Department of Agricultural Economics • 1991

Application of Stochastic Processes to Summarize Dynamic Programming Solutions

Department

TECHNICAL REPORT NO. 91-1

Texas Agricultural Experiment Station/Charles J. Arntzen, Director/The Texas A&M University System/College Station, Texas

APPLICATION OF STOCHASTIC PROCESSES TO SUMMARIZE DYNAMIC PROGRAMMING SOLUTIONS

James W. Mjelde
Wesley D. Harris
Gary D. Schnitkey
J. Richard Conner
Michael K. Glover
Lee Garoian

Department of Agricultural Economics
Texas A&M University
Texas Agricultural Experiment Station
College Station, Texas

The authors are associate professor and graduate research assistant, Texas A&M University; assistant professor, The Ohio State University; professor, Texas A&M University and econometricians American Express, TRS Inc. The authors thank Mike Mazzocco, Teo Ozuna, Bruce McCarl, and David Bessler for helpful comments on earlier drafts.

APPLICATIONS OF STOCHASTIC PROCESSES TO SUMMARIZE DYNAMIC PROGRAMMING SOLUTIONS

Contents Page #

APPLICATION OF STOCHASTIC PROCESSES TO SUMMARIZE DYNAMIC PROGRAMMING SOLUTIONS

Use of optimization techniques which accounts for both the stochastic and dynamic nature of agricultural problems is increasing. Dynamic programming (DP) is a commonly used technique in this setting. Although increasing, the use of DP is not widespread. Burt suggests two obstacles to the adoption of multi-period optimization techniques: 1) cursory treatment of this subject in graduate courses and 2) what Bellman termed the "curse of dimensionality." As noted by Taylor (1987), the solution to the first obstacle is obvious; however, the alleviation of the second obstacle is not as obvious.

Taylor (1987; 1989) argues that the "curse" as defined by Bellman is fading or more correctly being replaced by several curses including a "decision rule curse." We agree with Taylor that the curse is changing. Bellman termed the phrase, curse of dimensionality, to refer to computer storage and speed limitations which limits the size of dynamic optimization models. Dramatic advances in computer computational power and refinements in the DP technique have faded Bellman's original curse, such that models with a large number of total states can be solved (Taylor, 1987; 1989). Taylor (1987) defines the decision rule curse as "with a problem characterized by thousands or hundreds of states, the decision rule is such a large matrix that it is difficult for the researcher or decision maker to digest and fully understand" (p. 2).

Several techniques either directly or indirectly have been suggested or are being used to overcome the decision rule curse. Usually only convergent decision rules are presented, eliminating the decision rules at all stages before the rule converges. Graphical techniques have been used to summarize large models (Taylor, 1987; Schnitkey, Taylor and Barry). Building an expert system to recall the appropriate decision based on user input could help decision makers use a large DP model (Taylor, 1989). Burt and Allison suggest using infinite net returns and/or expected long-run net returns as procedures to compare models. Schnitkey, Taylor, and Barry; and Mjelde, Taylor, and Cramer calculate the probability of being in various states given initial conditions. Although these techniques only indirectly address the decision rule curse, they are techniques which summarize large complex models. These

techniques rely on the use of stochastic processes.

This paper is designed as a tutorial on the application of stochastic processes to summarize dynamic programming model's decision rules. As such, the paper contains basic background on stochastic processes and some extensions into areas that, to our knowledge, have not been previously addressed. Because the paper is a tutorial and readers will have varying backgrounds, the paper is divided into several almost self-contained sections. Readers with little background in Markov and stochastic processes should read all the sections. Other readers could easily skip the Markov review and still understand the application of Markov principles to summarize dynamic programming decision rules. Markov processes are discussed first to provide necessary background material. Within the discussion, several examples are given to clarify the material. Application of Markov theory to post-dynamic programming computations is then discussed. The discussion is then expanded to include discrete stochastic processes with multiple transition matrices. To our knowledge, this is the first formal discussion of such stochastic processes. Examples of this type of processes are given.

DISCRETE STOCHASTIC MARKOV PROCESSES

Two basic concepts within a discrete stochastic Markov process involve 1) the state of the system and 2) state transitions. The state of the system describes the condition of the process at a particular observation, usually defined as a stage. Normally, in agriculture, stages are defined as time periods. This definition is used throughout the remainder of this paper although it is not necessary. State transitions describe changes in the state of the system from one stage to the next. A stochastic process is a system in which the transitions can only be specified in terms of a probability distribution.

Let P_{ij} ($i, j = 1, \ldots, k$ and $0 \leq P_{ij} \leq 1$) represent the transition probabilities for a system with k possible states. The transition probability P_{ij} is the probability of the system moving from state i at stage t to state j at stage t+1. Furthermore, for any $i = 1, \ldots, k$, the following condition

must hold,

$$P_{i1} + P_{i2} + \ldots + P_{ik} = 1. \qquad (1)$$

This requirement states that if the system is in state i at stage t it must be in one of the k states at stage t+1. A transition probability matrix, P, is a k x k matrix of all the transition probabilities, P_{ij}. A fundamental characteristic of Markov processes is that P_{ij} is independent of the state of the system in stages one through t-1.

To clarify transition probabilities and matrices, consider the following example concerning the water level of a lake. Water level in the lake can be described by 3 states: below normal, normal, and above normal. Further, the level of the lake next year depends on the current state of the lake (below, normal, or above) and stochastic weather conditions that will occur over the year. Let the lake level transition matrix be:

$$P = [P_{ij}] = \begin{bmatrix} .5 & .3 & .2 \\ .2 & .6 & .2 \\ .1 & .5 & .4 \end{bmatrix} \qquad (2)$$

In this matrix the rows represent the current state of the process, and the columns represent the state of the process at stage t+1. Let the first row (column) represent below normal lake level, the second represent normal and the third represent above normal. With these definitions, elements in matrix P are interpreted as follows. If the lake is currently at a below normal level it has a 50% probability of being in the below normal level state next year, a 30% probability of being normal and 20% probability of being in state above normal. Similar interpretations are given to the other elements. This example (among others) will be used and expanded in the remainder of this paper.

State Occupancy Vectors - Limiting Probabilities

Let $\pi_i(t)$ be defined as the probability that the system will occupy state i at stage t (after t transitions). It follows that

$$\sum_{i=1}^{k} \pi_i(t) = 1 \qquad \textit{for all } t. \qquad (3)$$

Equation (3) simply forces the system to occupy some state defined in the system at each stage or transition. Define $\pi(t)$ with components $\pi_i(t)$ to be a row vector of the state occupancy probabilities. To find the probability that

the system occupies some state at time t, simply post-multiply the state occupancy probability vector at time t-1 by the transition matrix, that is,

$$\pi(t) = \pi(t\text{-}1)\ P. \tag{4}$$

In general, because of the recursive relationship present in equation (4), the vector π(t) can be found by,

$$\begin{aligned} \pi(1) &= \pi(0)\ P \\ \pi(2) &= \pi(1)\ P = \pi(0)\ P\ P = \pi(0)\ P^2 \\ \pi(3) &= \pi(2)\ P = \pi(0)\ P^3 \\ &\ \ \vdots \\ \pi(t) &= \pi(t\text{-}1)\ P = \pi(0)\ P^t \end{aligned} \tag{5}$$

where, (0) is the initial state occupancy vector.

Returning to our example, what is the probability of the lake having an above normal water level after 2 years if initially the lake is below normal. To find this probability, the following set of recursive calculations are necessary:

$$\pi(1) = \pi(0)\ P = [1.\ \ 0.\ \ 0.] \begin{bmatrix} .5 & .3 & .2 \\ .2 & .6 & .2 \\ .1 & .5 & .4 \end{bmatrix} = [.5\ \ .3\ \ .2]$$

and (6)

$$\pi(2) = \pi(1)\ P = [.5\ \ .3\ \ .2] \begin{bmatrix} .5 & .3 & .2 \\ .2 & .6 & .2 \\ .1 & .5 & .4 \end{bmatrix} = [.33\ \ .43\ \ .24].$$

After two years the lake has a 24% probability of having an above normal water level given it initially had a below normal water level. Further, the lake has a 33% probability of being below normal and 43% chance of having a normal water level given that the lake was initially in the below normal state. The state occupancy vectors for 6 time periods given the three possible starting lake levels are given in Table 1. Notice that the state occupancy probabilities given in Table 1 are tending to the same vector irrespective of the initial lake level. The state occupancy probabilities; therefore, appear to be independent of starting state of the system given a large number of transitions. This property is exhibited by many Markov processes.

A brief discussion of the properties of the transition matrix, P, necessary for this property to hold follows. Note, however, that state occupancy vectors which are independent of the initial conditions are known by several names including limiting probabilities (Howard), steady state

Table 1. State Occupancy Vectors at Different Stages Given Various Starting Lake Levels.[a]

	Stage (t)						
$\pi \backslash t$	0	1	2	3	4	5	6
	Lake Level Originally Below Normal						
$\pi_1(t)$	1.0	.500	.330	.275	.258	.252	.251
$\pi_2(t)$	0.0	.300	.430	.477	.493	.498	.499
$\pi_3(t)$	0.0	.200	.240	.248	.250	.250	.250
	Lake Level Originally Normal						
$\pi_1(t)$	0.0	.200	.240	.248	.250	.250	.250
$\pi_2(t)$	1.0	.600	.520	.504	.501	.500	.500
$\pi_3(t)$	0.0	.200	.240	.248	.250	.250	.250
	Lake Level Originally Above Normal						
$\pi_1(t)$	0.0	.100	.190	.229	.243	.248	.249
$\pi_2(t)$	0.0	.500	.530	.515	.506	.502	.501
$\pi_3(t)$	1.0	.400	.280	.256	.251	.250	.250

a) Probabilities rounded to 3 decimal places.

probabilities (Rorres and Anton; Gillett), limiting vectors (Kemeny and Snell), and equilibrium probability vectors. Here limiting probabilities are used.

Limiting probability vectors by definition have the following property

$$\pi = \pi P. \tag{7}$$

The condition given in equation (7) simply states that the probability of being in a given state will not change with an additional transition. That is, the probabilities are stage or transition independent. Given conditions discussed below, the vector π will be unique.

A sufficient condition for P to have a unique limiting probability vector is that P be a regular matrix (Rorres and Anton). A regular transition matrix has the property that some integer power of the matrix has all positive entries. This condition implies that the process can go from any state to any other state, but not necessarily in one transition. If P is a regular matrix then as t goes to positive infinity ($t \to \infty$) the following condition holds

$$p^t \to Q = \begin{bmatrix} q_1 & q_2 & \cdots & q_k \\ q_1 & q_2 & \cdots & q_k \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ q_1 & q_2 & \cdots & q_k \end{bmatrix} \tag{8}$$

where, q_i are positive numbers such that $q_1 + q_2 + \ldots + q_k = 1$. See Kemeny and Snell for a proof of this proposition. From the condition in equation (8), it can be seen that given any probability vector x (where $\Sigma x_i = 1$) that

$$q = x Q \tag{9}$$

where q is a row vector with elements $q_1, q_2, \ldots, q_k$. Equation (9) can easily be shown,

$$\begin{aligned} q &= [x_1 \; x_2 \; \cdots \; x_k] \begin{bmatrix} q_1 & q_2 & \cdots & q_k \\ q_1 & q_2 & \cdots & q_k \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ q_1 & q_2 & \cdots & q_k \end{bmatrix} \\ &= [q_1x_1 + q_1x_2 + \ldots + q_1x_k \quad q_2x_1 + q_2x_2 + \ldots + q_2x_k \quad \cdots \\ &\quad q_kx_1 + q_kx_2 + \ldots + q_kx_k] \\ &= (x_1 + x_2 + \ldots + x_k) \; [q_1 \quad q_2 \quad \cdots \quad q_k] = (1)q = q. \end{aligned} \tag{10}$$

Equation (10) results because for any probability vector, the sum of the elements must equal one. No matter what initial probability vector; therefore, is used, the probability of being in each state will be the same provided enough transitions have occurred. That is, provided t is sufficiently large. Equation (10) also shows that the limiting probability vector, π, will be equal to the vector q.

