=K
  110 CONTINUE
      KN=0
      DO 115 J=1,NST
      OP(I,J)=P(I,IIP,J)
  115 CONTINUE
```

```
      OC(I)=C(I,IIP)
      CV(I)=CVI
      IIPOL(I)=IIP
      BVI=CVI-V(I)
      FV(I,ITER)=V(I)
      IF(I.EQ.1) GO TO 116
      IF(BVMAX.GE.BVI) GO TO 117
      BVMAX=BVI
      GO TO 120
  116 BVMAX=BVI
      BVMIN=BVI
      GO TO 120
  117 IF(BVMIN.LE.BVI) GO TO 120
      BVMIN=BVI
  120 CONTINUE
      ERR=BVMAX-BVMIN
      IF(ERR.LE.DERR) GO TO 200
      NORD=0
  130 NORD=NORD+1
      DO 140 I=1,NST
      OPV=0.
      DO 135 J=1,NST
      OPV=OPV+OP(I,J)*CV(J)
  135 CONTINUE
      VI=OC(I)+DISCR*OPV
      V(I)=VI
      PBV=VI-CV(I)
      IF(I.EQ.1) GO TO 139
      IF(PBVMX.GE.PBV) GO TO 136
      PBVMX=PBV
      GO TO 140
  136 IF(PBVMN.LE.PBV) GO TO 140
      PBVMN=PBV
      GO TO 140
  139 PBVMX=PBV
      PBVMN=PBV
  140 CONTINUE
      ERR=PBVMX-PBVMN
      IF(ERR.LE.DERR) GO TO 151
      DO 141 I=1,NST
      CV(I)=V(I)
  141 CONTINUE
      GO TO 130
  151 WRITE (6,155) NORD
  155 FORMAT(10X,'NORD',5X,I4)
      IF(ITER.EQ.0) GO TO 60
      DO 163 K=1,ITER
      DO 162 I=1,NST
      DVIK=V(I)-FV(I,K)
      IF(I.EQ.1) GO TO 161
      IF(DVMAX.GE.DVIK) GO TO 162
  161 DVMAX=DVIK
  162 CONTINUE
      DVMX(K)=DVMAX
  163 CONTINUE
      ADD2=DISCR*PBVMN
      ELM=0.
      DO 171 I=1,NST
      ADVI=V(I)+ADD2
      DO 170 K=1,NAC
```

```
      L=FLAG(I,K)
      IF(L.LE.0) GO TO 165
      CPVAD=CPV(I,K)+DISCR*DVMX(L)
      IF(CPVAD.LT.ADVI) GO TO 167
      FLAG(I,K)=0.
      GO TO 170
  165 CPVAD=CPV(I,K)+DISCR*DVMX(ITER)
      IF(CPVAD.GE.ADVI) GO TO 170
      FLAG(I,K)=ITER
  167 ELM=ELM+1.
  170 CONTINUE
  171 CONTINUE
      PRCNT=ELM*100./TNM
      WRITE(6,172) PRCNT
  172 FORMAT(30X,'PRCNT',5X,F10.5)
      GO TO 60
  200 CALL TIME(1,1)
      WRITE (6,210)
  210 FORMAT (10X,'STATE',10X,'POLICY',15X,'RETURN',10X,'OPT. POLICY')
      AD=DISCR*BVMIN/(1.-DISCR)
      DO 205 I=1,NST
      V(I)=CV(I)+AD
  205 CONTINUE
      WRITE (6,250) (I,NAC,V(I),IIPOL(I),I=1,NST)
  250 FORMAT (10X,I3,12X,I3,10X,E16.7,10X,I3/)
      WRITE (6,260)
  260 FORMAT(10X,'DISCR',10X,'DERR',10X,'ITERATION')
      WRITE (6,300) DISCR,DERR,ITER
  300 FORMAT(10X,F5.3,8X,F8.5,8X,I5)
      STOP
      END
```

```
C***
C*** J) CODE FOR GENERATING RANDOM DATA
C***
      DIMENSION P(400,100),IP(100),C(100,400)
      REWIND8
      REWIND9
      READ(5,10)NST,NAC,NZERO,S,R
   10 FORMAT(3I5,2F10.3)
      X=RANDN(S)
      Y=RAND(R)
      JNZ=IRAND(-NST)
      DO 200 I=1,NST
      DO 100 K=1,NAC
   20 X=0.-FRANDN(S)
      IF(X.LT.0.) X=-X
      IF(X.GE.2.) GO TO 20
      ICIK=1000.-X*100.
      C(I,K)=ICIK-I*10
  100 CONTINUE
  200 CONTINUE
      WRITE(8) ((C(I,K),I=1,NST),K=1,NAC)
      I=1
  250 DO 400 K=1,NAC
      DO 300 J=1,NST
      P(K,J)=0.
  300 CONTINUE
      ISUM=0
      DO 320 L=1,NZERO
      IY=FRAND(R)*100.+100.
      ISUM=ISUM+IY
      IP(L)=IY
  320 CONTINUE
      NMZR=NZERO-1
      M=1
      SSUM=0.
  340 PKL=100*IP(M)/ISUM
  350 JNZ=IRAND(NST)
      IF(P(K,JNZ).GT.0.0) GO TO 350
      P(K,JNZ)=PKL*0.01
      SSUM=SSUM+P(K,JNZ)
      IF(M.GE.NMZR) GO TO 360
      M=M+1
      GO TO 340
  360 JNZ=IRAND(NST)
      IF(P(K,JNZ).GT.0.0) GO TO 360
      P(K,JNZ)=1.-SSUM
  400 CONTINUE
      WRITE(9) ((P(K,J),K=1,NAC),J=1,NST)
      IF(I.GE.NST) GO TO 500
      I=I+1
      GO TO 250
  500 WRITE(9,510)
  510 FORMAT('$CONTINUE WITH DR40X40')
      STOP
      END
```

```
C***
C*** K) CODE FOR GENERATING DATA OF TWO CRITERION AUTOMOBILE
C       REPLACEMENT PROBLEM
C***
      DIMENSION CS(50),T(50),E(50),PR(50),C(50,50),P(50,50),CTHR(50,50)
      REWIND8
      REWIND9
      DO 200 I=1,41
      READ(5,100) CS(I),T(I),E(I),PR(I)
  100 FORMAT (F10.1,F10.1,F10.1,F10.3)
      WRITE(6,100) CS(I),T(I),E(I),PR(I)
  200 CONTINUE
      DO 500 I=1,40
      M=I+1
      AI=M
      DO 400 K=1,41
      IF(K.EQ.1)GO TO 210
      L=K-1
      AL=L
      C(I,K)=T(M)-CS(L)-E(L)
      CTHR(I,K)=400./(AL**0.5)-C(I,K)
      GO TO 220
  210 C(I,K)=-E(M)
      CTHR(I,K)=400./(AI**0.5)-C(I,K)
  220 DO 300 J=1,40
      IF(K.GE.2) GO TO 250
      IF(I.GE.39) GO TO 245
      IF(J.EQ.M) GO TO 230
      IF(J.EQ.40) GO TO 240
      P(K,J)=0.
      GO TO 300
  230 P(K,J)=PR(M)
      GO TO 300
  240 P(K,J)=1.-PR(M)
      GO TO 300
  245 IF(J.EQ.40) GO TO 246
      P(K,J)=0.
      GO TO 300
  246 P(K,J)=1.0
      GO TO 300
  250 IF(K.EQ.41) GO TO 280
      IF(J.EQ.L) GO TO 260
      IF(J.EQ.40) GO TO 270
      P(K,J)=0.
      GO TO 300
  260 P(K,J)=PR(L)
      GO TO 300
  270 P(K,J)=1.-PR(L)
      GO TO 300
  280 IF(J.EQ.40) GO TO 290
      P(K,J)=0.0
      GO TO 300
  290 P(K,J)=1.0
  300 CONTINUE
  400 CONTINUE
      WRITE(9)((P(K,J),K=1,41),J=1,40)
  500 CONTINUE
      WRITE(8)((C(I,K),CTHR(I,K),I=1,40),K=1,41)
      STOP
      END
```

```
C***
C*** L) CODE FOR PARAMETRIC MDP WITH SELECTION RULE (B)
C***
      REAL*8 TALPH,DALPH,ALPHA,A,B,T,V,DET,COND,U,RV,DJ,BV,BU,PV,PU,
     1OBV,OBU,BVIK,BUIK,OOBU,BVMX,BVMN,BUMX,BUMN,RBV,DBJ,DJMX,PDUMX,
     2DUMAX,EXALP,SALPH,ADEPS,SBEPS
      DIMENSION V(50),R(50,50),P(50,50,50),IIPOL(50),D(50,50),A(50,50),
     1T(50,50),B(50,1),IPERM(100),KKPOL(50),U(50),RV(50),DJ(50),
     2BV(50,50),BU(50,50),BUI(50)
      REWIND8
      REWIND9
      ERR=0.001
      EPSLN=0.000001
      NDIMA=50
      NDIMBX=50
      NDIMT=50
      NSOL=1
      READ(5,1) NST,NAC,DISCR,ALPMN,ALPMX
    1 FORMAT(2I5,3F10.5)
      ALPHA=ALPMN
      READ(8)((R(I,K),D(I,K),I=1,NST),K=1,NAC)
      ITER=0
      ITR=0
      DO 2 I=1,NST
      READ(9)((P(I,K,J),K=1,NAC),J=1,NST)
    2 CONTINUE
      DO 50 I=1,NST
      DO 20 K=1,NAC
      RDIK=R(I,K)+ALPHA*D(I,K)
      IF(K.EQ.1) GO TO 10
      IF(ORI.GE.RDIK) GO TO 20
   10 ORI=RDIK
      IIP=K
   20 CONTINUE
      B(I,1)=ORI
      KKPOL(I)=IIP
      DO 40 J=1,NST
      IF(I.EQ.J) GO TO 30
      A(I,J)=-DISCR*P(I,IIP,J)
      GO TO 40
   30 A(I,J)=1.-DISCR*P(I,IIP,J)
   40 CONTINUE
   50 CONTINUE
      GO TO 130
   60 ITER=ITER+1
      DO 120 I=1,NST
      DO 110 K=1,NAC
      PV=0.
      DO 100 J=1,NST
      PV=PV+P(I,K,J)*V(J)
  100 CONTINUE
      RPVIK=R(I,K)+ALPHA*D(I,K)+DISCR*PV
      IF(K.EQ.1) GO TO 105
      IF(RVI.GE.RPVIK) GO TO 110
  105 RVI=RPVIK
      IIP=K
  110 CONTINUE
      DO 115 J=1,NST
      IF(I.EQ.J) GO TO 113
      A(I,J)=-DISCR*P(I,IIP,J)
```

```
      GO TO 115
  113 A(I,J)=1.-DISCR*P(I,IIP,J)
  115 CONTINUE
      B(I,1)=R(I,IIP)+ALPHA*D(I,IIP)
      IIPOL(I)=IIP
  120 CONTINUE
      DO 125 I=1,NST
      IF(IIPOL(I).NE.KKPOL(I)) GO TO 127
  125 CONTINUE
      GO TO 190
  127 DO 129 I=1,NST
      KKPOL(I)=IIPOL(I)
  129 CONTINUE
C*** SOLVE THE SYSTEM OF EQUATIONS AV=B BY CALLING SUBROUTINE SLE
  130 CALL SLE(NST,NDIMA,A,NSOL,NDIMBX,B,V,IPERM,NDIMT,T,DET,JEXP)
      IF(DET) 135,175,135
  135 GO TO 60
  175 WRITE(6,180)
  180 FORMAT(5X,'SOLUTION FAILED')
      GO TO 700
  190 DO 200 I=1,NST
      IIP=IIPOL(I)
      PV=0.
      DO 192 J=1,NST
      PV=PV+P(I,IIP,J)*V(J)
  192 CONTINUE
      RV(I)=R(I,IIP)+ALPHA*D(I,IIP)+DISCR*PV
      RBV=RV(I)-V(I)
      IF(I.EQ.1) GO TO 198
      IF(RBV.GT.BVMX) GO TO 194
      IF(RBV.LT.BVMN) GO TO 196
      GO TO 200
  194 BVMX=RBV
      GO TO 200
  196 BVMN=RBV
      GO TO 200
  198 BVMX=RBV
      BVMN=RBV
  200 CONTINUE
      VERR=BVMX-BVMN
      IF(ITR.EQ.0.AND.VERR.LE.ERR) GO TO 206
      ADD=DISCR*BVMN/(1.-DISCR)
      DO 202 I=1,NST
      V(I)=RV(I)+ADD
  202 CONTINUE
      IF(VERR.LE.ERR) GO TO 204
      ITR=ITR+1
      GO TO 190
  204 WRITE(6,205) ITR
  205 FORMAT(5X,'NUMBER OF ITERATION IN T-OPERATION FOR V IS',5X,I5)
      GO TO 60
  206 WRITE(6,205) ITR
      WRITE(6,208)ALPHA
  208 FORMAT(5X,'ALPHA =',F20.14)
      ADD=BVMN/(1.-DISCR)
      DO 209 I=1,NST
      V(I)=V(I)+ADD
  209 CONTINUE
      WRITE(6,210)
  210 FORMAT(10X,'STATE',10X,'POLICY',15X,'RETURN',10X,'OPT. POLICY')
```

```
      WRITE(6,220)(I,NAC,V(I),IIPOL(I),I=1,NST)
  220 FORMAT(10X,I3,12X,I3,10X,E16.7,10X,I3/)
      WRITE(6,230)
  230 FORMAT(10X,'DISCR',10X,'ITERATION')
      WRITE(6,240)DISCR,ITER
  240 FORMAT(10X,F7.5,6X,I5)
      WRITE(6,242)
  242 FORMAT(///1X,'*** OPTIMAL ACTION(S) TO BE CHOSEN FOR THE CURRENT
     1OPTIMAL POLICY'///)
C*** FIND THE INVERSE OF MATRIX A BY CALLING SUBROUTINE INV
      CALL INV(NST,NDIMA,A,IPERM,NDIMT,T,DET,JEXP,COND)
      IF(DET)245,620,245
  245 DO 250 I=1,NST
      TD=0.
      DO 247 J=1,NST
      IIP=IIPOL(J)
      TD=TD+T(I,J)*D(J,IIP)
  247 CONTINUE
      U(I)=TD
  250 CONTINUE
      ITR=0
  260 DO 300 I=1,NST
      IIP=IIPOL(I)
      PU=0.
      DO 265 J=1,NST
      PU=PU+P(I,IIP,J)*U(J)
  265 CONTINUE
      DU(I)=D(I,IIP)+DISCR*PU
      DBU=DU(I)-U(I)
      IF(I.EQ.1) GO TO 290
      IF(DBU.GT.BUMX) GO TO 270
      IF(DBU.LT.BUMN) GO TO 280
      GO TO 300
  270 BUMX=DBU
      GO TO 300
  280 BUMN=DBU
      GO TO 300
  290 BUMX=DBU
      BUMN=DBU
  300 CONTINUE
      UERR=BUMX-BUMN
      IF(UERR.LE.ERR) GO TO 310
      DO 305 I=1,NST
      U(I)=DU(I)
  305 CONTINUE
      ITR=ITR+1
      GO TO 260
  310 ADD=DISCR*BUMN/(1.-DISCR)
      DO 320 I=1,NST
      U(I)=DU(I)+ADD
  320 CONTINUE
      WRITE(6,590) ITR
      DO 325 I=1,NST
      DO 323 K=1,NAC
      PV=0.
      DO 321 J=1,NST
      PV=PV+P(I,K,J)*V(J)
  321 CONTINUE
      BV(I,K)=R(I,K)+ALPHA*D(I,K)+DISCR*PV-V(I)
  323 CONTINUE
```

