ENN582: Reinforcement Learning and Optimal Control
Dynamic Programming

Daniel E. Quevedo[‡]

VERSION 1.

Abstract

The purpose of these notes is to give a succinct introduction to Dynamic Programming methods for determinsitic and stochastic nonlinear control systems. More detailed expositions can be found in a number of books, such as [3].

1 Key Learning Objectives

Key Content Points:

- Principle of Optimality
- Dynamic Programming Algorithm
- Markov Decision Process

[‡]Please report errors within this document to daniel.quevedo@qut.edu.au

Key Learning Objectives - in conjunction with the practical:

- Understand the principle of optimality
- Be able to implement dynamic programming algorithms for deterministic and stochastic optimal control problems
- Formulate Markov Decision processes as an abstraction to engineering design scenarios

2 Principle of Optimality

Dynamic Programming (DP) principles can be applied to a large variety of multi-stage decision making scenarios. The key idea is to solve a complex and difficult problem through solving a collection of subproblems. DP ideas are very intuitive and mirror how in many everyday situations (such as the maze in Figure 1) we solve problems by first setting a goal and then recursively (going backwards) work out the steps to reach it. The remainder of this unit builds strongly on associated concepts. We will next introduce the main ideas in an intuitive way –guided by an example– before proceeding to a more precise mathematical exposition.

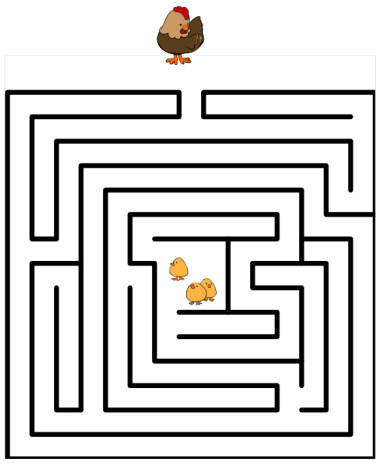

Figure 1: Moving backwards to find the chicks

Consider the shortest path problem depicted in Figure 2. The goal is to minimise the cost (e.g., travel time, distance or energy use) when travelling from Node S to Node E. The costs when travelling between individual nodes are displayed on the edges (arrows). From a graph-theoretic perspective, this problem can be interpreted as finding a path from S to E such that the sum of weights on its constituent edges is minimised.

To solve this problem, we make the key, and intuitive, observation that "If any Node X is on the optimal path, then the sub-path from Node X to Node E must be optimal as well."

More generally, Bellman's principle of optimality holds (see, e.g., [2, Chapter 3]):
"An optimal trajectory has the property that at an intermediate point, no matter how it was reached, the rest of the trajectory must coincide with an optimal trajectory as computed from this intermediate point as the starting point."

The principle of optimality states that from any state on an optimal trajectory, the remaining trajectory (the "tail") is optimal for the corresponding problem initiated at that state. The principle is intuitive and leads to the basic DP algorithm. The DP algorithm proceeds sequentially by solving all tail subproblems of a given length using the solution to every tail subproblem of shorter length [3]. Thus, the algorithm proceeds backwards. The DP algorithm provides the "cost-to-go" of the tail subproblems, as well as the control policies. The latter specify, which action to take at each state.

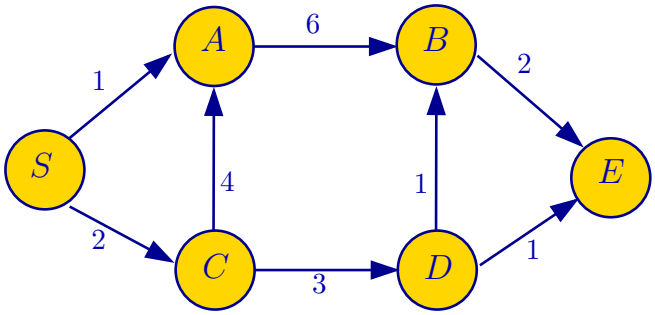

Figure 2: Example with finite state and action spaces: shortest path problem

To use the DP algorithm to solve the shortest path problem in Figure 2, it is convenient to define, for any Node X in the graph, the cost to go-functions

$$\text{dist}(X) \triangleq \text{minimum cost } X \to E.$$

Bellman's principle of optimality and the DP algorithm then provide:

$$\begin{aligned}
&\text{dist}(E) = 0 \\
&\text{dist}(B) = 2 \\
&\text{dist}(D) = \min\{1 + \text{dist}(E), 1 + \text{dist}(B)\} = \min\{1, 1+2\} = 1 \quad \Longrightarrow \quad \textbf{If at D, go to E} \\
&\text{dist}(A) = 6 + \text{dist}(B) = 6 + 2 = 8 \\
&\text{dist}(C) = \min\{3 + \text{dist}(D), 4 + \text{dist}(A)\} = \min\{3+1, 4+8\} = 4 \quad \Longrightarrow \quad \textbf{If at C, go to D} \\
\Rightarrow\ &\text{dist}(S) = \min\{2 + \text{dist}(C), 1 + \text{dist}(A)\} = \min\{2+4, 1+8\} = 6 \quad \Longrightarrow \quad \textbf{If at S, go to C}
\end{aligned}$$

In the above, we have denoted the optimal feedback policy with **boldface**. The associated optimal control sequence then becomes $\{\text{go to C}, \text{go to D}, \text{go to E}\}$.