Several methods exist to calculate the limiting probability vector. One method makes use of equations (3) and (7). These two equations give k + 1 equations in k unknowns. To find a unique solution, one equation given by equation (7) is replaced with equation (3) and the resulting system of k equations is solved. To clarify this method, return to the lake level example. Applying equation (7) to the example gives

$$\pi_1 = .5\pi_1 + .2\pi_2 + .1\pi_3 \qquad \textbf{(11a)}$$

$$\pi_2 = .3\pi_1 + .6\pi_2 + .5\pi_3 \qquad \textbf{(11b)}$$

$$\pi_3 = .2\pi_1 + .2\pi_2 + .4\pi_3. \qquad \textbf{(11c)}$$

Equation (13) for this example gives

$$\pi_1 + \pi_2 + \pi_3 = 1. \qquad \textbf{(12)}$$

Equation (12) is then substituted for any one of the equations in (11) usually equation (11c) and the resulting system of 3 linear equations and 3 unknowns is solved. The solution is independent of the equation being substituted. Using this method, the limiting probability vector is

$$\pi = [.25 \quad .50 \quad .25]. \qquad \textbf{(13)}$$

As expected, the limiting probabilities given in equation (13) are similar to these given in the sixth column of Table 1.

Several limitations of the above method of solving a system of linear equations are apparent. Using this method, only information about the limiting probabilities is gained. No information about transitions from initial conditions to the limiting probabilities is gained. Another limitation is potential problems associated with trying to solve a system of k linear equations as k increases.

To overcome these limitations, the use of the recursive relationship in

equation (5) can be used. Probability vector $\pi(t)$ can be calculated either by $\pi(t) = \pi(t-1)\ P$ or $\pi(t) = \pi(0)\ P^t$. Gillett presents a FORTRAN program to calculate powers of the P matrix. Appendix A presents a FORTRAN program to calculate $\pi(t) = \pi(t-1)\ P$. In both methods a tolerance limit is used to check for convergence. Convergence check for the powers method is to check

$$P_{ij}^{t} - P_{ij}^{t-1} \leq tol \text{ for all } i, j = 1, 2, \ldots, k \qquad \textbf{(14a)}$$

or

$$P_{ij}^{t} - P_{in}^{t} \leq tol \text{ for all } i, j, n = 1, 2, \ldots, k \qquad \textbf{(14b)}$$

where tol is the tolerance limit. For the method of calculating $\pi(t)$ at each stage the check is

$$\pi_i(t) - \pi_i(t-1) \leq tol \text{ for all } i = 1, 2, \ldots, k. \qquad \textbf{(15)}$$

The method of calculating state occupancy vectors appears to be the more efficient method. Consider for example, a system with 100 states, therefore; a P matrix of 100 x 100. The powers method requires 10,000 elements be calculated at each stage with each element requiring 100 multiplications and additions. These 10,000 elements need to be checked for convergence. For the state occupancy vector only 100 elements need to be calculated and checked for convergence. Each element still requires 100 multiplications and additions. The method which converges faster depends on the number of stages each method need for convergence. Our limited experience is that both methods converge in approximately the same number of iterations giving the state occupancy vector method an advantage, because of its property of requiring less elements to be calculated.

The condition that the probability matrix, P, be regular is more restrictive than necessary for a system to converge to a unique limiting probability vector (Kemeny and Snell). Three properties of Markov transition matrices need to be introduced to fully understand less restrictive conditions on the P matrix. These are transient sets, ergodic sets, and absorbing states. The elements of a transient set are called transient states, likewise, the elements of ergodic set are called ergodic states. Furthermore, there must be at least one ergodic set for every Markov process (Howard). A process, however, does not necessarily have to contain a transient set or an absorbing state. These properties classify states within a Markov process.

Classification of states within a Markov process is based on whether it is possible to go from a given state to another given state. This movement does not necessarily have to occur in one transition. A transient set has the following characteristics: the system can move between the states in the transient set or the system can leave the transient set and enter another set. Once a process has left a transient set and entered an ergodic set it can never return to the transient set. An ergodic set has the characteristic that once a process enters the ergodic set it can never leave this set, but the process can move between states within the set. An absorbing state is a special case of an ergodic set in which only one state exists within the ergodic set. Once a process has entered an absorbing state, it can not leave that state, thus state i is absorbing if and only if $P_{ii} = 1$ (Kemeny and Snell).

To clarify these definitions, consider the following transition matrix,

$$A = \begin{bmatrix} .8 & .2 & 0 & 0 & 0 \\ .5 & .5 & 0 & 0 & 0 \\ .25 & .25 & .25 & .25 & 0 \\ 0 & 0 & .25 & .5 & .25 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \qquad (16)$$

Within transition matrix A there are 2 ergodic sets and 1 transient set. Further, one of the ergodic sets is an absorbing state. States 1 and 2 comprise 1 ergodic set. If the process enters either state 1 or 2, it can never leave this set of states. States 3 and 4 are transient states. If the process is in state 3 or 4 it can enter any of the 4 remaining states, but not necessarily with 1 transition. State 5 is an absorbing ergodic state, that is, if the process enters state 5 it can never leave this state.

The question is why are these concepts important in terms of Markov processes, in general, and DP, in particular? Table 2 lists the state occupancy vectors at stage 6 for probability matrix A given different initial probability vectors (calculated using $\pi(t) = \pi(t-1)\ A$). It is obvious from the probabilities in Table 2 that the initial state is important in determining the probability of being in each of the 5 states. If the process started in state 5, it remains in this state no matter how many transitions occur. Further, if the process started in either state 1 or 2 it remained in this ergodic set. Finally, starting in states 3 or 4 leads to a probability

of being in each state, although it differs depending on where the process started. But, as $t \to \infty$ the probability of being in either state 3 or 4 approaches zero. Probability transition matrix A shows by example, that a matrix with 2 ergodic sets will not satisfy equation (7). This could be extended to any transition matrix with greater than 2 ergodic sets. Such matrices; therefore, do not have unique limiting probability vectors. The system does not lose complete memory of its initial state.

Now consider the following transition probability matrix that has only 1 ergodic set,

$$B = \begin{bmatrix} .33 & .33 & .34 \\ 0 & .2 & .8 \\ 0 & .7 & .3 \end{bmatrix}. \tag{17}$$

In matrix B, state 1 is a transient state and states 2 and 3 define an ergodic set. Table 3 lists the state occupancy vectors for different time periods given 3 possible initial states for matrix B. The state occupancy vectors appear to be tending toward a unique vector regardless of the initial conditions. In fact, matrix B has the following limiting probability vector,

$$\pi = [\, 0 \quad .467 \quad .533 \,]. \tag{18}$$

A matrix with 1 ergodic set and 1 transient set, will possess a unique limiting probability vector.

As a last example, consider the following ergodic matrix

$$C = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{19}$$

This is the simplest example of what is referred to as a cyclic Markov process. In matrix C, if the process is in state 1, it will be in state 2 and if the process starts in state 2, it will be in state 1 in the next stage. For a given starting point the process; therefore, moves through the states in a definite order. At any given stage and a given initial condition the state of the process will be known; therefore, it does not have a unique limiting probability vector. The concept of limiting probabilities in a theoretical sense are not relevant because the state of the process will be known. In a practical sense, this concept, however, may be useful (Howard).

Table 2. State Occupancy Vectors at Stage 6 for Probability Transition Matrix A Given Different Initial States[a].

Initial State	Probability of Being in a Given State				
	1	2	3	4	5
1	.714	.286	0	0	0
2	.714	.286	0	0	0
3	.542	.226	.021	.035	.175
4	.239	.110	.035	.057	.559
5	0	0	0	0	1

a) Rounded to 3 decimal places.

Using the method of solving a k system of linear equations gives a limiting probability vector with elements .5 and .5. This result is intuitively appealing given the cyclic nature of matrix C, that is, 50% of the time the process is in state 1 and 50% of the time it is in state 2.

How do the above examples relate back to the original question; are there less restrictive conditions than regularity that can be placed on the transition probability matrix and still have a unique limiting probability vector? As can be seen from the above examples, the answer is yes. Kemeny and Snell (p.37) subdivide Markov processes into the following classifications:

I. Processes without transient sets.
   a. Ergodic set is regular.
   b. Ergodic set is cyclical.

II. Processes with transient sets
   a. All ergodic sets contain a single absorbing state.
   b. All ergodic sets are regular, containing more than one state.
   c. All ergodic sets are cyclical.
   d. Both cyclic and regular ergodic sets exist.

Within this classification, the major difference is the existence of transient sets. In category I if more than one ergodic set is present, each can be considered as a separate process because no communication between the sets occur (Kemeny and Snell), whereas, category II, if more than 1 ergodic set exists, transition from the transient set(s) to the ergodic sets occurs. By example, it was shown that, in general, if the process contains more than one ergodic set, the process will not possess a unique limiting probability vector. It was also shown, by example, that in theory a cyclic process will not possess a unique limiting probability vector. But in practice, it may be possible to treat such a process as if the process possessed such a vector (Howard). With the examples, all Markov processes, except Ia, IIa with single absorbing state, and IIb containing only a single regular ergodic set, were eliminated from having a unique limiting probability vector. Further, IIa is not very exciting, as the process will become trapped in the absorbing state, that is, its limiting probability equals one for the absorbing state. The

Table 3. State Occupancy Vectors at Different Stages Given Different Starting States[a].

	Stage (t)						
π\t	0	1	2	3	4	5	6
			Process Initially in State 1				
$\pi_1(t)$	1	.33	.109	.036	.012	.004	.001
$\pi_2(t)$	0	.33	.413	.453	.460	.466	.466
$\pi_3(t)$	0	.34	.478	.511	.528	.531	.533
			Process Initially in State 2				
$\pi_1(t)$	0	0	0	0	0	0	0
$\pi_2(t)$	1	.2	.6	.4	.5	.45	.475
$\pi_3(t)$	0	.8	.4	.6	.5	.55	.525
			Process Initially in State 3				
$\pi_1(t)$	0	0	0	0	0	0	0
$\pi_2(t)$	0	.7	.35	.525	.438	.481	.459
$\pi_3(t)$	1	.3	.65	.475	.563	.519	.541

a) Rounded to 3 decimal places.

remainder of the discussion in this paper will concentrate on regular ergodic sets. For a discussion of cyclic ergodic sets, see Kemeny and Snell. Much of the discussion presented here applies to cyclical systems. Throughout the remainder of this paper, ergodic will be used to represent regular ergodic. A Markov transition matrix which contains a single ergodic set is defined as an ergodic Markov process. The necessary condition for a Markov process to possess a unique limiting probability vector; therefore, is that the process must be an ergodic process. This condition encompasses both transition matrices which are regular and matrices that contain transient sets and a single ergodic set. A regular transition matrix is a special case of ergodic chain, namely that no transient states exist in the matrix. The limiting probability vector in the latter case will contain zero's in the transient states and positive components for the states contained in the ergodic set. See Kemeny and Snell for mathematical proofs of the above conditions for existence of unique limiting probability vectors.

DETERMINATION OF SETS WITHIN A TRANSITION MATRIX

Obviously, classification of sets within a transition matrix is necessary to understand how a discrete stochastic Markov process behaves over time. Set identification methods should be computer adaptable to facilitate examination of medium to large matrices encountered in applied problems. Previously defined methods for determining sets within a transition matrix are reviewed in the section. A row-by-row examination method is proposed as an alternative to the previous methods. Each method has advantages and disadvantages, such that no method can be singled out as the one best method. These advantages and disadvantages are discussed.

Previously Defined Methods

Previously defined methods for determining sets within a matrix can be divided into four categories: 1) visual inspection methods, 2) Z-transformation method, 3) power-multiplication method, and 4) limiting probability vector method. Each method is discussed.

Visual inspection methods: Visual inspection relies on visual examination of the matrix to determine which states are assessable from other states. Up to this point, this paper has relied on visual inspection for set classification. Use of this method is appropriate only for small matrices.

Because of this drawback, this method is usually only appropriate for illustrative purposes and not for applied problems.

Z-transformation method: Howard proposes the z-transformation method for determining state occupancy probability vectors (pp 10 - 16). The z-transformations allows for the construction of the following equation:

$$J^t = [\, H + c^t K \,] \tag{20}$$

where J^t gives the probability matrix at time t, H and K are k by k matrices of constants, and c is a constant real number ($-1 < c < 1$). State probability vectors for each of the k possible initial states are given by the respective row of the J^t matrix. Any J^t matrix is calculated by setting t to the appropriate number and performing the calculation on the right-hand-side of equation (20).

Determining H, J and c is appealing because state probability vectors for all states and all t are given by equation (20). Limiting probability vectors for all states and all t are given by the H matrix. Unfortunately, determining the H and K matrices and the c constant requires use of z-transformations. These methods are difficult to numerically implement. As such, the z-transformation method is not discussed further.

Power-multiplication method: The power-multiplication method determines the converged transition probability matrix. Use of the method requires repeated post-multiplication of matrices in the following manner (Clark and Disney):

$$P^{t+g} = P^t P^g. \tag{21}$$

For a given transition matrix, the first multiplication yields $P^2 = P * P$. The P^2 then is used to determine $P^4 = P^2 * P^2$. This process is repeated until convergence occurs. Convergence can be checked using equation (14). Use of equation (21) instead of calculating $P^t = P^{t-1} P$ results in faster convergence. The converged matrix contains all limiting probability vectors which are given by rows in the matrix.