```
325 CONTINUE
    CALL TIME(0)
    ITER=1
330 DO 340 I=1,NST
    DO 335 K=1,NAC
    PU=0.
    DO 333 J=1,NST
    PU=PU+P(I,K,J)*U(J)
333 CONTINUE
    BU(I,K)=D(I,K)+DISCR*PU-U(I)
335 CONTINUE
340 CONTINUE
    KN=0
    DO 380 I=1,NST
    IIP=IIPOL(I)
    OBV=BV(I,IIP)
    OBU=BU(I,IIP)
    DO 370 K=1,NAC
    IF(K.EQ.IIP) GO TO 370
    BVIK=BV(I,K)
    BUIK=BU(I,K)
    IF(BUIK.LE.OBU) GO TO 370
    IF(BVIK.GT.OBV) GO TO 360
    KN=KN+1
    TALPH=(-BVIK+OBV)/(BUIK-OBU)
    IF(KN.EQ.1) GO TO 350
    ADEPS=DALPH+EPSLN
    SBEPS=DALPH-EPSLN
    IF(TALPH.GE.ADEPS) GO TO 370
    IF(TALPH.LE.SBEPS) GO TO 350
    IF(BUIK.LE.BUI(I)) GO TO 370
    KKPOL(I)=K
    BUI(I)=BUIK
    GO TO 370
350 DALPH=TALPH
    DO 357 J=1,NST
    IF(J.EQ.I) GO TO 355
    KKPOL(J)=IIPOL(J)
    BUI(J)=0.
    GO TO 357
355 KKPOL(J)=K
    BUI(J)=BUIK
357 CONTINUE
    GO TO 370
360 WRITE(6,365)I,K,BVIK,BUIK,OBV,OBU
365 FORMAT(5X,'BETTER POLICY EXIST',5X,2I5,4E16.7)
370 CONTINUE
380 CONTINUE
385 IF(KN.EQ.0) GO TO 600
    ALPHA=ALPHA+DALPH
    DO 393 I=1,NST
    IF(KKPOL(I).EQ.IIPOL(I)) GO TO 393
    WRITE(6,390)ALPHA,I,KKPOL(I)
390 FORMAT(5X,'ALPHA =',F20.14,5X,'I =',I5,5X,'K =',I5)
393 CONTINUE
    IF(ALPHA.GE.ALPMX) GO TO 700
    IF(ALPHA.LT.ALPMN) GO TO 700
399 DO 950 K=1,NST
    IF(KKPOL(K).EQ.IIPOL(K)) GO TO 900
    IFRST=K
```

```
      KFRST=KKPOL(K)
      DO 410 J=1,NST
      IF(J.EQ.IFRST) GO TO 400
      RV(J)=-DISCR*P(IFRST,KFRST,J)
      GO TO 410
  400 RV(J)=1.-DISCR*P(IFRST,KFRST,J)
  410 CONTINUE
      DO 430 I=1,NST
      PU=0.
      DO 420 J=1,NST
      PU=PU+RV(J)*T(J,I)
  420 CONTINUE
      DU(I)=PU
  430 CONTINUE
      PU=DU(IFRST)
      IF(PU.EQ.0.) GO TO 620
      DO 450 J=1,NST
      IF(J.EQ.IFRST) GO TO 440
      DU(J)=-DU(J)/PU
      GO TO 450
  440 DU(J)=1/PU
  450 CONTINUE
      DO 480 I=1,NST
      PV=T(I,IFRST)
      DO 470 J=1,NST
      IF(J.EQ.IFRST) GO TO 460
      T(I,J)=T(I,J)+PV*DU(J)
      GO TO 470
  460 T(I,J)=PV*DU(J)
  470 CONTINUE
  480 CONTINUE
  900 IIPOL(K)=KKPOL(K)
  950 CONTINUE
      DO 490 I=1,NST
      V(I)=V(I)+DALPH*U(I)
      PU=0.
      DO 485 J=1,NST
      IIP=IIPOL(J)
      PU=PU+T(I,J)*BU(J,IIP)
  485 CONTINUE
      U(I)=U(I)+PU
  490 CONTINUE
      ITR=0
  500 DO 550 I=1,NST
      IIP=IIPOL(I)
      PU=0.
      DO 510 J=1,NST
      PU=PU+P(I,IIP,J)*U(J)
  510 CONTINUE
      DU(I)=D(I,IIP)+DISCR*PU
      DBU=DU(I)-U(I)
      IF(I.EQ.1) GO TO 540
      IF(DBU.GT.BUMX) GO TO 520
      IF(DBU.LT.BUMN) GO TO 530
      GO TO 550
  520 BUMX=DBU
      GO TO 550
  530 BUMN=DBU
      GO TO 550
  540 BUMX=DBU
```

```
      BUMN=DBU
  550 CONTINUE
      UERR=BUMX-BUMN
      IF(UERR.LE.ERR) GO TO 570
      DO 560 I=1,NST
      U(I)=DU(I)
  560 CONTINUE
      ITR=ITR+1
      GO TO 500
  570 ADD=DISCR*BUMN/(1.-DISCR)
      DO 580 I=1,NST
      U(I)=DU(I)+ADD
  580 CONTINUE
      WRITE(6,590) ITR
  590 FORMAT(5X,'NUMBER OF ITERATION IN S-OPERATION FOR U IS',5X,I5)
      DO 599 I=1,NST
      DO 597 K=1,NAC
      BV(I,K)=BV(I,K)+DALPH*BU(I,K)
  597 CONTINUE
  599 CONTINUE
      ITER=ITER+1
      GO TO 330
  600 WRITE(6,610)
  610 FORMAT(5X,'OPTIMAL FOR ALL ALPHA GREATER THAN THE CURRENT VALUE')
      GO TO 700
  620 WRITE (6,630)
  630 FORMAT (5X,'INVERSION FAILED')
  700 CALL TIME(1,1)
      STOP
      END
```

```
C***
C*** M) CODE FOR PARAMETRIC MDP WITH SELECTION RULE (C)
C***
      REAL*8 TALPH,DALPH,ALPHA,A,B,T,V,DET,COND,U,RV,DU,BV,BU,PV,PU,
     1OBV,OBU,BVIK,BUIK,OOBU,BVMX,BVMN,BUMX,BUMN,RBV,DBJ,DJMX,PDUMX,
     2DUMAX,EXALP,SALPH,ADEPS,SBEPS
      DIMENSION V(50),R(50,50),P(50,50,50),IIPOL(50),D(50,50),A(50,50),
     1T(50,50),B(50,1),IPERM(100),KKPOL(50),U(50),RV(50),DU(50),
     2BV(50,50),BU(50,50),BUI(50)
      REWIND8
      REWIND9
      ERR=0.001
      EPSLN=0.000001
      NDIMA=50
      NDIMBX=50
      NDIMT=50
      NSOL=1
      READ(5,1) NST,NAC,DISCR,ALPMN,ALPMX
    1 FORMAT(2I5,3F10.5)
      ALPHA=ALPMN
      READ(8)((R(I,K),D(I,K),I=1,NST),K=1,NAC)
      ITER=0
      ITR=0
      DO 2 I=1,NST
      READ(9)((P(I,K,J),K=1,NAC),J=1,NST)
    2 CONTINUE
      DO 50 I=1,NST
      DO 20 K=1,NAC
      RDIK=R(I,K)+ALPHA*D(I,K)
      IF(K.EQ.1) GO TO 10
      IF(ORI.GE.RDIK) GO TO 20
   10 ORI=RDIK
      IIP=K
   20 CONTINUE
      B(I,1)=ORI
      KKPOL(I)=IIP
      DO 40 J=1,NST
      IF(I.EQ.J) GO TO 30
      A(I,J)=-DISCR*P(I,IIP,J)
      GO TO 40
   30 A(I,J)=1.-DISCR*P(I,IIP,J)
   40 CONTINUE
   50 CONTINUE
      GO TO 130
   60 ITER=ITER+1
      DO 120 I=1,NST
      DO 110 K=1,NAC
      PV=0.
      DO 100 J=1,NST
      PV=PV+P(I,K,J)*V(J)
  100 CONTINUE
      RPVIK=R(I,K)+ALPHA*D(I,K)+DISCR*PV
      IF(K.EQ.1) GO TO 105
      IF(RVI.GE.RPVIK) GO TO 110
  105 RVI=RPVIK
      IIP=K
  110 CONTINUE
      DO 115 J=1,NST
      IF(I.EQ.J) GO TO 113
      A(I,J)=-DISCR*P(I,IIP,J)
```

```
      GO TO 115
  113 A(I,J)=1.-DISCR*P(I,IIP,J)
  115 CONTINUE
      B(I,1)=R(I,IIP)+ALPHA*D(I,IIP)
      IIPOL(I)=IIP
  120 CONTINUE
      DO 125 I=1,NST
      IF(IIPOL(I).NE.KKPOL(I)) GO TO 127
  125 CONTINUE
      GO TO 190
  127 DO 129 I=1,NST
      KKPOL(I)=IIPOL(I)
  129 CONTINUE
C*** SOLVE THE SYSTEM OF EQUATIONS AV=B BY CALLING SUBROUTINE SLE
  130 CALL SLE(NST,NDIMA,A,NSOL,NDIMBX,B,V,IPERM,NDIMT,T,DET,JEXP)
      IF(DET) 135,175,135
  135 GO TO 60
  175 WRITE(6,180)
  180 FORMAT(5X,'SOLUTION FAILED')
      GO TO 700
  190 DO 200 I=1,NST
      IIP=IIPOL(I)
      PV=0.
      DO 192 J=1,NST
      PV=PV+P(I,IIP,J)*V(J)
  192 CONTINUE
      RV(I)=R(I,IIP)+ALPHA*D(I,IIP)+DISCR*PV
      RBV=RV(I)-V(I)
      IF(I.EQ.1) GO TO 198
      IF(RBV.GT.BVMX) GO TO 194
      IF(RBV.LT.BVMN) GO TO 196
      GO TO 200
  194 BVMX=RBV
      GO TO 200
  196 BVMN=RBV
      GO TO 200
  198 BVMX=RBV
      BVMN=RBV
  200 CONTINUE
      VERR=BVMX-BVMN
      IF(ITR.EQ.0.AND.VERR.LE.ERR) GO TO 206
      ADD=DISCR*BVMN/(1.-DISCR)
      DO 202 I=1,NST
      V(I)=RV(I)+ADD
  202 CONTINUE
      IF(VERR.LE.ERR) GO TO 204
      ITR=ITR+1
      GO TO 190
  204 WRITE(6,205) ITR
  205 FORMAT(5X,'NUMBER OF ITERATION IN T-OPERATION FOR V IS',5X,I5)
      GO TO 60
  206 WRITE(6,205) ITR
      WRITE(6,208)ALPHA
  208 FORMAT(5X,'ALPHA =',F20.14)
      ADD=BVMN/(1.-DISCR)
      DO 209 I=1,NST
      V(I)=V(I)+ADD
  209 CONTINUE
      WRITE(6,210)
  210 FORMAT(10X,'STATE',10X,'POLICY',15X,'RETURN',10X,'OPT. POLICY')
```

```
      WRITE(6,220)(I,NAC,V(I),IIPOL(I),I=1,NST)
  220 FORMAT(10X,I3,12X,I3,10X,E16.7,10X,I3/)
      WRITE(6,230)
  230 FORMAT(10X,'DISCR',10X,'ITERATION')
      WRITE(6,240)DISCR,ITER
  240 FORMAT(10X,F7.5,6X,I5)
      WRITE(6,242)
  242 FORMAT(///1X,'*** OPTIMAL ACTION(S) TO BE CHOSEN FOR THE CURRENT
     1OPTIMAL POLICY'///)
C*** FIND THE INVERSE OF MATRIX A BY CALLING SUBROUTINE INV
      CALL INV(NST,NDIMA,A,IPERM,NDIMT,T,DET,JEXP,COND)
      IF(DET)245,620,245
  245 DO 250 I=1,NST
      TD=0.
      DO 247 J=1,NST
      IIP=IIPOL(J)
      TD=TD+T(I,J)*D(J,IIP)
  247 CONTINUE
      U(I)=TD
  250 CONTINUE
      ITR=0
  260 DO 300 I=1,NST
      IIP=IIPOL(I)
      PU=0.
      DO 265 J=1,NST
      PU=PU+P(I,IIP,J)*U(J)
  265 CONTINUE
      DU(I)=D(I,IIP)+DISCR*PU
      DBU=DU(I)-U(I)
      IF(I.EQ.1) GO TO 290
      IF(DBU.GT.BUMX) GO TO 270
      IF(DBU.LT.BUMN) GO TO 280
      GO TO 300
  270 BUMX=DBU
      GO TO 300
  280 BUMN=DBU
      GO TO 300
  290 BUMX=DBU
      BUMN=DBU
  300 CONTINUE
      UERR=BUMX-BUMN
      IF(UERR.LE.ERR) GO TO 310
      DO 305 I=1,NST
      U(I)=DU(I)
  305 CONTINUE
      ITR=ITR+1
      GO TO 260
  310 ADD=DISCR*BUMN/(1.-DISCR)
      DO 320 I=1,NST
      U(I)=DU(I)+ADD
  320 CONTINUE
      WRITE(6,590) ITR
      DO 325 I=1,NST
      DO 323 K=1,NAC
      PV=0.
      DO 321 J=1,NST
      PV=PV+P(I,K,J)*V(J)
  321 CONTINUE
      BV(I,K)=R(I,K)+ALPHA*D(I,K)+DISCR*PV-V(I)
  323 CONTINUE
```