3 Dynamic Programming Algorithm for Deterministic Systems

To state the DP algorithm in more precise terms, we will proceed as in Chapter 1 of [4, 5] and focus on discrete-time nonlinear time-varying state space systems of the form:

$$x_{k+1} = f_k(x_k, u_k), \quad k \in \mathbb{N}_0. \tag{1}$$

In the above, the system state x_k is an element of the state space $\mathbb{X}$, whereas the control input $u_k \in \mathbb{U}$, the control space. These two spaces could also depend on k.

We will consider additive cost functions with stage costs $g_k(\cdot,\cdot)$ and final stage weighting $g_N(\cdot)$: [1]

$$J(x_0, \vec{u}) = g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k), \tag{2}$$

where $\vec{u} = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix}$ and $x_1 = f_0(x_0, u_0)$, $x_2 = f_1(x_1, u_1)$, etc.

We seek to find an optimal sequence, say $\vec{u}^*$, which minimises this cost, i.e., is such that

$$J(x_0, \vec{u}^*) = \min_{\vec{u} \in \mathbb{U}^N} J(x_0, \vec{u}). \tag{3}$$

We note that this optimal cost is a function of x_0 and we write $J^*(x_0) = J(x_0, \vec{u}^*)$.

Application of the principle of optimality leads to the basic DP algorithm for deterministic systems. It proceeds backwards in time from period $N-1$ to 0 and provides the optimal sequence $\vec{u}^*$, as well as the cost-to-go functions $J_k(x_k)$. The latter quantify the optimal cost of the tail subproblem, which starts at state x_k and time k, and ends at time N, i.e.,

$$J_k(x_k) = \min_{u_k, \ldots, u_{N-1}} \left\{ g_N(x_N) + \sum_{m=k}^{N-1} g_m(x_m, u_m) \right\}. \tag{4}$$

The DP algorithm starts from

$$J_N(x_N) = g_N(x_N)$$

and computes the remaining cost-to-go functions $\{J_{N-1}(x_{N-1}), J_{N-2}(x_{N-2}), \ldots, J_0(x_0)\}$ through the recursions:

$$J_k(x_k) = \min_{u_k \in \mathbb{U}} \Big\{ g_k(x_k, u_k) + J_{k+1}(f_k(x_k, u_k)) \Big\}, \quad k = 0, 1, \ldots, N-1 \tag{5}$$

The optimal cost is obtained at the last step, $J^*(x_0) = J_0(x_0)$ and the optimal control sequence $\{u_0^*, u_1^*, \ldots, u_{N-1}^*\}$ can be recovered by simply setting

$$u_0^* \in \arg\min_{u_0 \in \mathbb{U}} \Big\{ g_0(x_0, u_0) + J_1(f_0(x_0, u_0)) \Big\}, \quad x_1^* = f_0(x_0, u_0^*).$$

[1] Since we are not dealing with receding horizon optimisations, we can use a simpler notation than for MPC.

and going forward in time:

$$u_k^* \in \arg\min_{u_k \in \mathbb{U}} \Big\{ g_k(x_k^*, u_k) + J_{k+1}(f_k(x_k^*, u_k)) \Big\}, \quad x_{k+1}^* = f_k(x_k^*, u_k^*).$$

We emphasize that in the above algorithm, the cost-to-go functions need to be calculated for each state value x_k. This is due to the fact that, due to causality, at the start we don't know which subproblems we will encounter. In practical implementations, the algorithm would typically require one that the state space $\mathbb{X}$ be finitely countable. This may require one to discretise (quantise) the state space. Unless the number of possible states and the time horizon N are small, this will be computationally intensive. However, for some problem instances, closed form (analytical) solutions can be found. One such case will be discussed in the following section.

4 Solution of LQR using Dynamic Programming

The example in Section 2 considers control/decision problems where the state space (the nodes) and action space (the choice of edge) are discrete. Throughout this unit we will mostly focus on such scenarios. However, DP can also be used to establish solutions to some control problems with continuous state and action spaces. In particular, the MPC control law for LTI systems and with a quadratic cost function (i.e., the LQR controller) corresponds to the dynamic system model

$$x_{k+1} = f(x_k, u_k), \quad f(x, u) = Ax + Bu$$

and a cost function of the form (2) with

$$g_k(x, u) = x^T Q x + u^T R u, \quad k = 0, 1, \ldots, N-1$$

and $g_N(x) = x^T Q_F x$.

