Dynamic Programming and Reinforcement Learning

Lecture 2: Infinite-Horizon Dynamic Programming

Mengdi Wang

Operations Research and Financial Engineering
Princeton University

July 25-29, 2015

Abstract DP Model

Descrete-time System

State transition:

$$x_{k+1} = f_k(x_k, u_k, w_k), \qquad k = 0, 1, \ldots, N-1$$

- x_k: state; summarizing past information that is relevant for future optimization
- u_k: control/action; decision to be selected at time k from a given set U_k
- w_k: random disturbance or noise
- $g_k(x_k, u_k, w_k)$: state transitional cost incurred at time k given current state x_k and control u_k
- For every k and every x_k, we want an optimal action. We look for a mapping μ from states to actions.

Abstract DP Model

Objective - control the system to minimize overall cost

$$\min_{\mu} \mathbf{E}\left\{ g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k) \right\},$$
$$\text{subject to } u_k = \mu(x_k), \qquad k = 0, \ldots, N-1.$$

We look for a policy/strategy μ, which is a mapping from states to actions.

Finite-Horizon DP: Theory and Algorithm

- Value function is defined as the optimal revenue one can earn (by using the optimal policy onward) starting at time period t with inventory x

$$V_t(x) = \max_{\mu_t,\ldots,\mu_T} \mathbf{E}\left[\sum_{k=t}^{T} g_k(x_k, \mu_k(x_k), w_k) \mid x_t = x\right]$$

- Bellman Equation: for all x, $t \leq T-1$,

$$V_t(x) = \max_{\mu_t} \mathbf{E}\left[g_t(x, \mu_t(x), w_t) + V_{t+1}(x_{t+1}) \mid x_t = x\right]$$

- Tail Optimality: A strategy $\mu_1^*, \ldots, \mu_T^*$ is optimal, if and only if every tail strategy $\mu_t^*, \ldots, \mu_T^*$ is optimal for the tail problem starting at stage t.
- DP Algorithm: solve the Bellman equation directly using backward induction. This is value iteration.

Option Pricing: A Simple Binomial Model

We focus on American call options.

- Strike price: K
- Duration: T days
- Stock price of tth day: S_t
- Growth rate: $u \in (1, \infty)$
- Diminish rate: $d \in (0, 1)$
- Probability of growth: $p \in [0, 1]$

Binomial Model of Stock Price

$$S_{t+1} = \begin{cases} uS_t & \text{with probability p} \\ dS_t & \text{with probability 1-p} \end{cases}$$

As the discretization of time becomes finer, the binomial model approaches the Brownian motion model.

Option Pricing: Bellman's Equation

- Given S_0, T, K, u, r, p.
- State: S_t, finite number of possible values
- Cost vector: $V_t(S)$, the value of option at the tth day when the current stock price is S.

Bellman equation for binomial option

$$V_t(S_t) = \max\Big\{S_t - K, \quad pV_{t+1}(uS_t) + (1-p)V_{t+1}(dS_t)\Big\},$$
$$V_t(S_T) = \max\{S_T - K, 0\}.$$

Infinite-Horizon Discounted Problems/Bounded Cost

- Stationary system

$$x_{k+1} = f(x_k, u_k, w_k), \qquad k = 0, 1, \ldots$$

- Cost of a policy $\pi = \{\mu_0, \mu_1, \ldots\}$

$$J_\pi(x_0) = \lim_{N\to\infty} \mathbf{E}_{\substack{w_k \\ k=0,1,\ldots}} \left\{ \sum_{k=0}^{N-1} \alpha^k g\left(x_k, \mu_k(x_k), w_k\right) \right\}$$

with $\alpha < 1$, and g is bounded [for some M, we have $|g(x, u, w)| \leq M$ for all (x, u, w)]

- Optimal cost function is defined as

$$J^*(x) = \min_\pi J_\pi(x)$$

Infinite-Horizon Discounted Problems/Bounded Cost

- Boundedness of g guarantees that all costs are well-defined and bounded:

$$\left|J_\pi(x)\right| \leq \frac{M}{1-\alpha}$$

- All spaces are arbitrary - only boundedness of g is important (there are math fine points, e.g. measurability, but they don't matter in practice)
- Important special case with finite space: Markovian Decision Problem
- All algorithms ultimately work with a finite spaces MDP approximating the original problem

Shorthand notation for DP mappings

- For any function J of x, denote

$$(TJ)(x) = \min_{u \in U(x)} \mathbf{E}_w \left\{ g(x,u,w) + \alpha J\left(f(x,u,w)\right) \right\}, \ \forall \ x$$

- TJ is the optimal cost function for the one-stage problem with stage cost g and terminal cost function αJ.
- T operates on bounded functions of x to produce other bounded functions of x
- For any stationary policy μ, denote

$$(T_\mu J)(x) = \mathbf{E}_w \left\{ g\left(x,\mu(x),w\right) + \alpha J\left(f(x,\mu(x),w)\right) \right\}, \ \forall \ x$$

- The critical structure of the problem is captured in T and T_μ
- The entire theory of discounted problems can be developed in shorthand using T and T_μ
- True for many other DP problems.
- T and T_μ provide a powerful unifying framework for DP. This is the essence of the book "Abstract Dynamic Programming"