```
325 CONTINUE
    CALL TIME(0)
    ITER=1
330 DO 340 I=1,NST
    DO 335 K=1,NAC
    PU=0.
    DO 333 J=1,NST
    PU=PU+P(I,K,J)*U(J)
333 CONTINUE
    BU(I,K)=D(I,K)+DISCR*PU-U(I)
335 CONTINUE
340 CONTINUE
    KN=0
    DO 380 I=1,NST
    IIP=IIPOL(I)
    OBV=BV(I,IIP)
    OBU=BU(I,IIP)
    DO 370 K=1,NAC
    IF(K.EQ.IIP) GO TO 370
    BVIK=BV(I,K)
    BUIK=BU(I,K)
    IF(BUIK.LE.OBU) GO TO 370
    IF(BVIK.GT.OBV) GO TO 360
    KN=KN+1
    BUIK=BUIK-OBU
    TALPH=(-BVIK+OBV)/BUIK
    IF(KN.EQ.1) GO TO 350
    IF(DALPH.LT.TALPH) GO TO 370
    IF(DALPH.GT.TALPH) GO TO 350
    IF(BUIK.LE.OOBU) GO TO 370
350 DALPH=TALPH
    IFRST=I
    KFRST=K
    OOBU=BUIK
    GO TO 370
360 WRITE(6,365)I,K,BVIK,BUIK,OBV,OBU
365 FORMAT(5X,'BETTER POLICY EXIST',5X,2I5,4E16.7)
370 CONTINUE
380 CONTINUE
385 IF(KN.EQ.0) GO TO 600
    ALPHA=ALPHA+DALPH
    IIPOL(IFRST)=KFRST
    WRITE(6,390)ALPHA,IFRST,KFRST
390 FORMAT(5X,'ALPHA =',F20.14,5X,'I =',I5,5X,'K =',I5)
    IF(ALPHA.GE.ALPMX) GO TO 700
    IF(ALPHA.LT.ALPMN) GO TO 700
    DO 410 J=1,NST
    IF(J.EQ.IFRST) GO TO 400
    RV(J)=-DISCR*P(IFRST,KFRST,J)
    GO TO 410
400 RV(J)=1.-DISCR*P(IFRST,KFRST,J)
410 CONTINUE
    DO 430 I=1,NST
    PU=0.
    DO 420 J=1,NST
    PU=PU+RV(J)*T(J,I)
420 CONTINUE
    DU(I)=PU
430 CONTINUE
    PU=DU(IFRST)
```

```
      IF(PU.EQ.0.) GO TO 620
      DO 450 J=1,NST
      IF(J.EQ.IFRST) GO TO 440
      DU(J)=-DU(J)/PU
      GO TO 450
  440 DU(J)=1/PU
  450 CONTINUE
      DO 480 I=1,NST
      PV=T(I,IFRST)
      DO 470 J=1,NST
      IF(J.EQ.IFRST) GO TO 460
      T(I,J)=T(I,J)+PV*DU(J)
      GO TO 470
  460 T(I,J)=PV*DU(J)
  470 CONTINUE
  480 CONTINUE
      DO 490 I=1,NST
      V(I)=V(I)+DALPH*U(I)
      PU=0.
      DO 485 J=1,NST
      IIP=IIPOL(J)
      PU=PU+T(I,J)*BU(J,IIP)
  485 CONTINUE
      U(I)=U(I)+PU
  490 CONTINUE
      ITR=0
  500 DO 550 I=1,NST
      IIP=IIPOL(I)
      PU=0.
      DO 510 J=1,NST
      PU=PU+P(I,IIP,J)*U(J)
  510 CONTINUE
      DU(I)=D(I,IIP)+DISCR*PU
      DBU=DU(I)-U(I)
      IF(I.EQ.1) GO TO 540
      IF(DBU.GT.BUMX) GO TO 520
      IF(DBU.LT.BUMN) GO TO 530
      GO TO 550
  520 BUMX=DBU
      GO TO 550
  530 BUMN=DBU
      GO TO 550
  540 BUMX=DBU
      BUMN=DBU
  550 CONTINUE
      UERR=BUMX-BUMN
      IF(UERR.LE.ERR) GO TO 570
      DO 560 I=1,NST
      U(I)=DU(I)
  560 CONTINUE
      ITR=ITR+1
      GO TO 500
  570 ADD=DISCR*BUMN/(1.-DISCR)
      DO 580 I=1,NST
      U(I)=DU(I)+ADD
  580 CONTINUE
      WRITE(6,590) ITR
  590 FORMAT(5X,'NUMBER OF ITERATION IN S-OPERATION FOR U IS',5X,I5)
      DO 599 I=1,NST
      DO 597 K=1,NAC
```

```
      BV(I,K)=BV(I,K)+DALPH*BU(I,K)
  597 CONTINUE
  599 CONTINUE
      ITER=ITER+1
      GO TO 330
  600 WRITE(6,610)
  610 FORMAT(5X,'OPTIMAL FOR ALL ALPHA GREATER THAN THE CURRENT VALUE')
      GO TO 700
  620 WRITE (6,630)
  630 FORMAT (5X,'INVERSION FAILED')
  700 CALL TIME(1,1)
      STOP
      END
```

```
C***
C*** N) CODE FOR PARAMETRIC MDP WITH PROCEDURE (I-1)
C***                                                                    140.
      REAL*8 TALPH,DALPH,ALPHA,A,B,T,V,DET,COND,U,RV,DJ,BV,BU,PV,PU,
     1OBV,OBU,BVIK,BUIK,OOBU,BVMX,BVMN,BUMX,BUMN,RBV,DBU,DJMX,PDUMX,
     2DUMAX,EXALP,SALPH,ADEPS,SBEPS
      DIMENSION V(50),R(50,50),P(50,50,50),IIPOL(50),D(50,50),A(50,50),
     1T(50,50),B(50,1),IPERM(100),KKPOL(50),U(50),RV(50),DU(50),
     2BV(50,50),BU(50,50),FLAG(50,50),BUI(50)
      REWIND8
      REWIND9
      ERR=0.001
      EPSLN=0.000001
      NDIMA=50
      NDIMBX=50
      NDIMT=50
      NSOL=1
      READ(5,1) NST,NAC,DISCR,ALPMN,ALPMX
    1 FORMAT(2I5,3F10.5)
      ALPHA=ALPMN
      EXALP=ALPMX
      TNAC=NST*NAC
      ELM=0.
      APRNT=0.
      READ(8)((R(I,K),D(I,K),I=1,NST),K=1,NAC)
      ITER=0
      ITR=0
      DO 2 I=1,NST
      READ(9)((P(I,K,J),K=1,NAC),J=1,NST)
    2 CONTINUE
      DO 50 I=1,NST
      DO 20 K=1,NAC
      FLAG(I,K)=-1
      RDIK=R(I,K)+ALPHA*D(I,K)
      IF(K.EQ.1) GO TO 10
      IF(ORI.GE.RDIK) GO TO 20
   10 ORI=RDIK
      IIP=K
   20 CONTINUE
      B(I,1)=ORI
      KKPOL(I)=IIP
      DO 40 J=1,NST
      IF(I.EQ.J) GO TO 30
      A(I,J)=-DISCR*P(I,IIP,J)
      GO TO 40
   30 A(I,J)=1.-DISCR*P(I,IIP,J)
   40 CONTINUE
   50 CONTINUE
      GO TO 130
   60 ITER=ITER+1
      DO 120 I=1,NST
      DO 110 K=1,NAC
      PV=0.
      DO 100 J=1,NST
      PV=PV+P(I,K,J)*V(J)
  100 CONTINUE
      RPVIK=R(I,K)+ALPHA*D(I,K)+DISCR*PV
      IF(K.EQ.1) GO TO 105
      IF(RVI.GE.RPVIK) GO TO 110
  105 RVI=RPVIK
```

```
      IIP=K
  110 CONTINUE
      DO 115 J=1,NST
      IF(I.EQ.J) GO TO 113
      A(I,J)=-DISCR*P(I,IIP,J)
      GO TO 115
  113 A(I,J)=1.-DISCR*P(I,IIP,J)
  115 CONTINUE
      B(I,1)=R(I,IIP)+ALPHA*D(I,IIP)
      IIPOL(I)=IIP
  120 CONTINUE
      DO 125 I=1,NST
      IF(IIPOL(I).NE.KKPOL(I)) GO TO 127
  125 CONTINUE
      GO TO 190
  127 DO 129 I=1,NST
      KKPOL(I)=IIPOL(I)
  129 CONTINUE
C*** SOLVE THE SYSTEM OF EQUATIONS AV=B BY CALLING SUBROUTINE SLE
  130 CALL SLE(NST,NDIMA,A,NSOL,NDIMBX,B,V,IPERM,NDIMT,T,DET,JEXP)
      IF(DET) 135,175,135
  135 GO TO 60
  175 WRITE(6,180)
  180 FORMAT(5X,'SOLUTION FAILED')
      GO TO 700
  190 DO 200 I=1,NST
      IIP=IIPOL(I)
      PV=0.
      DO 192 J=1,NST
      PV=PV+P(I,IIP,J)*V(J)
  192 CONTINUE
      RV(I)=R(I,IIP)+ALPHA*D(I,IIP)+DISCR*PV
      RBV=RV(I)-V(I)
      IF(I.EQ.1) GO TO 198
      IF(RBV.GT.BVMX) GO TO 194
      IF(RBV.LT.BVMN) GO TO 196
      GO TO 200
  194 BVMX=RBV
      GO TO 200
  196 BVMN=RBV
      GO TO 200
  198 BVMX=RBV
      BVMN=RBV
  200 CONTINUE
      VERR=BVMX-BVMN
      IF(ITR.EQ.0.AND.VERR.LE.ERR) GO TO 206
      ADD=DISCR*BVMN/(1.-DISCR)
      DO 202 I=1,NST
      V(I)=RV(I)+ADD
  202 CONTINUE
      IF(VERR.LE.ERR) GO TO 204
      ITR=ITR+1
      GO TO 190
  204 WRITE(6,205) ITR
  205 FORMAT(5X,'NUMBER OF ITERATION IN T-OPERATION FOR V IS',5X,I5)
      GO TO 60
  206 WRITE(6,205) ITR
      WRITE(6,208)ALPHA
  208 FORMAT(5X,'ALPHA =',F20.14)
      ADD=BVMN/(1.-DISCR)
```

```
      DO 209 I=1,NST
      V(I)=V(I)+ADD
  209 CONTINUE
      WRITE(6,210)
  210 FORMAT(10X,'STATE',10X,'POLICY',15X,'RETURN',10X,'OPT. POLICY')
      WRITE(6,220)(I,NAC,V(I),IIPOL(I),I=1,NST)
  220 FORMAT(10X,I3,12X,I3,10X,E16.7,10X,I3/)
      WRITE(6,230)
  230 FORMAT(10X,'DISCR',10X,'ITERATION')
      WRITE(6,240)DISCR,ITER
  240 FORMAT(10X,F7.5,6X,I5)
      WRITE(6,242)
  242 FORMAT(///1X,'*** OPTIMAL ACTION(S) TO BE CHOSEN FOR THE CURRENT
     1OPTIMAL POLICY'///)
C*** FIND THE INVERSE OF MATRIX A BY CALLING SUBROUTINE INV
      CALL INV(NST,NDIMA,A,IPERM,NDIMT,T,DET,JEXP,COND)
      IF(DET)245,620,245
  245 DO 250 I=1,NST
      TD=0.
      DO 247 J=1,NST
      IIP=IIPOL(J)
      TD=TD+T(I,J)*D(J,IIP)
  247 CONTINUE
      U(I)=TD
  250 CONTINUE
      ITR=0
  260 DO 300 I=1,NST
      IIP=IIPOL(I)
      PU=0.
      DO 265 J=1,NST
      PU=PU+P(I,IIP,J)*U(J)
  265 CONTINUE
      DU(I)=D(I,IIP)+DISCR*PU
      DBU=DU(I)-U(I)
      IF(I.EQ.1) GO TO 290
      IF(DBU.GT.BUMX) GO TO 270
      IF(DBU.LT.BUMN) GO TO 280
      GO TO 300
  270 BUMX=DBU
      GO TO 300
  280 BUMN=DBU
      GO TO 300
  290 BUMX=DBU
      BUMN=DBU
  300 CONTINUE
      UERR=BUMX-BUMN
      IF(UERR.LE.ERR) GO TO 310
      DO 305 I=1,NST
      U(I)=DU(I)
  305 CONTINUE
      ITR=ITR+1
      GO TO 260
  310 ADD=DISCR*BUMN/(1.-DISCR)
      DO 320 I=1,NST
      U(I)=DU(I)+ADD
  320 CONTINUE
      WRITE(6,590) ITR
      DO 325 I=1,NST
      DO 323 K=1,NAC
      PV=0.
```

```
      DO 321 J=1,NST
      PV=PV+P(I,K,J)*V(J)
  321 CONTINUE
      BV(I,K)=R(I,K)+ALPHA*D(I,K)+DISCR*PV-V(I)
  323 CONTINUE
  325 CONTINUE
      CALL TIME(0)
      ITER=1
  330 DO 340 I=1,NST
      B(I,1)=U(I)
      DO 335 K=1,NAC
      PU=0.
      DO 333 J=1,NST
      PU=PU+P(I,K,J)*U(J)
  333 CONTINUE
      BU(I,K)=D(I,K)+DISCR*PU-U(I)
  335 CONTINUE
  340 CONTINUE
      KN=0
      DO 380 I=1,NST
      IIP=IIPOL(I)
      OBV=BV(I,IIP)
      OBU=BU(I,IIP)
      DO 370 K=1,NAC
      IF(K.EQ.IIP) GO TO 370
      BVIK=BV(I,K)
      BUIK=BU(I,K)
      IF(BUIK.LE.OBU) GO TO 370
      IF(BVIK.GT.OBV) GO TO 360
      KN=KN+1
      TALPH=(-BVIK+OBV)/(BUIK-OBU)
      IF(KN.EQ.1) GO TO 350
      ADEPS=DALPH+EPSLN
      SBEPS=DALPH-EPSLN
      IF(TALPH.GE.ADEPS) GO TO 370
      IF(TALPH.LE.SBEPS) GO TO 350
      IF(BUIK.LE.BUI(I)) GO TO 370
      KKPOL(I)=K
      BUI(I)=BUIK
      GO TO 370
  350 DALPH=TALPH
      DO 357 J=1,NST
      IF(J.EQ.I) GO TO 355
      KKPOL(J)=IIPOL(J)
      BUI(J)=0.
      GO TO 357
  355 KKPOL(J)=K
      BUI(J)=BUIK
  357 CONTINUE
      GO TO 370
  360 WRITE(6,365)I,K,BVIK,BUIK,OBV,OBU
  365 FORMAT(5X,'BETTER POLICY EXIST',5X,2I5,4E16.7)
  370 CONTINUE
  380 CONTINUE
  385 IF(KN.EQ.0) GO TO 600
      ALPHA=ALPHA+DALPH
      DO 393 I=1,NST
      IF(KKPOL(I).EQ.IIPOL(I)) GO TO 393
      WRITE(6,390)ALPHA,I,KKPOL(I)
  390 FORMAT(5X,'ALPHA =',F20.14,5X,'I =',I5,5X,'K =',I5)
```