In this case, the optimal control values at all time steps $k = 0, 1, \ldots, N-1$ are given by

$$u_k^* = -(R + B^T S_{k+1} B)^{-1} B^T S_{k+1} A x_k, \quad k = 0, 1, \ldots, N-1. \tag{6}$$

The matrix S_k evolves according to the so-called backward Riccati recursion

$$S_k = A^T S_{k+1} A + Q - A^T S_{k+1} B (B^T S_{k+1} B + R)^{-1} B^T S_{k+1} A$$

with initial condition $S_N = Q_F$. Furthermore, the optimal cost is given by

$$J^*(x_0) = x_0^T S_0 x_0.$$

We will outline how to establish this result by analysing the cost-to-go function of the tail problem from stage k to stage N, see also [10, Chapter 1] or [3, Chapter 4]:

$$J_k(x_k) = \min_{u_k, \ldots, u_{N-1}} \left\{ x_N^T Q_F x_N + \sum_{\ell=k}^{N-1} \left(x_\ell^T Q x_\ell + u_\ell^T R u_\ell \right) \right\}.$$

We will find that the cost-to-go function J_k is quadratic, i.e., $J_k(x) = x^T S_k x$ and that S_k can be found recursively, starting from $k = N$. Further, the optimal control values are easily expressed in closed form in terms of S_k.

We first note that the cost-to-go at the last stage is simply the final state cost:

$$J_N(x_N) = x_N^T S_N x_N, \quad S_N = Q_F.$$

To show that J_k is quadratic in x_k for all k, we examine stage $k = N - 1$:

$$J_{N-1}(x_{N-1}) = \min_{u_{N-1}} \left\{ x_{N-1}^T Q x_{N-1} + u_{N-1}^T R u_{N-1} + J_N(x_N) \right\}$$

Using $x_N = Ax_{N-1} + Bu_{N-1}$ gives

$$\begin{aligned} J_{N-1}(x_{N-1}) &= \min_{u_{N-1}} \left\{ x_{N-1}^T Q x_{N-1} + u_{N-1}^T R u_{N-1} + (Ax_{N-1} + Bu_{N-1})^T S_N (Ax_{N-1} + Bu_{N-1}) \right\} \\ &= x_{N-1}^T Q x_{N-1} + x_{N-1}^T (A^T S_N A) x_{N-1} \\ &\quad + \min_{u_{N-1}} \left\{ u_{N-1}^T (R + B^T S_N B) u_{N-1} + 2 x_{N-1}^T A^T S_N B u_{N-1} \right\} \end{aligned}$$

Differentiating with respect to u_{N-1} and setting the derivative equal to zero gives:[2]

$$u_{N-1}^* = L_{N-1} x_{N-1},$$

with $L_{N-1} \triangleq -(R + B^T S_N B)^{-1} B^T S_N A$.

This provides the optimal cost-to-go for stage $N - 1$

$$\begin{aligned} J_{N-1}(x_{N-1}) &= x_{N-1}^T \big(Q + A^T S_N A - A^T S_N B (R + B^T S_N B)^{-1} (R + B^T S_N B)(R + B^T S_N B)^{-1} B^T S_N A \big) x_{N-1} \\ &= x_{N-1}^T S_{N-1} x_{N-1} \end{aligned}$$

The rest can be proved by induction.

Dynamic programming algorithm for LQR

1. Set $S_N := Q_F$

2. For $k = N, N-1 \ldots, 1$, compute the discrete-time Riccati recursion

$$S_{k-1} = A^T S_k A + Q - A^T S_k B (R + B^T S_k B)^{-1} B^T S_k A$$

3. For $k = 0, \cdots, N-1$, compute

$$\begin{aligned} L_k &= -(R + B^T S_{k+1} B)^{-1} B^T S_{k+1} A \\ u_k^* &= L_k x_k \end{aligned}$$

[2] Note that

$$\frac{\partial (u^T X u)}{\partial u} = 2Xu, \qquad \frac{\partial (a^T u)}{\partial u} = a,$$

where a and u are column vectors, and X is a symmetric matrix.

5 Stochastic Systems

We will next turn our attention to stochastic systems of the form

$$x_{k+1} = f_k(x_k, u_k, w_k), \quad k \in \mathbb{N}_0. \tag{7}$$

When comparing to (1), we note the additional "driving noise" term $w_k \in \mathbb{W}$. This is a random stochastic process, which can represent a disturbance, model inaccuracies, etc.

