

Chapter 6

Linear Quadratic Regulator (LQR)

Reading

1. <http://underactuated.csail.mit.edu/lqr.html>, Lecture 3-4 at <https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-323-principles-of-optimal-control-spring-2008/lecture-notes>
2. Optional: Applied Optimal Control by Bryson & Ho, Chapter 4-5

This chapter is the analogue of Chapter 3 on Kalman filtering. Just like Chapter 2, the previous chapter gave us two algorithms, namely value iteration and policy iteration, to solve dynamic programming problems for a finite number of states and a finite number of controls. Solving dynamic programming problems is difficult if the state/control space are infinite. In this chapter, we will look at an important and powerful special case, called the Linear Quadratic Regulator (LQR), when we can solve dynamic programming problems easily. Just like a lot of real-world state-estimation problems can be solved using the Kalman filter and its variants, a lot of real-world control problems can be solved using LQR and its variants.

6.1 Discrete-time LQR

Consider a deterministic, *linear* dynamical system given by

$$x_{k+1} = Ax_k + Bu_k; \quad x_0 \text{ is given.}$$

where $x_k \in \mathbb{R}^d$ and $u_k \in \mathbb{R}^m$ which implies that $A \in \mathbb{R}^{d \times d}$ and $B \in \mathbb{R}^{d \times m}$. In this chapter, we are interested in calculating a feedback control $u_k = u(x_k)$ for such a system. Just like we formulated the problem in dynamic programming, we want to pick a feedback control which leads to a trajectory

20 that achieves a minimum of some run-time cost and a terminal cost. We will
 21 assume that both the run-time and terminal costs are *quadratic* in the state and
 22 control input, i.e.,

$$q(x, u) = \frac{1}{2}x^\top Qx + \frac{1}{2}u^\top Ru \quad (6.1)$$

23 where $Q \in \mathbb{R}^{d \times d}$ and $R \in \mathbb{R}^{m \times m}$ are symmetric, positive semi-definite
 24 matrices

$$Q = Q^\top \succeq 0, \quad R = R^\top \succeq 0.$$

25 Effectively, if Q were a diagonal matrix, a large diagonal entry would Q_{ii}
 26 models our desire that the trajectory of the system should not have a large value
 27 of the state x_i along its trajectories. We want these matrices to be positive
 28 semi-definitive to prevent dynamic programming from picking a trajectory
 29 which drives down the run-time cost to negative infinity by picking.

30 **Example** Consider the discrete-time equivalent of the so-called double inte-
 31 grator $\ddot{z}(t) = u(t)$. The linear system in this case (obtained by creating two
 32 states $x := [z(t), \dot{z}(t)]$ is

$$x_{k+1} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} 0 \\ \Delta t \end{bmatrix} u_k.$$

33
 34 First, note that a continuous-time linear dynamical system $\dot{x} = Ax$ is
 35 asymptotically stable, i.e., from any initial condition $x(0)$ its trajectories go
 36 to the equilibrium point $x = 0$ ($x(t) \rightarrow 0$ as $t \rightarrow \infty$). Asymptotic stability
 37 occurs if all eigenvalues of A are strictly negative. A discrete-time linear
 38 dynamical system $x_{k+1} = Ax_k$ is asymptotically stable if all eigenvalues of
 39 A have magnitude strictly smaller than 1, $|\lambda(A)| < 1$.

A typical trajectory of the double integrator will look as follows. Suppose

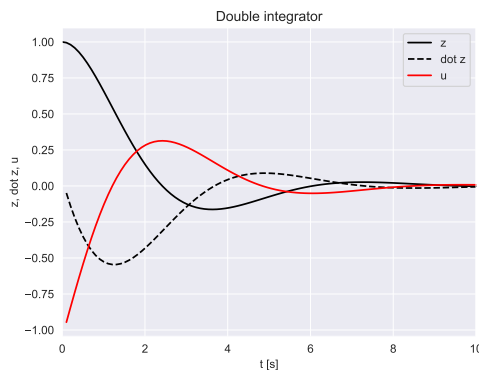


Figure 6.1: The trajectory of $z(t)$ as a function of time t for a double integrator $\ddot{z}(t) = u$ where we have chosen a stabilizing (i.e., one that makes the system asymptotically stable) controller $u = -z(t) - \dot{z}(t)$. Notice how the trajectory starts from some initial condition (in this case $z(0) = 1$ and $\dot{z}(0) = 0$) and moves towards its equilibrium point $z = \dot{z} = 0$.

i This system is called the double integrator because of the structure $\ddot{z} = u$; if z denotes the position of an object the equation is simply Newton's law which connects the force applied u to the acceleration.