```
393 CONTINUE
    PRCNT=(ELM/TNAC)*100.
    WRITE(6,394) PRCNT
394 FORMAT(5X,'PRCNT',5X,F10.6)
    APRNT=APRNT+PRCNT
    ELM=0.
    IF(ALPHA.GE.ALPMX) GO TO 700
    IF(ALPHA.LT.ALPMN) GO TO 700
399 DO 950 K=1,NST
    IF(KKPOL(K).EQ.IIPOL(K)) GO TO 900
    IFRST=K
    KFRST=KKPOL(K)
    DO 410 J=1,NST
    IF(J.EQ.IFRST) GO TO 400
    RV(J)=-DISCR*P(IFRST,KFRST,J)
    GO TO 410
400 RV(J)=1.-DISCR*P(IFRST,KFRST,J)
410 CONTINUE
    DO 430 I=1,NST
    PU=0.
    DO 420 J=1,NST
    PU=PU+RV(J)*T(J,I)
420 CONTINUE
    DU(I)=PU
430 CONTINUE
    PU=DU(IFRST)
    IF(PU.EQ.0.) GO TO 620
    DO 450 J=1,NST
    IF(J.EQ.IFRST) GO TO 440
    DU(J)=-DU(J)/PU
    GO TO 450
440 DU(J)=1/PU
450 CONTINUE
    DO 480 I=1,NST
    PV=T(I,IFRST)
    DO 470 J=1,NST
    IF(J.EQ.IFRST) GO TO 460
    T(I,J)=T(I,J)+PV*DU(J)
    GO TO 470
460 T(I,J)=PV*DU(J)
470 CONTINUE
480 CONTINUE
900 IIPOL(K)=KKPOL(K)
950 CONTINUE
    DO 490 I=1,NST
    V(I)=V(I)+DALPH*U(I)
    PU=0.
    DO 485 J=1,NST
    IIP=IIPOL(J)
    PU=PU+T(I,J)*BU(J,IIP)
485 CONTINUE
    U(I)=U(I)+PU
490 CONTINUE
    ITR=0
500 DO 550 I=1,NST
    IIP=IIPOL(I)
    PU=0.
    DO 510 J=1,NST
    PU=PU+P(I,IIP,J)*U(J)
510 CONTINUE
```

```
      DU(I)=D(I,IIP)+DISCR*PU
      DBU=DU(I)-U(I)
      IF(I.EQ.1) GO TO 540
      IF(DBU.GT.BUMX) GO TO 520
      IF(DBU.LT.BUMN) GO TO 530
      GO TO 550
  520 BUMX=DBU
      GO TO 550
  530 BUMN=DBU
      GO TO 550
  540 BUMX=DBU
      BUMN=DBU
  550 CONTINUE
      UERR=BUMX-BUMN
      IF(UERR.LE.ERR) GO TO 570
      DO 560 I=1,NST
      U(I)=DU(I)
  560 CONTINUE
      ITR=ITR+1
      GO TO 500
  570 ADD=DISCR*BUMN/(1.-DISCR)
      DO 580 I=1,NST
      U(I)=DU(I)+ADD
      DU(I)=U(I)-B(I,1)
      IF(I.EQ.1) GO TO 575
      IF(DU(I).LE.DUMX) GO TO 580
  575 DUMX=DU(I)
  580 CONTINUE
      WRITE(6,590) ITR
  590 FORMAT(5X,'NUMBER OF ITERATION IN S-OPERATION FOR U IS',5X,I5)
      DUMX=DISCR*DUMX
      DO 599 I=1,NST
      DO 597 K=1,NAC
      BV(I,K)=BV(I,K)+DALPH*BU(I,K)
      BU(I,K)=BU(I,K)+DUMX-DU(I)
  597 CONTINUE
  599 CONTINUE
      ITER=ITER+1
  820 KN=0
      DO 880 I=1,NST
      B(I,1)=U(I)
      IIP=IIPOL(I)
      PU=0.
      DO 830 J=1,NST
      PU=PU+P(I,IIP,J)*U(J)
  830 CONTINUE
      BU(I,IIP)=D(I,IIP)+DISCR*PU-U(I)
      OBV=BV(I,IIP)
      OBU=BU(I,IIP)
      DBU=OBV+(EXALP-ALPHA)*OBU
      DO 870 K=1,NAC
      IF(K.EQ.IIP) GO TO 870
      RBV=BV(I,K)+(EXALP-ALPHA)*BU(I,K)
      IF(FLAG(I,K).LE.0.) GO TO 840
      IF(RBV.LT.DBU) GO TO 853
      FLAG(I,K)=-1.
      PV=0.
      PU=0.
      DO 835 J=1,NST
      PV=PV+P(I,K,J)*V(J)
```

```
      PU=PU+P(I,K,J)*U(J)
  835 CONTINUE
      BV(I,K)=R(I,K)+ALPHA*D(I,K)+DISCR*PV-V(I)
      BU(I,K)=D(I,K)+DISCR*PU-U(I)
      GO TO 855
  840 IF(RBV.LT.OBU) GO TO 850
      PU=0.
      DO 845 J=1,NST
      PU=PU+P(I,K,J)*U(J)
  845 CONTINUE
      BU(I,K)=D(I,K)+DISCR*PU-U(I)
      GO TO 855
  850 FLAG(I,K)=1.
  853 ELM=ELM+1.
      GO TO 870
  855 BVIK=BV(I,K)
      BUIK=BU(I,K)
      IF(BUIK.LE.OBU) GO TO 870
      IF(BVIK.GT.OBV) GO TO 860
      KN=KN+1
      TALPH=(-BVIK+OBV)/(BUIK-OBU)
      IF(KN.EQ.1) GO TO 857
      ADEPS=DALPH+EPSLN
      SBEPS=DALPH-EPSLN
      IF(TALPH.GE.ADEPS) GO TO 870
      IF(TALPH.LE.SBEPS) GO TO 857
      IF(BUIK.LE.BUI(I)) GO TO 870
      KKPOL(I)=K
      BUI(I)=BUIK
      GO TO 870
  857 DALPH=TALPH
      DO 859 J=1,NST
      IF(J.EQ.I) GO TO 858
      KKPOL(J)=IIPOL(J)
      BUI(J)=0.
      GO TO 859
  858 KKPOL(J)=K
      BUI(J)=BUIK
  859 CONTINUE
      GO TO 870
  860 WRITE(6,865)I,K,BVIK,BUIK,OBV,OBU
  865 FORMAT(5X,'BETTER POLICY EXIST',5X,2I5,4E16.7)
  870 CONTINUE
  880 CONTINUE
      GO TO 385
  600 WRITE(6,610)
  610 FORMAT(5X,'OPTIMAL FOR ALL ALPHA GREATER THAN THE CJRRENT VALUE')
      GO TO 700
  620 WRITE (6,630)
  630 FORMAT (5X,'INVERSION FAILED')
  700 CALL TIME(1,1)
      PV=ITER
      APRNT=APRNT/PV
      WRITE(6,750) APRNT
  750 FORMAT(5X,'AVERAGE PRCNT =',5X,F10.6)
      STOP
      END
```

```
C***
C*** O) CODE FOR PARAMETRIC MDP WITH PROCEDURE (I-2)
C***
      REAL*8 TALPH,DALPH,ALPHA,A,B,T,V,DET,COND,U,RV,DJ,BV,BU,PV,PU,
     1OBV,OBU,BVIK,BUIK,OOBJ,BVMX,BVMN,BUMX,BUMN,RBV,DBJ,DJMX,PDUMX,
     2DUMAX,EXALP,SALPH,ADEPS,SBEPS
      DIMENSION V(50),R(50,50),P(50,50,50),IIPOL(50),D(50,50),A(50,50),
     1T(50,50),B(50,1),IPERM(100),KKPOL(50),U(50),RV(50),DJ(50),
     2BV(50,50),BU(50,50),FLAG(50,50),DUMX(50),PDUMX(50),BUI(50)
      REWIND8
      REWIND9
      ERR=0.001
      EPSLN=0.000001
      NDIMA=50
      NDIMBX=50
      NDIMT=50
      NSOL=1
      READ(5,1) NST,NAC,DISCR,ALPMN,ALPMX
    1 FORMAT(2I5,3F10.5)
      ALPHA=ALPMN
      EXALP=ALPMX
      TNAC=NST*NAC
      ELM=0.
      APRNT=0.
      READ(8)((R(I,K),D(I,K),I=1,NST),K=1,NAC)
      ITER=0
      ITR=0
      DO 2 I=1,NST
      READ(9)((P(I,K,J),K=1,NAC),J=1,NST)
    2 CONTINUE
      DO 50 I=1,NST
      DO 20 K=1,NAC
      FLAG(I,K)=-1
      RDIK=R(I,K)+ALPHA*D(I,K)
      IF(K.EQ.1) GO TO 10
      IF(ORI.GE.RDIK) GO TO 20
   10 ORI=RDIK
      IIP=K
   20 CONTINUE
      B(I,1)=ORI
      KKPOL(I)=IIP
      DO 40 J=1,NST
      IF(I.EQ.J) GO TO 30
      A(I,J)=-DISCR*P(I,IIP,J)
      GO TO 40
   30 A(I,J)=1.-DISCR*P(I,IIP,J)
   40 CONTINUE
   50 CONTINUE
      GO TO 130
   60 ITER=ITER+1
      DO 120 I=1,NST
      DO 110 K=1,NAC
      PV=0.
      DO 100 J=1,NST
      PV=PV+P(I,K,J)*V(J)
  100 CONTINUE
      RPVIK=R(I,K)+ALPHA*D(I,K)+DISCR*PV
      IF(K.EQ.1) GO TO 105
      IF(RVI.GE.RPVIK) GO TO 110
  105 RVI=RPVIK
```

```
      IIP=K
  110 CONTINUE
      DO 115 J=1,NST
      IF(I.EQ.J) GO TO 113
      A(I,J)=-DISCR*P(I,IIP,J)
      GO TO 115
  113 A(I,J)=1.-DISCR*P(I,IIP,J)
  115 CONTINUE
      B(I,1)=R(I,IIP)+ALPHA*D(I,IIP)
      IIPOL(I)=IIP
  120 CONTINUE
      DO 125 I=1,NST
      IF(IIPOL(I).NE.KKPOL(I)) GO TO 127
  125 CONTINUE
      GO TO 190
  127 DO 129 I=1,NST
      KKPOL(I)=IIPOL(I)
  129 CONTINUE
C*** SOLVE THE SYSTEM OF EQUATIONS AV=B BY CALLING SUBROUTINE SLE
  130 CALL SLE(NST,NDIMA,A,NSOL,NDIMBX,B,V,IPERM,NDIMT,T,DET,JEXP)
      IF(DET) 135,175,135
  135 GO TO 60
  175 WRITE(6,180)
  180 FORMAT(5X,'SOLUTION FAILED')
      GO TO 700
  190 DO 200 I=1,NST
      IIP=IIPOL(I)
      PV=0.
      DO 192 J=1,NST
      PV=PV+P(I,IIP,J)*V(J)
  192 CONTINUE
      RV(I)=R(I,IIP)+ALPHA*D(I,IIP)+DISCR*PV
      RBV=RV(I)-V(I)
      IF(I.EQ.1) GO TO 198
      IF(RBV.GT.BVMX) GO TO 194
      IF(RBV.LT.BVMN) GO TO 196
      GO TO 200
  194 BVMX=RBV
      GO TO 200
  196 BVMN=RBV
      GO TO 200
  198 BVMX=RBV
      BVMN=RBV
  200 CONTINUE
      VERR=BVMX-BVMN
      IF(ITR.EQ.0.AND.VERR.LE.ERR) GO TO 206
      ADD=DISCR*BVMN/(1.-DISCR)
      DO 202 I=1,NST
      V(I)=RV(I)+ADD
  202 CONTINUE
      IF(VERR.LE.ERR) GO TO 204
      ITR=ITR+1
      GO TO 190
  204 WRITE(6,205) ITR
  205 FORMAT(5X,'NUMBER OF ITERATION IN T-OPERATION FOR V IS',5X,I5)
      GO TO 60
  206 WRITE(6,205) ITR
      WRITE(6,208)ALPHA
  208 FORMAT(5X,'ALPHA =',F20.14)
      ADD=BVMN/(1.-DISCR)
```

```
      DO 209 I=1,NST
      V(I)=V(I)+ADD
  209 CONTINUE
      WRITE(6,210)
  210 FORMAT(10X,'STATE',10X,'POLICY',15X,'RETURN',10X,'OPT. POLICY')
      WRITE(6,220)(I,NAC,V(I),IIPOL(I),I=1,NST)
  220 FORMAT(10X,I3,12X,I3,10X,E16.7,10X,I3/)
      WRITE(6,230)
  230 FORMAT(10X,'DISCR',10X,'ITERATION')
      WRITE(6,240)DISCR,ITER
  240 FORMAT(10X,F7.5,6X,I5)
      WRITE(6,242)
  242 FORMAT(///1X,'*** OPTIMAL ACTION(S) TO BE CHOSEN FOR THE CURRENT
     1OPTIMAL POLICY'///)
C*** FIND THE INVERSE OF MATRIX A BY CALLING SUBROUTINE INV
      CALL INV(NST,NDIMA,A,IPERM,NDIMT,T,DET,JEXP,COND)
      IF(DET)245,620,245
  245 DO 250 I=1,NST
      TD=0.
      DO 247 J=1,NST
      IIP=IIPOL(J)
      TD=TD+T(I,J)*D(J,IIP)
  247 CONTINUE
      U(I)=TD
  250 CONTINUE
      ITR=0
  260 DO 300 I=1,NST
      IIP=IIPOL(I)
      PU=0.
      DO 265 J=1,NST
      PU=PU+P(I,IIP,J)*U(J)
  265 CONTINUE
      DU(I)=D(I,IIP)+DISCR*PU
      DBU=DU(I)-U(I)
      IF(I.EQ.1) GO TO 290
      IF(DBU.GT.BUMX) GO TO 270
      IF(DBU.LT.BUMN) GO TO 280
      GO TO 300
  270 BUMX=DBU
      GO TO 300
  280 BUMN=DBU
      GO TO 300
  290 BUMX=DBU
      BUMN=DBU
  300 CONTINUE
      UERR=BUMX-BUMN
      IF(UERR.LE.ERR) GO TO 310
      DO 305 I=1,NST
      U(I)=DU(I)
  305 CONTINUE
      ITR=ITR+1
      GO TO 260
  310 ADD=DISCR*BUMN/(1.-DISCR)
      DO 320 I=1,NST
      U(I)=DU(I)+ADD
  320 CONTINUE
      WRITE(6,590) ITR
      DO 325 I=1,NST
      DO 323 K=1,NAC
      PV=0.
```

```
      DO 321 J=1,NST
      PV=PV+P(I,K,J)*V(J)
  321 CONTINUE
      BV(I,K)=R(I,K)+ALPHA*D(I,K)+DISCR*PV-V(I)
  323 CONTINUE
  325 CONTINUE
      CALL TIME(0)
      ITER=1
  330 DO 340 I=1,NST
      A(I,ITER)=U(I)
      DO 335 K=1,NAC
      PU=0.
      DO 333 J=1,NST
      PU=PU+P(I,K,J)*U(J)
  333 CONTINUE
      BU(I,K)=D(I,K)+DISCR*PU-U(I)
  335 CONTINUE
  340 CONTINUE
      KN=0
      DO 380 I=1,NST
      IIP=IIPOL(I)
      OBV=BV(I,IIP)
      OBU=BU(I,IIP)
      DO 370 K=1,NAC
      IF(K.EQ.IIP) GO TO 370
      BVIK=BV(I,K)
      BUIK=BU(I,K)
      IF(BUIK.LE.OBU) GO TO 370
      IF(BVIK.GT.OBV) GO TO 360
      KN=KN+1
      TALPH=(-BVIK+OBV)/(BUIK-OBU)
      IF(KN.EQ.1) GO TO 350
      ADEPS=DALPH+EPSLN
      SBEPS=DALPH-EPSLN
      IF(TALPH.GE.ADEPS) GO TO 370
      IF(TALPH.LE.SBEPS) GO TO 350
      IF(BUIK.LE.BUI(I)) GO TO 370
      KKPOL(I)=K
      BUI(I)=BUIK
      GO TO 370
  350 DALPH=TALPH
      DO 357 J=1,NST
      IF(J.EQ.I) GO TO 355
      KKPOL(J)=IIPOL(J)
      BUI(J)=0.
      GO TO 357
  355 KKPOL(J)=K
      BUI(J)=BUIK
  357 CONTINUE
      GO TO 370
  360 WRITE(6,365)I,K,BVIK,BUIK,OBV,OBU
  365 FORMAT(5X,'BETTER POLICY EXIST',5X,2I5,4E16.7)
  370 CONTINUE
  380 CONTINUE
  385 IF(KN.EQ.0) GO TO 600
      ALPHA=ALPHA+DALPH
      DO 393 I=1,NST
      IF(KKPOL(I).EQ.IIPOL(I)) GO TO 393
      WRITE(6,390)ALPHA,I,KKPOL(I)
  390 FORMAT(5X,'ALPHA =',F20.14,5X,'I =',I5,5X,'K =',I5)
```