Control policies

A key difference in this stochastic scenario, when compared to the deterministic case, is that we do not optimise over control sequences. Instead, we optimise over admissible control policies (or "laws"). These consist of a sequence of functions

$$\pi = \{\mu_0, \mu_1, \ldots, \mu_{N-1}\},$$

where each

$$\begin{aligned} \mu_k : \mathbb{X} &\to \mathbb{U} \\ x_k &\to u_k = \mu_k(x_k). \end{aligned} \tag{8}$$

Control policies take into account the fact that, in the future, additional state information will be available. Thus, at every time k, the control action u_k is allowed to depend on x_k. For stochastic systems, control policies typically outperform control sequences, see also [1]. For deterministic systems, such as the LQR policy (6), control policies are equivalent to control sequences.

Cost function

Due to the presence of w_k, it turns out that, given an initial state x_0 and admissible policy, the states x_k and inputs $u_k = \mu_k(x_k)$ are random variables with distributions defined through (7). The associated cost incurred is typically a random variable as well. To account for this issue, we will consider the optimal control problem as one of minimising the expected cost:

$$J_{\pi^*}(x_0) = \min_{\pi \in \Pi} J_\pi(x_0), \quad \text{where} \quad J_\pi(x_0) = \underset{w_k}{E}\left\{ g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k) \right\}, \tag{9}$$

where Π is the set of all admissible policies, see (8). As in the deterministic case, the optimal cost depends on the initial state x_0 and we write:

$$J^*(x_0) = \min_{\pi \in \Pi} J_\pi(x_0).$$

Dynamic programming algorithm for stochastic systems

In (9) expectation is taken with respect to the probability distribution of w_k. We shall assume that w_k is characterised by a probability distribution $P_k^w(\cdot|x_k, u_k)$ that may depend on x_k and u_k, but not on previous values $w_{k-1}, \ldots, w_0$. Under this assumption, it turns out that one can apply Bellman's principle of optimality to (9) by considering the expectation of the cost-to-go of the tail subproblems:

$$J_k(x_k) = \min_{\mu_k,\ldots,\mu_{N-1}} \underset{w_k}{E}\left\{ g_N(x_N) + \sum_{m=k}^{N-1} g_m(x_m, \mu_m(x_m), w_m) \right\}.$$

The dynamic programming algorithm then proceeds similar to the deterministic case. The key difference is the use of an expectation in the recursion, cf. (5):

$$J_k(x_k) = \min_{u_k \in \mathbb{U}} \underset{w_k}{E}\Big\{ g_k(x_k, u_k, w_k) + J_{k+1}(f_k(x_k, u_k, w_k)) \Big\}, \quad k = 0, 1, \ldots, N-1. \tag{10}$$

The optimal cost satisfies $J^*(x_0) = J_0(x_0)$. If $u_k^* = \mu_k^*(x_k)$ minimises the right hand side of (10), for each x_k and k, then the policy

$$\pi^* = \{\mu_0^*, \mu_1^*, \ldots, \mu_{N-1}^*\}$$

is optimal.[3]

The computation of the cost-to-go functions provides the optimal policy π^* through the offline minimisations in (10). This policy can then be used online once the state information becomes available. Alternatively, one can also store the cost-to-go functions[4] and perform the optimisation (10) online.

LQR with intermittent communications

As an example, recall linear systems and a quadratic cost function, as seen in Section 4. We here consider a more general model affected by zero-mean driving noises w_k, that are assumed independent with zero-mean and having a finite second moment:

$$x_{k+1} = Ax_k + B_k u_k + w_k.$$

The matrix B_k switches randomly between a given value B and 0:

$$B_k = \begin{cases} B, & \text{with Prob} = \Gamma, \\ 0, & \text{with Prob} = 1 - \Gamma. \end{cases} \tag{11}$$

Such a model can be used to describe communication imperfections (data dropouts), often encountered in wireless control situations [8], see Figure 5.

[3] A proof of these statements is given in [3].

[4] Instead of storing the cost-to-go-functions, one may alternatively store suitable approximations. This will be discussed in later parts of this unit within the context of "approximations in value space."

