Chapter 7

Discrete-Time Discrete-Space Dynamic Models

With this chapter, we begin our study of dynamic economic models. Many economic models are inherently dynamic in that an agent or group of agents must make decisions over time such that an action taken at one date may affect or constrain the actions that may be taken at a later date. Dynamic models arise in many areas of economics. For example, dynamic models have been used to study consumption-savings decisions by individuals, production-investment decisions by firms, and arbitrage pricing in markets for physical and financial assets.

Dynamic economic models often present three complications rarely encountered together in static economic models or dynamic physical science models. First, humans are cogent, future-regarding beings capable of assessing how their actions will affect their future well-being as well as their present well-being. Thus, most useful dynamic economic models are future-looking as opposed to simply forward-moving. Second, human behavior is to a great extent unpredictable. Thus, most useful dynamic economic models are inherently stochastic. Third, the predictable component of human behavior is often complex. Thus, most useful dynamic economic models are inherently nonlinear.

The complications inherent in forward-looking, stochastic, nonlinear dynamic models render the classical tools of Algebra, Calculus, and Analysis of limited usefulness in solving and analyzing dynamic economic models. However, the proliferation of affordable personal computers and their phenomenal increase in speed over the last decade now make it possible for economists to analyze dynamic economic models using numerical methods. The tools of numerical analysis have been applied extensively to dynamic models in the physical sciences, particularly in engineering, physics, and chemistry. Economists can draw many lessons from the experiences of computational physical scientists. However, Economists must also develop novel numerical methods and strategies to solve their own unique problems.

The next three chapters are devoted to the numerical analysis of stochastic dynamic economic models in discrete time. In this chapter we study the simplest such model: the discrete-time, discrete-space Markov decision model. In the discrete Markov decision model, an agent controls an economic process that can attain only finitely many states by choosing from among finitely many possible actions. The discrete Markov decision model is relatively simple to analyze conceptually and numerically. The methods used to analyze the model, moreover, lay the foundations for the methods developed in subsequent chapters to analyze models in continuous time and space.

7.1 Discrete Dynamic Programming

The discrete-time, discrete-space Markov decision model has the following structure: In every period t, an agent observes the state of an economic process s_t, takes an action x_t, and earns a reward $f(s_t, x_t)$ that depends on both the state of the process and the action taken. The state space S, which contains all the states attainable by the process, and the action space X, which contains all actions that may be taken by the agent, are both finite. The state of the economic process follows a controlled Markov probability law. That is, the distribution of next period's state conditional on all currently available information depends only on the current state of the process and the agent's action:

$$\Pr(s_{t+1} = s' | s_t = s, x_t = x, \text{ other information at } t\ ) = P(s'|s, x).$$

The agent seeks a policy or rule $x^* = \{x_t^*\}_{t=1}^T$ of state-contingent actions $x_t = x_t^*(s_t)$ that maximizes the present value of current and expected future rewards over time, discounted at a per-period factor δ:

$$E \sum_t \delta^t f(s_t, x_t).$$

A discrete Markov decision model may have an infinite horizon, $T = \infty$, or may have a finite horizon, $T < \infty$. A discrete Markov decision model may also be stochastic or deterministic. A Markov decision model is deterministic if next period's state is known with certainty once the current period's state and action are known. In this case, it is sometimes beneficial to dispense with the probability transition law as a description of how the state evolves over time and use instead a deterministic state transition function g:

$$s_{t+1} = g(s_t, x_t).$$

Discrete Markov decision models may be analyzed using the dynamic programming principle developed by Richard Bellman (1956). Dynamic programming is a strategy for analyzing dynamic optimization models that is applicable not only to discrete Markov decision problems, but also to Markov decision models in continuous time and space. Dynamic programming effectively decomposes a dynamic optimization problem into a sequence of two period optimization problems. Dynamic programming is based on the Principle of Optimality, which was first articulated by Bellman as follows:

> "An optimal policy has the property that, whatever the initial state and decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."

The Principle of Optimality can be formally expressed in terms of the value functions V_t. For each period t and state s, $V_t(s)$ specifies the maximum attainable sum of current and expected future rewards, given that the process is in state s in period t. The Principle of Optimality implies that the value functions must satisfy Bellman's recursion equation

$$V_t(s) = \max_{x \in X(s)} \{f(s,x) + \delta \sum_{s' \in S} P(s'|s,x) V_{t+1}(s')\} \qquad s \in S.$$

Bellman's equation captures the essential problem faced by a dynamic, future-regarding optimizing agent: the need to balance the immediate reward $f(s_t, x_t)$ against expected future rewards $\delta E_t V_{t+1}(s_{t+1})$. Given the value function, the optimal policy $x_t^*(s)$ is simply the optimal solution to the optimization problems embedded in Bellman's equation.

For the discrete Markov decision model, the value function is simply a real vector and Bellman's equation is a nonlinear equation of real vectors. Since the number of states n is finite, the number of values is also finite. More specifically, if the n states are identified with the first n positive integers, then $V_t \in \Re^n$ in a natural way. Bellman's equation then asserts that the n-vector V_{t+1} is a function of the n-vector V_t. The function does not have a conventional analytic form, but is a function just the same. For each possible n-vector V_t, there is one and only one possible n-vector V_{t+1}.

For the finite horizon discrete Markov decision model to be well posed, a post-terminal value function V_{T+1} must be specified by the analyst. The post-terminal value function is fixed by some economically relevant terminal condition. In many applications, V_{T+1} is identically zero, indicating that no rewards are earned by the agent beyond the terminal decision period. In other applications, V_{T+1} may specify a salvage value earned by the agent after making his final decision. By convention, the agent makes no decision after the terminal period T, but may earn a final reward in the post-terminal period $T+1$.

Given the post-terminal value function, the finite horizon decision model in principle may be solved recursively by repeatedly solving Bellman's equation: Having V_{T+1}, solve for $V_T(s)$ for all states s; having V_T, solve for $V_{T-1}(s)$ for all states s; having V_{T-1}, solve for $V_{T-2}(s)$ for all states s; and so on. The process continues until $V_1(s)$ is derived for all states s. Because only finitely many actions are possible in a discrete Markov decision problem, the optimization problem embedded in Bellman's equation can always be solved by performing finitely many arithmetic operations. Thus, the value function of a finite horizon discrete Markov decision model is always well-defined, although in some cases more than one policy of state-contingent actions may yield the maximum expected stream of rewards.

If the discrete Markov decision model is infinite horizon then the value functions are the same for every period and thus may be commonly denoted simply by V. The infinite horizon value function V satisfies the n-vector fixed-point equation

$$V(s) = \max_{x \in X(s)} \{ f(s,x) + \delta \sum_{s' \in S} P(s'|s,x) V(s') \} \qquad s \in S.$$

If the discount factor δ is less than one, the mapping underlying Bellman's equation is a strong contraction. The Contraction Mapping Theorem thus guarantees the existence and uniqueness of the infinite horizon value function.

7.2 Economic Examples

7.2.1 Mine Management

A mine operator must determine how to optimally extract ore for a mine that will be shut down and abandoned at the end of year T. The market price of one ton of ore is p and the total cost of extracting x tons of ore in any year is $c = x^2/(1+s)$ where s is the stock of ore available at the beginning of the year in tons. The mine currently contains $\bar{s}$ tons of ore. Assuming the tons of ore extracted in any year must be an integer, what extraction schedule maximizes profits?