```
393 CONTINUE
    PRCNT=(ELM/TNAC)*100.
    WRITE(6,394) PRCNT
394 FORMAT(5X,'PRCNT',5X,F10.6)
    APRNT=APRNT+PRCNT
    ELM=0.
    IF(ALPHA.GE.ALPMX) GO TO 700
    IF(ALPHA.LT.ALPMN) GO TO 700
399 DO 950 K=1,NST
    IF(KKPOL(K).EQ.IIPOL(K)) GO TO 900
    IFRST=K
    KFRST=KKPOL(K)
    DO 410 J=1,NST
    IF(J.EQ.IFRST) GO TO 400
    RV(J)=-DISCR*P(IFRST,KFRST,J)
    GO TO 410
400 RV(J)=1.-DISCR*P(IFRST,KFRST,J)
410 CONTINUE
    DO 430 I=1,NST
    PU=0.
    DO 420 J=1,NST
    PU=PU+RV(J)*T(J,I)
420 CONTINUE
    DU(I)=PU
430 CONTINUE
    PU=DU(IFRST)
    IF(PU.EQ.0.) GO TO 620
    DO 450 J=1,NST
    IF(J.EQ.IFRST) GO TO 440
    DU(J)=-DU(J)/PU
    GO TO 450
440 DU(J)=1/PU
450 CONTINUE
    DO 480 I=1,NST
    PV=T(I,IFRST)
    DO 470 J=1,NST
    IF(J.EQ.IFRST) GO TO 460
    T(I,J)=T(I,J)+PV*DU(J)
    GO TO 470
460 T(I,J)=PV*DU(J)
470 CONTINUE
480 CONTINUE
900 IIPOL(K)=KKPOL(K)
950 CONTINUE
    DO 490 I=1,NST
    V(I)=V(I)+DALPH*U(I)
    PU=0.
    DO 485 J=1,NST
    IIP=IIPOL(J)
    PU=PU+T(I,J)*BU(J,IIP)
485 CONTINUE
    U(I)=U(I)+PU
490 CONTINUE
    ITR=0
500 DO 550 I=1,NST
    IIP=IIPOL(I)
    PU=0.
    DO 510 J=1,NST
    PU=PU+P(I,IIP,J)*U(J)
510 CONTINUE
```

```
      DU(I)=D(I,IIP)+DISCR*PU
      DBU=DU(I)-U(I)
      IF(I.EQ.1) GO TO 540
      IF(DBU.GT.BUMX) GO TO 520
      IF(DBU.LT.BUMN) GO TO 530
      GO TO 550
  520 BUMX=DBU
      GO TO 550
  530 BUMN=DBU
      GO TO 550
  540 BUMX=DBU
      BUMN=DBU
  550 CONTINUE
      UERR=BUMX-BUMN
      IF(UERR.LE.ERR) GO TO 570
      DO 560 I=1,NST
      U(I)=DU(I)
  560 CONTINUE
      ITR=ITR+1
      GO TO 500
  570 ADD=DISCR*BUMN/(1.-DISCR)
      DO 580 I=1,NST
      U(I)=DU(I)+ADD
  580 CONTINUE
      WRITE(6,590) ITR
  590 FORMAT(5X,'NUMBER OF ITERATION IN S-OPERATION FOR U IS',5X,I5)
      DO 594 K=1,ITER
      DO 592 I=1,NST
      DUIK=U(I)-A(I,K)
      IF(K.EQ.ITER) DU(I)=DUIK
      IF(I.EQ.1) GO TO 591
      IF(DUMAX.GE.DUIK) GO TO 592
  591 DUMAX=DUIK
  592 CONTINUE
      IF(ITER.EQ.1) GO TO 593
      IF(K.EQ.ITER) GO TO 593
      PDUMX(K)=DUMX(K)
  593 DUMX(K)=DISCR*DUMAX
  594 CONTINUE
      DO 599 I=1,NST
      DO 597 K=1,NAC
      BV(I,K)=BV(I,K)+DALPH*BU(I,K)
      L=FLAG(I,K)
      IF(L.LE.0.OR.L.EQ.ITER) GO TO 595
      BU(I,K)=BU(I,K)+DUMX(L)-PDUMX(L)-DU(I)
      GO TO 597
  595 BU(I,K)=BU(I,K)+DUMX(ITER)-DU(I)
  597 CONTINUE
  599 CONTINUE
      ITER=ITER+1
  820 KN=0
      DO 880 I=1,NST
      A(I,ITER)=U(I)
      IIP=IIPOL(I)
      PU=0.
      DO 830 J=1,NST
      PU=PU+P(I,IIP,J)*U(J)
  830 CONTINUE
      BU(I,IIP)=D(I,IIP)+DISCR*PU-U(I)
      OBV=BV(I,IIP)
```

```
      OBU=BU(I,IIP)
      DBU=OBV+(EXALP-ALPHA)*OBU
      DO 870 K=1,NAC
      IF(K.EQ.IIP) GO TO 870
      RBV=BV(I,K)+(EXALP-ALPHA)*BU(I,K)
      IF(FLAG(I,K).LE.0.) GO TO 840
      IF(RBV.LT.DBU) GO TO 853
      FLAG(I,K)=-1.
      PV=0.
      PU=0.
      DO 835 J=1,NST
      PV=PV+P(I,K,J)*V(J)
      PU=PU+P(I,K,J)*U(J)
  835 CONTINUE
      BV(I,K)=R(I,K)+ALPHA*D(I,K)+DISCR*PV-V(I)
      BU(I,K)=D(I,K)+DISCR*PU-U(I)
      GO TO 855
  840 IF(RBV.LT.DBU) GO TO 850
      PU=0.
      DO 845 J=1,NST
      PU=PU+P(I,K,J)*U(J)
  845 CONTINUE
      BU(I,K)=D(I,K)+DISCR*PU-U(I)
      GO TO 855
  850 FLAG(I,K)=ITER-1
  853 ELM=ELM+1.
      GO TO 870
  855 BVIK=BV(I,K)
      BUIK=BU(I,K)
      IF(BUIK.LE.OBU) GO TO 870
      IF(BVIK.GT.OBV) GO TO 860
      KN=KN+1
      TALPH=(-BVIK+OBV)/(BUIK-OBU)
      IF(KN.EQ.1) GO TO 857
      ADEPS=DALPH+EPSLN
      SBEPS=DALPH-EPSLN
      IF(TALPH.GE.ADEPS) GO TO 870
      IF(TALPH.LE.SBEPS) GO TO 857
      IF(BUIK.LE.BUI(I)) GO TO 870
      KKPOL(I)=K
      BUI(I)=BUIK
      GO TO 870
  857 DALPH=TALPH
      DO 859 J=1,NST
      IF(J.EQ.I) GO TO 858
      KKPOL(J)=IIPOL(J)
      BUI(J)=0.
      GO TO 859
  858 KKPOL(J)=K
      BUI(J)=BUIK
  859 CONTINUE
      GO TO 870
  860 WRITE(6,865)I,K,BVIK,BUIK,OBV,OBU
  865 FORMAT(5X,'BETTER POLICY EXIST',5X,2I5,4E16.7)
  870 CONTINUE
  880 CONTINUE
      GO TO 385
  600 WRITE(6,610)
  610 FORMAT(5X,'OPTIMAL FOR ALL ALPHA GREATER THAN THE CURRENT VALUE')
      GO TO 700
```

```
620 WRITE (6,630)
630 FORMAT (5X,'INVERSION FAILED')
700 CALL TIME(1,1)
    PV=ITER
    APRNT=APRNT/PV
    WRITE(6,750) APRNT
750 FORMAT(5X,'AVERAGE PRCNT =',5X,F10.6)
    STOP
    END
```

```
C***
C*** P) CODE FOR PARAMETRIC MDP WITH PROCEDURE (II-1)
C***
      REAL*8 TALPH,DALPH,ALPHA,A,B,T,V,DET,COND,U,RV,DJ,BV,BJ,PV,PU,
     1OBV,OBU,BVIK,BUIK,OOBU,BVMX,BVMN,BUMX,BUMN,RBV,DBJ,DUMX,PDUMX,
     2DUMAX,EXALP,SALPH,ADEPS,SBEPS
      DIMENSION V(50),R(50,50),P(50,50,50),IIPOL(50),D(50,50),A(50,50),
     1T(50,50),B(50,1),IPERM(100),KKPOL(50),U(50),RV(50),DJ(50),
     2BV(50,50),BU(50,50),FLAG(50,50),BUI(50)
      REWIND8
      REWIND9
      ERR=0.001
      EPSLN=0.000001
      NDIMA=50
      NDIMBX=50
      NDIMT=50
      NSOL=1
      READ(5,1) NST,NAC,DISCR,ALPMN,ALPMX
    1 FORMAT(2I5,3F10.5)
      ALPHA=ALPMN
      TNAC=NST*NAC
      ELM=0.
      APRNT=0.
      READ(8)((R(I,K),D(I,K),I=1,NST),K=1,NAC)
      ITER=0
      ITR=0
      DO 2 I=1,NST
      READ(9)((P(I,K,J),K=1,NAC),J=1,NST)
    2 CONTINUE
      DO 50 I=1,NST
      DO 20 K=1,NAC
      FLAG(I,K)=-1
      RDIK=R(I,K)+ALPHA*D(I,K)
      IF(K.EQ.1) GO TO 10
      IF(ORI.GE.RDIK) GO TO 20
   10 ORI=RDIK
      IIP=K
   20 CONTINUE
      B(I,1)=ORI
      KKPOL(I)=IIP
      DO 40 J=1,NST
      IF(I.EQ.J) GO TO 30
      A(I,J)=-DISCR*P(I,IIP,J)
      GO TO 40
   30 A(I,J)=1.-DISCR*P(I,IIP,J)
   40 CONTINUE
   50 CONTINUE
      GO TO 130
   60 ITER=ITER+1
      DO 120 I=1,NST
      DO 110 K=1,NAC
      PV=0.
      DO 100 J=1,NST
      PV=PV+P(I,K,J)*V(J)
  100 CONTINUE
      RPVIK=R(I,K)+ALPHA*D(I,K)+DISCR*PV
      IF(K.EQ.1) GO TO 105
      IF(RVI.GE.RPVIK) GO TO 110
  105 RVI=RPVIK
      IIP=K
```

```
  110 CONTINUE
      DO 115 J=1,NST
      IF(I.EQ.J) GO TO 113
      A(I,J)=-DISCR*P(I,IIP,J)
      GO TO 115
  113 A(I,J)=1.-DISCR*P(I,IIP,J)
  115 CONTINUE
      B(I,1)=R(I,IIP)+ALPHA*D(I,IIP)
      IIPOL(I)=IIP
  120 CONTINUE
      DO 125 I=1,NST
      IF(IIPOL(I).NE.KKPOL(I)) GO TO 127
  125 CONTINUE
      GO TO 190
  127 DO 129 I=1,NST
      KKPOL(I)=IIPOL(I)
  129 CONTINUE
C*** SOLVE THE SYSTEM OF EQUATIONS AV=B BY CALLING SUBROUTINE SLE
  130 CALL SLE(NST,NDIMA,A,NSOL,NDIMBX,B,V,IPERM,NDIMT,T,DET,JEXP)
      IF(DET) 135,175,135
  135 GO TO 60
  175 WRITE(6,180)
  180 FORMAT(5X,'SOLUTION FAILED')
      GO TO 700
  190 DO 200 I=1,NST
      IIP=IIPOL(I)
      PV=0.
      DO 192 J=1,NST
      PV=PV+P(I,IIP,J)*V(J)
  192 CONTINUE
      RV(I)=R(I,IIP)+ALPHA*D(I,IIP)+DISCR*PV
      RBV=RV(I)-V(I)
      IF(I.EQ.1) GO TO 198
      IF(RBV.GT.BVMX) GO TO 194
      IF(RBV.LT.BVMN) GO TO 196
      GO TO 200
  194 BVMX=RBV
      GO TO 200
  196 BVMN=RBV
      GO TO 200
  198 BVMX=RBV
      BVMN=RBV
  200 CONTINUE
      VERR=BVMX-BVMN
      IF(ITR.EQ.0.AND.VERR.LE.ERR) GO TO 206
      ADD=DISCR*BVMN/(1.-DISCR)
      DO 202 I=1,NST
      V(I)=RV(I)+ADD
  202 CONTINUE
      IF(VERR.LE.ERR) GO TO 204
      ITR=ITR+1
      GO TO 190
  204 WRITE(6,205) ITR
  205 FORMAT(5X,'NUMBER OF ITERATION IN T-OPERATION FOR V IS',5X,I5)
      GO TO 60
  206 WRITE(6,205) ITR
      WRITE(6,208)ALPHA
  208 FORMAT(5X,'ALPHA =',F20.14)
      ADD=BVMN/(1.-DISCR)
      DO 209 I=1,NST
```

```
      V(I)=V(I)+ADD
  209 CONTINUE
      WRITE(6,210)
  210 FORMAT(10X,'STATE',10X,'POLICY',15X,'RETURN',10X,'OPT. POLICY')
      WRITE(6,220)(I,NAC,V(I),IIPOL(I),I=1,NST)
  220 FORMAT(10X,I3,12X,I3,10X,E16.7,10X,I3/)
      WRITE(6,230)
  230 FORMAT(10X,'DISCR',10X,'ITERATION')
      WRITE(6,240)DISCR,ITER
  240 FORMAT(10X,F7.5,6X,I5)
      WRITE(6,242)
  242 FORMAT(///1X,'*** OPTIMAL ACTION(S) TO BE CHOSEN FOR THE CURRENT
     1OPTIMAL POLICY'///)
C*** FIND THE INVERSE OF MATRIX A BY CALLING SUBROUTINE INV
      CALL INV(NST,NDIMA,A,IPERM,NDIMT,T,DET,JEXP,COND)
      IF(DET)245,620,245
  245 DO 250 I=1,NST
      TD=0.
      DO 247 J=1,NST
      IIP=IIPOL(J)
      TD=TD+T(I,J)*D(J,IIP)
  247 CONTINUE
      U(I)=TD
  250 CONTINUE
      ITR=0
  260 DO 300 I=1,NST
      IIP=IIPOL(I)
      PU=0.
      DO 265 J=1,NST
      PU=PU+P(I,IIP,J)*U(J)
  265 CONTINUE
      DU(I)=D(I,IIP)+DISCR*PU
      DBU=DU(I)-U(I)
      IF(I.EQ.1) GO TO 290
      IF(DBU.GT.BUMX) GO TO 270
      IF(DBU.LT.BUMN) GO TO 280
      GO TO 300
  270 BUMX=DBU
      GO TO 300
  280 BUMN=DBU
      GO TO 300
  290 BUMX=DBU
      BUMN=DBU
  300 CONTINUE
      UERR=BUMX-BUMN
      IF(UERR.LE.ERR) GO TO 310
      DO 305 I=1,NST
      U(I)=DU(I)
  305 CONTINUE
      ITR=ITR+1
      GO TO 260
  310 ADD=DISCR*BUMN/(1.-DISCR)
      DO 320 I=1,NST
      U(I)=DU(I)+ADD
  320 CONTINUE
      WRITE(6,590) ITR
      DO 325 I=1,NST
      DO 323 K=1,NAC
      PV=0.
      DO 321 J=1,NST
```