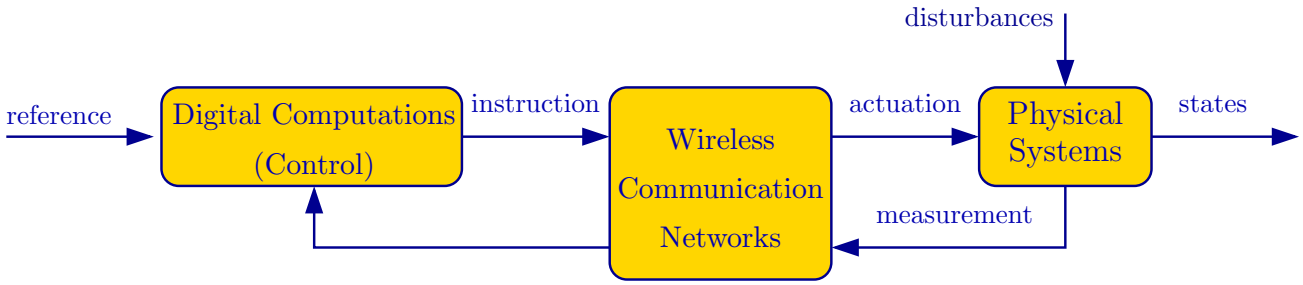

Figure 3: Remote control over a wireless network

The cost function is as in (9), with

$$g_k(x,u,w) = x^T Q x + u^T R u, \quad g_N(x) = x^T Q_F x.$$

By considering the disturbance as (w_k, B_k) we can apply the DP algorithm of (10), which takes the form:

$$J_N(x_N) = x_N^T Q_F x_N$$
$$J_k(x_k) = \min_{u_k} \underset{w_k, B_k}{E} \Big\{ x_k^T Q x_k + u_k^T R u_k + J_{k+1}(A x_k + B_k u_k + w_k) \Big\}, \quad k = 0, 1, \ldots, N-1.$$

As in the deterministic case examined in Section 4, the cost-to-go functions are quadratic and the optimal control policies are linear functions of the current state. This can be established by induction, similar to the deterministic case:

$$\begin{aligned}
J_{N-1}(x_{N-1}) &= \min_{u_{N-1}} \underset{w_{N-1}, B_{N-1}}{E} \Big\{ x_{N-1}^T Q x_{N-1} + u_{N-1}^T R u_{N-1} \\
&\qquad + (A x_{N-1} + B_{N-1} u_{N-1} + w_{N-1})^T Q_F (A x_{N-1} + B_{N-1} u_{N-1} + w_{N-1}) \Big\} \\
&= x_{N-1}^T Q x_{N-1} + \min_{u_{N-1}} \Big\{ u_{N-1}^T R u_{N-1} \\
&\qquad + \underset{w_{N-1}, B_{N-1}}{E} \Big\{ (A x_{N-1} + B_{N-1} u_{N-1} + w_{N-1})^T Q_F (A x_{N-1} + B_{N-1} u_{N-1} + w_{N-1}) \Big\} \Big\} \\
&= x_{N-1}^T (Q + A^T Q_F A) x_{N-1} + \underset{w_{N-1}}{E} \{ w_{N-1}^T Q_F w_{N-1} \} \\
&\quad + \min_{u_{N-1}} \Big\{ u_{N-1}^T R u_{N-1} + \underset{B_{N-1}}{E} \{ u_{N-1}^T B_{N-1}^T Q_F B_{N-1} u_{N-1} + 2 x_{N-1}^T A^T Q_F B_{N-1} u_{N-1} \} \Big\}, \\
&= x_{N-1}^T (Q + A^T Q_F A) x_{N-1} + \underset{w_{N-1}}{E} \{ w_{N-1}^T Q_F w_{N-1} \} \\
&\quad + \min_{u_{N-1}} \Big\{ u_{N-1}^T R u_{N-1} + \Gamma u_{N-1}^T B^T Q_F B u_{N-1} + 2\Gamma x_{N-1}^T A^T Q_F B u_{N-1} \Big\},
\end{aligned}$$

where we have used (11) and the fact that w_k is zero-mean. Upon calculating all cost-to-go functions, we obtain the optimal control policy

$$\mu_k^*(x_k) = L_k x_k,$$

where

$$L_k = -(R + \Gamma B^T S_{k+1} B)^{-1} \Gamma B^T S_{k+1} A.$$

The matrices S_k follow the recursion:

$$S_{k-1} = A^T S_k A + Q - A^T S_k \Gamma B (R + \Gamma B^T S_k B)^{-1} \Gamma B^T S_k A$$

starting from $S_N = Q_F$.

6 Markov Decision Processes

As an alternative to the state-space model in (7), some of the literature uses Markov Decision Process (MDP) models, see, e.g., [11, 6, 9]. Similar to (7), MDPs are characterised by a cost function, state and action spaces, $\mathbb{X}$ and $\mathbb{U}$. The key difference to (7) lies in that, instead of using a stochastic recursion, the state at the next step is described through a probability function.

Discrete-space models

MDP models are especially useful, in scenarios where the state space $\mathbb{X}$, the action space $\mathbb{U}$ and the noise space $\mathbb{W}$ are discrete (i.e., are finite, or have an infinitely countable number of elements, such as the integers). The state at the next time step is characterised by a transition probability.