40 we would like to pick a different controller that more quickly brings the system
 41 to its equilibrium. One way of doing so is to minimize
 42

$$J = \sum_{k=0}^T \|x_k\|^2$$

43 which represents how far away both the position and velocity are from zero
 44 over all times k . The following figure shows the trajectory that achieves a
 45 small value of J .

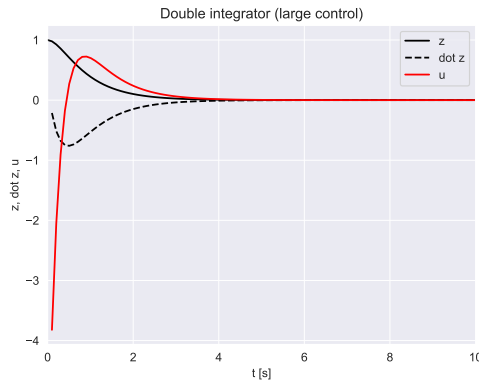


Figure 6.2: The trajectory of $z(t)$ as a function of time t for a double integrator $\ddot{z}(t) = u$ where we have chosen a large stabilizing control at each time $u = -5z(t) - 5\dot{z}(t)$. Notice how quickly the state trajectory converges to the equilibrium without much oscillation as compared to Figure 6.1 but how large the control input is at certain times.

46 This is obviously undesirable for real systems where we may want the
 47 control input to be bounded between some reasonable values (a car cannot
 48 accelerate by more than a certain threshold). A natural way of enforcing this
 49 is to modify our our desired cost of the trajectory to be

$$J = \sum_{k=0}^T (\|x_k\|^2 + \rho \|u_k\|^2)$$

50 where the value of the parameter ρ is something chosen by the user to give
 51 a good balance of how quickly the trajectory reaches the equilibrium point
 52 and how much control is exerted while doing so. Linear-Quadratic-Regulator
 53 (LQR) is a generalization of this idea, notice that the above example is equiva-
 54 lent to setting $Q = I_{d \times d}$ and $R = \rho I_{m \times m}$ for the run-time cost in (6.1).

55 **Back to LQR** With this background, we are now ready to formulate the
 56 Linear-Quadratic-Regulator (LQR) problem which is simply dynamic pro-
 57 gramming for a linear dynamical system with quadratic run-time cost. In order
 58 to enable the system to reach the equilibrium state even if we have only a finite
 59 time-horizon, we also include a quadratic cost

$$q_f(x) = \frac{1}{2} x^\top Q_f x. \quad (6.2)$$

60 The dynamic programming problem is now formulated as follows.

Finite time-horizon LQR problem Find a sequence of control inputs $(u_0, u_1, \dots, u_{T-1})$ such that the function

$$J(x_0; u_0, u_1, \dots, u_{T-1}) = \frac{1}{2} x_T^\top Q_f x_T + \frac{1}{2} \sum_{k=0}^{T-1} (x_k^\top Q x_k + u_k^\top R u_k) \quad (6.3)$$

is minimized under the constraint that $x_{k+1} = Ax_k + Bu_k$ for all times $k = 0, \dots, T-1$ and x_0 is given.

61 6.1.1 Solution of the discrete-time LQR problem

62 We know the principle of dynamic programming and can apply it to solve the
63 LQR problem. As usual, we will compute the cost-to-go of a trajectory that
64 starts at some state x and goes further by $T - k$ time-steps, $J_k(x)$ backwards.