This is a finite horizon, deterministic model with time $t = \{1, 2, \ldots, T\}$ measured in years. The state is

$$\begin{aligned} s_t &= \text{tons of ore in mine at beginning of year } t \\ s_t &\in S = \{0, 1, 2, \ldots, \bar{s}\} \end{aligned}$$

and the action is

$$\begin{aligned} x_t &= \text{tons of ore extracted in year } t \\ x_t &\in X(s_t) = \{0, 1, 2, \ldots, s_t\}. \end{aligned}$$

The deterministic state transition function is

$$s_{t+1} = g(s_t, x_t) = s_t - x_t$$

and the reward function is

$$f(s, x) = px - x^2/(1+s).$$

The value function

$$V_t(s) = \text{value of mine with } s \text{ tons of ore at } t$$

satisfies Bellman's equation

$$V_t(s) = \max_{x=0,1,2,\ldots,s} \{px - x^2/(1+s) + \delta V_{t+1}(s-x)\}, \qquad s \in S$$

subject to the post-terminal condition

$$V_{T+1}(s) = 0, \qquad s \in S.$$

7.2.2 Deterministic Asset Replacement

At the end of each lactation cycle a dairy producer must decide whether to lactate a cow again or replace it with a new one. A cow yields y_i tons of milk over lactation cycle i, up to ten lactations. Upon completion of the 10^{th} lactation, a cow becomes unproductive and must be replaced. The net cost of replacing a cow is c dollars and the profit contribution of milk is p dollars per ton. What replacement policy maximizes profits?

This is an infinite horizon, deterministic model with time t measured in lactation cycles. The state is

$$\begin{aligned} i_t &= \text{lactation number of cow in cycle } t \\ i_t &\in S = \{1, 2, \dots, 10\} \end{aligned}$$

and the action is

$$\begin{aligned} k_t &= \text{replacement decision in cycle } t \\ k_t &\in X(i_t) = \begin{cases} \{1 = \text{lactate}, 2 = \text{replace}\} & i_t < 10 \\ \{2 = \text{replace}\} & i_t = 10. \end{cases} \end{aligned}$$

The deterministic state transition function is

$$i_{t+1} = g(i_t, k_t) = \begin{cases} i_t + 1 & k_t = 1 \quad \text{(lactate)} \\ 1 & k_t = 2 \quad \text{(replace)} \end{cases}$$

and the reward function is

$$f(i, k) = \begin{cases} py_i & k = 1 \quad \text{(lactate)} \\ py_i - c & k = 2 \quad \text{(replace)}. \end{cases}$$

The value function

$$V(i) = \text{value of cow entering lactation } i,$$

must satisfy Bellman's equation

$$V(i) = \begin{cases} \max\{py_i + \delta V(i+1), py_i - c + \delta V(1)\}, & i < 10, \\ py_i - c + \delta V(1), & i = 10. \end{cases}$$

Bellman's equation asserts that if we decide to keep the cow at the end of lactation i, we receive net earnings py_i and begin the subsequent cycle with a cow worth $V(i+1)$; if we decide to replace the cow at the end of lactation i, we receive net earnings of $py_i - c$ and begin the subsequent cycle with a cow worth $V(1)$.

7.2.3 Stochastic Asset Replacement

Suppose now that dairy cows vary in productivity. Each cow belongs to one of m productivity classes. A cow in productivity class j yields $q_j y_i$ tons of milk over lactation cycle i, where q_j is a quality multiplier and y_i is an industry baseline yield. When replacing a dairy cow, the farmer will not know how productive the new cow will be until the end of its first lactation. Cows of quality class j are obtained from the replacement pool with probability w_j. What is the optimal lactation-replacement policy?

This is an infinite horizon, stochastic model with time t measured in lactation cycles. The two-dimensional state is

$$\begin{aligned} i_t &= \text{lactation number of cow} \\ i_t &\in I = \{1, 2, \ldots, 10\} \end{aligned}$$

and

$$\begin{aligned} j_t &= \text{quality class of cow} \\ j_t &\in J = \{1, 2, \ldots, m\} \end{aligned}$$

and the action is

$$\begin{aligned} k_t &= \text{replacement decision in cycle } t \\ k_t &\in X(i_t) = \begin{cases} \{1 = \text{lactate}, 2 = \text{replace}\} & i_t < 10 \\ \{2 = \text{replace}\} & i_t = 10. \end{cases} \end{aligned}$$

The state transition probability rule is

$$P(i', j' | i, j, k) = \begin{cases} 1 & i' = i + 1, j' = j, k = 1 \\ w_{j'} & i' = 1, k = 2 \\ 0 & \text{otherwise} \end{cases}$$

and the reward function is

$$f(i, j, k) = \begin{cases} p q_j y_i & k = 1 \quad \text{(lactate)} \\ p q_j y_i - c & k = 2 \quad \text{(replace).} \end{cases}$$

The value function

$$V(i, j) = \text{value of cow of quality } j \text{ entering lactation } i$$

must satisfy Bellman's equation

$$V(i,j) = \max\{pq_j y_i + \delta V(i+1,j), pq_j y_i - c + \delta \sum_{j' \in J} w_{j'} V(1,j')\},$$

for $i < 10$ and

$$V(i,j) = pq_j y_i - c + \delta \sum_{j' \in J} w_{j'} V(1,j')$$

for $i = 10$.

7.2.4 Bioeconomic Model

In order to survive, an animal must forage for food in one of m distinct areas. In area k, the animal survives predation with probability p_k, finds food with probability q_k, and, if it finds food, gains e_k energy units. The animal expends one energy unit every period and has a maximum energy carrying capacity $\bar{s}$. If the animal's energy stock drops to zero, it dies. What foraging pattern maximizes the animal's probability of surviving to reproduce after period T?

This is a finite horizon, stochastic model with time $t = \{1, 2, \ldots, T\}$ measured in foraging periods. The state is

$$\begin{aligned} s_t &= \text{stock of energy at beginning of period } t \\ s_t &\in S = \{0, 1, 2, \ldots, \bar{s}\}; \end{aligned}$$

the action is

$$\begin{aligned} x_t &= \text{foraging area in } t \\ x_t &\in X = \{1, 2, \ldots, m\}. \end{aligned}$$

The state transition probability rule is

$$P(s'|s,x) = \begin{cases} p_x q_x & s > 0, s' = s - 1 + e_x & \text{(survives, finds food)} \\ p_x(1-q_x) & s > 0, s' = s - 1 & \text{(survives, finds no food)} \\ (1-p_x) & s > 0, s' = 0 & \text{(victim of predator)} \\ 1 & s = 0, s' = 0 & \text{(death is permanent)} \\ 0 & \text{otherwise;} \end{cases}$$

and the reward function is

$$f(s,x) = 0.$$

The value function

$$V_t(s) = \text{probability of procreating, given energy stocks } s \text{ in period } t,$$

must satisfy Bellman's equation

$$V_t(s) = \max_{x \in X} \{p_x q_x V_{t+1}(\min(\bar{s}, s - 1 + e)) + p_x(1 - q_x)V_{t+1}(s - 1)\},$$

subject to the post-terminal condition

$$V_{T+1}(s) = \begin{cases} 1 & s > 0 \\ 0 & s = 0 \end{cases}$$

Note that in this problem, the value function is just a conditional expectation of survival until the post-terminal period, and Bellman's equation is just an application of the Law of iterated expectations. The post-terminal value function simply asserts that in the post-terminal period, an animal with positive energy stocks will be alive with probability one and one with zero energy stocks will be dead with probability one.

7.2.5 Option Pricing

An American put option gives the holder the right, but not the obligation, to sell a specified quantity of a commodity at a specified strike price prior to or on a specified expiration date. In the Cox-Ross-Rubenstein binomial option pricing model, the price of the commodity is assumed to follow a two-state discrete jump process. Specifically, if the price of the commodity is p in period t, then its price in period $t + 1$ will be pu with probability q and p/u with probability $1 - q$ where:

$$\begin{aligned} u &= \exp(\sigma\tau^{0.5}) > 1 \\ q &= 0.5 + \tau^{0.5}(r - \sigma^2/2)/(2\sigma) \\ \delta &= \exp(-r\tau). \end{aligned}$$

Here, r is the annualized interest rate, continuously compounded, σ is the annualized volatility of the commodity price, and τ is the length of a period in years. Assuming the current price of the commodity is p_1, what is the value of an American put option if it has a strike price $\bar{p}$ and if it expires N periods from today?