Express Finite-Horizon Cost using T

- Consider an N-stage policy $\pi_0^N = \{\mu_0, \mu_1, \ldots, \mu_{N-1}\}$ with a terminal cost J:

$$\begin{aligned} J_{\pi_0^N}(x_0) &= \mathbf{E}\left\{\alpha^N J(x_k) + \sum_{\ell=0}^{N-1} \alpha^\ell g\left(x_\ell, \mu_\ell(x_\ell), w_\ell\right)\right\} \\ &= \mathbf{E}\left\{g\left(x_0, \mu_0(x_0), w_0\right) + \alpha J_{\pi_1^N}(x_1)\right\} \\ &= (T_{\mu_0} J_{\pi_1^N})(x_0) \end{aligned}$$

  where $\pi_1^N = \{\mu_1, \mu_2, \ldots, \mu_{N-1}\}$
- By induction we have

$$J_{\pi_0^N}(x) = (T_{\mu_0} T_{\mu_1} \cdots T_{\mu_{N-1}} J)(x), \qquad \forall\ x$$

- For a stationary policy μ the N-stage cost function (with terminal cost J) is

$$J_{\pi_0^N} = T_\mu^N J$$

  where T_μ^N is the N-fold composition of T_μ
- Similarly the optimal N-stage cost function (with terminal cost J) is $T^N J$
- $T^N J = T(T^{N-1} J)$ is just the DP algorithm

Markov Chain

- A Markov chain is a random process that takes values on the state space $\{1, \ldots, n\}$.
- The process evolves according to a certain transition probability matrix $P \in \Re^{n \times n}$ where

$$P(i_{k+1} = j \mid i_k, i_{k-1}, \ldots, i_0) = P(i_{k+1} = j \mid i_k = i) = P_{ij}$$

- Markov chain is memoryless, i.e., further evolvements are independent with past trajectory conditioned on the current state.
- The "memoryless" property is equivalent to "Markov."
- A state i is recurrent if it will be visited infinitely many times with probability 1.
- A Markov chain is said to be irreducible if its state space is a single communicating class; in other words, if it is possible to get to any state from any state.
- When states are modeled appropriately, all stochastic processes are Markov.

Markovian Decision Problem

We will mostly assume the system is an n-state (controlled) Markov chain

- States $i = 1, \ldots, n$ (instead of x)
- Transition probabilities $p_{i_k i_{k+1}}(u_k)$ [instead of $x_{k+1} = f(x_k, u_k, w_k)$]
- Stage cost $g(i_k, u_k, i_{k+1})$ [instead of $g(x_k, u_k, w_k)$]
- Cost functions $J = \big(J(1), \ldots, J(n)\big)$ (vectors in $\Re^n$)
- Cost of a policy $\pi = \{\mu_0, \mu_1, \ldots\}$

$$J_\pi(i) = \lim_{N\to\infty} \mathbf{E}_{\substack{i_k \\ k=1,2,\ldots}} \left\{ \sum_{k=0}^{N-1} \alpha^k g\left(i_k, \mu_k(i_k), i_{k+1}\right) \mid i_0 = i \right\}$$

- MDP is the most important problem in infinite-horizon DP

Markovian Decision Problem

- Shorthand notation for DP mappings

$$(TJ)(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha J(j)\big), \quad i = 1, \ldots, n,$$

$$(T_\mu J)(i) = \sum_{j=1}^{n} p_{ij}\big(\mu(i)\big)\big(g\left(i, \mu(i), j\right) + \alpha J(j)\big), \quad i = 1, \ldots, n$$

- Vector form of DP mappings

$$TJ = \min_{\mu} \{g_\mu + P_\mu J\}$$

and

$$T_\mu J = g_\mu + P_\mu J,$$

where

$$g_\mu(i) = \sum_{j=1}^{n} p_{ij}(\mu(i)) g(i, \mu(i), j), \qquad P_\mu(i,j) = p_{ij}(\mu(i)).$$

Two Key properties

- Monotonicity property: For any J and J' such that $J(x) \leq J'(x)$ for all x, and any μ

$$(TJ)(x) \leq (TJ')(x), \qquad \forall\ x,$$

$$(T_\mu J)(x) \leq (T_\mu J')(x), \qquad \forall\ x.$$

- Constant Shift property: For any J, any scalar r, and any μ

$$(T(J + re))(x) = (TJ)(x) + \alpha r, \qquad \forall\ x,$$

$$(T_\mu(J + re))(x) = (T_\mu J)(x) + \alpha r, \qquad \forall\ x,$$

  where e is the unit function $[e(x) \equiv 1]$.
- Monotonicity is present in all DP models (undiscounted, etc)
- Constant shift is special to discounted models
- Discounted problems have another property of major importance: T and T_μ are contraction mappings (we will show this later)

Convergence of Value Iteration

Theorem

For all bounded J_0, we have $J^*(x) = \lim_{k\to\infty}(T^k J_0)(x)$, for all x

Proof. For simplicity we give the proof for $J_0 \equiv 0$. For any initial state x_0, and policy $\pi = \{\mu_0, \mu_1, \ldots\}$,

$$J_\pi(x_0) = \mathbf{E}\left\{\sum_{\ell=0}^{\infty} \alpha^\ell g\left(x_\ell, \mu_\ell(x_\ell), w_\ell\right)\right\}$$

$$= \mathbf{E}\left\{\sum_{\ell=0}^{k-1} \alpha^\ell g\left(x_\ell, \mu_\ell(x_\ell), w_\ell\right)\right\} + \mathbf{E}\left\{\sum_{\ell=k}^{\infty} \alpha^\ell g\left(x_\ell, \mu_\ell(x_\ell), w_\ell\right)\right\}$$