65 Set

$$J_T^*(x) = \frac{1}{2} x^\top Q_f x \quad \text{for all } x.$$

66 Using the principle of dynamic programming, the cost-to-go J_{T-1} is given by

$$\begin{aligned} J_{T-1}^*(x_{T-1}) &= \min_u \left\{ \frac{1}{2} (x_{T-1}^\top Q x_{T-1} + u^\top R u) + J_T^*(Ax_{T-1} + Bu) \right\} \\ &= \min_u \left\{ \frac{1}{2} (x_{T-1}^\top Q x_{T-1} + u^\top R u + (Ax_{T-1} + Bu)^\top Q_f (Ax_{T-1} + Bu)) \right\}. \end{aligned}$$

67 We can now take the derivative of the right-hand side with respect to u to get

$$\begin{aligned} 0 &= \frac{d\text{RHS}}{du} \\ &= \{Ru + B^\top Q_f (Ax_{T-1} + Bu)\} \\ \Rightarrow u_{T-1}^* &= -(R + B^\top Q_f B)^{-1} B^\top Q_f A x_{T-1} \\ &\equiv -K_{T-1} x_{T-1}. \end{aligned} \quad (6.4)$$

68 where

$$K_{T-1} = (R + B^\top Q_f B)^{-1} B^\top Q_f A$$

69 is (surprisingly) also called the Kalman gain. The second derivative is positive
70 semi-definite

$$\frac{d^2\text{RHS}}{du^2} = R + B^\top Q_f B \succeq 0$$

71 so we know that u_{T-1}^* is a minimum of the convex quantity on the right-hand
72 side. Notice that the optimal control u_{T-1}^* is a linear function of the state
73 x_{T-1} . Let us now expand the cost-to-go J_{T-1} using this optimal value (the
74 subscript $T-1$ on the curly bracket simply means that all quantities are at

75 time $T - 1$)

$$\begin{aligned}
 J_{T-1}^*(x_{T-1}) &= \frac{1}{2} \left\{ x^\top Q x + u^{*\top} R u^* + (Ax + Bu^*)^\top Q_f (Ax + Bu^*) \right\}_{T-1} \\
 &= \frac{1}{2} x_{T-1}^\top \left\{ Q + K^\top R K + (A - BK)^\top Q_f (A - BK) \right\}_{T-1} x_{T-1} \\
 &\equiv \frac{1}{2} x_{T-1}^\top P_{T-1} x_{T-1}
 \end{aligned}$$

76 where we set the stuff inside the curly brackets to the matrix P which is also
 77 positive semi-definite. This is great, the cost-to-go is also a quadratic function
 78 of the state x_{T-1} . Let us assume that this pattern holds for all time steps
 79 and the cost-to-go of the optimal LQR trajectory starting from a state x and
 80 proceeding forwards for $T - k$ time-steps is

$$J_k^*(x) = \frac{1}{2} x^\top P_k x.$$

81 We can now repeat the same exercise to get a recursive formula for P_k in terms
 82 of P_{k+1} . This is the *solution* of dynamic programming for the LQR problem
 83 as looks as follows.

$$\begin{aligned}
 P_T &= Q_f \\
 K_k &= (R + B^\top P_{k+1} B)^{-1} B^\top P_{k+1} A \\
 P_k &= Q + K_k^\top R K_k + (A - BK_k)^\top P_{k+1} (A - BK_k),
 \end{aligned} \tag{6.5}$$

84 for $k = T - 1, T - 2, \dots, 0$. There are a number of important observations to
 85 be made from this calculation:

- 86 1. The optimal controller $u_k^* = -K_k x_k$ is a linear function of the state
 87 x_k . This is only true for linear dynamical systems with quadratic costs.
 88 Notice that both the state and control space are infinite sets but we have
 89 managed to solve the dynamic programming problem to get the optimal
 90 controller. We could not have done it if the run-time/terminal costs were
 91 not quadratic or if the dynamical system were not linear. Can you say
 92 why?
- 93 2. The cost-to-go matrix P_k and the Kalman gain K_k do not depend upon
 94 the state and can be computed ahead of time if we know what the time
 95 horizon T is going to be.
- 96 3. The Kalman gain changes with time k . Effectively, the LQR controller
 97 picks a large control input to quickly reduce the run-time cost at the
 98 beginning (if the initial condition were such that the run-time cost of
 99 the trajectory would be very large) and then gets into a balancing act
 100 where it balances the control effort and the state-dependent part of the
 101 run-time cost. LQR is an optimal way to strike a balance between the
 102 two examples in Figure 6.1 and Figure 6.2.

103 The careful reader will notice how the equations in (6.5) and our remarks
 104 about them are similar to the update equations of the Kalman filter and our
 105 remarks there. In fact we will see shortly how spookily similar the two are.
 106 The key difference is that Kalman filter updates run forwards in time and

107 update the covariance while LQR updates run backwards in time and update
 108 the cost-to-go matrix P . This is not surprising because LQR is an optimal
 control problem, its update equations run backward in time.