Sets contained within the transition matrix can be determined by applying the following three steps to matrix P^t:

1. Determine which rows have identical elements. All or a portion of the states represented by the rows with similar

elements constitutes an ergodic set. Some of the states represented may be members of a transient sets.

2. Determine the ergodic set for all rows with similar elements. All members of an ergodic set must have positive limiting probabilities. Suppose, for example, that rows 1, 2, and 3 are identical and states 1 and 2 have positive probabilities. In this case, the ergodic set consists of states 1 and 2.

3. Determine transient states for all rows with similar elements. Members of transient sets will have zero probability. In the above example, state 3 represents a transient set.

The above process is illustrated using the following transition matrix:

$$F = \begin{bmatrix} .5 & .5 & 0 & 0 & 0 \\ .5 & .5 & 0 & 0 & 0 \\ 0 & .2 & .8 & 0 & 0 \\ 0 & 0 & 0 & .5 & .5 \\ 0 & 0 & 0 & .5 & .5 \end{bmatrix}. \tag{22}$$

Using the power-multiplication methods, matrix F has the following converged probability matrix:

$$\begin{bmatrix} .5 & .5 & 0 & 0 & 0 \\ .5 & .5 & 0 & 0 & 0 \\ .5 & .5 & 0 & 0 & 0 \\ 0 & 0 & 0 & .5 & .5 \\ 0 & 0 & 0 & .5 & .5 \end{bmatrix}. \tag{23}$$

This converged matrix has two sets of rows with identical elements: 1) states 1, 2, and 3, and 2) states 3 and 4. For states 1, 2, and 3, step (2) reveals that the ergodic set consists of states 1 and 2. Both of these states have positive limiting state occupancy probabilities. Step 3 indicates that state 3 is a transient state that communicates with the ergodic set consisting of states 1 and 2. Repeating steps 1, 2, and 3 for the second set of rows with identical elements reveals that states 4 and 5 constitute an ergodic set.

The power-multiplication method is adaptable to numeric methods. Its major advantage is that all limiting probability vectors are found and all ergodic states are identified.

The method has three disadvantages. First, extensive resources are

required for large matrices. In most cases, two matrices, the matrix to be post-multiplied and the resulting matrix, must be in random access memory (RAM). For a 1000 by 1000 transition matrix, memory requirements for these matrices equal 7,812.5 K of RAM, presuming that each element requires 4 bits of storage. Memory requirements could be reduced by one-half by storing the resulting matrix in external storage. Preforming these storage operations will dramatically increase input-output operations, thereby increasing computing time. Also, accuracy may be reduced during the storage operations.

Second, convergence may require extensive post-multiplications. This could require extensive computer resources. Moreover, convergence will not occur when the matrix includes cyclical components.

Third, the method may not identify all transient sets. As an example, examine the following matrix:

$$G = \begin{bmatrix} .5 & .5 & 0 & 0 \\ 0 & .5 & .5 & 0 \\ 0 & .5 & .5 & 0 \\ 0 & 0 & .5 & .5 \end{bmatrix}. \tag{24}$$

This matrix contains three sets: (1) an ergodic set consisting of states 2 and 3, (2) a transient set consisting of state 1, and (3) a transient set consisting of state 4. Matrix G's converged probability matrix is:

$$\begin{bmatrix} 0 & .5 & .5 & 0 \\ 0 & .5 & .5 & 0 \\ 0 & .5 & .5 & 0 \\ 0 & .5 & .5 & 0 \end{bmatrix}. \tag{25}$$

The power-multiplication method will identify only one transient set consisting of states 1 and 4. Only transient states and not transient sets; therefore, are identified.

Limiting probability vector method: This method finds a limiting probability vector for a given initial state by using equation (5). For a given limiting vector, all states with positive probability will constitute an ergodic set. States with zero limiting probability vectors are either transient states or constitute another ergodic set. To classify the states with zero limiting probabilities, calculation of a new limiting probability vector must occur using a different initial state. The new initial state is a state that had a limiting probability of zero. States with a positive probability are eliminated because they have been classified. A transient

state is identified when the state is used as the initial state and its limiting probability equal zero. This process is repeated until all states are either identified as either part of an ergodic set or are identified as transient states.

The limiting probability vector method is a special case of the power-multiplication method. Operations are performed for a single state rather than on the whole matrix. The method may be preferred because memory requirements are reduced. Under the power method, two k by k matrices must be stored in RAM. Under the limiting vector method, only one transition matrix and two 1 by k vectors need to be stored. The method still suffers from the later two disadvantages identified under the power-multiplication method.

Row-by-Row Examination

Of the above methods, the power-multiplication and limiting probability vector methods are numerically applicable. Both methods have two disadvantages: convergence may require extensive calculations, and the methods do not necessarily identify transient sets. To overcome these two disadvantages, the following row-by-row examination method is proposed. The row-by-row method is similar to Heyman and Sobel's discussion on communicating states (p. 230-235). A computer program using the row-by-row method is contained in Appendix B. This method required a finite number of iterations, thus avoiding convergence difficulties. It also identifies all transient and ergodic sets. The method requires three steps: (1) determining communication between states, (2) determining sets within the matrix, and (3) determining ergodic and transient sets.

Determining communication between states: This step determines states obtainable from other states by creating a matrix that containing only 1's and 0's. A 1 in the ith row and jth column indicates movement from state i to state j is possible but not necessarily in one transition. A zero indicates movement from state i to state j is not possible. Construction of this matrix requires an initial conditioning operation. This initial operation replaces all positive entries in a transition matrix with a 1. Suppose that you have

the following transition matrix:

$$\begin{bmatrix} .5 & .5 & 0 & 0 & 0 \\ .5 & .5 & 0 & 0 & 0 \\ 0 & .2 & .8 & 0 & 0 \\ 0 & 0 & 0 & .5 & .5 \\ 0 & 0 & 0 & .5 & .5 \end{bmatrix}. \tag{26}$$

After completing the initial conditioning operation, the matrix becomes:

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}. \tag{27}$$

the 1's in the matrix indicate that you can move directly from state i to state j.

After the conditioning operation, an iteration is required for each state (row). Each iteration has two steps. First, the current state is examined to determine if transition is possible to any state examined during previous iterations, hereafter referred to as previously considered state(s). If transition is possible, the current state's row may require updating. This updating replaces any 0 element in the current row with a 1 if the corresponding element in the previously considered row equals 1. Second, previously considered states are examined again. This examination determines if transition is possible from the previously considered state to the current state. If possible, as indicated by the appropriate element equalling 1, the row corresponding to the previously considered row may require updating. This updating will replace any 0 element in the previously considered row with a 1 if the corresponding element in the current row equals 1.

For clarification, consider determining communication between states for the matrix given in equation (27). The first iteration examines state 1. No previously considered states, however, exist causing the iteration to end. The second iteration examines state 2. The first step examines the element in the first column of the second row. The element equals 1 indicating the updating of row 2 may be required. The elements of row 1 and 2 are identical; therefore, updating is not required. The second step requires examination of the first element in the first row. This element equals 1 indicating that row 2 may have to be updated. Rows 1 and 2, however, are identical such that

updating does not change row 1.

The third iteration examines row 3. The first step requires examination of the first and second elements of row 3. The first element is zero such that no updating occurs. The second element equals 1 such that updating of row 3 may be warranted. This updating will replace all zero elements in row 3 with 1's if the corresponding element in row 2 equals 1. After updating, row 3 appears as:

$$[\,1 \;\; 1 \;\; 1 \;\; 0 \;\; 0\,]. \tag{28}$$

This row indicates that the first three states are obtainable from the third state. The second step requires consideration of the third element in rows 1 and 2. Both of these elements equal 0 such that no updating occurs. This completes iteration 3.

The fourth and fifth iterations proceed as described above. These iterations do not change any of the rows. At the end of the iteration the matrix appears as:

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}. \tag{29}$$

Each row in the matrix indicates all possible states obtainable from the respective states.

Determining sets within the matrix: States contained within a given set have identical rows. In the above example, there are three sets; the first set contains states 1 and 2, the second set contains state 3, and the third set contains states 4 and 5.

Determining ergodic and transient sets: Ergodic sets are absorbing. In a matrix, absorption will be indicated by having no communication with another set. In other words, an ergodic set will have zeros in columns whose states are in other sets. Transient sets, on the other hand, communicate with other sets. This communication is indicated by a 1 for states that are part of another set. In the example, two sets are ergodic: the set containing states 1 and 2 and the set containing states 4 and 5. The matrix contains one transient set, state 3.

Finally, an examination of all columns needs to be completed. If a

column contains all zeros, the respective state is part of a transient set. The state should be identified as a separate transient set if it has not already been identified as part of a transient set.

The row-by-row examination method possesses several advantages over the previously defined methods. First, the method identifies all transient and ergodic sets. Second, as noted earlier, the method avoids convergence difficulties associated with the other methods. Finally, the method works with integer and not real numbers, thereby decreasing computer storage requirements. The main disadvantage of the row by row examination method is that the limiting probabilities are not determined. Limiting probabilities must be determined outside of the method. As with the power-multiplication and limiting probability methods, computer storage and time requirements associated with the row-by-row examinations may be large.

Comparison of Methods

The power-multiplication, limiting probability vector, and row-by-row examination methods all are numerically implementable. The first two methods suffer from convergence and transient set identification problems. If these issues are not of concern, as is likely for most transition matrices, the power-multiplication and limiting probability vector methods can accomplish ergodic set identification. In terms of computational time, the power-multiplication and row-by-row examination methods are more efficient than the limiting vector method. In some cases, the power-multiplication method is more efficient than the row-by-row examination method. Efficiency is matrix specific. The row-by-row method advantages over the power-multiplication method include lower RAM requirements, no convergence problems, and no transient set identification problems. The power-multiplication method's advantage over the row-by-row method is that limiting probability vectors for all states are identified once convergence is obtained.

APPLICATION OF MARKOV PROCESSES TO DYNAMIC PROGRAMMING SOLUTIONS

The preceding discussion provides the basics necessary to apply Markov processes to post-dynamic programming calculations to help in overcoming the decision rule and acceptance curse. Usually, limiting probabilities are used in procedures to compare decision rules obtained from DP models to decision rules obtained elsewhere (Burt and Allison). Further, state occupancy vectors

are used to obtain the probability of being in various states given an initial state (Schnitkey, Taylor, and Barry). Probabilities of being in various states, not conditional on the initial state have also been reported (Mjelde, Taylor, and Cramer). The objective of these procedures has been to reduce a complicated model or decision rule into a number or set of numbers which provide meaningful information. These procedures along with examination of the decision rules helps a decision maker ascertain the relevancy a model or models.

In applied DP models, the models are usually sufficiently large that the method of solving linear equations to obtain limiting probability vectors is not practical. The recursive relationship in equation (5); therefore, is normally employed. To use equation (5), a probability matrix, P, must be specified. For equation (7) to hold, matrix P must represent an ergodic process. Specification of P in applied DP problems usually is accomplished in the following manner. Matrix P normally represents the probability of going from state i to state j, given the decision maker is following an optimal convergent decision rule. Convergent decision rules have the property that for a given state the optimal decision remains constant between stages. This does not mean the process remains in the same state, but just if the process is in a given state the decision is independent of the stage. Most applied DP model's decision rules converge given enough stages. Convergence; therefore, is not a restrictive condition. Once P is defined, the application of equation (5) and possibly equation (7) is straight forward.

Reporting state occupancy vectors at various stages, given initial conditions, can help summarize a DP model. This procedure is most useful when the Markov process possesses at least one of the following three conditions: 1) the process contains an absorbing state, 2) the process contains more than one ergodic set, or 3) the initial state of the process is known. Calculating the limiting probability vector for the first condition, as seen earlier, is not very exciting. All the probability will be in the absorbing state. A process possessing condition 2 will not have a unique limiting probability vector. Under condition 3 the decision maker knows where he/she currently is and is concerned with where the process will be after several transitions. Reporting state occupancy vectors requires the simple application of equation

(5). Still in large DP models with hundreds or thousands of states, reporting of state occupancy vectors may be as difficult as reporting the decision rules, because there is a probability associated with each state.

Because of the problem associated with reporting state occupancy vectors, limiting probability vectors can be calculated and used to obtain expected long run net returns. Application of this procedure to DP models requires that net returns associated with each state be based on convergent decision rules, because the transition matrix, P, is based on the convergent rules. These net returns, better known as immediate stage net returns, are only one component of Bellman's recursive equation. The sum of the multiplication of the limiting probability for each state by its associated net return gives expected long run yearly net returns, that is,

$$E(NR) = \pi R \tag{30}$$

where E(NR) is expected long run net returns, π represents the limiting probability vector, and R is a vector of net returns associated with each state.