For discrete-space systems this transition probability can be written as:

$$p_{ij}(u,k) = \text{Prob}(x_{k+1} = j | x_k = i, u_k = u), \quad i,j \in \mathbb{X},\ u_k \in \mathbb{U}.$$

It is easy to see that this model can be rewritten in the form (7), by setting:

$$x_{k+1} = w_k,$$

where the noise distribution satisfies $P_k^w(w_k = j | x_k = i, u_k = u) = p_{ij}(u,k)$. Conversely, a model of the form (7), can be rewritten using state transition probabilities, upon noting that

$$p_{ij}(u,k) = \text{Prob}(\mathbb{W}_k(i,u,j) | x_k = i, u_k = u),$$

where the set $\mathbb{W}_k(i,u,j) = \{w \in \mathbb{W} \,|\, j = f_k(i,u,w)\}$.

In terms of transition probabilities, and if we denote by $g_k(i,u)$ the expected cost per stage at state $x_k = i$ when the control u is applied at time k, the DP algorithm recursions (10) reduce to:

$$J_k(i) = \min_{u \in \mathbb{U}} \Big\{ g_k(i,u) + \sum_j p_{ij}(u,k) J_{k+1}(j) \Big\}, \quad k = 0, 1, \ldots, N-1. \tag{12}$$

Example: Controlling the rover

Consider an autonomous rover on planet Pluto. The rover is equipped with PV panels, but sunshine on Pluto is scarce. Thus, the rover needs to drive up a hill in order to harvest energy. Unfortunately, the rover is clumsy and has a tendency to roll off the hill. Following as in [7, Section 7.7], we model this by abstracting the Rover states to belong to the finite set $\mathbb{X} = \{\mathrm{T}, \mathrm{R}, \mathrm{B}\}$, referring to the positions "Top", "Rolling" and "Bottom," respectively. Our goal is to design a control policy for the driving actions. The control constraint set is chosen as $\mathbb{U} = \{0, 1\}$, where 0 refers to "not driving" and 1 denotes "driving."

The state dynamics is described as follows (taken directly from [7, p.264]):

> (1) Rover on top. If driving, then it is still at the top of the hill in the next period with probability 0.8, and rolling down in the next period with probability 0.2. If not driving, these probabilities are 0.75 and 0.25, respectively.
> (2) Rolling. If driving, then with probability 0.9 it is at the top of the hill in the next period, and with probability 0.1 it moves to the bottom.
> If not driving, then with probability 1 it is at the bottom in the next period.
> (3) Bottom. If driving, then it remains at the bottom with probability 0.9, and is rolling in the next period with probability 0.1. If not driving, it remains on the bottom with probability 1.

The above description leads to the transition probabilities in Table 1 below.

$i\backslash j$	T	R	B
T	0.75	0.25	0
R	0	0	1
B	0	0	1

$i\backslash j$	T	R	B
T	0.8	0.2	0
R	0.9	0	0.1
B	0	0.1	0.9

Table 1: Transition probabilities, left: $p_{ij}(0, k)$, right: $p_{ij}(1, k)$

To formulate the cost function, we assume that at the top of the hill, the rover is able to harvest $h = 3$ units of energy. When in the other two states, it cannot harvest energy at all. Driving uses $e = 2$ units in all three states. We model this by the costs[5]

$$g_k(\mathrm{T}, 0) = -h = -3,\ g_k(\mathrm{T}, 1) = -h + e = -1, \quad g_k(\mathrm{R}, 0) = g_k(\mathrm{B}, 0) = 0, \quad g_k(\mathrm{R}, 1) = g_k(\mathrm{B}, 1) = e = 2.$$

We choose $N = 3$ and set $g_3(i) = 0$ for all states.

Use of the DP algorithm in (12) then allows us to calculate the optimal control policy through simple, but somewhat tedious, calculations as follows:

$$J_3(\mathrm{T}) = J_3(\mathrm{R}) = J_3(\mathrm{B}) = 0.$$

[5]Rewards would be positive.

At stage $k = 2$, we have

$$J_2(\mathrm{T}) = \min_{u\in\{0,1\}} \big\{g_2(\mathrm{T},u)\big\} = \min\{-3,-1\} = -3, \quad \Longrightarrow \quad \mu_2(\mathrm{T}) = 0.$$

$$J_2(\mathrm{R}) = \min_{u\in\{0,1\}} \big\{g_2(\mathrm{R},u)\big\} = \min\{0,2\} = 0 = J_2(\mathrm{B}), \quad \Longrightarrow \quad \mu_2(\mathrm{R}) = 0, \quad \mu_2(\mathrm{B}) = 0.$$