```
      PV=PV+P(I,K,J)*V(J)
  321 CONTINUE
      BV(I,K)=R(I,K)+ALPHA*D(I,K)+DISCR*PV-V(I)
  323 CONTINUE
  325 CONTINUE
      CALL TIME(0)
      ITER=1
  330 DO 340 I=1,NST
      B(I,1)=U(I)
      DO 335 K=1,NAC
      PU=0.
      DO 333 J=1,NST
      PU=PU+P(I,K,J)*U(J)
  333 CONTINUE
      BU(I,K)=D(I,K)+DISCR*PU-U(I)
  335 CONTINUE
  340 CONTINUE
      KN=0
      DO 380 I=1,NST
      IIP=IIPOL(I)
      OBV=BV(I,IIP)
      OBU=BU(I,IIP)
      DO 370 K=1,NAC
      IF(K.EQ.IIP) GO TO 370
      BVIK=BV(I,K)
      BUIK=BU(I,K)
      IF(BUIK.LE.OBU) GO TO 370
      IF(BVIK.GT.OBV) GO TO 360
      KN=KN+1
      TALPH=(-BVIK+OBV)/(BUIK-OBU)
      IF(KN.EQ.1) GO TO 350
      ADEPS=DALPH+EPSLN
      SBEPS=DALPH-EPSLN
      IF(TALPH.GE.ADEPS) GO TO 370
      IF(TALPH.LE.SBEPS) GO TO 350
      IF(BUIK.LE.BUI(I)) GO TO 370
      KKPOL(I)=K
      BUI(I)=BUIK
      GO TO 370
  350 DALPH=TALPH
      DO 357 J=1,NST
      IF(J.EQ.I) GO TO 355
      KKPOL(J)=IIPOL(J)
      BUI(J)=0.
      GO TO 357
  355 KKPOL(J)=K
      BUI(J)=BUIK
  357 CONTINUE
      GO TO 370
  360 WRITE(6,365)I,K,BVIK,BUIK,OBV,OBU
  365 FORMAT(5X,'BETTER POLICY EXIST',5X,2I5,4E16.7)
  370 CONTINUE
  380 CONTINUE
  385 IF(KN.EQ.0) GO TO 600
      ALPHA=ALPHA+DALPH
      DO 393 I=1,NST
      IF(KKPOL(I).EQ.IIPOL(I)) GO TO 393
      WRITE(6,390)ALPHA,I,KKPOL(I)
  390 FORMAT(5X,'ALPHA =',F20.14,5X,'I =',I5,5X,'K =',I5)
  393 CONTINUE
```

```
      PRCNT=(ELM/TNAC)*100.
      WRITE(6,394) PRCNT
  394 FORMAT(5X,'PRCNT',5X,F10.6)
      APRNT=APRNT+PRCNT
      ELM=0.
      IF(ALPHA.GE.ALPMX) GO TO 700
      IF(ALPHA.LT.ALPMN) GO TO 700
      IF(KN.EQ.1) GO TO 398
      KN=0
      DO 397 I=1,NST
      KKP=KKPOL(I)
      OOBU=BU(I,KKP)
      IIP=IIPOL(I)
      OBV=BV(I,IIP)
      OBU=BU(I,IIP)
      DO 396 K=1,NAC
      IF(K.EQ.IIP) GO TO 396
      IF(FLAG(I,K).GT.0.) GO TO 396
      BUIK=BU(I,K)
      IF(BUIK.LE.OOBU) GO TO 396
      BVIK=BV(I,K)
      IF(BVIK.GT.OBV) GO TO 396
      BUIK=BUIK-OBU
      TALPH=(OBV-BVIK)/BUIK
      KN=KN+1
      IF(KN.EQ.1) GO TO 395
      IF(SALPH.LT.TALPH) GO TO 396
      IF(SALPH.GT.TALPH) GO TO 395
      IF(BUIK.LE.PU) GO TO 396
  395 SALPH=TALPH
      ISCND=I
      KSCND=K
      PU=BUIK
  396 CONTINUE
  397 CONTINUE
      GO TO 399
  398 KN=0
  399 DO 950 K=1,NST
      IF(KKPOL(K).EQ.IIPOL(K)) GO TO 900
      IFRST=K
      KFRST=KKPOL(K)
      DO 410 J=1,NST
      IF(J.EQ.IFRST) GO TO 400
      RV(J)=-DISCR*P(IFRST,KFRST,J)
      GO TO 410
  400 RV(J)=1.-DISCR*P(IFRST,KFRST,J)
  410 CONTINUE
      DO 430 I=1,NST
      PU=0.
      DO 420 J=1,NST
      PU=PU+RV(J)*T(J,I)
  420 CONTINUE
      DU(I)=PU
  430 CONTINUE
      PU=DU(IFRST)
      IF(PU.EQ.0.) GO TO 620
      DO 450 J=1,NST
      IF(J.EQ.IFRST) GO TO 440
      DU(J)=-DU(J)/PU
      GO TO 450
```

```
440 DU(J)=1/PU
450 CONTINUE
    DO 480 I=1,NST
    PV=T(I,IFRST)
    DO 470 J=1,NST
    IF(J.EQ.IFRST) GO TO 460
    T(I,J)=T(I,J)+PV*DU(J)
    GO TO 470
460 T(I,J)=PV*DU(J)
470 CONTINUE
480 CONTINUE
900 IIPOL(K)=KKPOL(K)
950 CONTINUE
    DO 490 I=1,NST
    V(I)=V(I)+DALPH*U(I)
    PU=0.
    DO 485 J=1,NST
    IIP=IIPOL(J)
    PU=PU+T(I,J)*BU(J,IIP)
485 CONTINUE
    U(I)=U(I)+PU
490 CONTINUE
    ITR=0
500 DO 550 I=1,NST
    IIP=IIPOL(I)
    PU=0.
    DO 510 J=1,NST
    PU=PU+P(I,IIP,J)*U(J)
510 CONTINUE
    DU(I)=D(I,IIP)+DISCR*PU
    DBU=DU(I)-U(I)
    IF(I.EQ.1) GO TO 540
    IF(DBU.GT.BUMX) GO TO 520
    IF(DBU.LT.BUMN) GO TO 530
    GO TO 550
520 BUMX=DBU
    GO TO 550
530 BUMN=DBU
    GO TO 550
540 BUMX=DBU
    BUMN=DBU
550 CONTINUE
    UERR=BUMX-BUMN
    IF(UERR.LE.ERR) GO TO 570
    DO 560 I=1,NST
    U(I)=DU(I)
560 CONTINUE
    ITR=ITR+1
    GO TO 500
570 ADD=DISCR*BUMN/(1.-DISCR)
    DO 580 I=1,NST
    U(I)=DU(I)+ADD
    DU(I)=U(I)-B(I,1)
    IF(I.EQ.1) GO TO 575
    IF(DU(I).LE.DUMX) GO TO 580
575 DUMX=DU(I)
580 CONTINUE
    WRITE(6,590) ITR
590 FORMAT(5X,'NUMBER OF ITERATION IN S-OPERATION FOR U IS',5X,I5)
    DUMX=DISCR*DUMX
```

```
      DO 599 I=1,NST
      DO 597 K=1,NAC
      BV(I,K)=BV(I,K)+DALPH*BU(I,K)
      BU(I,K)=BU(I,K)+DUMX-DU(I)
  597 CONTINUE
  599 CONTINUE
      ITER=ITER+1
      IF(KN.EQ.0) GO TO 810
      IIP=IIPOL(ISCND)
      PU=0.
      PV=0.
      DO 800 J=1,NST
      PU=PU+P(ISCND,IIP,J)*U(J)
      PV=PV+P(ISCND,KSCND,J)*U(J)
  800 CONTINUE
      PU=D(ISCND,IIP)+DISCR*PU-U(ISCND)
      PV=D(ISCND,KSCND)+DISCR*PV-U(ISCND)
      IF(PU.GE.PV) GO TO 810
      EXALP=ALPHA+(BV(ISCND,IIP)-BV(ISCND,KSCND))/(PV-PU)
      IF(EXALP.LT.ALPHA.OR.EXALP.GT.ALPMX) GO TO 810
      GO TO 820
  810 EXALP=ALPMX
  820 KN=0
      DO 880 I=1,NST
      B(I,1)=U(I)
      IIP=IIPOL(I)
      PU=0.
      DO 830 J=1,NST
      PU=PU+P(I,IIP,J)*U(J)
  830 CONTINUE
      BU(I,IIP)=D(I,IIP)+DISCR*PU-U(I)
      OBV=BV(I,IIP)
      OBU=BU(I,IIP)
      DBU=OBV+(EXALP-ALPHA)*OBU
      DO 870 K=1,NAC
      IF(K.EQ.IIP) GO TO 870
      RBV=BV(I,K)+(EXALP-ALPHA)*BU(I,K)
      IF(FLAG(I,K).LE.0.) GO TO 840
      IF(RBV.LT.DBU) GO TO 853
      FLAG(I,K)=-1.
      PV=0.
      PU=0.
      DO 835 J=1,NST
      PV=PV+P(I,K,J)*V(J)
      PU=PU+P(I,K,J)*U(J)
  835 CONTINUE
      BV(I,K)=R(I,K)+ALPHA*D(I,K)+DISCR*PV-V(I)
      BU(I,K)=D(I,K)+DISCR*PU-U(I)
      GO TO 855
  840 IF(RBV.LT.DBU) GO TO 850
      PU=0.
      DO 845 J=1,NST
      PU=PU+P(I,K,J)*U(J)
  845 CONTINUE
      BU(I,K)=D(I,K)+DISCR*PU-U(I)
      GO TO 855
  850 FLAG(I,K)=1.
  853 ELM=ELM+1.
      GO TO 870
  855 BVIK=BV(I,K)
```

```
      BUIK=BU(I,K)
      IF(BUIK.LE.OBU) GO TO 870
      IF(BVIK.GT.OBV) GO TO 860
      KN=KN+1
      TALPH=(-BVIK+OBV)/(BUIK-OBU)
      IF(KN.EQ.1) GO TO 857
      ADEPS=DALPH+EPSLN
      SBEPS=DALPH-EPSLN
      IF(TALPH.GE.ADEPS) GO TO 870
      IF(TALPH.LE.SBEPS) GO TO 857
      IF(BUIK.LE.BUI(I)) GO TO 870
      KKPOL(I)=K
      BUI(I)=BUIK
      GO TO 870
857   DALPH=TALPH
      DO 859 J=1,NST
      IF(J.EQ.I) GO TO 858
      KKPOL(J)=IIPOL(J)
      BUI(J)=0.
      GO TO 859
858   KKPOL(J)=K
      BUI(J)=BUIK
859   CONTINUE
      GO TO 870
860   WRITE(6,865)I,K,BVIK,BUIK,OBV,OBU
865   FORMAT(5X,'BETTER POLICY EXIST',5X,2I5,4E16.7)
870   CONTINUE
880   CONTINUE
      GO TO 385
600   WRITE(6,610)
610   FORMAT(5X,'OPTIMAL FOR ALL ALPHA GREATER THAN THE CURRENT VALUE')
      GO TO 700
620   WRITE (6,630)
630   FORMAT (5X,'INVERSION FAILED')
700   CALL TIME(1,1)
      PV=ITER
      APRNT=APRNT/PV
      WRITE(6,750) APRNT
750   FORMAT(5X,'AVERAGE PRCNT =',5X,F10.6)
      STOP
      END
```

```
C***
C*** Q) CODE FOR PARAMETRIC MDP WITH PROCEDURE (II-2)
C***
      REAL*8 TALPH,DALPH,ALPHA,A,B,T,V,DET,COND,U,RV,DJ,BV,BU,PV,PU,
     1OBV,OBU,BVIK,BUIK,OOBU,BVMX,BVMN,BUMX,BUMN,RBV,DBJ,DUMX,PDUMX,
     2DUMAX,EXALP,SALPH,ADEPS,SBEPS
      DIMENSION V(50),R(50,50),P(50,50,50),IIPOL(50),D(50,50),A(50,50),
     1T(50,50),B(50,1),IPERM(100),KKPOL(50),U(50),RV(50),DJ(50),
     2BV(50,50),BU(50,50),FLAG(50,50),DUMX(50),PDUMX(50),BUI(50)
      REWIND8
      REWIND9
      ERR=0.001
      EPSLN=0.000001
      NDIMA=50
      NDIMBX=50
      NDIMT=50
      NSOL=1
      READ(5,1) NST,NAC,DISCR,ALPMN,ALPMX
    1 FORMAT(2I5,3F10.5)
      ALPHA=ALPMN
      TNAC=NST*NAC
      ELM=0.
      APRNT=0.
      READ(8)((R(I,K),D(I,K),I=1,NST),K=1,NAC)
      ITER=0
      ITR=0
      DO 2 I=1,NST
      READ(9)((P(I,K,J),K=1,NAC),J=1,NST)
    2 CONTINUE
      DO 50 I=1,NST
      DO 20 K=1,NAC
      FLAG(I,K)=-1
      RDIK=R(I,K)+ALPHA*D(I,K)
      IF(K.EQ.1) GO TO 10
      IF(ORI.GE.RDIK) GO TO 20
   10 ORI=RDIK
      IIP=K
   20 CONTINUE
      B(I,1)=ORI
      KKPOL(I)=IIP
      DO 40 J=1,NST
      IF(I.EQ.J) GO TO 30
      A(I,J)=-DISCR*P(I,IIP,J)
      GO TO 40
   30 A(I,J)=1.-DISCR*P(I,IIP,J)
   40 CONTINUE
   50 CONTINUE
      GO TO 130
   60 ITER=ITER+1
      DO 120 I=1,NST
      DO 110 K=1,NAC
      PV=0.
      DO 100 J=1,NST
      PV=PV+P(I,K,J)*V(J)
  100 CONTINUE
      RPVIK=R(I,K)+ALPHA*D(I,K)+DISCR*PV
      IF(K.EQ.1) GO TO 105
      IF(RVI.GE.RPVIK) GO TO 110
  105 RVI=RPVIK
      IIP=K
```