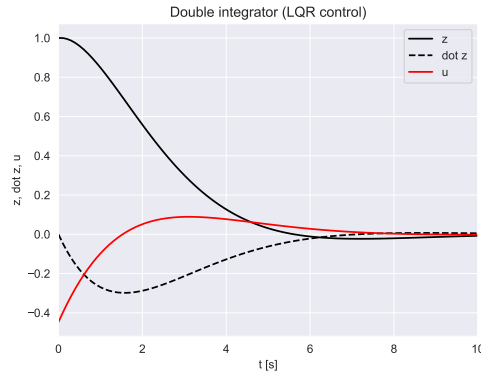


Figure 6.3: The trajectory of $z(t)$ as a function of time t for a double integrator $\ddot{z}(t) = u$ where we have chosen a controller obtained from LQR with $Q = I$ and $R = 5$. This gives the controller to be about $u = -0.45z(t) - 1.05\dot{z}(t)$. Notice how we still get stabilization but the control acts more gradually. Using different values of R , we can get many different behaviors. Another key aspect of LQR as compared to Figure 6.1 where the control was chosen in an ad hoc fashion is to let us prescribe the quality of state trajectories using high-level quantities like Q, R .

109

110 6.2 Hamilton-Jacobi-Bellman equation

111 This section will show how the principle of dynamic programming looks for
 112 continuous-time deterministic dynamical systems

$$\dot{x} = f(x, u), \quad \text{with } x(0) = x_0.$$

113 As we discussed in Chapter 3, we can think of this as the limit of discrete-time
 114 dynamical system $x_{k+1} = f^{\text{discrete}}(x_k, u_k)$ as the time discretization goes to
 115 zero. Just like we have a sequence of controls in the discrete-time case, we
 116 have a continuous curve that determines the control (let us also call it the
 117 control sequence)

$$\{u(t) : t \in \mathbb{R}_+\}$$

118 which gives rise to a trajectory of the states

$$\{x(t) : t \in \mathbb{R}_+\}$$

119 for the dynamical system. Let us consider the case when we want to find
 120 control sequences that minimize the integral of the cost along the trajectory
 121 that stops at some fixed, finite time-horizon T :

$$q_f(x(T)) + \int_0^T q(x(t), u(t)) dt.$$

122 This cost is again a function of the run-time cost and a terminal cost.

i If you are trying this example yourself, I used the formula for continuous-time LQR and then discretized the controller while implementing it. We will see this in Section 6.2

i Since $\{x(t)\}_{t \geq 0}$ and $\{u(t)\}_{t \geq 0}$ are continuous curves and the cost is now a function of a continuous-curve, mathematicians say that the cost is a “functional” of the state and control trajectory.

Continuous-time optimal control problem We again want to solve for

$$J^*(x_0) = \min_{u(t), t \in [0, T]} \left\{ q_f(x(T)) + \int_0^T q(x(t), u(t)) dt \right\} \quad (6.6)$$

with the system satisfying $\dot{x} = f(x, u)$ at each time instant. Notice that the minimization is over a function of time $\{u(t) : t \in [0, T]\}$ as opposed to a discrete-time sequence of controls that we had in the discrete-time case. We will next look at the Hamilton-Jacobi-Bellman equation which is a method to solve optimal-control problems of this kind.

123 The principle of dynamic programming principle is still valid: if we have
 124 an optimal control trajectory $\{u^*(t) : t \in [0, T]\}$ we can chop it up into two
 125 parts at some intermediate time $t \in [0, T]$ and claim that the tail is optimal.
 126 In preparation for this, let us define the cost-to-go of going forward by $T - t$
 127 time as

$$J^*(x, t) = \min_{u(s), s \in [t, T]} \left\{ q_f(x(T)) + \int_t^T q(x(s), u(s)) ds \right\},$$

the cost incurred if the trajectory starts at state x and goes forward by $T - t$ time. This is very similar to the cost-to-go $J_k^*(x)$ we had in discrete-time dynamic programming. Dynamic programming now gives

$$\begin{aligned} J^*(x(t), t) &= \min_{u(s), t \leq s \leq T} \left\{ q_f(x(T)) + \int_t^T q(x(s), u(s)) ds \right\} \\ &= \min_{u(s), t \leq s \leq T} \left\{ q_f(x(T)) + \int_t^{t+\Delta t} q(x(s), u(s)) ds + \int_{t+\Delta t}^T q(x(s), u(s)) ds \right\} \\ &= \min_{u(s), t \leq s \leq t+\Delta t} \left\{ J^*(x(t+\Delta t), t+\Delta t) + \int_t^{t+\Delta t} q(x(s), u(s)) ds \right\}. \end{aligned}$$