Returning to the lake level example, what is the expected long run net returns associated with the lake under the following scenario? Assume that the owners of the property rights to the water in the lake sell the water. If the level of the lake is below normal, no water can be sold, but the owners still must pay for upkeep on the dam; therefore, they experience a loss of $25 thousand. For normal and above normal levels, net returns (water sold minus dam upkeep costs) are $10 and $15 thousand. Expected long run net returns are then (using equations (13 and 30)):

$$\begin{aligned} E(NR) &= \pi R \\ &= \begin{bmatrix} .25 & .50 & .25 \end{bmatrix} \begin{bmatrix} -25. \\ 10. \\ 15. \end{bmatrix} \\ &= \$2.50. \end{aligned} \tag{31}$$

Equation (31) shows that the expected long run yearly net returns to the owners of the lake's water is $2.5 thousand per year.

Use of limiting probabilities has reduced the problem to a single expected net return. It is obvious, that such a procedure allows for a more succinct reporting of a DP model solution, but caution must be used because much information is not reported using this procedure. Calculation of

expected long run net returns can also be used to compare decision rules. To compare decision rules, a different transition matrix P is specified for each alternative decision rule. Usually, one matrix is specified as discussed above using the DP convergent decision rule. Other transition matrices are specified based on the transitions associated with the decision rule to be compared to the DP rule. Then using the above procedure, expected long run yearly net returns are calculated for each decision rule. Comparison of the expected net returns allows statements to be made about how one decision rule dominates another in terms of expected yearly returns. Note also that expected net returns are not the only item that could be calculated. Expected yearly yields is an example of other factors that could be calculated.

Detailed Example - Comparison of Decision Rules

As a more complicated example, consider the planting problem addressed by Burt and Allison. They consider the problem of either planting wheat or leaving land to fallow (land which is not planted in hopes of increasing its soil moisture content) for a dry-land farming operation. They developed a simple DP model which included five soil moisture levels. Transitions for the soil moisture state variable are stochastic, and the distribution is affected by the plant/no plant decision. The convergent DP solution was to fallow in the lowest moisture state and plant in the remaining states. Burt and Allison compare their DP decision rule to a decision rule of continuous cropping of wheat and a rule which consists of alternating wheat and fallow. Using the information in Table 4, the expected long run yearly net returns (LRYNR) for each of the three decision rules are calculated below.

The first step in calculating LRYNR is to develop a transition matrix and an immediate net return vector. For the convergent DP decision rule the transition matrix is

$$P(DP) = \begin{bmatrix} 0 & 1/20 & 5/20 & 7/20 & 7/20 \\ 9/23 & 7/23 & 7/23 & 0 & 0 \\ 9/23 & 7/23 & 7/23 & 0 & 0 \\ 9/23 & 7/23 & 7/23 & 0 & 0 \\ 9/23 & 7/23 & 7/23 & 0 & 0 \end{bmatrix}. \qquad (32)$$

This matrix is obtained from Table 4 using the convergent decision rule of fallowing in the lowest soil moisture state and planting in the remaining

Table 4. Burt and Allison's Wheat Cropping Decision Problem, Transition Probabilities and Expected Returns.

Current State[a]	Decision[b]	State Next Period 1	2	3	4	5	Expected Net Returns[c]
1	F	0	1/20	5/20	7/20	7/20	-2.33
	W	9/23	7/23	7/23	0	0	4.52
2	F	0	0	1/20	5/20	14/20	-2.33
	W	9/23	7/23	7/23	0	0	32.07
3	F	0	0	0	1/20	19/20	-2.33
	W	9/23	7/23	7/23	0	0	36.26
4	F	0	0	0	0	1	-2.33
	W	9/23	7/23	7/23	0	0	36.78
5	F	0	0	0	0	1	-2.33
	W	9/23	7/23	7/23	0	0	47.63

Source: Burt and Allison p. 130.

a) Soil moisture state, state 1 is the lowest soil moisture and state 5 is the highest soil moisture.

b) F is used to denote the fallow decision and W denotes plant wheat decision.

c) Expected immediate net return in dollars per acre given current soil moisture state and decision undertaken.

states. Expected immediate net returns vector for the DP decision rule is

$$R(DP) = \begin{bmatrix} -2.33 \\ 32.07 \\ 36.26 \\ 36.78 \\ 47.63 \end{bmatrix}. \tag{33}$$

Applying equations (3) and (7) to the probability matrix given by equation (32) and solving resulting the system of linear equations gives a limiting probability vector of

$$\pi(DP) = [\ .281 \quad .233 \quad .289 \quad .096 \quad .101\ ]. \tag{34}$$

Expected LRYNR for the convergent DP decision rule is:

$$\begin{aligned} E(NRDP) &= \pi(DP)\ R(DP) \\ &= \$25.63. \end{aligned} \tag{35}$$

Continuous wheat transition probability matrix is:

$$P(W) = \begin{bmatrix} 9/23 & 7/23 & 7/23 & 0 & 0 \\ 9/23 & 7/23 & 7/23 & 0 & 0 \\ 9/23 & 7/23 & 7/23 & 0 & 0 \\ 9/23 & 7/23 & 7/23 & 0 & 0 \\ 9/23 & 7/23 & 7/23 & 0 & 0 \end{bmatrix}. \tag{36}$$

It can easily be seen that the limiting probability vector for the continuous wheat decision rule is

$$\pi(W) = [\ 9/23 \quad 7/23 \quad 7/23 \quad 0 \quad 0\ ]. \tag{37}$$

The expected LRYNR for continuous wheat is then:

$$\begin{aligned} E(NRW) &= \pi(W)\ R(W) \\ &= \pi(W) \begin{bmatrix} 4.52 \\ 32.07 \\ 36.26 \\ 36.78 \\ 47.63 \end{bmatrix} \\ &= \$22.56. \end{aligned} \tag{38}$$

Calculation of the expected LRYNR for the decision rule of alternating wheat and fallow is not as straight forward as for the other two decision rules. A single transition matrix cannot be specified because the matrix is dependent on the decision. But modifying equation (5) gives

$$\begin{aligned} \pi(BF) &= \pi(BW)\ P(W), \text{ and} \\ \pi(BW) &= \pi(BF)\ P(F) \end{aligned} \tag{39}$$

where π(BW)is the state occupancy vector before wheat is planted (after a year

of fallow), π(BF) is the state occupancy vector before fallow (after wheat has been planted and harvested), P(F) is the probability transition matrix given the producer fallowed, and P(W) is the probability transition matrix given the producer planted wheat. In equation (39), P(F) can be obtained from the information in Table 4. Given the information in Table 4, π(BF) is independent of the soil moisture state before wheat was planted. This can easily be shown by specifying matrix P(W), which can be specified from the information in Table 4 as,

$$P(W) = \begin{bmatrix} 9/23 & 7/23 & 7/23 & 0 & 0 \\ 9/23 & 7/23 & 7/23 & 0 & 0 \\ 9/23 & 7/23 & 7/23 & 0 & 0 \\ 9/23 & 7/23 & 7/23 & 0 & 0 \\ 9/23 & 7/23 & 7/23 & 0 & 0 \end{bmatrix}. \tag{40}$$

With P(W) specified as in equation (40), any π(BW) will result in the same π(BF), specifically,

$$\pi(BF) = [\ 9/23 \quad 7/23 \quad 7/23 \quad 0 \quad 0\ ]. \tag{41}$$

Burt and Allison base this assumption on the argument that "...at some time during the season of wheat growth, essentially all of the moisture in the root zone will be exhausted, making soil moisture at planting time have a negligible effect on soil moisture the following year at planting time" (p. 130). With this assumption, π(BW) is calculated as

$$\pi(BW) = [\ 9/23 \quad 7/23 \quad 7/23 \quad 0 \quad 0\ ] \begin{bmatrix} 0 & 1/20 & 5/20 & 7/20 & 7/20 \\ 0 & 0 & 1/20 & 5/20 & 14/20 \\ 0 & 0 & 0 & 1/20 & 19/20 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{42}$$

$$= [\ 0 \quad .020 \quad .113 \quad .228 \quad .639\ ].$$

Expected LRYNR is then calculated as

$$
\begin{aligned}
E(NRFW) &= .5\{\pi(BW)\ R(W) + \pi(BF)\ R(F)\} \\
&= .5\ \{\ [0 \quad .020 \quad .113 \quad .228 \quad .639] \begin{bmatrix} 4.52 \\ 32.07 \\ 36.26 \\ 36.78 \\ 47.63 \end{bmatrix} \\
&\qquad + [9/23 \quad 7/23 \quad 7/23 \quad 0 \quad 0] \begin{bmatrix} -2.33 \\ -2.33 \\ -2.33 \\ -2.33 \\ -2.33 \end{bmatrix} \} \qquad (43) \\
&= .5(43.56 - 2.33) \\
&= \$20.62
\end{aligned}
$$

where R(F) is the cost of fallowing. The value in equation (43) is divided by one-half because the decision maker plants or fallows every other year. Calculated expected LRYNRs indicate that the optimal DP decision rule is approximately $3.00 better than the continuous wheat decision rule and approximately $5.00 better than alternating wheat and fallow.

Practical Aspects

As noted earlier, most applied DP models contain hundreds or thousands of states. Application of the methodology described in this paper changes little as the size of the transition matrix increases. But, some practical issues should be discussed. One of the most important issues is determining the classification of the stochastic process given by the convergent DP solution. That is, does the transition matrix have an absorbing state, a single ergodic set or does it contain several ergodic sets? With hundreds of states, it is no longer possible to visibly examine a matrix and determine how many ergodic sets are present. As previously illustrated by several examples, determining the classification of the transition matrix is important in determining if the matrix has a unique limiting probability vector. In practice; therefore, it is suggested that one of the previously described methodologies is used to determine the number of ergodic and transient sets. If more than one ergodic set exists in the matrix, it is suggested that limiting probabilities for initial conditions that encompass each ergodic set and all transient states be calculated. Expected long run net returns can then be presented for a set of initial conditions. It should be noted that

any initial state within an ergodic set gives a unique probability vector for that ergodic set. Only one initial condition; therefore, needs to be specified for each ergodic set. This condition, in general, is not true for transient sets. Limiting probabilities for each transient state need to be calculated.

Our experiences using equations (5) and (15) have been that in general convergences of the limiting probabilities occurs fairly rapidly. Obviously, the speed at which convergence occurs depends on the convergent tolerance. Tolerances of 0.001 and 0.0001 have been used successfully with the FORTRAN program given in Appendix A. Further, a convergence tolerance of 0.000001 has been successfully tested on several matrices. One exception to rapid convergence is the 3 x 3 transition matrix example given in Gillett. This matrix required over 40 iterations to converge. This illustrates convergence is matrix specific.

Two other points need to be mentioned. First, be innovative when applying the principles discussed here. Burt and Allison's decision rule of planting wheat and fallowing illustrate an innovative use of Markov principles. Finally, when calculating limiting probabilities it is suggested that, as a check, the sum of the state occupancy probabilities be calculated to determine if they sum to one. This procedure has been found to be a useful check of the state occupancy calculation program and a check of the transfer of the transition probability matrix from the DP program to the state occupancy program.

PROCESSES WITH MULTIPLE TRANSITION MATRICES

What if in Burt and Allison's problem under the wheat/fallow decision rule π(BF) was not independent of the soil moisture before planting? Could limiting probabilities and the Markov principles still be applied? Further, DP models are starting to define more than one stage per year. Mjelde, Garoian, and Conner's hay inventory model, for example, defines months as the stages. Usually, in this type of models the transition matrix varies by months within a year, but are identical between years. To clarify, the transition matrix from one month to another, for example, January to February is the same for each year, but the transition matrix varies between months within a year (Jan. to Feb. differs from Feb. to March, etc.). If the

transition matrices did not vary, then the principles discussed earlier would apply.

The following discussion illustrates how a process with multiple transition matrices can be analyzed using the principles previously discussed. It will be proved that under certain conditions unique limiting probability vectors exist. Equation (5) again provides the basis for the discussion. Applying equation (5) to a discrete stochastic process with two transition matrices (e.g. six month periods within a year) gives

$$\begin{aligned} \pi 2(0) &= \pi 1(0)\ P1 \\ \pi 1(1) &= \pi 2(0)\ P2 \\ \pi 2(1) &= \pi 1(1)\ P1 \\ \pi 1(2) &= \pi 2(1)\ P2 \\ . &= . \quad . \\ . &= . \quad . \\ . &= . \quad . \\ \pi 1(t) &= \pi 2(t\text{-}1)\ P2 \\ \pi 2(t) &= \pi 1(t)\ P1 \end{aligned} \tag{44}$$

where, $\pi 1(t)$ is the state occupancy vector at time period 1 within transition time (year) t, $\pi 2(t)$ is the state occupancy vector at time period 2 within year t, P1 and P2 are the transition probability matrices at the two time periods within the year, and $\pi 1(0)$ is the initial state occupancy vector.