For time $k = 1$, we calculate

$$\begin{aligned} J_1(\mathrm{T}) &= \min_{u\in\{0,1\}} \left\{ g_1(\mathrm{T},u) + \sum_{j\in\{\mathrm{T,R,B}\}} p_{\mathrm{T}j}(u,k) J_2(j) \right\} \\ &= \min\left\{ \left(g_1(\mathrm{T},0) + \sum_{j\in\{\mathrm{T,R,B}\}} p_{\mathrm{T}j}(0,k) J_2(j) \right), \left(g_1(\mathrm{T},1) + \sum_{j\in\{\mathrm{T,R,B}\}} p_{\mathrm{T}j}(1,k) J_2(j) \right) \right\} \\ &= \min\big\{ \big(-3 + 0.75\cdot(-3) + 0.25\cdot 0 + 0\cdot 0\big), \big(-1 + 0.8\cdot(-3) + 0.2\cdot 0 + 0\cdot 0\big) \big\} \\ &= \min\{-5.25, -3.4\} = -5.25 \quad \Longrightarrow \quad \mu_1(\mathrm{T}) = 0. \end{aligned}$$

$$\begin{aligned} J_1(\mathrm{R}) &= \min\left\{ \left(g_1(\mathrm{R},0) + \sum_{j\in\{\mathrm{T,R,B}\}} p_{\mathrm{R}j}(0,k) J_2(j) \right), \left(g_1(\mathrm{R},1) + \sum_{j\in\{\mathrm{T,R,B}\}} p_{\mathrm{R}j}(1,k) J_2(j) \right) \right\} \\ &= \min\big\{ \big(0 + 0\cdot(-3) + 0\cdot 0 + 1\cdot 0\big), \big(2 + 0.9\cdot(-3) + 0\cdot 0 + (0.1)\cdot 0\big) \big\} \\ &= \min\{0, -0.7\} = -0.7 \quad \Longrightarrow \quad \mu_1(\mathrm{R}) = 1. \end{aligned}$$

$$\begin{aligned} J_1(\mathrm{B}) &= \min\left\{ \left(g_1(\mathrm{B},0) + \sum_{j\in\{\mathrm{T,R,B}\}} p_{\mathrm{B}j}(0,k) J_2(j) \right), \left(g_1(\mathrm{B},1) + \sum_{j\in\{\mathrm{T,R,B}\}} p_{\mathrm{B}j}(1,k) J_2(j) \right) \right\} \\ &= \min\big\{ \big(0 + 0\cdot(-3)\big), \big(2 + 0\cdot(-3)\big) \big\} = \min\{0,2\} = 0 \quad \Longrightarrow \quad \mu_1(\mathrm{B}) = 0. \end{aligned}$$

At the start, $k = 0$, we obtain:

$$\begin{aligned} J_0(\mathrm{T}) &= \min\left\{ \left(g_0(\mathrm{T},0) + \sum_{j\in\{\mathrm{T,R,B}\}} p_{\mathrm{T}j}(0,k) J_1(j) \right), \left(g_0(\mathrm{T},1) + \sum_{j\in\{\mathrm{T,R,B}\}} p_{\mathrm{T}j}(1,k) J_1(j) \right) \right\} \\ &= \min\big\{ \big(-3 + 0.75\cdot(-5.25) + 0.25\cdot(-0.7) + 0\cdot 0\big), \big(-1 + 0.8\cdot(-5.25) + 0.2\cdot(-0.7) + 0\cdot 0\big) \big\} \\ &= \min\{-7.1125, -5.34\} = -7.1125 \quad \Longrightarrow \quad \mu_0(\mathrm{T}) = 0. \end{aligned}$$

$$\begin{aligned} J_0(\mathrm{R}) &= \min\left\{ \left(g_0(\mathrm{R},0) + \sum_{j\in\{\mathrm{T,R,B}\}} p_{\mathrm{R}j}(0,k) J_1(j) \right), \left(g_0(\mathrm{R},1) + \sum_{j\in\{\mathrm{T,R,B}\}} p_{\mathrm{R}j}(1,k) J_1(j) \right) \right\} \\ &= \min\big\{ \big(0 + 0\cdot(-5.25) + 0\cdot(-0.7) + 1\cdot 0\big), \big(2 + 0.9\cdot(-5.25) + 0\cdot(-0.7) + 0.1\cdot 0\big) \big\} \\ &= \min\{0, -2.725\} = -2.725 \quad \Longrightarrow \quad \mu_0(\mathrm{R}) = 1. \end{aligned}$$

$$
\begin{aligned}
J_0(\mathrm{B}) &= \min\left\{\left(g_0(\mathrm{B},0)+\sum_{j\in\{\mathrm{T,R,B}\}} p_{\mathrm{B}j}(0,k)J_1(j)\right),\left(g_0(\mathrm{B},1)+\sum_{j\in\{\mathrm{T,R,B}\}} p_{\mathrm{B}j}(1,k)J_1(j)\right)\right\}\\
&= \min\left\{\big(0+0\cdot(-5.25)+0\cdot(-0.7)+1\cdot 0\big),\big(2+0\cdot(-5.25)+0.1\cdot(-0.7)+0.9\cdot 0\big)\right\}\\
&= \min\{0,1.93\}=0 \quad\Longrightarrow\quad \mu_0(\mathrm{B})=0.
\end{aligned}
$$