This is a finite horizon, stochastic model where time $t \in \{1, 2, \ldots, N\}$ is measured in periods of length τ years each. The state is

$$\begin{aligned} p_t &= \text{commodity price in period } t \\ p_t &\in S = \{p_1 u^i | i = -N, \ldots, 0, \ldots, N\} \end{aligned}$$

and the action is

$$\begin{aligned} k_t &= \text{decision to keep or exercise at } t \\ k_t &\in X = \{1 = \text{keep}, 2 = \text{exercise}\}. \end{aligned}$$

The state transition probability rule is

$$P(p'|p,k) = \begin{cases} q & p' = pu \\ 1-q & p' = p/u \\ 0 & \text{otherwise} \end{cases}$$

and the reward function is

$$f(p,k) = \begin{cases} 0 & k = 1 \quad \text{(keep)} \\ \bar{p} - p & k = 2 \quad \text{(exercise)} \end{cases}$$

The value function

$$V_t(p) = \text{option value at } t, \text{ if commodity price is } p.$$

must satisfy Bellman's equation

$$V_t(p) = \max\{\ \bar{p} - p,\ q\delta V_{t+1}(pu) + (1-q)\delta V_{t+1}(p/u)\ \}$$

subject to the post-terminal condition

$$V_{N+1}(p) = \max\{\ \bar{p} - p,\ 0\ \}$$

Note that if the option is exercised in period t, then the owner receives $\bar{p} - p_t$. If he does not exercise the option, however, he earns no immediate reward but will have an option in hand the following period worth $V_{t+1}(p_t u)$ with probability q and $V_{t+1}(p_t/u)$ with probability $1-q$. In the post-terminal period, the option is exercised if the commodity price is below the strike price; otherwise the option is allowed to expire.

7.2.6 Job Search

At the beginning of each week, an infinitely-lived worker finds himself either employed or unemployed and must decide whether to be active in the labor market over the coming week by working, if he is employed, or by searching for a job, if he is unemployed. An active employed worker earns a wage w. An active unemployed worker earns an unemployment benefit u. An inactive worker earns a psychic benefit v from additional leisure, but no income. An unemployed worker that looks for a job will find one with probability p by the end of the week. An employed worker that remains at his job will be fired with probability q at the end of the week. What is the worker's optimal labor policy?

This is a infinite horizon, stochastic model with time $t = \{1, 2, \ldots, \infty\}$ measured in weeks. The state is

$$\begin{aligned} s_t &= \text{employment state at beginning of } t \\ s_t &\in S = \{1 = \text{unemployed}, 2 = \text{employed}\} \end{aligned}$$

and the action is

$$\begin{aligned} x_t &= \text{labor force participation decision } t \\ x_t &\in X = \{1 = \text{inactive}, 2 = \text{active}\}. \end{aligned}$$

The state transition probability rule is

$$P(s'|s,x) = \begin{cases} 1 & x = 1, s' = 1 & \text{(inactive worker)} \\ 1-p & x = 2, s = 1, s' = 1 & \text{(searches, finds no job)} \\ p & x = 2, s = 1, s' = 2 & \text{(searches, finds job)} \\ q & x = 2, s = 2, s' = 1 & \text{(works, loses job)} \\ 1-q & x = 2, s = 2, s' = 2 & \text{(works, keeps job)} \\ 0 & \text{otherwise;} \end{cases}$$

and the reward function is

$$f(s,x) = \begin{cases} v & x = 1 & \text{(inactive, receives leisure)} \\ u & x = 2, s = 1 & \text{(searching, receives benefit)} \\ w & x = 2, s = 2 & \text{(working, receives wage)} \end{cases}$$

The value function

$$V(s) = \text{Value of being in employment state } s \text{ at beginning of week,}$$

must satisfy Bellman's equation

$$V(s) = \begin{cases} \max\{v + \delta V(1), u + \delta p V(2) + \delta(1-p)V(1)\}, & s = 1 \\ \max\{v + \delta V(1), w + \delta q V(1) + \delta(1-q)V(2)\}, & s = 2 \end{cases}$$

7.2.7 Optimal Irrigation

Water from a dam can be used for either irrigation or recreation. Irrigation during the spring benefits farmers, but reduces the dam's water level during the summer, damaging recreational users. Specifically, farmer and recreational user benefits in year t are, respectively, $F(x_t)$ and $G(y_t)$, where x_t are the units of water used for irrigation and y_t are the units of water remaining for recreation. Water levels are replenished by random rainfall during the winter. With probability p, it rains one unit; with probability $1-p$ is does not rain at all. The dam has a capacity of M units of water and excess rainfall flows out of the dam without benefit to either farmer or recreational user. Derive the irrigation flow policy that maximizes the sum of farmer and recreational user benefits over an infinite time horizon.

This is a infinite horizon, stochastic model with time $t = \{1, 2, \ldots, \infty\}$ measured in years. The state is

$$\begin{aligned} s_t &= \text{units of water in dam at beginning of year } t \\ s_t &\in S = \{0, 1, 2, \ldots, M\} \end{aligned}$$

and

$$\begin{aligned} x_t &= \text{units of water released for irrigation in year } t \\ x_t &\in X(s_t) = \{0, 1, 2, \ldots, s_t\}. \end{aligned}$$

The state transition probability rule is

$$P(s'|s,x) = \begin{cases} p & s' = \min(s - x + 1, M) \quad \text{(rain)} \\ 1-p & s' = s - x, \quad\quad\quad\quad\quad\; \text{(no rain)} \\ 0 & \text{otherwise} \end{cases}$$

and the reward function is

$$f(s,x) = F(x) + G(s-x).$$

The value function

$$V(s) = \text{Value of } s \text{ units of water in dam at beginning of year } t.$$

must satisfy Bellman's equation:

$$V(s) = \max_{x=0,1,\ldots,s} \{f(s,x) + \delta p V(\min(s-x+1, M)) + \delta(1-p)V(s-x)\}.$$

7.3 Solution Algorithms

Below, we develop numerical solution algorithms for stochastic discrete-time, discrete-space Markov decision models. The algorithms apply to deterministic models as well, provided one views a deterministic model as a degenerate special case of the stochastic model for which the transition probabilities are all zeros or ones.

To develop solution algorithms, we must introduce some vector notation and operations. Assume that the states $S = \{1, 2, \ldots, n\}$ and actions $X = \{1, 2, \ldots, m\}$ are indexed by the first n and m integers, respectively. Also, let $v \in \Re^n$ denote an arbitrary value vector:

$$v_i \in \Re = \text{value in state } i;$$

and let $x \in X^n$ denote an arbitrary policy vector:

$$x_i \in X = \text{action in state } i.$$

For each action $k \in X$, let $f(k) \in \Re^n$ denote the vector of rewards at different states, given the action:

$$f_i(k) = \text{reward in state } i, \text{ given action } k \text{ taken}$$

and let $P(k) \in \Re^{n \times n}$ denote the transition probability matrix, given the action:

$$P_{ij}(k) = \text{probability of jump from state } i \text{ to } j, \text{ given action } k \text{ taken.}$$

For each policy $x \in X^n$, let $f(x) \in \Re^n$ denote the vector of rewards under the prescribed policy:

$$f_i(x) = \text{reward in state } i, \text{ given action } x_i \text{ taken;}$$

and let $P(x) \in \Re^{n \times n}$ denote the transition probability matrix under the prescribed policy:

$$P_{ij}(x) = \text{probability of jump from state } i \text{ to } j, \text{ given action } x_i \text{ is taken.}$$