The tail portion satisfies

$$\left|\mathbf{E}\left\{\sum_{\ell=k}^{\infty} \alpha^\ell g\left(x_\ell, \mu_\ell(x_\ell), w_\ell\right)\right\}\right| \leq \frac{\alpha^k M}{1-\alpha}$$

where $M \geq |g(x,u,w)|$. Take min over π of both sides, then lim as $k \to \infty$. ■

Proof of Bellman's equation

Theorem

The optimal cost function J^* is a solution of Bellman's equation, $J^* = TJ^*$, i.e., for all x,

$$J^*(x) = \min_{u \in U(x)} \mathbf{E}_w \left\{ g(x,u,w) + \alpha J^* \left(f(x,u,w) \right) \right\}$$

Proof. For all x and k,

$$J^*(x) - \frac{\alpha^k M}{1-\alpha} \leq (T^k J_0)(x) \leq J^*(x) + \frac{\alpha^k M}{1-\alpha},$$

where $J_0(x) \equiv 0$ and $M \geq |g(x,u,w)|$. Applying T to this relation, and using Monotonicity and Constant Shift,

$$(TJ^*)(x) - \frac{\alpha^{k+1} M}{1-\alpha} \leq (T^{k+1} J_0)(x) \leq (TJ^*)(x) + \frac{\alpha^{k+1} M}{1-\alpha}$$

Taking the limit as $k \to \infty$ and using the fact $\lim_{k\to\infty}(T^{k+1}J_0)(x) = J^*(x)$ we obtain $J^* = TJ^*$. ■

The Contraction Property

- Contraction property: For any bounded functions J and J', and any μ,

$$\max_x |(TJ)(x) - (TJ')(x)| \leq \alpha \max_x |J(x) - J'(x)|,$$

$$\max_x |(T_\mu J)(x) - (T_\mu J')(x)| \leq \alpha \max_x |J(x) - J'(x)|.$$

Proof. Denote $c = \max_{x \in S} |J(x) - J'(x)|$. Then

$$J(x) - c \leq J'(x) \leq J(x) + c, \qquad \forall\ x$$

Apply T to both sides, and use the Monotonicity and Constant Shift properties:

$$(TJ)(x) - \alpha c \leq (TJ')(x) \leq (TJ)(x) + \alpha c, \qquad \forall\ x$$

Hence

$$|(TJ)(x) - (TJ')(x)| \leq \alpha c, \qquad \forall\ x.$$

■

- This implies that T, T_μ have unique fixed points. Then J^* is the unique solution of $J^* = TJ^*$, and J_μ is the unique solution of $J_\mu = T_\mu J_\mu$

Nec. and Sufficient Opt. Condition

Theorem

A stationary policy μ is optimal if and only if $\mu(x)$ attains the minimum in Bellman's equation for each x; i.e.,

$$TJ^* = T_\mu J^*,$$

or, equivalently, for all x,

$$\mu(x) \in \arg\min_{u \in U(x)} \mathbf{E}_w \left\{ g(x,u,w) + \alpha J^* \left(f(x,u,w) \right) \right\}$$

Proof of Optimality Condition

Proof. We have two directions.

- If $TJ^* = T_\mu J^*$, then using Bellman's equation ($J^* = TJ^*$), we have

$$J^* = T_\mu J^*,$$

  so by uniqueness of the fixed point of T_μ, we obtain $J^* = J_\mu$; i.e., μ is optimal.

- Conversely, if the stationary policy μ is optimal, we have $J^* = J_\mu$, so

$$J^* = T_\mu J^*.$$

  Combining this with Bellman's Eq. ($J^* = TJ^*$), we obtain $TJ^* = T_\mu J^*$. ■

Two Main Algorithms

Value Iteration (VI)

Solve the Bellman equation $J^* = TJ^*$ by iterating on the value functions:

$$J_{k+1} = TJ_k,$$

or

$$J_{k+1}(i) = \min_u \sum_{i=1}^{n} p_{ij}(u) \left(g(i,u,j) + \alpha J_k(j) \right),$$

for $i = 1, \ldots, n$.

- The program only needs to memorize the current value function J_k.
- We have shown that $J_k \to J^*$ as $k \to \infty$.

Policy Iteration (PI)

Solve the Bellman equation $J^* = TJ^*$ by iterating on the policies

Policy Iteration (PI)

Given μ^k, the k-th policy iteration has two steps

- Policy evaluation: Find J_{μ^k} by solving

$$J_{\mu^k}(i) = \sum_{j=1}^{n} p_{ij}\big(\mu^k(i)\big)\big(g\left(i, \mu^k(i), j\right) + \alpha J_{\mu^k}(j)\big), \quad i = 1, \ldots, n$$

  or $J_{\mu^k} = T_{\mu^k} J_{\mu^k}$

- Policy improvement: Let μ^{k+1} be such that

$$\mu^{k+1}(i) \in \arg\min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i, u, j) + \alpha J_{\mu^k}(j)\big), \quad \forall\ i$$

  or $T_{\mu^{k+1}} J_{\mu^k} = T J_{\mu^k}$

Policy iteration is a method that updates the policy instead of the value function.