Expanding the lake level example to include two transitions in each year is informative. In this scenario the managers of the lake are concerned with the level of the lake at two time periods during the year. These two periods, for example, may be after two rainy seasons of the year. Define the two time periods to be January and July. Let the following matrix, P1, represent the transition from January to July

$$P1 = \begin{bmatrix} .5 & .3 & .2 \\ .2 & .6 & .2 \\ .1 & .5 & .4 \end{bmatrix}. \tag{45}$$

Without loss of any generality, P1 can be the same matrix as used earlier. Let the transitions in lake levels from July to January, a dryer period, be

$$P2 = \begin{bmatrix} .7 & .2 & .1 \\ .4 & .5 & .1 \\ .3 & .4 & .3 \end{bmatrix}. \tag{46}$$

Applying equation (44) to the expanded lake level example gives the state

occupancy vectors listed in Table 5 for various initial lake levels. The state occupancy vectors are tending towards the same vector for January and for July, but these vectors differ between January and July. Calculating the state occupancy vector gives January's limiting probability vector as [.476 .379 .146], and July's as [.328 .443 .229].

As noted earlier, the type of stochastic process addressed in the expanded lake level example has not been formally addressed to our knowledge. It will be shown that, in general, a problem with m different transition matrices that repeat will have m unique limiting probability vectors if the vectors exist. The next section proves that unique limiting probability vectors exist under certain conditions. This type of stochastic process will be referred to as a discrete stochastic process with multiple transition matrices (DSPMTM).

Existence of Unique Limiting Probability Vectors

The proof that unique probability vectors exist in DSPMTM relies on the theorem that a transition matrix that contains a single ergodic set (including regular matrices) has a unique limiting probability vector. As stated earlier, a proof of the uniqueness with a single ergodic transition matrix can be found in Kemeny and Snell or Howard. Several examples which rely on this theorem were presented earlier. The proof that unique limiting probability vectors exist will be developed for a process with two transition matrices. This will then be generalized to the case with m transition matrices.

Theorem 1: If [P1 P2] is a single set regular ergodic matrix, unique limiting probability vectors exist for $\pi 1$ and $\pi 2$.

Proof: Rewriting equation (44) gives

$$
\begin{aligned}
&\pi 2(0) = \pi 1(0)\ P1 \\
&\pi 1(1) = \pi 2(0)\ P2 = \pi 1(0)\ P1\ P2 \\
&\pi 2(1) = \pi 1(1)\ P1 = \pi 1(0)\ P1\ P2\ P1 \\
&\pi 1(2) = \pi 2(1)\ P2 = \pi 1(0)\ P1\ P2\ P1\ P2 = \pi 1(0)\ [P1\ P2]^2 \\
&\pi 2(2) = \pi 1(2)\ P1 = \pi 1(0)\ [P1\ P2]^2\ P1 \\
&\qquad\vdots \qquad\qquad \vdots \qquad\qquad \vdots \\
&\pi 1(t) = \pi 2(t\text{-}1)\ P2 = \pi 1(0)\ [P1\ P2]^t \\
&\pi 2(t) = \pi 1(t)\ P1 = \pi 1(0)\ [P1\ P2]^t\ P1.
\end{aligned}
\tag{47}
$$

Table 5. State Occupancy Vectors for January and July for the Lake Level Example[a].

	Year									
	0		1		2		3		4	
*i	Jan.	July	Jan.	July	Jan.	July	Jan.	July	Jan.	July
				Lake Level Initial Below Normal						
π_1	1	.500	.530	.345	.481	.330	.476	.328	.476	.328
π_2	0	.300	.330	.427	.374	.441	.378	.443	.379	.443
π_3	0	.200	.140	.228	.146	.229	.146	.229	.146	.229
				Lake Level Initial Normal						
π_1	0	.200	.440	.318	.473	.327	.475	.328	.476	.328
π_2	1	.600	.420	.454	.382	.444	.379	.443	.379	.443
π_3	0	.200	.142	.228	.146	.229	.149	.229	.146	.229
				Lake Level Initial Above Normal						
π_1	0	.100	.390	.299	.466	.325	.475	.328	.475	.328
π_2	0	.500	.430	.467	.387	.445	.380	.443	.379	.443
π_3	1	.400	.180	.236	.147	.229	.146	.229	.146	.229

a) Rounded to 3 decimal places.

To prove that limiting probability vectors exist, first consider the calculation of $\pi 1(t)$,

$$\pi(t) = \pi 1(0)\ [P1\ P2]^t. \tag{48}$$

resulting from the multiplication of P1 and P2 is a transition matrix that is either regular or has a single regular ergodic set, $\pi 1$ has a unique limiting probability vector as t goes to positive infinity. For now assume the matrix [P1 P2] is a transition matrix with a single regular ergodic set, then $\pi 1$ has a unique limiting probability vector such that for a sufficiently large t the following condition holds

$$\pi 1(t) = \pi 1(t\text{-}1). \tag{49}$$

It, therefore, is shown for certain set of conditions a unique limiting probability vector for $\pi 1$ exists. Using this result, now consider $\pi 2$. For a unique limiting probability vector to exist for $\pi 2$ the following condition must hold for a sufficiently large t

$$\pi 2(t) = \pi 2(t\text{-}1). \tag{50}$$

The left and right side components of equation (50) can be calculated as

$$\pi 2(t\text{-}1) = \pi 1(t\text{-}1)\ P1, \text{ and} \tag{51a}$$

$$\pi 2(t) = \pi 1(t)\ P1. \tag{51b}$$

For a sufficiently large t, equation (49) holds. Combining equation (49) and (51b) gives

$$\pi 2(t) = \pi 1(t)\ P1 = \pi 1(t\text{-}1)\ P1, \tag{52}$$

but equation (51a) states

$$\pi 2(t\text{-}1) = \pi 1(t\text{-}1)\ P1, \tag{53}$$

therefore,

$$\pi 2(t) = \pi 2(t\text{-}1) \tag{54}$$

which is a necessary condition for a unique limiting probability vector to exist. Further, if $\pi 1$ is unique then $\pi 2$ will be unique because equation (53) reduces to a unique vector ($\pi 1$) post-multiplied by a given matrix (P1); therefore, $\pi 2$ must be unique.

The proof to this point has relied on the assumption that [P1 P2] was a transition matrix with a single ergodic set. Each component of this assumption is now examined. Both P1 and P2 must be transition matrices for the process to be a DSPMTM. It, therefore, must be shown that the multiplication of two transition matrices results in a transition matrix.

Recall that, in a transition matrix, all elements within the matrix must be equal to or greater than zero and less than or equal to one. Further, the rows must sum to one. If P1 and P2 have m rows and columns the n, r element is given by

$$[P1\ P2]_{nr} = \sum_{i=1}^{m} P1_{ni}\ P2_{ir} \tag{55}$$

where the subscripts denote row and column for an individual element in each matrix. The sum of the elements in the nth row is

$$\sum_{r=1}^{m} \sum_{i=1}^{m} P1_{ni}\ P2_{ir}. \tag{56}$$

For the matrix [P1 P2] to be a transition matrix, the sum given in equation (56) must be equal to one for each row. Equation (56) can be rewritten as

$$\sum_{i=1}^{m} P1_{ni} \sum_{r=1}^{m} P2_{ir}. \tag{57}$$

The two individual sums in equation (57) represent the sum of each row in the respective transition matrix, which by definition must sum to one. Equation (57); therefore, equals one. Next, it must be shown that every element in the matrix [P1 P2] satisfies the condition $0 \leq [P1\ P2]_{nr} \leq 1$ for all n, r = 1, 2, ..., m. The first part of this condition ($0 \leq [P1\ P2]_{nr}$) follows from multiplying and adding non-negative numbers. All elements in P1 and P2 are non-negative; therefore, multiplying elements and adding the resultant product must give a non-negative number. The second part of the condition ($[P1\ P2]_{nr} \leq 1$) holds because it has been shown elements in [P1 P2] are non-negative and the rows must sum to one; therefore, no element can be larger than one. The multiplication of two transition matrices; therefore, results in a transition matrix.

The proof then rests on the assumption that the matrix resulting from post-multiplying P1 by P2 has a single ergodic set. Unfortunately, the only way to determine if [P1 P2] contains a single ergodic set is to examine the matrices involved. The assumption that [P1 P2] contains a single ergodic set is similar to the assumption necessary for a Markov process with a single transition matrix, P. That is, the transition matrix, P, must contain only a single ergodic set for unique limiting probabilities to exist. As was the case with the single transition matrix, the existence of a single ergodic set

is process specific. This assumption is no more limiting in the two transition matrix case than in the one transition matrix case, in fact maybe less limiting.

Examples of transition matrices with differing ergodic sets and number of ergodic sets may help in clarifying the following two points. First, the existence of a single ergodic set is process specific. Second, the assumption of a single ergodic set in the two transition matrix process may be less limiting than in the one transition matrix case. Examples of multiplying various transition matrices are given in Table 6. Following Kemeny and Snell, an ergodic set can be represented by an absorbing state to simplify the matrix. This follows from the definition of an ergodic set, that is, once a process enters an ergodic set, it can never leave that set. Seven examples of different P1 and P2 matrices are given in Table 6 each with different ergodic sets.

In example 1, both transition matrices, P1 and P2, are regular, and the resulting [P1 P2] matrix is regular. This result can be explained intuitively using the definition of regular. Recall, a regular process is one in which the process can go from any given state to any other state. In a two transition matrix process, if the process can go from any state to any other state at both time period 1 and at time period 2, it follows that the overall process can move from a given state to any other state. In example 2, both matrices have a single ergodic set and this set is identical in each transition matrix. The overall process is represented by a single ergodic matrix. Matrices P1 and P2 in example 2 both have state 1 representing the ergodic set. Once the process enters state 1, it can not leave this state at either time period. The overall process; therefore, has state 1 as the ergodic set.

Examples 3 through 6 are variations of the following scheme. P1 has at least one ergodic set (represented by an absorbing state) which is a transient set in P2 and at least one ergodic set in P2 is a transient set in P1. In these examples, a matrix with a single ergodic set (recall a regular matrix has a single ergodic set) results from the multiplication of P1 and P2. Although, a set of states is ergodic at time period 1, the transient nature of these states in P2 allows the process to move out of the time period 1 ergodic

set at time period 2. This combination of transient and ergodic sets allows the process to move between the states. This is shown dramatically in examples 3 and 6 in which multiplying P1 by P2 results in a regular matrix. In examples 4 and 5, a transient and ergodic set result from multiplying P1 by P2. This result follows from the discussion of example 2.

In examples 1 through 6, the process will have unique limiting probability vectors at each time period, whereas example 7 shows that a unique limiting vector does not always exist. The examples show that the classification of the matrix resulting from multiplying P1 by P2 will be process specific. Some of the individual transition matrices do not have unique limiting probability vectors, if the individual transition matrices were considered separately (see example 6). A formal examination of the P1 and P2 matrices is given next. Necessary and sufficient conditions on the P1 or P2 matrices for the existence of unique limiting probability vectors are identified.

Necessary and Sufficient Conditions on P1 and P2

The proof of the existence of unique limiting probability vectors relies on the assumption that [P1 P2] has a single ergodic set. A sufficient condition for [P1 P2] to contain a single ergodic set is that either P1 or P2 contains a single ergodic set. This condition is given in the following theorem and proof.

Theorem 2: If either P1 or P2 is single set ergodic, then [P1 P2] is single set ergodic.

Proof by contradiction: Suppose that [P1 P2] is not single set ergodic. This would imply that [P1 P2] can be partitioned in the following manner after suitable row operations.

$$\begin{bmatrix} a_1 & 0 & 0 & 0 & 0 \\ 0 & a_2 & 0 & 0 & 0 \\ 0 & 0 & a_3 & 0 & 0 \\ . & . & . & . & . \\ 0 & 0 & 0 & 0 & a_m \end{bmatrix} \tag{58}$$

where a_i are submatrices representing different ergodic sets. It is not possible to partition one or both P1 and P2 as above because one or both are single set ergodic. Moreover, post-multiplication of a single set ergodic matrix by another matrix will not result in a matrix partitionable as above.