Finally, given m vectors $v_1, v_2, \ldots, v_m$ in $\Re^n$, let

$$v = \max\{v_1, v_2, \ldots, v_m\} \in \Re^n$$

be the vector that contains the row-wise maxima of the v_k vectors; that is, $v_i = \max_k(v_{ki})$ for $i = 1, 2, \ldots, n$. Also, let

$$x = \text{argmax}\{v_1, v_2, \ldots, v_m\} \in X^n$$

be the vector that contains the indices of the row-wise maximal elements; that is, $x_i = \min\{k \mid v_i = v_{ki}\}$ for $i = 1, 2, \ldots, n$. For example,

$$\begin{pmatrix} 7 \\ 9 \end{pmatrix} = \max\left\{\begin{pmatrix} 1 \\ 9 \end{pmatrix}, \begin{pmatrix} 7 \\ 2 \end{pmatrix}, \begin{pmatrix} 7 \\ 8 \end{pmatrix}\right\}$$

and

$$\begin{pmatrix} 2 \\ 3 \end{pmatrix} = \text{argmax}\left\{\begin{pmatrix} 1 \\ 9 \end{pmatrix}, \begin{pmatrix} 7 \\ 2 \end{pmatrix}, \begin{pmatrix} 7 \\ 8 \end{pmatrix}\right\}.$$

Given this notation, it is possible to express Bellman's equation precisely as a vector equation suitable for numerical analysis using a vector processing language. Specifically, if $v_t \in \Re^n$ denotes the value function in period t, then

$$v_t = \max_k\{f(k) + \delta P(k) v_{t+1}\}.$$

All solution algorithms for discrete Markov decision model are based on Bellman's equation. The specific details vary according to whether the model is finite or infinite horizon.

For an infinite horizon model, Bellman's equation reduces to finding a fixed-point of a function that maps $\Re^n$ into itself. Specifically, let $F : \Re^n \mapsto \Re^n$ be defined as follows:

$$F(v) = \max_k\{f(k) + \delta P(k) v\}.$$

Then Bellman's equation simply asserts that the infinite horizon value function v is a fixed-point of F:

$$F(v) = v.$$

If the discount factor δ is less than one, the nonlinear map F can be shown to be a contraction map. Thus, the infinite-horizon value function exists and is unique, and may be computed to an arbitrary accuracy using either function iteration or Newton's method.

7.3.1 Backward Recursion

The finite horizon discrete Markov decision model is solved by deriving the value functions in reverse order, beginning with the terminal decision period T value function. To execute the backward recursion algorithm, one must specify the rewards f, the transition probabilities P, the discount factor δ, the decision horizon T, and the post-terminal value function v_{T+1}. The backward recursion algorithm computes the value functions v_t and optimal policies x_t as follows:

Algorithm: Backward Recursion.

0. Initialization: Enter f, P, δ, T and v_{T+1}, and set $t \leftarrow T$.

1. Recursion Step: Given v_{t+1}, compute v_t and x_t:

$$\begin{aligned} v_t &\leftarrow \max_k \{f(k) + \delta P(k) v_{t+1}\} \\ x_t &\leftarrow \operatorname*{argmax}_k \{f(k) + \delta P(k) v_{t+1}\}. \end{aligned}$$

2. Termination Check: If $t = 1$, stop; otherwise set $t \leftarrow t - 1$ and return to step 1.

Each iteration of the backward recursion algorithm involves a finite number of well-defined matrix-vector operations, implying that the value functions are also well-defined for every period. Note however, that it may be possible to have more than one sequence of optimal policies if ties occur in Bellman's equation. Since the algorithm requires exactly T iterations, it terminates in finite time with the value function precisely computed and at least one optimal policy obtained.

7.3.2 Function Iteration

To execute the function iteration algorithm, one must specify the rewards f, the transition probabilities P, the discount factor δ, and an initial guess for the value function v. Since function iteration does not guarantee an exact answer in finitely many iterations, a convergence tolerance τ must also be specified. Given the data for the problem, the function iteration algorithm computes successive approximations to the value function v and optimal policy x as follows:

Algorithm: Function Iteration.

0. Initialization: Enter f, P, δ, τ and v.

1. Function Iteration: Update the value function v:

$$v \leftarrow \max_k \{f(k) + \delta P(k) v\}.$$

2. Termination Check: If $||\Delta v|| < \tau$, set

$$x \leftarrow \operatorname*{argmax}_k \{f(k) + \delta P(k) v\}$$

and stop; otherwise return to step 1.

Because function iteration does not generally compute the value function exactly, it is reassuring that one can compute an upper bound on the error associated with the final value function iterate. Specifically, if the algorithm terminates at iteration n, then

$$||v_n - v^*|| \leq \frac{\delta}{1-\delta} ||v_n - v_{n-1}||_\infty$$

where v^* is the true value function.

7.3.3 Policy Iteration

The function iteration algorithm is relatively easy to implement, but may converge slowly in some applications. Policy iteration offers an alternative to function iteration for solving the infinite horizon Markov decision model. The policy iteration algorithm designed to yield a sequence of successively superior policies, and in many cases will converge quicker than function iteration. Policy iteration is based on the observation that if a (possibly suboptimal) policy x is followed indefinitely, then the associated value v of doing so must satisfy

$$v = f(x) + \delta P(x) v,$$

or, equivalently,

$$v = (I - \delta P(x))^{-1} f(x).$$

Note that because $P(x)$ is a probability matrix, the matrix $I - \delta P(x)$ is always invertible, implying that the value associated with any policy is well-defined.

To execute the policy iteration algorithm, one must specify the rewards f, the transition probabilities P, the discount factor δ, and an initial guess for the policy function x. Given the data for the problem, the policy iteration computes the optimal value function v and policy x as follows:

Algorithm: Policy Iteration.

0. Initialization: Enter f, P, δ, τ, and an initial guess for x.

1. Value Computation: Given x, the current approximant to the policy function, compute the associated value function v:

$$v \leftarrow (I - \delta P(x))^{-1} f(x).$$

2. Policy Iteration: Given v, the current approximant to the value function, update the policy x:

$$x \leftarrow \underset{k}{\operatorname{argmax}} \{f(k) + \delta P(k) v\}.$$

3. Termination Check: If $\Delta x = 0$, stop; otherwise return to step 1.

At each iteration, policy iteration either finds the optimal policy or offers a strict improvement in the value function. Because the total number of states and actions is finite, the total number of admissible policies is also finite, guaranteeing that policy iteration will terminate after finitely many iterations with an exact optimal solution. Note however, that each policy iteration requires the solution of a linear equation system. If $P(x)$ is large and dense, the linear equation could be expensive to solve, making policy iteration slow and possibly impracticable.

7.4 Dynamic Analysis

The optimal value and policy functions provide some insight into the nature of the controlled dynamic economic process. The optimal value function describes the benefits of being in a given state and the optimal policy function prescribes the optimal action to be taken there. However, the optimal value and policy functions provide only a partial, essentially static, picture of the controlled dynamic process. Typically, one wishes to analyze the controlled process further to learn about its dynamic properties. Furthermore, one often wishes to know how the process is affected by changes in model parameters.

To analyze the dynamics of the controlled process, one will typically perform dynamic path analysis and steady-state analysis. Dynamic path analysis examines how the controlled dynamic process evolves over time starting from some initial state. Specifically, dynamic path analysis describes the path or expected path followed by the state or some other endogenous variable and how the path or expected path will vary with changes in model parameters.

Steady-state analysis examines the longrun tendencies of the controlled process over an infinite horizon, without regard to the path followed over time. Steady-state analysis of a deterministic model seeks to find the values to which the state or other endogenous variables will converge over time, and how the limiting values will vary with changes in the model parameters. Steady-state analysis of a stochastic model requires derivation of the steady-state distribution of the state or other endogenous variable. In many cases, one is satisfied to find the steady-state means and variances of these variables and their sensitivity to changes in exogenous model parameters.

The path followed by a controlled, finite horizon, deterministic, discrete, Markov decision process is easily computed. Given the state transition function g and the optimal policy functions x_t^*, the path taken by the state from an initial point s_1 can be computed as follows:

$$\begin{array}{lcl} s_2 & = & g(s_1, x_1^*(s_1)) \\ s_3 & = & g(s_2, x_2^*(s_2)) \\ s_4 & = & g(s_3, x_3^*(s_3)) \\ \vdots & & \\ s_{T+1} & = & g(s_T, x_T^*(s_T)). \end{array}$$

Given the path of the controlled state, it is straightforward to derive the path of actions through the relationship $x_t = x_t^*(s_t)$. Similarly, given the path taken by the controlled state and action allows one to derive the path taken by any function of the state and action.