128 We now take the Taylor approximation of the term $J^*(x(t+\Delta t), t+\Delta t)$ as
 129 follows

$$\begin{aligned} &J^*(x(t+\Delta t), t+\Delta t) - J^*(x(t), t) \\ &\approx \partial_x J^*(x(t), t) (x(t+\Delta t) - x(t)) + \partial_t J^*(x(t), t) \Delta t \\ &\approx \partial_x J^*(x(t), t) f(x(t), u(t)) \Delta t + \partial_t J^*(x(t), t) \Delta t \end{aligned}$$

130 where $\partial_x J^*$ and $\partial_t J^*$ denote the derivative of J^* with respect to its first and
 131 second argument respectively. We substitute this into the minimization and
 132 collect terms of Δt to get

$$0 = \partial_t J^*(x(t), t) + \min_{u(t) \in U} \left\{ q(x(t), u(t)) + f(x(t), u(t)) \partial_x J^*(x(t), t) \right\}. \quad (6.7)$$

133 Notice that the minimization in (6.7) is only over *one* control input $u(t) \in U$,
 134 this is the control that we should take at time t . (6.7) is called the Hamilton-

135 Jacobi-Bellman (HJB) equation. Just like the Bellman equation

$$J_k^*(x) = \min_{u \in U} \{q_k(x, u) + J_{k+1}^*(f(x, u))\}.$$

136 has two quantities x and the time k , the Hamilton-Jacobi-Bellman equation
 137 also has two quantities x and continuous time t . Just like the Bellman equation
 138 is solved backwards in time starting from T with $J_k^*(x) = q_f(x)$, the HJB
 139 equation is solved backwards in time by setting

$$J^*(x, T) = q_f(x).$$

You should think of the HJB equation as the continuous-time, continuous-space analogue of Dijkstra's algorithm when the number of nodes in the graph goes to infinity and the length of each edge is also infinitesimally small.

140 6.2.1 Infinite-horizon HJB

141 The infinite-horizon problem with the HJB equation is easy: since we know
 142 that the optimal cost-to-go is not a function of time, we have

$$\partial_t J^*(x, t) = 0$$

143 and therefore $J^*(x)$ satisfies

$$0 = \min_{u \in U} \{q(x, u) + f(x, u) \partial_x J^*(x)\}. \quad (6.8)$$

144 In this case, the above equation makes sense only if the integral of the run-time
 145 cost with the optimal controller $\int_0^\infty q(x(t), u^*(x(t))) dt$ remains bounded and
 146 does not diverge to infinity. Therefore typically in this problem we will set
 147 $q(0, 0) = 0$, i.e., there is no cost for the system being at the origin with zero
 148 control, otherwise the integral of the run-time cost will never be finite. This
 149 also gives the boundary condition $J^*(0) = 0$ for the HJB equation.

150 6.2.2 Solving the HJB equation

151 The HJB equation is a partial differential equation (PDE) because there is one
 152 cost-to-go from every state $x \in X$ and for every time $t \in [0, T]$. It belongs
 153 to a large and important class of PDEs, collectively known as Hamilton-
 154 Jacobi-type equations. As you can imagine, since dynamic programming is
 155 so pervasive and solutions of DP are very useful in practice for a number of
 156 problems, there have been many tools invented to solve the HJB equation.
 157 These tools have applications to a wide variety of problems, from under-
 158 standing how sound travels in crowded rooms to how light diffuses in an
 159 animated movie scene, to even obtaining better algorithms to train deep net-
 160 works (<https://arxiv.org/abs/1704.04932>). HJB equations are usually never
 161 exactly solvable and a number of approximations need to be made in order to
 162 solve it.

In this course, we will not solve the HJB equation. Rather, we are interested in seeing how the HJB equation looks for continuous-time linear dynamical systems (both deterministic and stochastic ones) and LQR problems for such systems, as done in the following section.