The optimal policy is displayed in Table 2. From the calculations, it is clear that the designed behaviour of the rover depends on the system problem parameters and on the chosen horizon length. How do you think the policy will be influenced if a longer horizon were chosen? How would the policies change if the energy used for driving were lower? As an extension you may want to examine how to incorporate energy storage and also scenarios where the harvested energy is random.[6]

	$\mu_0(\cdot)$	$\mu_1(\cdot)$	$\mu_2(\cdot)$
T	0	0	0
R	1	1	0
B	0	0	0

Table 2: The optimal policy for the Rover problem

Continuous-space models (optional)

MDPs can also be formulated for continuous-space problems, e.g., where the states, control and disturbances are vectors of real variables. The transition probabilities, say $\mathcal{Q}_k$, then capture the dynamics by focusing on sets $\mathbb{B}\subset\mathbb{X}$:

$$
\mathcal{Q}_k(\mathbb{B}|x,u)=\mathrm{Prob}(x_{k+1}\in\mathbb{B}|x_k=x,u_k=u),\quad \mathbb{B}\subset\mathbb{X}. \tag{13}
$$

Note that $x_{k+1}=f_k(x_k,u_k,w_k)$ with w_k characterised by its conditional probability distribution $P_k^w(\cdot|x_k,u_k)$ belongs to this model class. In fact, for any set $\mathbb{B}\subset\mathbb{X}$, we have that

$$
\begin{aligned}
\mathcal{Q}_k(\mathbb{B}|x,u) &= \mathrm{Prob}(f_k(x_k,u_k,w_k)\in\mathbb{B}|x_k=x,u_k=u)=\mathrm{Prob}(f_k(x,u,w_k)\in\mathbb{B})\\
&= \mathrm{Prob}\big(w_k\in\{w\in\mathbb{W}\text{ such that }f_k(x,u,w)\in\mathbb{B}\}\big)=\underset{w_k}{E}\big\{\mathcal{I}_{\mathbb{B}}\big(f_k(x,u,w)\big)\big\},
\end{aligned} \tag{14}
$$

where $\mathcal{I}_{\mathbb{B}}(\cdot)$ is the indicator function of the set $\mathbb{B}$.[7] If $\mathbb{B}$ contains only a single value, then Dirac-delta functions can be used in (14).

[6] A more challenging extension to this problem amounts to designing the rover control law assuming that the rover position is only available when it is located at the top of the hill. This leads to a partially observed control problem, beyond the scope of this unit. For further information, see [3, Chapter 5] or also Section 7.7 of [7].

[7] $\mathcal{I}_{\mathbb{B}}(x)=1$, if $x\in\mathbb{B}$, and $\mathcal{I}_{\mathbb{B}}(x)=0$ otherwise.

Mid-unit reflection

So far, we have investigated methods for formulating and solving optimal control problems over a finite horizon. We have allowed models to be nonlinear and possibly stochastic. Throughout the remainder of this unit we will build on the concepts presented so far and extend our formulation to infinite horizon problems. We will also elucidate optimal control design methods for scenarios where system models are unavailable and thus optimal control design involves learning.

References

[1] Y. Bar-Shalom and E. Tse, "Dual effect, certainty equivalence, and separation in stochastic control," *IEEE Transactions on Automatic Control*, vol. 19, pp. 494– 500, Oct. 1974.

[2] R. E. Bellman, *Dynamic Programming*. Princeton University Press, 1957.

[3] D. P. Bertsekas, *Dynamic Programming and Optimal Control, Vol.1*. Belmont, MA: Athena Scientific, 2005.

[4] D. P. Bertsekas, *Reinforcement Learning and Optimal Control*. Athena Scientific, 2019.

[5] D. P. Bertsekas, *Rollout, Policy Iteration, and Distributed Reinforcement Learning*. Athena Scientific, 2020.

[6] O. Hérnandez-Lerma and J. B. Laserre, *Discrete-Time Markov Control Processes*. New York, N.Y.: Springer-Verlag, 1996.

[7] S. Meyn, *Control Systems and Reinforcement Learning*. Cambridge University Press, 2022.

[8] E. G. W. Peters, D. E. Quevedo, and M. Fu, "Controller and scheduler codesign for feedback control over IEEE 802.15.4 networks," *IEEE Trans. Contr. Syst. Technol.*, vol. 24, pp. 2016–2030, Nov. 2016.

[9] M. L. Puterman, *Markov Decision Processes*. Hoboken, N.J.: Wiley-Interscience, 1994.

[10] J. B. Rawlings and D. Q. Mayne, *Model Predictive Control: Theory and Design*. Madison, WI: Nob Hill Publishing, 2009.

[11] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2nd ed., 2018.