A controlled, infinite horizon, deterministic, discrete Markov decision process can be analyzed similarly. Given the state transition function g and optimal policy function x^*, the path taken by the controlled state from an initial point s_1 can be computed from the iteration rule:

$$s_{t+1} = g(s_t, x^*(s_t)).$$

The steady-state of the controlled process can be computed by continuing to form iterates until they converge. The path and steady-state values of other endogenous variables, including the action variable, can then be computed from the path and steady-state of the controlled state.

Analysis of controlled, stochastic, discrete Markov decision processes is a bit more complicated because such processes follow a random, not deterministic, path. Consider a finite horizon process whose optimal policy x_t^* has been derived for each period t. Under the optimal policy, the controlled state will follow a finite horizon Markov chain with nonstationary transition probabilities:

$$\Pr(s_{t+1} = j | s_t = i) = P_{ij}(x_t^*).$$

Here, $P(x_t^*)$ is a probability matrix whose row i, column j element gives the probability of jumping from state i in period t to state j in period $t+1$, given that the optimal policy $x_t^*(i)$ is followed in period t.

The controlled state of an infinite horizon, stochastic, discrete Markov decision model with optimal policy x will follow a infinite horizon stationary Markov chain with transition probability matrix

$$\Pr(s_{t+1} = j | s_t = i) = P_{ij}(x^*),$$

which will be designated P^* for short. Given the transition probability matrix P^* for the controlled state it is possible to simulate a representative state path, or, for that matter, many representative state paths, by performing Monte Carlo simulation. To perform Monte Carlo simulation, one picks an initial state, say s_1. Having the simulated state $s_t = i$, one may simulate a jump to s_{t+1} by randomly picking a new state j with probability P_{ij}^*. Routines for simulating discrete random variables are generally available.

The path taken by the controlled state of an infinite horizon, stochastic, discrete Markov model may also be described probabilistically. To this end, let Q_t denote the matrix whose row i, column j entry gives the probability that the process will be in state j in period t, given that it is in state i in period 1. Then the t-period transition probability matrices Q_t are recursively defined by

$$Q_{t+1} = P^* Q_t$$

where $Q_1 = I$. Given the t-period transition probability matrices Q_t, one can fully describe, in a probabilistic sense, the path taken by the controlled process from any initial state $s_1 = i$ by looking at the i^{th} rows of the matrices Q_t.

In most economic applications, the multiperiod transition matrices Q_t will converge to a matrix Q. In such cases, each entry of Q will indicate the relative frequency with which the controlled decision process will visit a given state in the longrun, when starting from given initial state. In the event that all the columns of Q are identical and the longrun probability of visiting a given state is independent of initial state, then we say that the controlled state process possesses a steady-state distribution. The steady state distribution is given by the probability vector π that is the common row of the matrix Q. Given the steady-state distribution of the controlled state process, it becomes possible to compute summary measures about the longrun behavior of the controlled process, such as its longrun mean or variance. Also, it is possible to derive the longrun probability distribution of the optimal action variable or the longrun distribution of any other variables that are functions of the state and action.

7.5 Numerical Implementation

Discrete Markov decision models of all kinds may be routinely solved with two standard computer routines. One routine solves the optimization problem embedded in Bellman's equation. The second routine computes the state reward and transition probability matrix induced by any given policy.

In Matlab, it is simple to write a routine that solves the optimization problem embedded in Bellman's equation for any n-state, m-action discrete Markov decision model. Such a routine is `valmax`:

```
function [v,x] = valmax(v,f,P,delta)
[m,n]=size(f);
[v,x]=max(f+delta*reshape(P*v,m,n),[],2);
```

The routine requires as input:

`v`	An n by 1 vector containing next period's value function. Specifically, `v(i)` is the value of arriving in state `i` next period.
`f`	An n by m matrix of rewards whose rows represent states and columns represent actions.

`P`	An mn by n matrix of state transition probabilities whose rows represent this period's state and columns represent next period's state. The state transition probability matrices for the various actions are stacked vertically on top of each other, with the n by n transition probability matrix associated with action 1 at the top and the n by n transition probability matrix associated with action m at the bottom.
`delta`	The discount factor, a scalar.

The routine returns as output:

`v`	An n by 1 vector containing this period's value function. Specifically, `v(i)` is the value of being in state `i` this period.
`x`	An n by 1 vector containing this period's optimal policy. Specifically, `x(i)` is the action that maximizes the value of being in state `i` this period.

In Matlab it is also simple to write a routine that computes the provisional value function, state reward function, and transition probability matrix induced by a arbitrary prescribed policy. Such a routine is `valpol`:

```
function [v,pstar,fstar] = valpol(x,f,P,delta)
[n,m]=size(f); i=(1:n)';
pstar = P(n*(x(i)-1)+i,:);
fstar = f(n*(x(i)-1)+i);
v = (eye(n,n)-delta*pstar)\fstar;
```

The routine takes as input the reward matrix `f`, the state transition probability matrix `P`, and a policy vector `x`, in the format described above. The routine then returns as output:

`v`	An n by 1 vector specifying the value of different states under policy `x`. Specifically, `v(i)` is the value of being in state `i` given policy x is followed indefinitely.
`fstar`	An n by 1 vector specifying the rewards obtained under policy `x`. Specifically, `fstar(i)` is the reward obtained in state `i` if the process is in state `i` and action `x(i)` is taken.

`Pstar`	An n by n matrix specifying the probability of jumping from one state to another under policy `x`. Specifically, `Pstar(i,j)` is the probability of jumping from state `i` to state `j` if the process is in state `i` and action `x(i)` is taken.

Given the two subroutines, it is straightforward to implement the backward recursion, function iteration, and policy iteration algorithms used to solve discrete Markov decision models. As a practical matter, the greatest difficultly typically encountered when solving discrete Markov decision models using the two subroutines is correctly initializing the reward and state transition matrices. Consider the following examples:

7.5.1 Mine Management

Consider the mine management model with market price $p = 1$, initial stock of ore $\bar{s} = 100$, and annual discount factor $\delta = 0.95$.

The first step required to solve the model numerically is to enter the model parameters and construct the state and action spaces:

```
delta = 0.95;                                % discount factor
price = 1;                                   % price of ore
sbar = 100;                                  % initial ore stock
s = (0:sbar)';                               % vector of states
n = length(s);                               % number of states
x = (0:sbar)';                               % vector of actions
m = length(x);                               % number of actions
```

Next, one constructs the reward and transition probability matrices:

```
f = zeros(n,m); P = [];
for k=1:m
   f(:,k) = price*x(k)-(x(k)^2)./(1+s);
   i = find(x(k)>s); if ~isempty(i), f(i,k) = -inf; end;
   Pk = sparse(zeros(n,n));
   j = max(0,s-x(k)) + 1;
   for i=1:n
      Pk(i,j(i)) = 1;
   end
   P = [P;Pk];
end
```

Here, a reward matrix element is set to negative infinity if the extraction level exceeds the available stock. This guarantees that the value maximization algorithm will not chose an infeasible action. Also note that because the probability transition matrix contains mostly zeros, it is stored in sparse matrix format to speed up subsequent computations.

To solve the finite horizon model via backward recursion, one must set the terminal decision period and supply the post-terminal value function. Also, after each step one must store the optimal policy vector because it varies from period to period. The matrices `vout` and `xout` are created for this purpose.

```
T=11;                                    % terminal period
v = zeros(n,1);                          % post-terminal value
xout = [];                               % initial policy matrix
for t=T:-1:1                             % backward recursion
   [v,ix] = valmax(v,f,P,delta);         % Bellman equation
   xout = [x(ix) xout];                  % store optimal policy
end
```

Upon completion, `xout` is an `n` by `T` matrix containing the optimal ore extraction policy for all possible initial ore stock values and periods 1 to `T`. The columns of `xout` represent periods and its rows represent states.