Table 6. Examples of Multiplying Different Transition Matrices.

[P1]			[P2]				[P1 P2]		
Example 1 - Both Matrices Regular									
.2	0	.8	.4	.5	.1		.48	.10	.42
.1	.2	.7	.1	.6	.3	=	.41	.17	.42
.7	.3	0	.5	0	.5		.31	.53	.16
Example 2 - Both Matrices with One Ergodic Set - Similar									
1	0	0	1	0	0		1	0	0
.2	.2	.6	.4	.5	.1	=	.28	.16	.56
.1	.1	.8	0	.1	.9		.14	.13	.73
Example 3 - Both Matrices with One Ergodic Set - Differ									
0	.2	.8	1	0	0		.50	.40	.10
.5	.2	.3	.5	.4	.1	=	.75	.20	.05
0	0	1	.5	.4	.1		.50	.40	.10
Example 4 - One Matrix with One Ergodic Set Other Matrix Two Ergodic Sets									
1	0	0	.5	.2	.3		.50	.20	.30
.5	.3	.2	0	1	0	=	.25	.40	.35
0	0	1	0	0	1		0	0	1
Example 5 - Both Matrices with Two Ergodic Sets									
1	0	0	.5	.3	.2		.50	.30	.20
.5	.3	.2	0	1	0	=	.25	.45	.30
0	0	1	0	0	1		0	0	1
Example 6 - Both Matrices with Two Ergodic Sets									
1	0	0	0	.5	.2	.2	.1	.50	.20	.20	.10
.2	.2	.2	.4	0	1	0	0	.18	.28	.44	.10
.1	.5	.2	.2	0	0	1	0	.09	.54	.32	.05
0	0	0	1	.2	.1	.5	.2	.20	.10	.50	.20
Example 7 - Both Matrices with Two Ergodic Sets									
1	0	0	1	0	0		1	0	0
.5	.3	.2	.2	.3	.5	=	.56	.09	.35
0	0	1	0	0	1		0	0	1

One can always move to another state in an ergodic set. If one set is not single set ergodic, the size of the ergodic set may be reduced. The single set ergodic matrix, however, will allow movement into that set. A contradiction; therefore, has been reached and the [P1 P2] must be single set ergodic.

The above sufficient condition, however, is not a necessary condition. As illustrated in the above examples, both the P1 and P2 matrices can include many ergodic sets and the [P1 P2] matrix may contain only one ergodic set. In these cases, a necessary condition for the [P1 P2] matrix to contain only one set is that the ergodic sets in one period's matrix be overlapped by a set (either transient or ergodic) in the other period's matrix. To illustrate, suppose P1 has an ergodic set consisting of states 1 and 2 and another ergodic set containing states 3 and 4. The matrix [P1 P2] will be single set ergodic only if P2 contains a set (transient or ergodic) that has at a minimum state 1 or 2 and state 3 or 4.

Unique limiting probability vectors; therefore, may exist even if all the individual transition matrices contain more than one ergodic set. This illustrates the need to consider the [P1 P2] matrix and not the P1 and P2 matrices individually. It also explains why the assumption of [P1 P2] containing a single ergodic set is necessary. Determining the status of the [P1 P2] matrix can be accomplished using the same methods applicable to all transition matrices. These methods for examining matrices were presented earlier.

The status of sets in the [P1 P2] matrix also holds implications for the status of the [P2 P1] matrix. If the $\pi 1$ vector is unique, then the $\pi 2$ vector is also unique, thereby implying that the [P2 P1] matrix is single set ergodic. This implication occurs because a necessary condition for unique limiting probability vectors is that the transition matrix contain only one ergodic set. Conversely, if $\pi 1$ is not unique, which results because [P1 P2] is not single set ergodic, then $\pi 2$ also is not unique (if $\pi 2$ is unique then $\pi 1$ would also be unique by modification of the previous proof). This implies that the [P1 P2] matrix contains more than one ergodic set. Either [P1 P2] and [P2 P1]; therefore, are both single set ergodic or they both contain more than one ergodic set. This allows examination of only one of the

post-multiplied matrices when determining the status of all the possible combinations of post-multiplied individual transition matrices.

Generalization to m Transition Matrices

The preceding discussion and proof has centered on a DSPMTM with two different transition matrices that repeat. It will be shown here that the above proof can be generalized to a process with m transition matrices that repeat. Examples of processes with more than two transition matrices would be seasonal or monthly models.

Theorem 3: If [P1 P2 ... Pm] is a single set regular ergodic matrix, then m unique limiting probability vectors exist.

Proof: Rewriting equation (47) to represent m transition matrices instead of two results in

$$\pi 1(t) = \pi m(t\text{-}1)\ Pm = \pi 1(0)[\ P1\ \ P2\ \ldots\ Pm]^t \qquad \textbf{(59a)}$$

and for n ≠ 1

$$\pi n(t) = \pi n\text{-}1(t)\ Pn\text{-}1 = \pi 1(0)[\ P1\ P2\ \ldots Pm]^t\ \ P1\ P2\ \ldots\ Pn\text{-}1 \qquad \textbf{(59b)}$$

where n = 1, 2, ..., m and the remaining variables are as defined in equation (47) with appropriate modification for multiple transition matrices. From equation (59) it can be seen that the argument presented in the two transition matrix process applies with the following modification that [P1 P2 ... Pm] contains a single regular ergodic set. If [P1 P2 ... Pm] contains a single regular ergodic set then

$$[\ P1\ P2\ \ldots\ Pm\ ]^t = [\ P1\ P2\ \ldots\ Pm\ ]^{t\text{-}1} \qquad \textbf{(60)}$$

provided t is sufficiently large. From equation (60) and the multiplication of constants argument presented earlier, it follows that

$$\pi n(t) = \pi n(t\text{-}1),\ for\ all\ n, \qquad \textbf{(61)}$$

again provided t is sufficiently large. A process with m transition matrices; therefore, has m unique limiting probability vectors given the condition of [P1 P2 ... Pm] being a matrix containing a single ergodic set.

Be it a process with one transition matrix or m transition matrices, the relevant consideration is the entire process. This is shown in the necessary assumption for the existence of unique limiting probabilities, namely that [P1 P2 ... Pm] contains a single ergodic set. Another way of interpreting the necessary assumption for the previous theorems is that [P1 P2 ... P_m] gives a Markov process transition matrix covering the entire time period for the

original DSPMTM process.

Practical Aspects Concerning Use of m Transition Matrices

Most of the previous discussion on the use of limiting probabilities and practical aspects of calculating these probabilities given for Markov processes holds when the stochastic process has m transition matrices. Determining the number of ergodic and transients sets, for example is necessary when m transition matrices exist. All m state occupancy vectors must be compared using equation (15) as a check for convergence. The FORTRAN program given in Appendix A has been successfully used to calculate limiting probability vectors for problems with 4 transition matrices.

After calculating the limiting probability vectors, these vectors can be used, as before, to calculate expected long run net returns (yields, etc.). In general, expected long run net returns are calculated by

$$E(NR) = \sum_{i=1}^{m} \pi i \; Ri \qquad (62)$$

where πi represents the limiting probability vector at time period i of the transition period and Ri is a vector of net returns associated with time period i. From equation (62), it can easily be seen that expected yearly net returns can be calculated for each period and for the entire process.

Lake Level Example

What is the expected yearly net returns associated with the expanded lake level example given that January's and July's net returns are given by

$$NR(Jan) = \begin{bmatrix} -25. \\ 10. \\ 15. \end{bmatrix} \; and \; NR(July) = \begin{bmatrix} -35. \\ 25. \\ 28. \end{bmatrix} ? \qquad (63)$$

Using equation (62) expected net returns can be calculated by

$$E(NR) = \pi(Jan)\; NR(Jan) + \pi(July)\; NR(July). \qquad (64)$$

Limiting probability vectors for January and July were calculated using the FORTRAN program given in Appendix A. Using these limiting probabilities,

equation (64) can be rewritten as

$$E(NR) = \begin{bmatrix} .476 & .379 & .146 \end{bmatrix} \begin{bmatrix} -25. \\ 10. \\ 15. \end{bmatrix} + \begin{bmatrix} .328 & .443 & .229 \end{bmatrix} \begin{bmatrix} -35. \\ 25. \\ 28. \end{bmatrix} \quad (65)$$

$$= -5.91 + 6.00$$

$$= 0.09$$

The owners of the water rights can expect to lose \$5.91 thousand from January to July and gain \$6.00 thousand from July to January. For the entire year the owners will gain \$0.09 thousand from owning the water property rights.

This breakdown of net returns by time period is an example of the type of information that can be calculated with a process containing m transition matrices. As was the case in a one transition matrix process, calculation of limiting probabilities allows for a more succinct presentation of DP models and/or comparison of decision rules.

CONCLUDING REMARKS

This paper addressed several of the obstacles to the adoption of stochastic dynamic programming namely cursory treatment of the subject and the decision and acceptance rule curses. Basic principles of Markov processes were introduced and illustrated. These principles were then used in conjunction with dynamic programming models to provide a technique which allows a more succinct reporting of results. Although much of the discussion synthesizes previous literature, one area of the discussion has not previously appeared in the literature. This is discrete stochastic processes with multiple transition matrices.

Appendices A and B contain FORTRAN programs to determine limiting probabilities and determining ergodic and transient sets within a transition matrix. These programs have been successfully used on an 968 x 968 transition matrix (Mjelde *et al.*). Further, the programs have been tested on three different FORTRAN compilers, Ryan McFarland and Microsoft FORTRAN on IBM microcomputers, and FORTRAN Compiler for HP-UNIX operating system for a Hewlett Packard minicomputer. This paper is not intended as software documentation, but the programs are presented to aid in the use and development of dynamic programming models.

REFERENCES

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

Burt, O.R. "Dynamic Programming: Has Its Day Arrived." *West. J. Agr. Econ.* 7(1982):381-393.

Burt, O.R. and J.R. Allison. "Farm Management Decisions with Dynamic Programming." *J. Farm Econ.* 45(1963):121-136.

Clark, A.B. and R.L. Disney. *Probability and Random Processes for Engineers and Scientists*. New York: John Wiley & Sons, Inc. 1970.

Gillett, B.E. *Introduction to Operations Research: A Computer Oriented Algorithm Approach*. New York: McGrann Hill, Inc. 1976.

Heyman, D.P. and M.J. Sobel. *Stochastic Models in Operations Research Volume I Stochastic Processes and Operating Characteristics*. New York: McGraw-Hill, Inc. 1982.

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

Kemeny, J.G. and J.L. Snell. *Finite Markov Chains*. Princeton, N.J.: D. Van Nostrand Company, Inc., 1960.

Mjelde, J.W., Garoian, and J.R. Conner. "Counterintuitive Decision Rules in Complex Dynamic Models: A Case Study." In Proceedings: *Applications of Dynamic Programming to Agricultural Decision Problems*. C.R. Taylor, ed. In progress. 1990.

Mjelde, J.W., W.D. Harris, J.R. Conner, G.D. Schnitkey, M.K. Glover, and L. Garoian. "Existence of Unique Limiting Probability Vectors in Stochastic Processes with Multiple Transition Matrices." Working Paper. Dept. of Ag. Econ. Texas A&M University. 1991.

Mjelde, J.W., C.R. Taylor, and G.L. Cramer. "Optimal Marketing Strategies for Wheat and Corn Producers in the 1982 Farm Program." *North Central J. Agr. Econ.* 7(1985):51-60.

Rorres, C. and H. Anton. *Applications of Linear Algebra*. New York: John Wiley & Sons. 1984.

Schnitkey, G.D., C.R. Taylor, and P.J. Barry. "Evaluating Farmland Investments Considering Dynamic Stochastic Returns and Farmland Prices." *West. J. Agr. Econ.* 14(1989):143-156.

Taylor, C.R. "The Fading Curse of Dimensionality." Paper Presented at a Meeting of the Consortium for Research on Crop Production Systems, Allerton House, Monticello, Illinois. June 8-10, 1987. Staff Paper No. 87 E-385 Dept. Agr. Econ. University of Illinois. June, 1987.

Taylor, C.R. "Dynamic Programming and the Curses of Dimensionality." In Proceedings: *Applications of Dynamic Programming to Agricultural Decision Problems*. C.R. Taylor, ed. In Progress. 1990.

APPENDIX A

FORTRAN Program to Calculate Limiting Probability Vectors Using the equation $\pi(t) = \pi(t-1)P$.

To run the program the following data must be in a control file called "PCONV". The program reads the numeric data in free format and the Alphanumeric data in character *8 format.