Policy Iteration (PI)

More abstractly, the k-th policy iteration has two steps

- Policy evaluation: Find J_{μ^k} by solving

$$J_{\mu^k} = T_{\mu^k} J_{\mu^k} = g_{\mu^k} + P_{\mu^k} J_{\mu^k}$$

- Policy improvement: Let μ^{k+1} be such that

$$T_{\mu^{k+1}} J_{\mu^k} = T J_{\mu^k}.$$

Comments:

- Policy evaluation is equivalent to solving an $n \times n$ linear system of equations
- Policy improvement is equivalent to 1-step lookahead using the evaluated value function
- For large n, exact PI is out of the question . We use instead optimistic PI (policy evaluation with a few VIs)

Convergence of Policy Iteration

Theorem

Assume that the state and action spaces are finite. The policy iteration generates μ^k that converges to the optimal policy μ^* in a finite number of steps.

Proof. We show that $J_{\mu^k} \geq J_{\mu^{k+1}}$ for all k.

- For given k, we have

$$J_{\mu^k} = T_{\mu^k} J_{\mu^k} \geq T J_{\mu^k} = T_{\mu^{k+1}} J_{\mu^k}$$

  Using the monotonicity property of DP,

$$J_{\mu^k} \geq T_{\mu^{k+1}} J_{\mu^k} \geq T^2_{\mu^{k+1}} J_{\mu^k} \geq \cdots \geq \lim_{N\to\infty} T^N_{\mu^{k+1}} J_{\mu^k}$$

- Since

$$\lim_{N\to\infty} T^N_{\mu^{k+1}} J_{\mu^k} = J_{\mu^{k+1}}$$

  we have $J_{\mu^k} \geq J_{\mu^{k+1}}$.
- If $J_{\mu^k} = J_{\mu^{k+1}}$, all above inequalities hold as equations, so J_{μ^k} solves Bellman's equation. Hence $J_{\mu^k} = J^*$
- Thus at iteration k either the algorithm generates a strictly improved policy or it finds an optimal policy. For a finite spaces MDP, the algorithm terminates with an optimal policy.

"Shorthand" Theory – A Summary

- Infinite horizon cost function expressions [with $J_0(x) \equiv 0$]

$$J_\pi(x) = \lim_{N\to\infty} (T_{\mu_0} T_{\mu_1} \cdots T_{\mu_N} J_0)(x), \quad J_\mu(x) = \lim_{N\to\infty} (T_\mu^N J_0)(x)$$

- Bellman's equation: $J^* = TJ^*$, $J_\mu = T_\mu J_\mu$
- Optimality condition:

$$\mu \text{: optimal} \quad <==> \quad T_\mu J^* = TJ^*$$

- Value iteration: For any (bounded) J

$$J^*(x) = \lim_{k\to\infty} (T^k J)(x), \qquad \forall\ x$$

- Policy iteration: Given μ^k,
  - Policy evaluation: Find J_{μ^k} by solving

$$J_{\mu^k} = T_{\mu^k} J_{\mu^k}$$

  - Policy improvement : Find μ^{k+1} such that

$$T_{\mu^{k+1}} J_{\mu^k} = TJ_{\mu^k}$$

Q-Factors

- Optimal Q-factor of (x, u) :

$$Q^*(x,u) = \mathbf{E}\left\{g(x,u,w) + \alpha J^*(f(x,u,w))\right\}.$$

- It is the cost of starting at x, applying u is the 1st stage, and an optimal policy after the 1st stage
- The value function is equivalent to

$$J^*(x) = \min_{u \in U(x)} Q^*(x,u), \qquad \forall\ x.$$

- Q-factors are costs in an "augmented" problem where states are (x, u)
- Here (x, u) is a post-decision state.

VI in Q-factors

- We can equivalently write the VI method as

$$J_{k+1}(x) = \min_{u \in U(x)} Q_{k+1}(x,u), \qquad \forall\ x,$$

where Q_{k+1} is generated by

$$Q_{k+1}(x,u) = \mathbf{E}\left\{ g(x,u,w) + \alpha \min_{v \in U(\overline{x})} Q_k(f(x,u,w),v) \right\}$$

- VI converges for Q-factors

Q-Factors

- VI and PI for Q-factors are mathematically equivalent to VI and PI for costs
- They require equal amount of computation ... they just need more storage
- Having optimal Q-factors is convenient when implementing an optimal policy on-line by

$$\mu^*(x) = \min_{u \in U(x)} Q^*(x,u)$$

- Once $Q^*(x,u)$ are known, the model [g and $E\{\cdot\}$] is not needed. Model-free operation
- Q-Learning (to be discussed later) is a sampling method that calculates $Q^*(x,u)$ using a simulator of the system (no model needed)

MDP and Q-Factors

Optimal Q-factors - the function $Q(i,u)$ that satisfies the following Bellman equation

$$Q^*(i,u) = \sum_{j=1}^{n} p_{ij}(u) \left(g(i,u,j) + \alpha \min_{v \in U(j)} Q^*(j,v) \right),$$

or in short

$$Q^* = FQ^*$$

Interpretation: Q-factors can be viewed as J values by considering (i,u) as the post-decision state

DP Algorithm for Q-values instead of J-values

- Value Iteration: $Q_{k+1} = FQ_k$
- Policy Iteration: $F_{\mu_{k+1}} Q_{\mu_k} = FQ_{\mu_k}$
- VI and PI are convergent for Q-values
- Model-free.