To solve the infinite horizon model via function iteration, one must set a maximum number of iterations and a convergence tolerance, and must supply an initial guess for the infinite horizon value function. Because function iteration generates successive approximants to the infinite horizon value function, an approximant may be discarded once a new one is generated.

```
maxit = 300;                             % maximum iterations
tol = 1.e-8;                             % convergence tolerance
v = zeros(n,1);                          % initial value
for it=1:maxit                           % function iteration
   vold = v;                             % store old value
   [v,ix] = valmax(v,f,P,delta);         % Bellman equation
   change = norm(v-vold);                % change in policy
   fprintf ('\n%5i %10.1e',it,change)    % print progress
   if change<tol, break, end;            % convergence check
end
```

Upon convergence, `v` will be `n` by `1` matrix containing the value function and ix will be n by 1 matrix containing the indices of the optimal ore extractions. The optimal ore extractions may be recovered by typing x(ix).

To solve the infinite horizon model via policy iteration, one must set a maximum number of iterations and a convergence tolerance, and must supply an initial guess for the optimal extraction policy indices.

```
maxit = 300;                                % maximum iterations
tol = 1.e-8;                                % convergence tolerance
ix = ones(n,1);                             % initial policy indices
for it=1:maxit                              % policy iteration
   ixold = ix;                              % store old policy indices
   v = valpol(ix,f,P,delta);                % implied value
   [v,ix] = valmax(v,f,P,delta);            % update policy
   change = norm(ix-ixold);                 % change in policy indices
   fprintf ('\n%5i %10.1e',it,change)       % print progress
   if ix==ixold, break, end;                % convergence check
end
```

Upon convergence, v will be n by 1 matrix containing the value function and ix will be n by 1 matrix containing the indices of the optimal ore extractions. The optimal ore extractions may be recovered by typing x(ix). Note that the intermediate value functions v computed by valmax are not used by the algorithm.

Once the optimal solution has been computed, one may plot the optimal value and extraction policy functions:

```
figure(1); plot(s,x(ix)); xlabel('Stock'); ylabel('Optimal Extraction');
figure(2); plot(s,v);     xlabel('Stock'); ylabel('Optimal Value');
```

One may also plot the optimal path of the stock level over time, starting from the initial stock level:

```
st = sbar;
splot = [];
maxtime = 20;
for t=0:maxtime;
   [serr,is] = min(abs(s-st));              % Find stock index
   splot = [splot st];                      % Record stock
   st = st - x(ix(is));                     % Update stock
end
figure(3); plot(0:maxtime,splot); xlabel('Year'); ylabel('Stock');
```

As seen in figure *, optimal extraction plan thus calls for the mine's ore to be completely extracted by the end of year 20.

7.5.2 Deterministic Asset Replacement

Consider the deterministic cow replacement model with yield function $y_i = 8 + 2i - 0.25i^2$, replacement cost $c = 500$, milk price $p = 150$, and a per-cycle discount factor $\delta = 0.9$.

The first step required to solve the model numerically is to enter the model parameters and construct the state space:

```
delta = 0.9;                            % discount factor
cost  = 500;                            % replacement cost
price = 150;                            % milk price
s = (1:10)';                            % lactation states
```

There is no need to explicitly define an action space since actions are represented by integer indices.

Next, one constructs the reward and transition probability matrices: Here, action 1 is to keep the cow and action 2 is to replace the cow after the current lactation:

```
y = (-0.2*s.^2+2*s+8);                  % Yield per lactation
f = [price*y price*y-cost];             % Net revenue by action
i = find(s==10); f(i,1) = -999;         % Replace after lactation 10
P1 = zeros(10,10); P2 = zeros(10,10);
for i=1:10
   if i<10
      P1(i,i+1) = 1;                    % Up lactation number, if keep
   else
      P1(i,1) = 1;                      % Replace after lactation 10
   end
   P2(i,1) = 1;                         % Go to lactation 1, if replace
end
P = [P1;P2];
```

Here, a reward matrix element is set to negative infinity for a keep decision in the tenth and final lactation to ensure that the value maximization algorithm will elect replacement after the tenth lactation.

To solve the infinite horizon model via function iteration, one must set a maximum number of iterations and a convergence tolerance, and must supply an initial guess for the infinite horizon value function. Because function iteration generates successive approximants to the one infinite horizon value function, an approximant may be discarded once a new one is generated.

```
maxit = 300;                              % maximum iterations
tol = 1.e-8;                              % convergence tolerance
v = zeros(n,1);                           % initial value
for it=1:maxit                            % function iteration
   vold = v;                              % store old value
   [v,x] = valmax(v,f,P,delta);           % Bellman equation
   change = norm(v-vold);                 % change in policy
   fprintf ('\n%5i %10.1e',it,change)     % print progress
   if change<tol, break, end;             % convergence check
end
```

If convergence is reached, `v` will be `n` by `1` matrix containing the value function and `x` will be `n` by `1` matrix containing optimal replacement policy. Note how this code segment is identical to that used solve the infinite horizon mine management model via function iteration, except that the optimal policy and its index are one and the same.

To solve the infinite horizon model via policy iteration, one must set a maximum number of iterations and a convergence tolerance, and must supply an initial guess for the optimal extraction policy indices.

```
maxit = 300;                              % maximum iterations
tol = 1.e-8;                              % convergence tolerance
x = ones(n,1);                            % initial policy indices
for it=1:maxit                            % policy iteration
   xold = x;                              % store old policy indices
   v = valpol(x,f,P,delta);               % implied value
   [v,x] = valmax(v,f,P,delta);           % update policy
   change = norm(x-xold);                 % change in policy indices
   fprintf ('\n%5i %10.1e',it,change)     % print progress
   if x==xold, break, end;                % convergence check
end
```

If convergence is reached, `v` will be `n` by `1` matrix containing the value function and `x` will be `n` by `1` matrix containing optimal replacement policy. Note how this code segment is identical to that used solve the infinite horizon mine management model via function iteration, except that the optimal policy and its index are one and the same.

Once the optimal solution has been computed, one may plot the optimal value and extraction policy functions:

```
figure(1); plot(s,v); xlabel('Age'); ylabel('Optimal Value');
figure(2); bar(s,x);  xlabel('Age'); ylabel('Optimal Decision');
```

As seen in figure *, the optimal policy is to replace a cow after its fifth lactation.

7.5.3 Stochastic Asset Replacement

Suppose now that dairy cows vary in productivity. Each cow belongs to one of 3 productivity classes, yielding 0.8, 1.0, and 1.2 times the industry baseline, respectively. Also suppose that cows from these three classes are obtained from the replacement pool with probabilities 0.2, 0.6, and 0.2, respectively.

The first step required to solve the model numerically is to enter the model parameters: parameters:

```
delta = 0.9;                                % discount factor
cost  = 500;                                % replacement cost
price = 150;                                % milk price
```

Next, one constructs the state space grid by computing the vectors of values attainable by each of the two state variables and forming their Cartesian product:

```
scoord{1} = (1:10)';                        % lactation states
scoord{2} = [0.8;1.0;1.2];                  % productivity states
s = cartgrid(scoord);                       % combined state grid
n = length(s)                               % number of states
```

There is no need to explicitly define an action space since actions are represented by integer indices.