```
  110 CONTINUE
      DO 115 J=1,NST
      IF(I.EQ.J) GO TO 113
      A(I,J)=-DISCR*P(I,IIP,J)
      GO TO 115
  113 A(I,J)=1.-DISCR*P(I,IIP,J)
  115 CONTINUE
      B(I,1)=R(I,IIP)+ALPHA*D(I,IIP)
      IIPOL(I)=IIP
  120 CONTINUE
      DO 125 I=1,NST
      IF(IIPOL(I).NE.KKPOL(I)) GO TO 127
  125 CONTINUE
      GO TO 190
  127 DO 129 I=1,NST
      KKPOL(I)=IIPOL(I)
  129 CONTINUE
C*** SOLVE THE SYSTEM OF EQUATIONS AV=B BY CALLING SUBROUTINE SLE
  130 CALL SLE(NST,NDIMA,A,NSOL,NDIMBX,B,V,IPERM,NDIMT,T,DET,JEXP)
      IF(DET) 135,175,135
  135 GO TO 60
  175 WRITE(6,180)
  180 FORMAT(5X,'SOLUTION FAILED')
      GO TO 700
  190 DO 200 I=1,NST
      IIP=IIPOL(I)
      PV=0.
      DO 192 J=1,NST
      PV=PV+P(I,IIP,J)*V(J)
  192 CONTINUE
      RV(I)=R(I,IIP)+ALPHA*D(I,IIP)+DISCR*PV
      RBV=RV(I)-V(I)
      IF(I.EQ.1) GO TO 198
      IF(RBV.GT.BVMX) GO TO 194
      IF(RBV.LT.BVMN) GO TO 196
      GO TO 200
  194 BVMX=RBV
      GO TO 200
  196 BVMN=RBV
      GO TO 200
  198 BVMX=RBV
      BVMN=RBV
  200 CONTINUE
      VERR=BVMX-BVMN
      IF(ITR.EQ.0.AND.VERR.LE.ERR) GO TO 206
      ADD=DISCR*BVMN/(1.-DISCR)
      DO 202 I=1,NST
      V(I)=RV(I)+ADD
  202 CONTINUE
      IF(VERR.LE.ERR) GO TO 204
      ITR=ITR+1
      GO TO 190
  204 WRITE(6,205) ITR
  205 FORMAT(5X,'NUMBER OF ITERATION IN T-OPERATION FOR V IS',5X,I5)
      GO TO 60
  206 WRITE(6,205) ITR
      WRITE(6,208)ALPHA
  208 FORMAT(5X,'ALPHA =',F20.14)
      ADD=BVMN/(1.-DISCR)
      DO 209 I=1,NST
```

```
      V(I)=V(I)+ADD
  209 CONTINUE
      WRITE(6,210)
  210 FORMAT(10X,'STATE',10X,'POLICY',15X,'RETURN',10X,'OPT. POLICY')
      WRITE(6,220)(I,NAC,V(I),IIPOL(I),I=1,NST)
  220 FORMAT(10X,I3,12X,I3,10X,E16.7,10X,I3/)
      WRITE(6,230)
  230 FORMAT(10X,'DISCR',10X,'ITERATION')
      WRITE(6,240)DISCR,ITER
  240 FORMAT(10X,F7.5,6X,I5)
      WRITE(6,242)
  242 FORMAT(///1X,'*** OPTIMAL ACTION(S) TO BE CHOSEN FOR THE CURRENT
     1OPTIMAL POLICY'///)
C*** FIND THE INVERSE OF MATRIX A BY CALLING SUBROUTINE INV
      CALL INV(NST,NDIMA,A,IPERM,NDIMT,T,DET,JEXP,COND)
      IF(DET)245,620,245
  245 DO 250 I=1,NST
      TD=0.
      DO 247 J=1,NST
      IIP=IIPOL(J)
      TD=TD+T(I,J)*D(J,IIP)
  247 CONTINUE
      U(I)=TD
  250 CONTINUE
      ITR=0
  260 DO 300 I=1,NST
      IIP=IIPOL(I)
      PU=0.
      DO 265 J=1,NST
      PU=PU+P(I,IIP,J)*U(J)
  265 CONTINUE
      DU(I)=D(I,IIP)+DISCR*PU
      DBU=DU(I)-U(I)
      IF(I.EQ.1) GO TO 290
      IF(DBU.GT.BUMX) GO TO 270
      IF(DBU.LT.BUMN) GO TO 280
      GO TO 300
  270 BUMX=DBU
      GO TO 300
  280 BUMN=DBU
      GO TO 300
  290 BUMX=DBU
      BUMN=DBU
  300 CONTINUE
      UERR=BUMX-BUMN
      IF(UERR.LE.ERR) GO TO 310
      DO 305 I=1,NST
      U(I)=DU(I)
  305 CONTINUE
      ITR=ITR+1
      GO TO 260
  310 ADD=DISCR*BUMN/(1.-DISCR)
      DO 320 I=1,NST
      U(I)=DU(I)+ADD
  320 CONTINUE
      WRITE(6,590) ITR
      DO 325 I=1,NST
      DO 323 K=1,NAC
      PV=0.
      DO 321 J=1,NST
```

```
      PV=PV+P(I,K,J)*V(J)
  321 CONTINUE
      BV(I,K)=R(I,K)+ALPHA*D(I,K)+DISCR*PV-V(I)
  323 CONTINUE
  325 CONTINUE
      CALL TIME(0)
      ITER=1
  330 DO 340 I=1,NST
      A(I,ITER)=U(I)
      DO 335 K=1,NAC
      PU=0.
      DO 333 J=1,NST
      PU=PU+P(I,K,J)*U(J)
  333 CONTINUE
      BU(I,K)=D(I,K)+DISCR*PU-U(I)
  335 CONTINUE
  340 CONTINUE
      KN=0
      DO 380 I=1,NST
      IIP=IIPOL(I)
      OBV=BV(I,IIP)
      OBU=BU(I,IIP)
      DO 370 K=1,NAC
      IF(K.EQ.IIP) GO TO 370
      BVIK=BV(I,K)
      BUIK=BU(I,K)
      IF(BUIK.LE.OBU) GO TO 370
      IF(BVIK.GT.OBV) GO TO 360
      KN=KN+1
      TALPH=(-BVIK+OBV)/(BUIK-OBU)
      IF(KN.EQ.1) GO TO 350
      ADEPS=DALPH+EPSLN
      SBEPS=DALPH-EPSLN
      IF(TALPH.GE.ADEPS) GO TO 370
      IF(TALPH.LE.SBEPS) GO TO 350
      IF(BUIK.LE.BUI(I)) GO TO 370
      KKPOL(I)=K
      BUI(I)=BUIK
      GO TO 370
  350 DALPH=TALPH
      DO 357 J=1,NST
      IF(J.EQ.I) GO TO 355
      KKPOL(J)=IIPOL(J)
      BUI(J)=0.
      GO TO 357
  355 KKPOL(J)=K
      BUI(J)=BUIK
  357 CONTINUE
      GO TO 370
  360 WRITE(6,365)I,K,BVIK,BUIK,OBV,OBU
  365 FORMAT(5X,'BETTER POLICY EXIST',5X,2I5,4E16.7)
  370 CONTINUE
  380 CONTINUE
  385 IF(KN.EQ.0) GO TO 600
      ALPHA=ALPHA+DALPH
      DO 393 I=1,NST
      IF(KKPOL(I).EQ.IIPOL(I)) GO TO 393
      WRITE(6,390)ALPHA,I,KKPOL(I)
  390 FORMAT(5X,'ALPHA =',F20.14,5X,'I =',I5,5X,'K =',I5)
  393 CONTINUE
```

```
      PRCNT=(ELM/TNAC)*100.
      WRITE(6,394) PRCNT
  394 FORMAT(5X,'PRCNT',5X,F10.6)
      APRNT=APRNT+PRCNT
      ELM=0.
      IF(ALPHA.GE.ALPMX) GO TO 700
      IF(ALPHA.LT.ALPMN) GO TO 700
      IF(KN.EQ.1) GO TO 398
      KN=0
      DO 397 I=1,NST
      KKP=KKPOL(I)
      OOBU=BU(I,KKP)
      IIP=IIPOL(I)
      OBV=BV(I,IIP)
      OBU=BU(I,IIP)
      DO 396 K=1,NAC
      IF(K.EQ.IIP) GO TO 396
      IF(FLAG(I,K).GT.0.) GO TO 396
      BUIK=BU(I,K)
      IF(BUIK.LE.OOBU) GO TO 396
      BVIK=BV(I,K)
      IF(BVIK.GT.OBV) GO TO 396
      BUIK=BUIK-OBU
      TALPH=(OBV-BVIK)/BUIK
      KN=KN+1
      IF(KN.EQ.1) GO TO 395
      IF(SALPH.LT.TALPH) GO TO 396
      IF(SALPH.GT.TALPH) GO TO 395
      IF(BUIK.LE.PU) GO TO 396
  395 SALPH=TALPH
      ISCND=I
      KSCND=K
      PU=BUIK
  396 CONTINUE
  397 CONTINUE
      GO TO 399
  398 KN=0
  399 DO 950 K=1,NST
      IF(KKPOL(K).EQ.IIPOL(K)) GO TO 900
      IFRST=K
      KFRST=KKPOL(K)
      DO 410 J=1,NST
      IF(J.EQ.IFRST) GO TO 400
      RV(J)=-DISCR*P(IFRST,KFRST,J)
      GO TO 410
  400 RV(J)=1.-DISCR*P(IFRST,KFRST,J)
  410 CONTINUE
      DO 430 I=1,NST
      PU=0.
      DO 420 J=1,NST
      PU=PU+RV(J)*T(J,I)
  420 CONTINUE
      DU(I)=PU
  430 CONTINUE
      PU=DU(IFRST)
      IF(PU.EQ.0.) GO TO 620
      DO 450 J=1,NST
      IF(J.EQ.IFRST) GO TO 440
      DU(J)=-DU(J)/PU
      GO TO 450
```

```
440 DU(J)=1/PU
450 CONTINUE
    DO 480 I=1,NST
    PV=T(I,IFRST)
    DO 470 J=1,NST
    IF(J.EQ.IFRST) GO TO 460
    T(I,J)=T(I,J)+PV*DU(J)
    GO TO 470
460 T(I,J)=PV*DU(J)
470 CONTINUE
480 CONTINUE
900 IIPOL(K)=KKPOL(K)
950 CONTINUE
    DO 490 I=1,NST
    V(I)=V(I)+DALPH*U(I)
    PU=0.
    DO 485 J=1,NST
    IIP=IIPOL(J)
    PU=PU+T(I,J)*BU(J,IIP)
485 CONTINUE
    U(I)=U(I)+PU
490 CONTINUE
    ITR=0
500 DO 550 I=1,NST
    IIP=IIPOL(I)
    PU=0.
    DO 510 J=1,NST
    PU=PU+P(I,IIP,J)*U(J)
510 CONTINUE
    DU(I)=D(I,IIP)+DISCR*PU
    DBU=DU(I)-U(I)
    IF(I.EQ.1) GO TO 540
    IF(DBU.GT.BUMX) GO TO 520
    IF(DBU.LT.BUMN) GO TO 530
    GO TO 550
520 BUMX=DBU
    GO TO 550
530 BUMN=DBU
    GO TO 550
540 BUMX=DBU
    BUMN=DBU
550 CONTINUE
    UERR=BUMX-BUMN
    IF(UERR.LE.ERR) GO TO 570
    DO 560 I=1,NST
    U(I)=DU(I)
560 CONTINUE
    ITR=ITR+1
    GO TO 500
570 ADD=DISCR*BUMN/(1.-DISCR)
    DO 580 I=1,NST
    U(I)=DU(I)+ADD
580 CONTINUE
    WRITE(6,590) ITR
590 FORMAT(5X,'NUMBER OF ITERATION IN S-OPERATION FOR U IS',5X,I5)
    DO 594 K=1,ITER
    DO 592 I=1,NST
    DUIK=U(I)-A(I,K)
    IF(K.EQ.ITER) DU(I)=DUIK
    IF(I.EQ.1) GO TO 591
```

```
      IF(DUMAX.GE.DUIK) GO TO 592
  591 DUMAX=DUIK
  592 CONTINUE
      IF(ITER.EQ.1) GO TO 593
      IF(K.EQ.ITER) GO TO 593
      PDUMX(K)=DUMX(K)
  593 DUMX(K)=DISCR*DUMAX
  594 CONTINUE
      DO 599 I=1,NST
      DO 597 K=1,NAC
      BV(I,K)=BV(I,K)+DALPH*BU(I,K)
      L=FLAG(I,K)
      IF(L.LE.0.OR.L.EQ.ITER) GO TO 595
      BU(I,K)=BU(I,K)+DUMX(L)-PDUMX(L)-DU(I)
      GO TO 597
  595 BU(I,K)=BU(I,K)+DUMX(ITER)-DU(I)
  597 CONTINUE
  599 CONTINUE
      ITER=ITER+1
      IF(KN.EQ.0) GO TO 810
      IIP=IIPOL(ISCND)
      PU=0.
      PV=0.
      DO 800 J=1,NST
      PU=PU+P(ISCND,IIP,J)*U(J)
      PV=PV+P(ISCND,KSCND,J)*U(J)
  800 CONTINUE
      PU=D(ISCND,IIP)+DISCR*PU-U(ISCND)
      PV=D(ISCND,KSCND)+DISCR*PV-U(ISCND)
      IF(PU.GE.PV) GO TO 810
      EXALP=ALPHA+(BV(ISCND,IIP)-BV(ISCND,KSCND))/(PV-PJ)
      IF(EXALP.LT.ALPHA.OR.EXALP.GT.ALPMX) GO TO 810
      GO TO 820
  810 EXALP=ALPMX
  820 KN=0
      DO 880 I=1,NST
      A(I,ITER)=U(I)
      IIP=IIPOL(I)
      PU=0.
      DO 830 J=1,NST
      PU=PU+P(I,IIP,J)*U(J)
  830 CONTINUE
      BU(I,IIP)=D(I,IIP)+DISCR*PU-U(I)
      OBV=BV(I,IIP)
      OBU=BU(I,IIP)
      DBU=OBV+(EXALP-ALPHA)*OBU
      DO 870 K=1,NAC
      IF(K.EQ.IIP) GO TO 870
      RBV=BV(I,K)+(EXALP-ALPHA)*BU(I,K)
      IF(FLAG(I,K).LE.0.) GO TO 840
      IF(RBV.LT.DBU) GO TO 853
      FLAG(I,K)=-1.
      PV=0.
      PU=0.
      DO 835 J=1,NST
      PV=PV+P(I,K,J)*V(J)
      PU=PU+P(I,K,J)*U(J)
  835 CONTINUE
      BV(I,K)=R(I,K)+ALPHA*D(I,K)+DISCR*PV-V(I)
      BU(I,K)=D(I,K)+DISCR*PU-U(I)
```