Other DP Models

- We have looked so far at the (discrete or continuous spaces) discounted models for which the analysis is simplest and results are most powerful
- Other DP models include:
- Undiscounted problems ($\alpha = 1$): They may include a special termination state (stochastic shortest path problems)
- Continuous-time finite-state MDP : The time between transitions is random and state-and-control-dependent (typical in queueing systems, called Semi-Markov MDP). These can be viewed as discounted problems with state-and-control-dependent discount factors
- Continuous-time, continuous-space models : Classical automatic control, process control, robotics
- Substantial differences from discrete-time
- Mathematically more complex theory (particularly for stochastic problems)
- Deterministic versions can be analyzed using classical optimal control theory
- Admit treatment by DP, based on time discretization

Practical Difficulties of DP

- The curse of dimensionality
- Exponential growth of the computational and storage requirements as the number of state variables and control variables increases
- Quick explosion of the number of states in combinatorial problems
- The curse of modeling
- Sometimes a simulator of the system is easier to construct than a model
- There may be real-time solution constraints
- A family of problems may be addressed. The data of the problem to be solved is given with little advance notice
- The problem data may change as the system is controlled – need for on-line replanning
- All of the above are motivations for approximation and simulation

General Orientation to ADP

- ADP (late 80s - present) is a breakthrough methodology that allows the application of DP to problems with many or infinite number of states .
- Other names for ADP are:
- "reinforcement learning" (RL).
- "neuro-dynamic programming" (NDP).
- "adaptive dynamic programming" (ADP).
- We will mainly adopt an n-state discounted model (the easiest case - but think of HUGE n).
- Extensions to other DP models (continuous space, continuous-time, not discounted) are possible (but more quirky). We will set aside for later.
- There are many approaches:
  - Problem approximation
  - Simulation-based approaches (we will focus on these)
- Simulation-based methods are of three types:
  - Rollout (we will not discuss further)
  - Approximation in value space
  - Approximation in policy space

Why do we use Simulation?

- One reason: Computational complexity advantage in computing sums/expectations involving a very large number of terms
- Any sum

$$\sum_{i=1}^{n} a_i$$

can be written as an expected value :

$$\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} \xi_i \frac{a_i}{\xi_i} = E_\xi \left\{ \frac{a_i}{\xi_i} \right\},$$

where ξ is any prob. distribution over $\{1, \ldots, n\}$

- It can be approximated by generating many samples $\{i_1, \ldots, i_k\}$ from $\{1, \ldots, n\}$, according to distribution ξ, and Monte Carlo averaging:

$$\sum_{i=1}^{n} a_i = E_\xi \left\{ \frac{a_i}{\xi_i} \right\} \approx \frac{1}{k} \sum_{t=1}^{k} \frac{a_{i_t}}{\xi_{i_t}}$$

- Simulation is also convenient when an analytical model of the system is unavailable , but a simulation/computer model is possible.

Solve DP via Simulation

- Ideally, VI and PI solve the fixed equation: finding J^* such that

$$J^* = \min_{\mu} \{g_\mu + \alpha P_\mu J^*\}$$

- Practically, we often wish to solve Bellman's equation without knowing P_μ, g_μ.
- What we do have: a simulator that starts from state i, given action a, generate random samples of transition costs and future state

$$g(i, i_{next}, a), \quad i_{next}$$

Example: Optimize a trading policy to maximize profit

- Current transaction has unknown market impact
- Use current order book as states/features

Example: stochastic games, Tetris, hundreds of millions of states, captured using 22 features

Value Iteration

Exercise 2-1

Use VI to evaluate an infinite-time American call options.
The program should be a function of $S_0, \alpha, p, U, D, u, d, K$.