Control file---PCONV---Data Order and Description

Numeric (data type is indicated in parentheses)

iter - number of iterations (integer).

nstate - number of states (integer).

tol - convergence tolerance (real).

toll - tolerance used to check if rows sum to one (real).

itol - number of iteration before the program starts to check for convergence (real).

initial - state the system initial occupies (integer).

nmatrix - number of transition matrices (integer).

nfile - number of unique data files that contain the transition matrices, must be either one or equal to nmatrix (integer).

nret - number of unique data files that contain net return data, must equal either one or nmatrix (integer).

switch - equals zero when the transition matrix data are found in one file, anything else the transition data are found in nmatrix files (integer).

switch2 - equals zero when nets returns data are found in one file, anything else the net returns are found in nmatrix files (integer).

switch3 - equals zero print state occupancy vectors if convergence is reached equals anything else vectors will not be printed if convergence is reach. If convergence is not reached vectors will be printed regardless of the value for switch3 (integer).

switch4 - equals zero program will calculate long run returns, equals anything else returns will not be calculated (integer).

Alphanumeric - one filename per line

output - name of output file

transition - list all files which contain transition matrix data must be listed in order of occurrence.

return - list all files which contain net returns data. Must be in order of occurrence. A file name must be given even if expected net returns are not going to be calculated. The file will not be opened or read if switch4 does not equal zero.

Both the transition matrix data and the net returns data are read in free format. The transition data is read one row at a time, that is, the

probability of going from the current state to all states in the next period is read before going to the next current state. That is, using the notation in this paper, P_{ij}'s, for a given i and all j are read then i is varied. Returns are read by state (1 to n) again in order of occurrence.

When using the program, the dimension statements in the main program and the two subroutines need to be changed to your problem's dimensions. The first component in arrays **ret** and **pi** needs to be set equal to the number of states in your problem or larger, whereas, the first two components of array prob needs to be set to the number of states or larger.

Example. Transition data are in two files (TRAN.ONE and TRAN.TWO), returns are in one file (RET), and the system has three states. The listing of each file along with the generated output is given.

PCONV

```
100   3   .0001   .0001   2   1   2   2   1   1   0   0   0
OUTPUT
TRAN.ONE
TRAN.TWO
RET
```

TRAN.ONE

```
   .2   .3   .5
   .5   .3   .2
    0   .8   .2
```

TRAN.TWO

```
   .3   .3   .4
   .4   .4   .2
   .5   .1   .4
```

RET

```
   10.   11.   12.
   12.    5.    2.
```

OUTPUT

```
initial state =       1

convergence reached

state occupancy vectors are

 .4100575149  .2808485925  .3090940714
 .2224358022  .4545471072  .3230172694

number of iterations =      6
convergent tolerance =  .0001000000

expected returns for period  1 = $    10.899038

expected returns for period  2 = $     5.588000

total expected returns =       16.4870
```

```
*****************************************************************
*                              conv
*
*     program to calculate convergent probabilities and long run
*     probabilities by multiplication of state occupancy vector and
*     the transition probability matrix
*
*              department of agricultural economics
*                      texas a&m university
*
*                           september 1990
*
*****************************************************************
*   Variable Definitions:
*     prob(i,j,l) - matrices of transition probabilities -- i is the
*       current state, j is the state you are going to and l is the
*       matrix number
*     pi(i,l,2) - vector of state occupancy probabilities -- i is the
*       state, l is the matrix number, and 2 is used to check for convergence
*     ret(i,l) - vector of immediate returns -- i is for the state and
*       l is the matrix number
*     iter - the number of iterations
*     nstate - number of states in the system
*     tol - user specified convergence tolerance
*     toll - tolerance used to check if initial vectors sum to one
*     itol - number of iteration to start checking for convergence
*     initial - number of the initial state the system occupies
*     nmatrix - number of unique transition matrices
*     nfile - number of files the transition matrices are to be read from
*     nret - number of files the returns are to be read from
*     switch - equals zero transition matrices are in one file
*            -  equals anything else transition matrices are located in
*              nmatrix files in this case nmatrix=nfile
*     switch2 - equals zero returns are found in one file
*             - equals anything else returns are found in nret files
*               nret=nmatrix
*     switch3 - equals zero print state occupancy vectors only if
*               convergence is reached if not reached vectors will be printed
*             - equals anything else don't print vectors again with
*               convergence being reached
*     switch4 - equals zero calculate long run returns
*             - equals anything else don't calculate returns
*     output - alphanumeric character delineating the user specified
*              output file
*     matfile - alphanumeric vector of transition files
*     retfile - alphamumeric vector of return files
*     returns - calculated long run expected returns
*     nn,kn,i,j,k,l,ll,kp - counters within the program
*     temp,t1,temp1,test1 - temporary variables within the program
*****************************************************************
      integer switch,switch2,switch3,switch4
      real prob(968,968,2),pi(968,2,2)
      real ret(968,2)
      character*8 output,matfile(2),retfile(2)
      common /blk1/ prob
      common /blk2/ matfile
      common /blk3/ pi,ret
   30 format(a8)
   35 format(1x,10(f12.10,1x))
   70 format(1x/1x,'convergence not reached given your tolerance'/
     &1x,'and number of iterations')
   71 format(1x,'convergence reached'/)
   72 format(/1x,'state occupancy vectors are'/)
   75 format(/1x,'number of iterations = ',i5/
     &1x,'convergent tolerance = ',f12.10/)
   80 format(/1x,'sum of state occupancy vector for period',
```

```
     & i3,' = ',f12.6)
   81 format(/1x,'convergened probabilites do not sum to one'/
     & 1x,'at matrix = ',i3,3x,'sum equals = ', f10.8)
   82 format(/1x,'initial state = ',i6/)
   90 format(/1x,'expected returns for period',i3,' = $ ',f12.6)
 1012 format(/1x,'total expected returns = ', f13.4)
  101 format(1x,'probability in transition matrix does not sum to one',/
     & 1x,'matrix = ',i3,2x,'state = ',i3,2x,'sum = ',f10.8)
*****************************************************************
*     read in parmeters
*****************************************************************
      open(unit=9,file='pconv',status='unknown')
      read(9,*) iter,nstate,tol,toll,itol,initial,nmatrix,nfile,nret,
     & switch,switch2,switch3,switch4
      read(9,30)output
      read(9,30) (matfile(l), l = 1,nfile)
      read(9,30) (retfile(l), l = 1,nret)
      close (9)
      open(unit=10,file=output,status='unknown')
*****************************************************************
*     read in the probability matrix of size nstate by nstate
*****************************************************************
      if(switch .eq. 0) then
         call read1(nmatrix,nstate)
       else
         call read2(nmatrix,nstate)
      endif
*****************************************************************
*     check to see if probabilities sum to one
*****************************************************************
      do 501 l = 1,nmatrix
      do 500 i = 1,nstate
      t1= 0.0
      do 525 j=1,nstate
      t1=prob(i,j,l) + t1
  525 continue
*****************************************************************
*     if pij's don't sum to one within specified tolerance output caution
*****************************************************************
      if(abs(1.0-t1) .gt. toll) write(10,101) l,i,t1
  500 continue
  501 continue
*****************************************************************
*     initialize state occupancy vector to be zero except initial state
*****************************************************************
      do 675 i=1,nstate
      pi(i,1,1) = 0.
  675 continue
      pi(initial,1,1) = 1.0
      nn = 2
      write(10,82) initial
*****************************************************************
*    multiply pi times prob
*    transpose the pi vector subscript to allow for comparison
*****************************************************************
      do 700 k = 1,iter
      if(nn.eq.2) then
      nn = 1
      else
      nn = 2
      endif
      kn=nn
*****************************************************************
*     multiplication
*****************************************************************
      do 701 l = 1,nmatrix
```

```
      do 725 j = 1,nstate
      temp1 = 0.0
      do 750 i = 1,nstate
      temp1 = temp1 + pi(i,l,nn)*prob(i,j,l)
  750 continue
      ll = l + 1
      if(l .eq. nmatrix) then
        ll = 1
        kn = 1
        if(nn .eq. 1) kn=2
      endif
      pi(j,ll,kn) = temp1
  725 continue
  701 continue
*********************************************************************
*     test for tolerance of convergance
*     only test past iterations greater than itol
*********************************************************************
      if((k .gt. itol) .and. (k .gt.1)) then
      do 826 l = 1,nmatrix
      do 825 i = 1,nstate
      test1 = abs(pi(i,l,nn)-pi(i,l,kn))
      if(test1 .gt. tol) go to 1000
  825 continue
  826 continue
      go to 1100
 1000 continue
      endif
  700 continue
*********************************************************************
*     convergence not reached given the number of iterations
*********************************************************************
      write(10,70)
      kp = k - 1
      write(10,75) kp,tol
*********************************************************************
*   write pi to file #10
*********************************************************************
      write(10,72)
      do 851 l = 1,nmatrix
      write(10,35) (pi(i,l,nn),i=1,nstate)
  851 continue
*********************************************************************
*     test to see if state occupancy vectors sum to one
*********************************************************************
      do 853 l = 1,nmatrix
      test1 = 0.0
      do 850 i= 1,nstate
      test1 = test1 + pi(i,l,nn)
  850 continue
      write(10,80) l,test1
  853 continue
      go to 1200
*********************************************************************
*     convergance reached given a tolerance of tol
*     write the convergent probabilities for both vectors
*********************************************************************
 1100 continue
      write(10,71)
*********************************************************************
*   write pi to file #10
*********************************************************************
      if(switch4 .eq. 0) then
      write(10,72)
      do 1101 l = 1,nmatrix
      write(10,35) (pi(i,l,nn),i=1,nstate)
```

```
 1101 continue
      endif
      write(10,75) k,tol
*****************************************************************
*    test to see if state occupancy vectors sum to one
*****************************************************************
      do 876 l =1,nmatrix
      test1 = 0.0
      do 875 i= 1,nstate
      test1 = test1 + pi(i,l,nn)
  875 continue
      if(abs(1.0-test1) .gt. toll) write(10,81) l,test1
  876 continue
*****************************************************************
*     calculate expected long run yearly returns
*     read in returns and multiply by state occupancy vector
*****************************************************************
      if (switch4. eq. 0) then
      if(switch2 .eq. 0) then
*****************************************************************
*     all returns are in one file
*****************************************************************
      open(unit=14,file=retfile(1),status='unknown')
      do 877 l = 1,nmatrix
      read(14,*) (ret(i,l),i=1,nstate)
  877 continue
      close(14)
      else
*****************************************************************
*     returns are in seperate files
*****************************************************************
      do 878 l = 1,nret
      open(unit=14,file=retfile(l),status='unknown')
      read(14,*) (ret(i,l),i = 1,nstate)
      close(14)
  878 continue
      endif
      returns = 0.0
      do 901 l = 1,nmatrix
      temp = 0.
      do 900 i = 1,nstate
      temp = temp + (ret(i,l) * pi(i,l,nn))
  900 continue
      returns = returns + temp
      write(10,90) l,temp
  901 continue
      write(10,1012) returns
      endif
 1200 close(10)
      stop
      end
*****************************************************************
*     all probabilities are in one file
*****************************************************************
      subroutine read1(nmatrix,nstate)
      real prob(968,968,2)
      character*8 matfile(2)
      common /blk1/ prob
      common /blk2/ matfile
      open(unit=16,file=matfile(1),status='unknown')
      do 11 l = 1,nmatrix
      do 10 i = 1,nstate
      read(16,*) (prob(i,j,l),j=1,nstate)
   10 continue
   11 continue
      close (16)
```

```
      return
      end
*************************************************************************
*    probabilities are in seperate files
*************************************************************************
      subroutine read2(nmatrix,nstate)
      real prob(968,968,2)
      character*8 matfile(2)
      common /blk1/ prob
      common /blk2/ matfile
      do 11 l = 1,nmatrix
      open(unit=16,file=matfile(l),status='unknown')
      do 10 i = 1,nstate
      read(16,*) (prob(i,j,l),j=1,nstate)
   10 continue
      close (16)
   11 continue
      return
      end
```

APPENDIX B

FORTRAN Program to Determine Ergodic and Transient Sets.

To run the program the following data must be in a control file called "PARAM". The program reads the numeric data in free format and the Alphanumeric data in character *8 format.

Control file---PARAM---data order and description

Numeric (data type is indicated in parentheses)

n - number of states (INTEGER)

tol - tolerance level for binary conversion (REAL)

switch1 - (INTEGER)
= 0 when binary matrix is to be outputted.
= 1 (or nonzero) when binary matrix is not to be outputted.

switch2 - (INTEGER)
= 0 when transformed matrix is to be outputted.
= 1 (or nonzero) when transformed matrix is not to be outputted.

switch3 - (INTEGER)
= 0 when transition matrices are found in one file.
= 1 (or nonzero) when transition matrices are found in more than one file.

nmatrix - number of transition matrices to be inputted and multiplied (INTEGER).