```
      GO TO 855
  840 IF(RBV.LT.OBU) GO TO 850
      PU=0.
      DO 845 J=1,NST
      PU=PU+P(I,K,J)*U(J)
  845 CONTINUE
      BU(I,K)=D(I,K)+DISCR*PU-U(I)
      GO TO 855
  850 FLAG(I,K)=ITER-1
  853 ELM=ELM+1.
      GO TO 870
  855 BVIK=BV(I,K)
      BUIK=BU(I,K)
      IF(BUIK.LE.OBU) GO TO 870
      IF(BVIK.GT.OBV) GO TO 860
      KN=KN+1
      TALPH=(-BVIK+OBV)/(BUIK-OBU)
      IF(KN.EQ.1) GO TO 857
      ADEPS=DALPH+EPSLN
      SBEPS=DALPH-EPSLN
      IF(TALPH.GE.ADEPS) GO TO 870
      IF(TALPH.LE.SBEPS) GO TO 857
      IF(BUIK.LE.BUI(I)) GO TO 870
      KKPOL(I)=K
      BUI(I)=BUIK
      GO TO 870
  857 DALPH=TALPH
      DO 859 J=1,NST
      IF(J.EQ.I) GO TO 858
      KKPOL(J)=IIPOL(J)
      BUI(J)=0.
      GO TO 859
  858 KKPOL(J)=K
      BUI(J)=BUIK
  859 CONTINUE
      GO TO 870
  860 WRITE(6,865)I,K,BVIK,BUIK,OBV,OBU
  865 FORMAT(5X,'BETTER POLICY EXIST',5X,2I5,4E16.7)
  870 CONTINUE
  880 CONTINUE
      GO TO 385
  600 WRITE(6,610)
  610 FORMAT(5X,'OPTIMAL FOR ALL ALPHA GREATER THAN THE CURRENT VALUE')
      GO TO 700
  620 WRITE (6,630)
  630 FORMAT (5X,'INVERSION FAILED')
  700 CALL TIME(1,1)
      PV=ITER
      APRNT=APRNT/PV
      WRITE(6,750) APRNT
  750 FORMAT(5X,'AVERAGE PRCNT =',5X,F10.6)
      STOP
      END
```

*** R) COMPUTATIONAL RESULT OF TWO CRITERION AUTOMOBILE REPLACEMENT
*** PROBLEM USING CODE L WITH DISCOUNT RATE OF .96

NUMBER OF ITERATION IN T-OPERATION FOR V IS 0
ALPHA = 0.0

STATE	POLICY	RETURN	OPT. POLICY
1	41	-0.2552761E+04	18
2	41	-0.2772761E+04	18
3	41	-0.2882761E+04	18
4	41	-0.3062761E+04	18
5	41	-0.3132761E+04	18
6	41	-0.3202761E+04	18
7	41	-0.3272761E+04	18
8	41	-0.3327237E+04	1
9	41	-0.3377924E+04	1
10	41	-0.3427125E+04	1
11	41	-0.3474379E+04	1
12	41	-0.3518796E+04	1
13	41	-0.3560623E+04	1
14	41	-0.3600088E+04	1
15	41	-0.3637403E+04	1
16	41	-0.3672761E+04	1
17	41	-0.3705251E+04	1
18	41	-0.3736024E+04	1
19	41	-0.3765222E+04	1
20	41	-0.3792979E+04	1
21	41	-0.3819420E+04	1
22	41	-0.3844663E+04	1
23	41	-0.3868648E+04	1
24	41	-0.3891045E+04	1
25	41	-0.3910764E+04	1
26	41	-0.3927791E+04	1

27	41	-0.3942019E+04	1
28	41	-0.3952761E+04	18
29	41	-0.3962761E+04	18
30	41	-0.3967761E+04	18
31	41	-0.3972761E+04	18
32	41	-0.3977761E+04	18
33	41	-0.3982761E+04	18
34	41	-0.3992761E+04	18
35	41	-0.3997761E+04	18
36	41	-0.4002761E+04	18
37	41	-0.4007761E+04	18
38	41	-0.4017761E+04	18
39	41	-0.4025761E+04	18
40	41	-0.4032761E+04	18

```
DISCR          ITERATION
0.96000           6
```

*** OPTIMAL ACTION(S) TO BE CHOSEN FOR THE CURRENT OPTIMAL POLICY

```
NUMBER OF ITERATION IN S-OPERATION FOR J IS          21
ALPHA =      0.00223520805536      I =     7       K =    1
NUMBER OF ITERATION IN S-OPERATION FOR U IS           0
ALPHA =      0.04782711267229      I =     6       K =    1
NUMBER OF ITERATION IN S-OPERATION FOR U IS           0
ALPHA =      0.05753709652306      I =    27       K =   18
NUMBER OF ITERATION IN S-OPERATION FOR U IS           0
ALPHA =      0.06824467713929      I =     5       K =    1
NUMBER OF ITERATION IN S-OPERATION FOR J IS           0
ALPHA =      0.07544087286022      I =     4       K =    1
NUMBER OF ITERATION IN S-OPERATION FOR J IS           0
ALPHA =      0.08670211920300      I =     1       K =   14
ALPHA =      0.08670211920300      I =     2       K =   14
ALPHA =      0.08670211920300      I =     3       K =   14
ALPHA =      0.08670211920300      I =    27       K =   14
ALPHA =      0.08670211920300      I =    28       K =   14
ALPHA =      0.08670211920300      I =    29       K =   14
ALPHA =      0.08670211920300      I =    30       K =   14
ALPHA =      0.08670211920300      I =    31       K =   14
ALPHA =      0.08670211920300      I =    32       K =   14
ALPHA =      0.08670211920300      I =    33       K =   14
ALPHA =      0.08670211920300      I =    34       K =   14
ALPHA =      0.08670211920300      I =    35       K =   14
```

```
ALPHA =     0.08670211920300      I =    36      K =    14
ALPHA =     0.08670211920300      I =    37      K =    14
ALPHA =     0.08670211920300      I =    38      K =    14
ALPHA =     0.08670211920300      I =    39      K =    14
ALPHA =     0.08670211920300      I =    40      K =    14
NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
ALPHA =     0.21459945512724      I =    26      K =    14
NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
ALPHA =     0.25049353888991      I =     1      K =     1
NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
ALPHA =     0.25565032155634      I =     3      K =     1
NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
ALPHA =     0.26155063770537      I =     2      K =     1
NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
ALPHA =     0.32894307829057      I =    25      K =    14
NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
ALPHA =     0.42463104236151      I =    25      K =     2
ALPHA =     0.42463104236151      I =    26      K =     2
ALPHA =     0.42463104236151      I =    27      K =     2
ALPHA =     0.42463104236151      I =    28      K =     2
ALPHA =     0.42463104236151      I =    29      K =     2
ALPHA =     0.42463104236151      I =    30      K =     2
ALPHA =     0.42463104236151      I =    31      K =     2
ALPHA =     0.42463104236151      I =    32      K =     2
ALPHA =     0.42463104236151      I =    33      K =     2
ALPHA =     0.42463104236151      I =    34      K =     2
ALPHA =     0.42463104236151      I =    35      K =     2
ALPHA =     0.42463104236151      I =    36      K =     2
ALPHA =     0.42463104236151      I =    37      K =     2
ALPHA =     0.42463104236151      I =    38      K =     2
ALPHA =     0.42463104236151      I =    39      K =     2
ALPHA =     0.42463104236151      I =    40      K =     2
NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
ALPHA =     0.42561664618784      I =    24      K =     2
NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
ALPHA =     0.44289342537689      I =    22      K =     2
NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
ALPHA =     0.44428425680768      I =    23      K =     2
NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
ALPHA =     0.45444687773132      I =    21      K =     2
NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
ALPHA =     0.46745478620674      I =    20      K =     2
NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
ALPHA =     0.48028381234698      I =    18      K =     2
NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
ALPHA =     0.48105384274671      I =    19      K =     2
NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
ALPHA =     0.48582284110104      I =     7      K =     2
NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
ALPHA =     0.48738723143245      I =    11      K =     2
NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
ALPHA =     0.49032200229026      I =    17      K =     2
NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
ALPHA =     0.49438128081422      I =    12      K =     2
NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
ALPHA =     0.49584291440344      I =    15      K =     2
NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
ALPHA =     0.49798662127968      I =    13      K =     2
NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
ALPHA =     0.49927546453550      I =    16      K =     2
```

```
    NUMBER OF ITERATION IN S-OPERATION FOR U IS          0
    ALPHA =    0.50215696870027      I =   14     K =    2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS          0
    ALPHA =    0.50977427285892      I =   10     K =    2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS          0
    ALPHA =    0.52478618731602      I =    8     K =    2
    NUMBER OF ITERATION IN S-OPERATION FOR J IS          0
    ALPHA =    0.52601803898339      I =    9     K =    2
    NUMBER OF ITERATION IN S-OPERATION FOR J IS          0
    ALPHA =    0.55043773750936      I =    6     K =    2
    NUMBER OF ITERATION IN S-OPERATION FOR J IS          0
    ALPHA =    0.55600010971500      I =    3     K =    2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS          0
    ALPHA =    0.57498989312685      I =    5     K =    2
    NUMBER OF ITERATION IN S-OPERATION FOR J IS          0
    ALPHA =    0.60466183413904      I =    4     K =    2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS          0
    ALPHA =    0.66828568172717      I =    2     K =    2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS          0
    ALPHA =    0.78213344389196      I =    1     K =    2
    NUMBER OF ITERATION IN S-OPERATION FOR J IS          0
    OPTIMAL FOR ALL ALPHA GREATER THAN THE CURRENT VALUE
EXECUTION TERMINATED   17:40:19  T=11.094  RC=0     $3.79
  $3.87,   $4.78T


$SIG
```

*** S) COMPUTATIONAL RESULT OF TWO CRITERION AUTOMOBILE REPLACEMENT
*** PROBLEM USING CODE L WITH DISCOUNT RATE OF .97

NUMBER OF ITERATION IN T-OPERATION FOR V IS 0
ALPHA = 0.0

STATE	POLICY	RETURN	OPT. POLICY
1	41	-0.3924717E+04	14
2	41	-0.4044717E+04	14
3	41	-0.4154717E+04	14
4	41	-0.4331642E+04	1
5	41	-0.4398066E+04	1
6	41	-0.4462022E+04	1
7	41	-0.4522882E+04	1
8	41	-0.4580883E+04	1
9	41	-0.4635197E+04	1
10	41	-0.4687651E+04	1
11	41	-0.4737786E+04	1
12	41	-0.4784717E+04	1
13	41	-0.4828727E+04	1
14	41	-0.4870073E+04	1
15	41	-0.4908988E+04	1
16	41	-0.4945687E+04	1
17	41	-0.4979281E+04	1
18	41	-0.5010936E+04	1
19	41	-0.5040800E+04	1
20	41	-0.5069008E+04	1
21	41	-0.5095686E+04	1
22	41	-0.5120944E+04	1
23	41	-0.5144714E+04	1
24	41	-0.5166661E+04	1
25	41	-0.5185686E+04	1
26	41	-0.5201752E+04	1

27	41	-0.5214717E+04	14
28	41	-0.5224717E+04	14
29	41	-0.5234717E+04	14
30	41	-0.5239717E+04	14
31	41	-0.5244717E+04	14
32	41	-0.5249717E+04	14
33	41	-0.5254717E+04	14
34	41	-0.5264717E+04	14
35	41	-0.5269717E+04	14
36	41	-0.5274717E+04	14
37	41	-0.5279717E+04	14
38	41	-0.5289717E+04	14
39	41	-0.5297717E+04	14
40	41	-0.5304717E+04	14

DISCR	ITERATION
0.97000	9

*** OPTIMAL ACTION(S) TO BE CHOSEN FOR THE CURRENT OPTIMAL POLICY

```
NUMBER OF ITERATION IN S-OPERATION FOR U IS          34
ALPHA =     0.14066894675594     I =   26     K =   14
NUMBER OF ITERATION IN S-OPERATION FOR U IS           0
ALPHA =     0.20215348265718     I =    3     K =    1
NUMBER OF ITERATION IN S-OPERATION FOR U IS           0
ALPHA =     0.21270901910018     I =    2     K =    1
NUMBER OF ITERATION IN S-OPERATION FOR U IS           0
ALPHA =     0.21461143323469     I =    1     K =    1
NUMBER OF ITERATION IN S-OPERATION FOR U IS           0
ALPHA =     0.27632974583896     I =   25     K =   14
NUMBER OF ITERATION IN S-OPERATION FOR U IS           0
ALPHA =     0.39268378223707     I =   24     K =   14
NUMBER OF ITERATION IN S-OPERATION FOR U IS           0
ALPHA =     0.39883714527196     I =   24     K =    2
ALPHA =     0.39883714527196     I =   25     K =    2
ALPHA =     0.39883714527196     I =   26     K =    2
ALPHA =     0.39883714527196     I =   27     K =    2
ALPHA =     0.39883714527196     I =   28     K =    2
ALPHA =     0.39883714527196     I =   29     K =    2
ALPHA =     0.39883714527196     I =   30     K =    2
ALPHA =     0.39883714527196     I =   31     K =    2
ALPHA =     0.39883714527196     I =   32     K =    2
ALPHA =     0.39883714527196     I =   33     K =    2
```

```
    ALPHA =     0.39883714527196      I =    34      K =     2
    ALPHA =     0.39883714527196      I =    35      K =     2
    ALPHA =     0.39883714527196      I =    36      K =     2
    ALPHA =     0.39883714527196      I =    37      K =     2
    ALPHA =     0.39883714527196      I =    38      K =     2
    ALPHA =     0.39883714527196      I =    39      K =     2
    ALPHA =     0.39883714527196      I =    40      K =     2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
    ALPHA =     0.41737376800098      I =    22      K =     2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
    ALPHA =     0.41903877390046      I =    23      K =     2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
    ALPHA =     0.43056709447739      I =    21      K =     2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
    ALPHA =     0.44539850599727      I =    20      K =     2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
    ALPHA =     0.45988918502068      I =    18      K =     2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
    ALPHA =     0.46078685810491      I =    19      K =     2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
    ALPHA =     0.46917357999788      I =     7      K =     2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
    ALPHA =     0.46978789362211      I =    11      K =     2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
    ALPHA =     0.47235104315322      I =    17      K =     2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
    ALPHA =     0.47708660539642      I =    12      K =     2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
    ALPHA =     0.47839025272684      I =    15      K =     2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
    ALPHA =     0.48096076424851      I =    13      K =     2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
    ALPHA =     0.48208221706539      I =    16      K =     2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
    ALPHA =     0.48537861195174      I =    14      K =     2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
    ALPHA =     0.49538267455812      I =    10      K =     2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
    ALPHA =     0.51200593539467      I =     8      K =     2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
    ALPHA =     0.51317547135414      I =     9      K =     2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
    ALPHA =     0.54200845891815      I =     6      K =     2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
    ALPHA =     0.54925318241443      I =     3      K =     2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
    ALPHA =     0.56844182460447      I =     5      K =     2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
    ALPHA =     0.59933138657381      I =     4      K =     2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
    ALPHA =     0.66609386309097      I =     2      K =     2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
    ALPHA =     0.78164104185996      I =     1      K =     2
    NUMBER OF ITERATION IN S-OPERATION FOR U IS              0
    OPTIMAL FOR ALL ALPHA GREATER THAN THE CURRENT VALUE
EXECUTION TERMINATED   17:45:10  T=10.419  RC=0     $3.53
  $3.59,   $4.51T
```