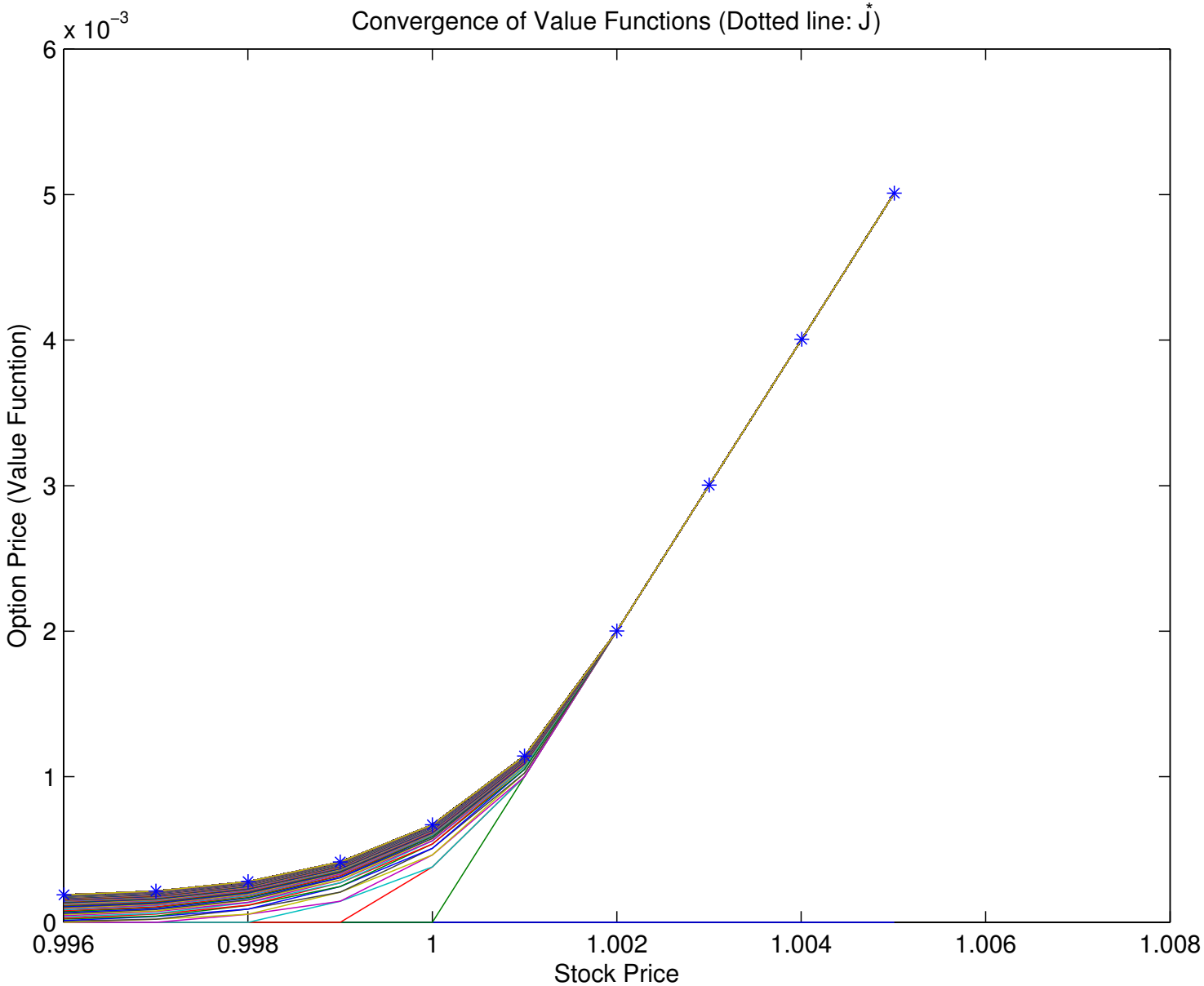
Value Iteration Algorithm

- Start with an arbitrary J_0.
- Repeat the following until convergence:
  - Given J_k.
  - Apply value iteration $J_{k+1} = TJ_k$.
- Obtain the optimal stopping policy from J^*.
- Plot Q values.

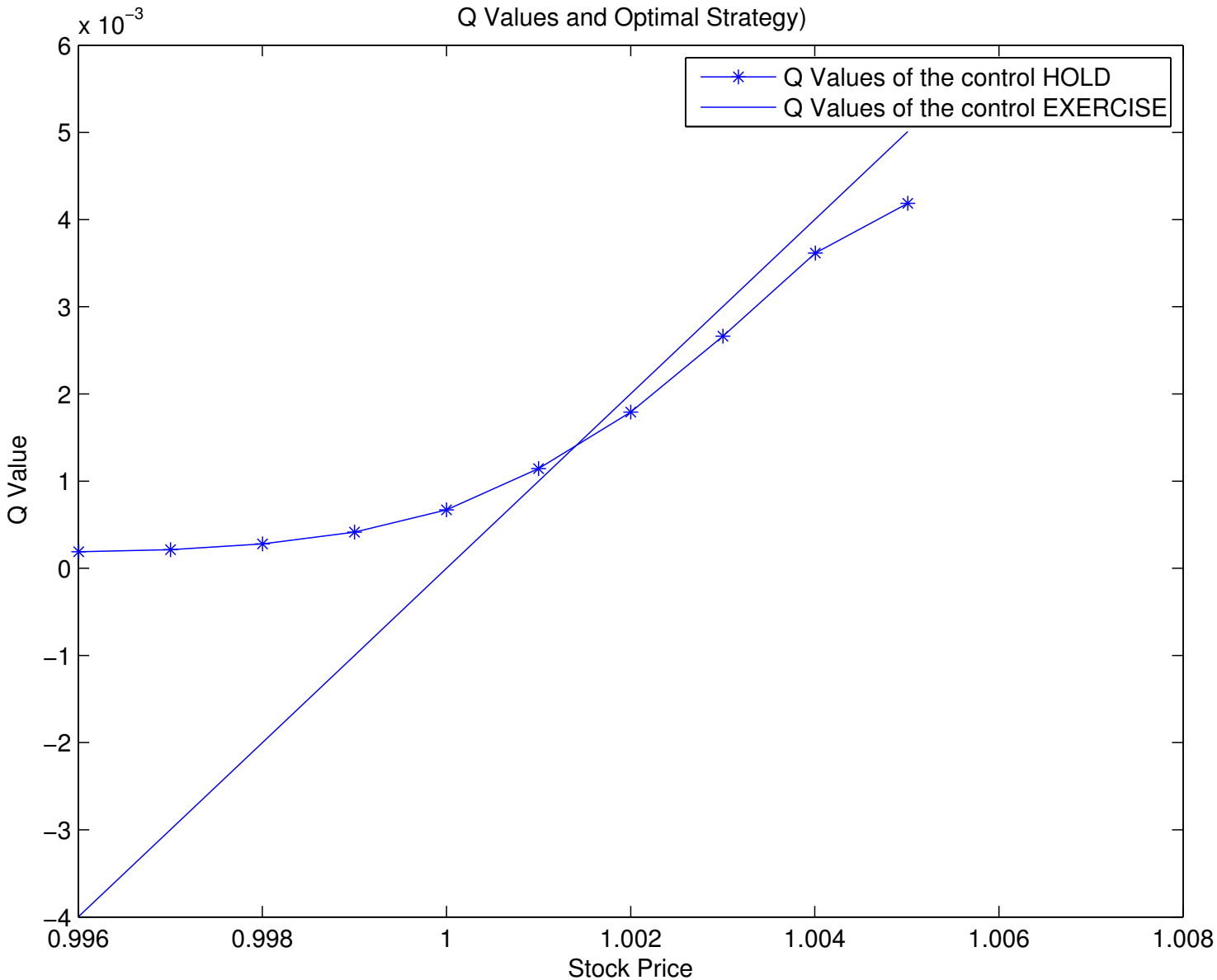
Suggestions:

- Define a finite state space through discretization.