Alphanumeric - one filename per line

output1 - name of output file

matfile(1) - name of first transition matrix file. This will be the only transition matrix file if nmatrix = 1.

matfile(2) - name of second transition matrix file. This will be the last transition matrix file if nmatrix = 2.

matfile(nmatrix) - name of last transition matrix file.

The transition matrix data are read in free format. The data are read in one row at a time, that is, the probability of going from the current state to all states in the next period are read before going to the next current state. That is, using the notation in the paper, P_{ij}'s, for a given i and all j are read then i is varied. Returns are read by state (1 to n) again in order of occurrence.

This program is able to input a single transition matrix from a single input file. Two different ways can be used to input more than one transition matrix. The first is to place all the transition matrices in a single file with no special recognition of the separate matrices (such as a blank line or a title, i.e. transition matrix #2). In this case switch3 would be set to 0 in the parameter file PARAM. The second method involves placing each and

every transition matrix in a separate file. In this case, switch3 would be set to 1 (or any other nonzero integer), the number of files would be indicated, as well as, the name of each individual file. In either of the two cases it is important that all matrices have the same number of states.

When using the program, the dimension statements in the main program and the two subroutines need to be changed to your problem's dimensions. The arrays **aa, a, b, d, c, e,** and **f** all need to be dimensioned at the number of states in your problem or larger.

Example: This example uses the same data files as the previous example.

```
n = 3
tol = .0001
switch1 = 1
switch2 = 1
switch3 = 1
output1 = ERG.OUT
nmatrix = 2
```

The listing of each file along with the generated output is given below.

PARAM

```
3   .0001   1    1    1
ERG. OUT
2
TRAN.ONE
TRAN.TWO
```

TRAN.ONE

```
   .2   .3   .5
   .5   .3   .2
    0   .8   .2
```

TRAN.TWO

```
   .3   .3   .4
   .4   .4   .2
   .5   .1   .4
```

ERG.OUT

```
number of states = 3
the total number of sets = 1
```

```
************************************************************************
**                              ergodic                               **
**                                                                    **
**          program to calculate ergodic & transient sets             **
**                                                                    **
**               department of agricultural economics                 **
**                                                                    **
**                      texas a&m university                          **
**                        september 1990                              **
**                                                                    **
************************************************************************

      real aa(10,10)
      integer a(10,10),b(10,10),d(10),c(10),e(10),f(10)
      integer c1,c2,c3,c4,c5,c6,c7,c8,c9
      integer sumc,sumc1,sumc2,nmatrix,q
      integer switch1,switch2,switch3
      character*8 param,output1,matfile(2)
      common/blk1/ aa,b,c
      common/blk2/ matfile
      common/blk3/ a,d,e

************************************************************************
**                              formats                               **
************************************************************************

  45  format(1x,'number of states = ',i5/
     &1x,'the total number of sets = ',i5/)
  50  format(1x,'*************** set # ',i5,'  ***************',/,
     &1x,'the set is ergodic',/,
     &1x,'the following row(s) comprise set # ',i5/)
  55  format(1x,i5)
  60  format(1x,'*************** set # ',i5,' ***************',/,
     &1x,'the set is transient',/,
     &1x,'the following row(s) comprise set # ',i5/)
  65  format(1x,'set # ',i5,' communicates with the following sets:'/)
  70  format(20i3)
  79  format(1x)
  82  format(a8)

************************************************************************
**     read in parameter file named PARAM which should have:          **
**               1) n -> number of states                             **
**               2) tol -> tolerance level for binary conversion      **
**               3) switch1                                           **
**                    0  denotes user requests output of binary       **
**                       matrix                                       **
**                    1  (or nonzero) denotes user does not request   **
**                       output of binary matrix                      **
**               4) switch2                                           **
**                    0  denotes user requests output of transformed  **
**                       matrix                                       **
**                    1  (or nonzero) denotes user does not request   **
**                       output of transformed matrix                 **
**               5) switch3                                           **
**                    0  denotes transition matrix or matrices are    **
**                       located in a single file                     **
**                    1  (or nonzero) denotes transition matrices     **
**                       are located in more than a single file       **
**                       (only one transition matrix per file)        **
**               6) output1 -> name of the output file                **
**               7) nmatrix -> number of transition matrices to       **
**                             be inputted and multiplied             **
**               8) matfile(i) ->                                     **
**                     matfile(1) -> name of first transition         **
**                                   matrix file                      **
```

```
**                    matfile(2) -> name of second transition     **
**                                  matrix file                   **
**                     .     .                                    **
**                    matfile(nmatrix) -> name of last tranition  **
**                                        matrix file             **
*********************************************************************

      open(unit=14,file='param',status='unknown')

      read(14,*) n,tol,switch1,switch2,switch3
      read(14,82) output1
      read(14,*) nmatrix
      do 99 i=1,nmatrix
      read(14,82) matfile(i)
  99  continue
      close(14)

      open(unit=10,file=output1,status='unknown')

      if(switch3 .eq. 0) then
        call read1(nmatrix,n)
       else
        call read2(nmatrix,n)
      endif

      do 125 i=1, n
      c(i)=0
      d(i)=0
      e(i) = 0
      f(i) = 0
  125 continue

*********************************************************************
**         convert original (inputted) matrix to binary form       **
*********************************************************************

      do 140 i=1,n
      do 150 j=1,n
      if(aa(i,j).gt.tol) a(i,j)=1.
      if(aa(i,j).le.tol) a(i,j)=0.
  150 continue
  140 continue

*********************************************************************
**   output the binary version of the original (inputted) matrix   **
*********************************************************************

      if(switch1.ne.0) go to 210

      do 175 i=1, n
      write(10,70) (a(i,j), j=1,n)
  175 continue
      write(10,79)
      do 176 i=1, n
      write(10,70) (aa(i,j), j=1,n)
  176 continue
      write(10,79)

*********************************************************************
**                  determine the transformed matrix               **
*********************************************************************

  210 do 225 i = 2, n
      do 250 j = 1, i - 1
      if (a(i, j).ne.1) go to 250
      do 275 k = 1, n
```

```
      if (((a(i, k).ne.a(j, k))).and.(a(j, k).eq.1)) a(i,k)=1
  275 continue
  250 continue
      do 300 l = 1, i - 1
      if (a(l, i).ne.1) go to 300
      do 325 m = 1, n
      if (((a(l, m).ne.a(i, m))).and.(a(i, m).eq.1)) a(l,m)=1
  325 continue
  300 continue
  225 continue

**********************************************************************
**                      output transformed matrix                   **
**********************************************************************

      if(switch2.ne.0) go to 380

      do 350 i=1, n
      write(10,70) (a(i,j), j=1,n)
  350 continue
      write(10,79)

**********************************************************************
**      determine transient rows by finding columns with all zeros  **
**********************************************************************

  380 do 370 i = 1, n
      c8 = 0
      do 375 j = 1, n
      if (a(j,i).ne.0) go to 370
      c8 = c8 + 1
      if (c8.eq.n) then
      f(i) = 1
      else
      endif
  375 continue
  370 continue

**********************************************************************
**             calculate the sets (which rows are alike)            **
**********************************************************************

      c1 = 0
      c2 = c1
      c3 = 0
      do 400 i=1, n
      if (e(i).ne.0) go to 400
      c4 = 0
      if (i.eq.n) then
      if (e(i).eq.0) then
      c1 = c1 + 1
      d(c1) = i
      else
      endif
      go to 430
      else
      endif
      do 425 j =i+1, n
      if (e(j).ne.0) go to 425
      c4 = c4 + 1
      if (c4.eq.1) then
      c1 = c1 + 1
      d(c1) = i
      else
      endif
      c7 = 0
```

```
      do 450 k=1, n
      if (a(i, k).ne.a(j, k)) go to 425
      c7 = c7 + 1
      if (c7.eq.n) then
      c9 = f(i) + f(j)
      if ((c9.lt.2).and.(c9.gt.0)) go to 425
      e(j) = 1
      c1 = c1 + 1
      d(c1) = j
      else
      endif
  450 continue
  425 continue
  430 c3 = c3 + 1
      if (((c1-c2).eq.0).and.(i.ne.n)) then
      c1=c1+1
      d(c1)=i
      else
      endif
      c(c3) = c1 - c2
      c2=c1
  400 continue

**********************************************************************
**       calculate and output the total number of sets & states     **
**********************************************************************

      c5=0
      do 475 i=1, n
      if (c(i).eq.0) go to 500
      c5 = c5 + 1
  475 continue
  500 write(10,45) n,c5

**********************************************************************
**                              determine:                          **
**     1) which sets are ergodic                                    **
**     2) which sets are transient                                  **
**     3) which ergodic sets the transient sets communicate with    **
**     4) output all of the above                                   **
**********************************************************************

      sumc = 1
      do 525 i=1, c5
      c7 = 0
      c6 = 0
      do 550 j=1, n
      do 575 k=sumc, (sumc + c(i) - 1)
      if (j.eq.d(k)) go to 550
  575 continue
      if ((a(d(sumc), j).eq.0).and.(f(d(sumc)).eq.0)) then
      c7 = c7 + 1
      if (c7.eq.(n-c(i))) then
      write(10,50)i,i
      do 600 l = sumc, (sumc + c(i) - 1)
      write(10,55) d(l)
  600 continue
      write(10,79)
      else
      endif
      else
      endif
      if ((a(d(sumc), j).ne.0).or.(f(d(sumc)).ne.0)) then
      if (c6.eq.0) then
      write(10,60)i,i
      do 625 l = sumc, (sumc + c(i) - 1)
```

```
      write(10,55) d(l)
  625 continue
      write(10,79)
      write(10,65) i
      c6 = 1
      else
      endif
      if (c6.ne.0) then
      do 650 l = 1, n
      do 675 m = sumc, (sumc + c(i) - 1)
      if (l.eq.d(m)) go to 650
  675 continue
      if (a(d(sumc), l).ne.0) then
      sumc1 = 1
      sumc2 = sumc1 + c(1)
      if (d(sumc1).eq.l) then
      do 700 q = 1, c5
      if (q.eq.i)  go to 700
      if ((l.ge.sumc1).and.(l.lt.sumc2)) write(10,55) q
      if ((l.lt.sumc1).or.(l.ge.sumc2)) then
      sumc1 = sumc2
      sumc2 = sumc2 + c(q + 1)
      else
      endif
  700 continue
      write(10,70)
      else
      endif
      else
      endif
  650 continue
      go to 725
      else
      endif
      else
      endif
  550 continue
  725 sumc = sumc + c(i)
  525 continue

      close(10)

      stop
      end

**********************************************************************
**  subroutine read1: all probability matrices are in one data file  **
**********************************************************************

      subroutine read1(nmatrix,n)
      real aa(10,10),b(10,10),c(10)
      character*8 matfile(2)
      common/blk1/ aa,b,c
      common/blk2/ matfile
      open(unit=16,file=matfile(1),status='unknown')
      do 800 i=1,n
      read(16,*) (aa(i,j),j=1,n)
  800 continue
      if(nmatrix.eq.1) go to 930
      do 825 m=2,nmatrix
      do 850 i=1,n
      read(16,*) (b(i,j),j=1,n)
  850 continue
      do 875 i=1,n
      do 900 k=1,n
      c(k)=aa(i,k)
```

```
  900 continue
      do 910 j=1,n
      temp = 0.0
      do 925 l=1,n
      temp = temp + (c(l)*b(l,j))
  925 continue
      aa(i,j) = temp
  910 continue
  875 continue
  825 continue
  930 close(16)
      return
      end

**************************************************************************
**  subroutine read1: probability matrices are in several data files **
**************************************************************************

      subroutine read2(nmatrix,n)
      real aa(10,10),b(10,10),c(10)
      character*8 matfile(2)
      common/blk1/ aa,b,c
      common/blk2/ matfile
      open(unit=16,file=matfile(1),status='unknown')
      do 950 i=1,n
      read(16,*) (aa(i,j),j=1,n)
  950 continue
      close(16)
      do 975 m=2,nmatrix
      open(unit=16,file=matfile(m),status='unknown')
      do 1000 i=1,n
      read(16,*) (b(i,j),j=1,n)
 1000 continue
      do 1025 i=1,n
      do 1075 k=1,n
      c(k)=aa(i,k)
1075  continue
      do 1050 j=1,n
      temp = 0.0
      do 1060 l=1,n
      temp = temp + (c(l)*b(l,j))
 1060 continue
      aa(i,j) = temp
 1050 continue
 1025 continue
      close(16)
  975 continue
      return
      end
```