One then constructs the reward matrix, identifying action 1 with 'keep' and action 2 with 'replace':

```
y = (-0.2*s(:,1).^2+2*s(:,1)+8).*s(:,2); % yield per lactation
f = [price*y price*y-cost];              % net revenue by action
i = find(s(:,1)==10); f(i,1) = -999;     % force replace at lactation 10
```

Constructing the state transition probability matrix is a bit involved due to the multidimensional state space. Here, we set up, for each action, a four dimensional transition probability array - two dimensions for the current values of the two state variables and two dimensions for the future values of the two state variables. The four dimensional arrays are then reshaped into two-dimensional probability transition matrices and stacked for subsequent computation.

```
P1 = zeros(3,10,3,10); P2 = zeros(3,10,3,10);
for i=1:3
for j=1:10
   if j<10
      P1(i,j,i,j+1) = 1;                 % Increment lactation, if keep
   else
      P1(i,j,1,1) = 0.2;                 % Replace after lactation 10
      P1(i,j,2,1) = 0.6;
      P1(i,j,3,1) = 0.2;
   end
   P2(i,j,1,1) = 0.2;                    % Elected replacement
   P2(i,j,2,1) = 0.6;
   P2(i,j,3,1) = 0.2;
end
end
P1 = reshape(P1,30,30);
P2 = reshape(P2,30,30);
P = [P1;P2];
```

To solve the infinite horizon model via function iteration, one must set a maximum number of iterations and a convergence tolerance, and must supply an initial guess for the infinite horizon value function. Because function iteration generates successive approximants to the one infinite horizon value function, an approximant may be discarded once a new one is generated.

```
maxit = 300;                             % maximum iterations
tol = 1.e-8;                             % convergence tolerance
v = zeros(n,1);                          % initial value
for it=1:maxit                           % function iteration
   vold = v;                             % store old value
   [v,x] = valmax(v,f,P,delta);          % Bellman equation
   change = norm(v-vold);                % change in policy
   fprintf ('\n%5i %10.1e',it,change)    % print progress
   if change<tol, break, end;           % convergence check
end
```

Upon convergence, v will be n by 1 matrix containing the value function and x will be n by 1 matrix containing optimal replacement policy. Note how this code segment is identical to that used solve the deterministic cow replacement problem. The only significant difference between solving up the deterministic and stochastic problems comes in the construction of the reward and transition probability matrices.

To solve the infinite horizon model via policy iteration, one must set a maximum number of iterations and a convergence tolerance, and must supply an initial guess for the optimal replacement policy.

```
maxit = 300;                            % maximum iterations
tol = 1.e-8;                            % convergence tolerance
x = ones(n,1);                          % initial policy
for it=1:maxit                          % policy iteration
   xold = x;                            % store old policy
   v = valpol(x,f,P,delta);             % implied value
   [v,x] = valmax(v,f,P,delta);         % update policy
   change = norm(x-xold);               % change in policy
   fprintf ('\n%5i %10.1e',it,change)   % print progress
   if x==xold, break, end;              % convergence check
end
```

If convergence is reached, v will be n by 1 matrix containing the value function and x will be n by 1 matrix containing optimal replacement policy.

Once the optimal solution has been computed, one may plot the optimal value and replacement policy functions:

```
figure(1); bar(scoord{1},reshape(x,3,10)')
xlabel('Age'); ylabel('Optimal Decision');
legend('Low','Med','Hi')
set(gca,'YTick',[0 1 2]);
figure(2); plot(scoord{1},reshape(v,3,10)')
xlabel('Age'); ylabel('Optimal Value');
legend('Low','Med','Hi')
```

To perform dynamic analysis, one first computes the probability transition and payoff matrices for the optimally controlled state process:

```
[vstar,Pstar,fstar] = valpol(x,f,P,delta);
```

Given Pstar and fstar, it is straight forward to plot the expected net revenue over time given that cow is in lactation 1 and has low productivity at present

```
figure(3)
pi = zeros(size(s(:,1))); pi(1)=1;       % Current state with certainty
maxtime = 20;
for t=0:maxtime;
   exprevenue(t+1) = pi'*fstar;          % Compute expected revenue
   pi = Pstar'*pi;                       % Update conditional distribution
end
plot(0:maxtime,exprevenue); xlabel('Year'); ylabel('Expected Revenue');
```

If the probability vector pi has converged to its steady-state, then one can further compute the average age and productivity of cows in the longrun:

```
avgage = pi'*s(:,1); fprintf('\nSteady-state Age          %8.2f\n',avgage)
avgpri = pi'*s(:,2); fprintf('\nSteady-state Productivity %8.2f\n',avgpri)
```

7.5.4 Bioeconomic Model

Consider the bioeconomic model with three foraging areas, predation survival probabilities $p_1 = 1$, $p_2 = 0.98$, and $p_3 = 0.90$, and foraging success probabilities $q_1 = 0$, $q_2 = 0.3$, and $q_3 = 0.8$. Also assume that successful foraging delivers $e = 4$ units of energy in all areas and that the procreation horizon is 10 periods.

The first step required to solve the model numerically is to enter the model parameters and construct the state and action spaces:

```
T = 10;                                 % foraging periods
eadd =  4;                              % energy from foraging
emax = 10;                              % energy capacity
s = 0:emax;                             % energy levels
n = length(s);                          % number of states
m = 3;                                  % number of forage areas
```

There is no need to explicitly define an action space since actions are represented by integer indices.

Next, one constructs the reward and transition probability matrices:

```
f = zeros(n,m);
p = [1 .98 .9];                         % predation survival prob.
```

```
q = [0 .30 .8];                           % foraging success prob.
P = [];
for k=1:m
  Pk = zeros(n,n);
  Pk(1,1) = 1;
  for i=2:n;
     Pk(i,min(n,i-1+eadd)) = p(k)*q(k);
     Pk(i,i-1)             = p(k)*(1-q(k));
     Pk(i,1)               = Pk(i,1) + (1-p(k));
  end
  P = [ P ; Pk ];
end
```

The reward matrix is zero because the reward is not earned until the post-terminal period. Upon the reaching the post-terminal period, either the animal is alive, earning reward of 1, or is dead, earning a reward of 0.

To solve the finite horizon model via backward recursion, one must set the terminal decision period and supply the post-terminal value function. Also, after each step one must store the optimal value and policy vectors because they vary from period to period. Cell arrays v and x are created for this purpose:

```
x = cell(T,1); v = cell(T+1,1);           % declare cell arrays
v{T+1} = zeros(n,1);                      % preallocate post-terminal value
v{T+1}(2:n) = 1;                          % post-terminal value: survive
for t=T:-1:1                              % backward recursion
   [v{t},x{t}] = valmax(v{t+1},f,P,1);    % Solve Bellman's equation
end
```

Upon completion, x is an T by 1 cell array, one cell for each foraging period, with each cell an n by 1 matrix containing the optimal foraging policy for all possible initial energy stock levels.

Once the optimal solution has been computed, one may print out the survival probabilities:

```
fprintf('\nProbability of Survival\n')
fprintf('\n                                 Stocks\n')
fprintf('Period ');fprintf('%5i ',s);fprintf('\n');
for t=1:T
   fprintf('%5i ',t);fprintf('%6.2f',v{t}');fprintf('\n')
end
```

A similar script can be executed to print out the optimal foraging strategy.

7.5.5 Option Pricing

Consider the binomial option pricing model with current asset price $p_1 = 2.00$, strike price $\bar{p} = 2.10$, annual interest rate $r = 0.05$, annual volatility $\sigma = 0.2$, and time to expiration $T = 0.5$ years that is to be divided into $N = 50$ intervals.

The first step required to solve the model numerically is to enter the model parameters:

```
T = 0.5;                                 % years to expiration
sigma = 0.2;                             % annual volatility
r = 0.05;                                % annual interest rate
strike = 2.1;                            % option strike price
p1 = 2;                                  % current asset price
N = 200;                                 % number of time intervals
tau = T/N;                               % length of time intervals
delta = exp(-r*tau);                     % discount factor
u = exp( sigma*sqrt(tau));               % up jump factor
q = 0.5+tau^2*(r-(sigma^2)/2)/(2*sigma); % up jump probability
```

The next step is to construct the state space:

```
price = p1*(u.^(-N:N))';                 % asset prices
n = length(price);                       % number of states
```

There is no need to explicitly define an action space since actions are represented by integer indices.