Convergence of Value Functions J_t



Optimal Q Values and Exercising Policies



Policy Iteration for Option Pricing

Algorithm

- Starts with any μ_0.
- Policy evaluation:
  - Evaluate J_{μ_t} by VI: applying $T_{\mu_k} J \mapsto J$ till convergence, and yielding J_{μ_k}.
  - Evaluate the Q-values by $Q_{\mu_t}(i_t) = \alpha \mathbf{E}\left[J_{\mu_t}(i_{t+1})\right]$.
- Policy improvement:

$$\mu_{t+1}(i) = \begin{cases} \text{HOLD} & \text{if } S(i) - K \leq Q_{\mu_t}(i), \\ \text{EXERCISE} & \text{Otherwise.} \end{cases}$$

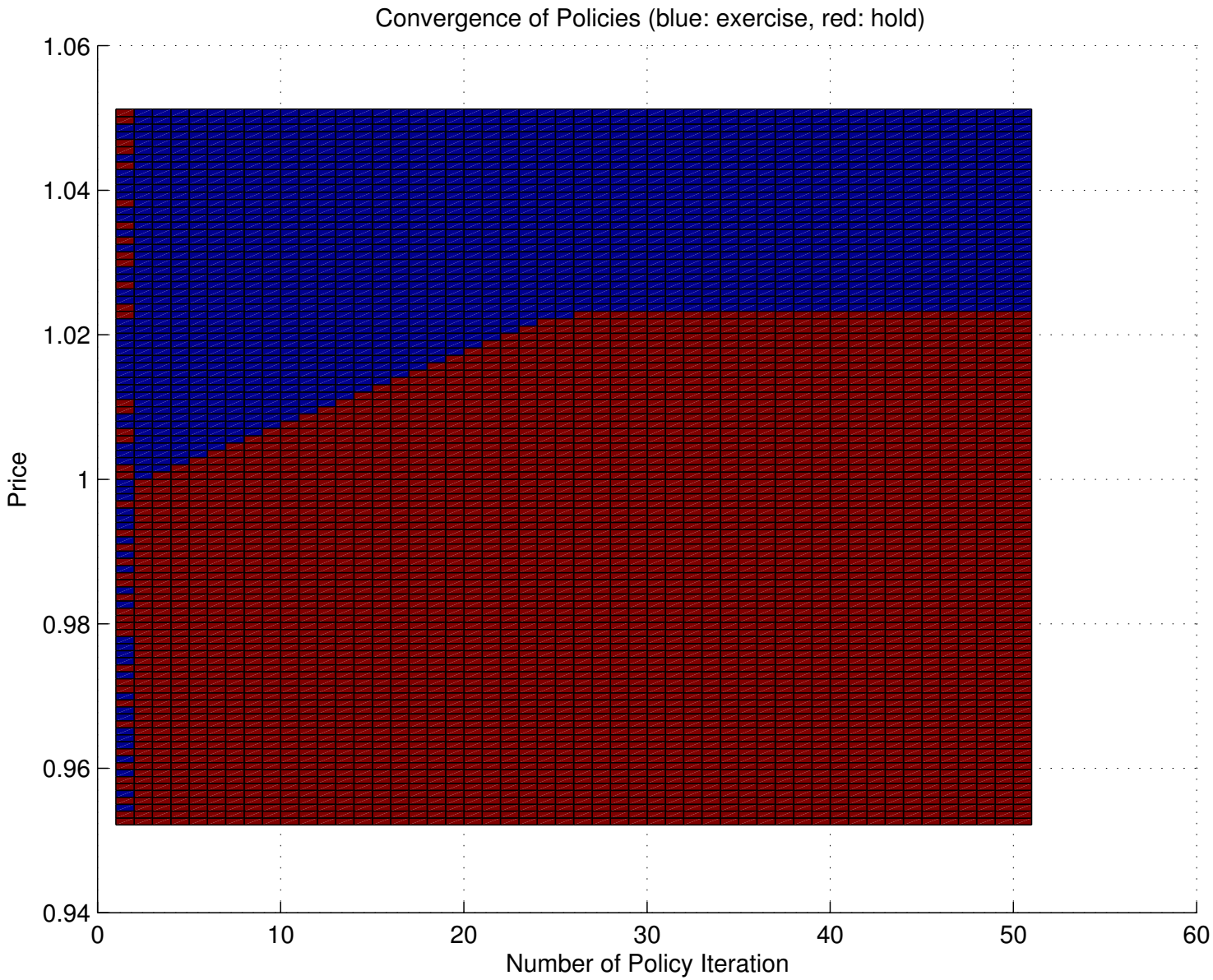
Exercise 2-2

Modify the code of Exercise 2-1, to use PI to evaluate an infinite-time American call options. The program should be a function of $S_0, \alpha, p, U, D, u, d, K$.

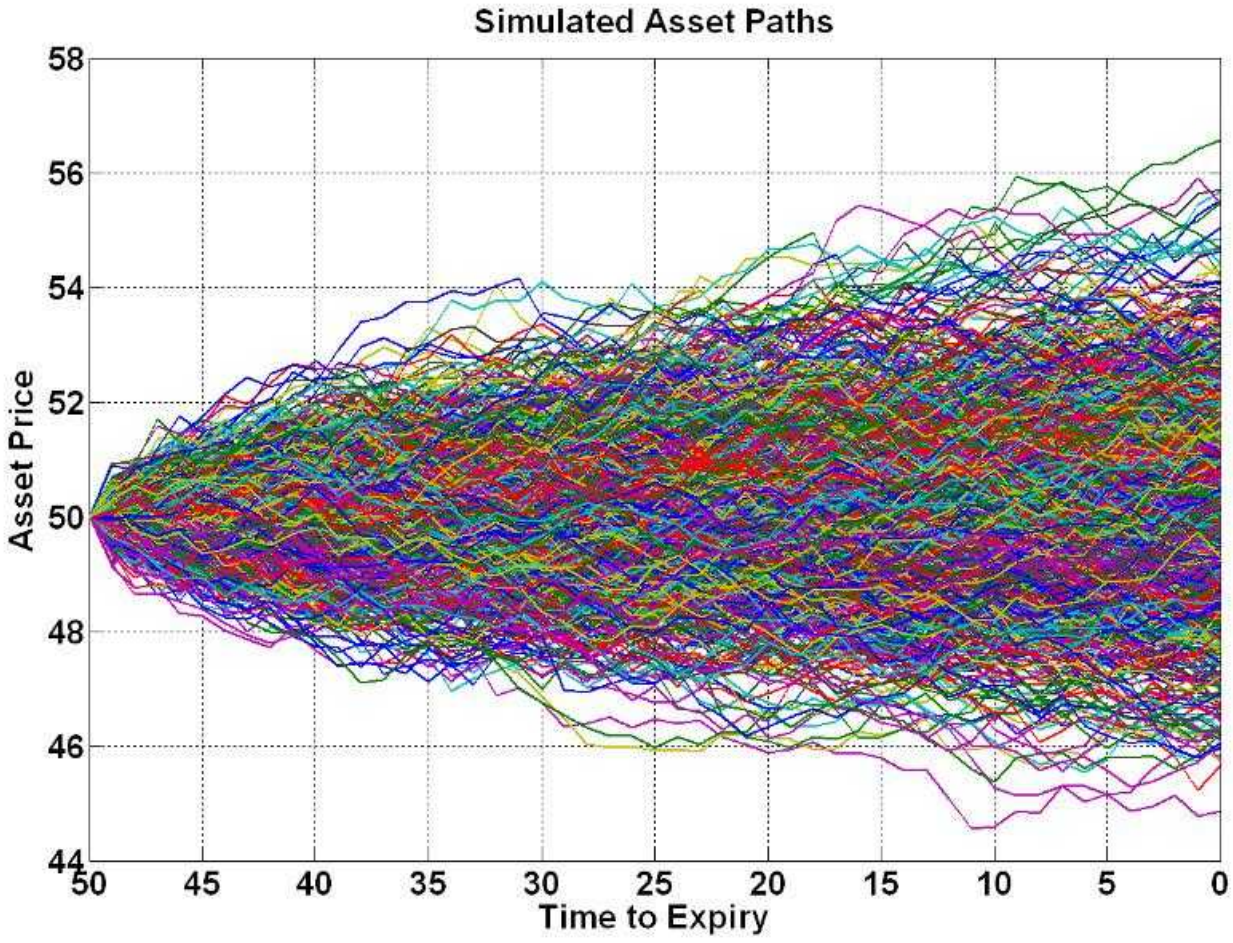
Suggestions:

- Start with a randomly generated policy $\mu_0 : \{1, \ldots, n\} \mapsto \{HOLD, EXERCISE\}$.
- Use VI to evaluate J_{μ_t} for a given policy μ_t.
- Plot the trajectories of μ_t.

Convergence of Policies



Option pricing without the price model



- Use simulated/historical trajectories
- Backward induction on trajectories data

Data-Based DP Algorithm

Suppose that we have N price trajectories, from either simulation or historical data,

$$\{S_t^{(i)}\}_{t\in[0,T]}, \qquad \text{where } i = 1, \ldots, T$$

Let us price an option based on the data without knowing the price model.

Backward induction based on sample trajectories

Calculate the value for possible price S_t at time t

$$V_t(S_t) = \max\left\{S_t - K, \quad \frac{1}{\sum_{i=1}^N \mathbf{1}_{S_t^{(i)}=S_t}} \sum_{i=1}^N V_{t+1}(S_{t+1}^{(i)})\mathbf{1}_{S_t^{(i)}=S_t}\right\},$$

$$V_t(S_T) = \max\{S_T - K, 0\}.$$

- The value functions $V_t(S)$ is the fair price of the call option at time T if the current price is S.
- This approach is model-free, which is very important to price exotic options and price the effects of future events.

Exercise 2-3

Modify the code of Exercise 2-1. Generate random trajectories of stock prices. Use simulation-based VI to evaluate an infinite-time American call options. The program should be a function of $S_0, \alpha, p, U, D, u, d, K$.