Next, one constructs the reward and transition probability matrices:

```
% Construct reward function (actions exercise=1, hold=2)
f = [ strike-price zeros(n,1) ];
P = zeros(n,n);
for i=1:n
  P(i,min(i+1,n)) = q;
  P(i,max(i-1,1)) = 1-q;
end
P = [ zeros(n,n); P ];
P = sparse(P);
```

Here, action 1 is identified with the exercise decision and action 2 is identified with the hold decision. Note how the transition probability matrix associated with the decision to exercise the option is identically the zero matrix.

This is done to ensure that the expected future value of an exercised option always computes to zero. Also note that because the probability transition matrix contains mostly zeros, it is stored in sparse matrix format to speed up subsequent computations.

To solve the finite horizon model via backward recursion, one must set the terminal decision period and supply the post-terminal value function. Since one is interested only in pricing the option in the current period, period 1, the last value function produced by the backward recursion algorithm contains all necessary information, and other value and optimal policy functions may be discarded once new ones are generated. Thus one only needs single arrays `v` and `x` to keep track of the optimal value and policy functions, respectively.

```
v = max(0,strike-price);                         % post-terminal value
for t=N:-1:1                                     % backward recursion
   [v,x] = valmax(v,f,P,delta);                  % Bellman equation
end
```

Upon completion, `v` is an `n` by 1 array that contains the value of the American option in period 1 for different asset prices.

Once the optimal solution has been computed, one may plot the optimal value function.

```
plot(price,v); axis([0 strike*2 -inf inf]);
xlabel('Asset Price'); ylabel('Put Option Premium');
```

7.5.6 Job Search

Consider the job search model with weekly wage $w = 100$, unemployment benefit $u = 55$, and psychic benefit from leisure $v = 60$. Also assume the probability of finding a job is $p = 0.90$ and the probability of being fired is $q = 0.05$. Further assume a weekly discount rate of $\delta = 0.99$.

The first step required to solve the model numerically is to enter the model parameters:

```
w = 100;                                  % weekly wage
u =  55;                                  % weekly unemp. benefit
v =  60;                                  % weekly value of leisure
pfind = 0.90;                             % prob. of finding job
pfire = 0.05;                             % prob. of being fired
delta = 0.99;                             % discount factor
```

Note that by identifying both states and actions with their integer indices, one does not need to explicitly generate the state and action space.

Next, one constructs the reward and transition probability matrices. Here, we identify state 1 with joblessness and state 2 with employment, and identify action 1 with inactivity and action 2 with activity:

```
% Construct reward function
f(:,1) = v;                                 % gets leisure
f(1,2) = u;                                 % gets benefit
f(2,2) = w;                                 % gets wage
P1 = zeros(2,2); P2 = zeros(2,2);
P1(:,1) = 1;                                % remains unemployed
P2(1,1) = 1-pfind;                          % finds no job
P2(1,2) = pfind;                            % finds job
P2(2,1) = pfire;                            % gets fired
P2(2,2) = 1-pfire;                          % keeps job
P = [P1;P2];
```

To solve the infinite horizon model via policy iteration, one must set a maximum number of iterations and a convergence tolerance, and must supply an initial guess for the labor policy indices.

```
maxit = 1000;                               % maximum iterations
tol = 1.e-8;                                % convergence tolerance
x = ones(2,1);                              % initial policy
for it=1:maxit                              % policy iteration
   xold = x;                                % store old policy
   v = valpol(x,f,P,delta);                 % implied value
   [v,x] = valmax(v,f,P,delta);             % update policy
   change = norm(x-xold);                   % change in policy
   if x==xold, break, end;                  % convergence check
end
```

If convergence is reached, `v` will be `n` by `1` matrix containing the value function and `x` will be `n` by `1` matrix containing optimal labor policy.

7.5.7 Optimal Irrigation

The first step required to solve the model numerically is to enter the model parameters and construct the state and action spaces:

```
delta =  0.9;                              % discount factor
irrben = [-3;5;9;11];                      % Irrigation Benefits to Farmers
recben = [-3;3;5;7];                       % Recreational Benefits to Users
maxcap = 3;                                % maximum dam capacity
n = maxcap+1;                              % number of states
s = (0:1:maxcap)';                         % vector of states
m = maxcap+1;                              % number of actions
x = (0:1:maxcap)';                         % vector of actions
```

Next, one constructs the reward matrix:

```
f = zeros(n,m);
for i=1:n;
for k=1:m;
   if k>i
      f(i,k) = -inf;
   else
      f(i,k) = irrben(k) + recben(i-k+1);
   end
end
end
```

Here, a reward matrix element is set to negative infinity if the irrigation level exceeds the available water stock. This guarantees that the value maximization algorithm will not chose an infeasible level.

Next, one constructs the transition probability matrix:

```
P = [];
for k=1:m
   Pk = sparse(zeros(n,n));
   for i=1:n;
      j=i-k+1; j=max(1,j); j=min(n,j);
      Pk(i,j) = Pk(i,j) + 0.4;
      j=j+1; j=max(1,j); j=min(n,j);
      Pk(i,j) = Pk(i,j) + 0.6;
   end
   P = [P;Pk];
end
```

To solve the infinite horizon model via function iteration, one must set a maximum number of iterations and a convergence tolerance, and must supply an initial guess for the infinite horizon value function. Because function iteration generates successive approximants to the one infinite horizon value function, the old approximant may be discarded once a new one is generated:

```
maxit = 300;                              % maximum iterations
tol = 1.e-8;                              % convergence tolerance
v = zeros(length(s),1);                   % initial value
for it=1:maxit                            % function iteration
   vold = v;                              % store old value
   [v,ix] = valmax(v,f,P,delta);          % Bellman equation
   change = norm(v-vold);                 % change in policy
   fprintf ('\n%5i %10.1e',it,change)     % print progress
   if change<tol, break, end;             % convergence check
end
```

Upon convergence, `v` will be `n` by `1` matrix containing the value function and `ix` will be `n` by `1` matrix containing optimal irrigation policy indices. The optimal irrigation levels may be recovered by typing `x(ix)`.

To solve the infinite horizon model via policy iteration, one must set a maximum number of iterations and a convergence tolerance, and must supply an initial guess for the optimal irrigation policy indices.

```
maxit = 300;                              % maximum iterations
tol = 1.e-8;                              % convergence tolerance
ix = ones(length(s),1);                   % initial policy
for it=1:maxit                            % policy iteration
   ixold = ix;                            % store old policy
   v = valpol(ix,f,P,delta);              % implied value
   [v,ix] = valmax(v,f,P,delta);          % update policy
   change = norm(ix-ixold);               % change in policy
   fprintf ('\n%5i %10.1e',it,change)     % print progress
   if ix==ixold, break, end;              % convergence check
end
```

Upon convergence, `v` will be `n` by `1` matrix containing the value function and `ix` will be `n` by `1` matrix containing optimal irrigation level indices. The irrigation levels may be recovered by typing `x(ix)`. Note that the intermediate value functions `v` computed by `valmax` are not used by the algorithm.

Once the optimal solution has been computed, one may plot the optimal value and irrigation policy functions:

```
figure(1); plot(s,x(ix)); xlabel('Stock'); ylabel('Optimal Irrigation');
figure(2); plot(s,v); xlabel('Stock'); ylabel('Optimal Value');
```

One can also produce a picture of the expected net social benefit over time beginning from a stock level of, say, zero:

```
figure(3)
[vstar,Pstar,fstar] = valpol(ix,f,P,delta);
pi = zeros(size(s)); pi(1)=1;            % Current state with certainty
maxtime = 20;
for t=0:maxtime;
   expbenefit(t+1) = pi'*fstar;          % Compute expected benefit
   pi = Pstar'*pi;                       % Update conditional distribution
end
plot(0:maxtime,expbenefit); xlabel('Year'); ylabel('Expected Benefit');
```