163 **An example** We will look at a classical example of the so-called car-on-the-hill problem given below. The state of the problem is the position and

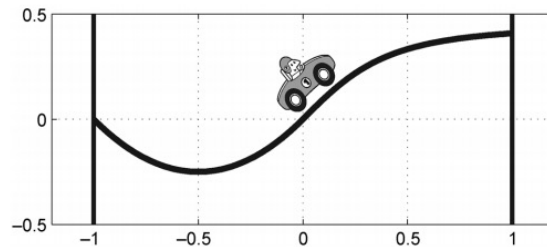


Figure 6.4: A car whose position is given by $z(t)$ would like to climb the hill to its right and reach the top with minimal velocity. The car rolls on the hill without friction. The run-time reward is zero everywhere inside the state-space. Terminal reward is -1 for hitting the left boundary ($z = -1$) and $1 - \dot{z}/2$ for reaching the right boundary ($z = 1$). The car is a single integrator, i.e., $\dot{z} = u$ with only two controls ($u = 4$ and $u = -4$) and cannot exceed a given velocity (in this case $|\dot{z}| \leq 4$). This looks like a simple dynamic programming problem but it is quite hard due to the constraint on the velocity. The car may need to make multiple swing ups before it gains enough velocity (but not too much) to climb up the hill.

164
165 velocity (z, \dot{z}) and we can solve a two-dimensional HJB equation to obtain the
166 optimal cost-to-go from any state, as done by the authors Yuval Tassa and Tom
167 Erez in “Least Squares Solutions of the HJB Equation With Neural Network
168 Value-Function Approximators”

169 (<https://homes.cs.washington.edu/~fodorov/courses/amath579/reading/NeuralNet.pdf>).

170 In practice, while solving the HJB PDE, one discretizes the state-space at given
171 set of states and solves the HJB equation (6.7) on this grid using numerical
172 methods (these authors used neural networks to solve it). The end result looks
173 as follows.

174 6.2.3 Continuous-time LQR

175 Consider a linear continuous-time dynamical system given by

$$\dot{x} = A x + B u; \quad x(0) = x_0.$$

176 In the LQR problem, we are interested in finding a control trajectory that
177 minimizes, as usual, a cost function that is quadratic in states and controls,
178 except that we have an integral of the run-time cost because our system is a

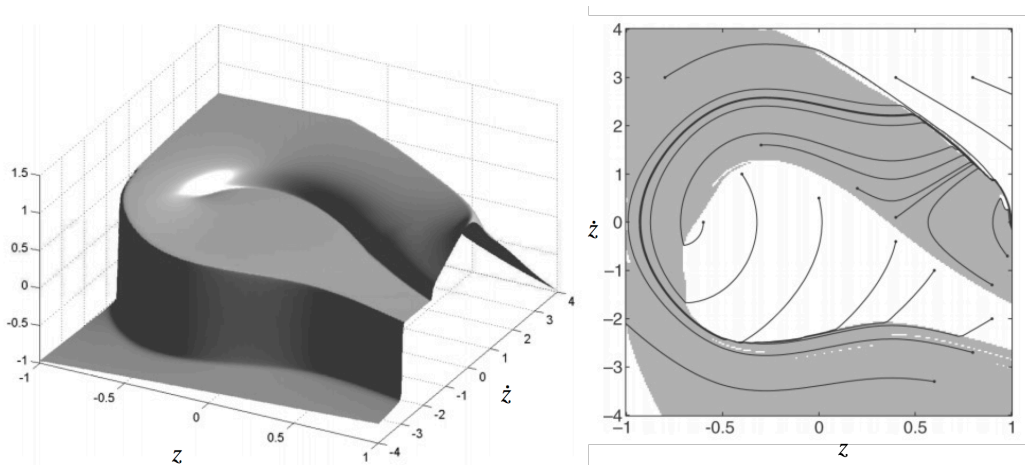


Figure 6.5: The left-hand side picture shows the infinite-horizon cost-to-go $J^*(z, \dot{z})$ for the car-on-the-hill problem. Notice how the value function is non-smooth at various places. This is quite typical of difficult dynamic programming problems. The right-hand side picture shows the optimal trajectories of the car $(z(t), \dot{z}(t))$; gray areas indicate maximum control and white areas indicate minimum control. The black lines show a few optimal control sequences taken the car starting from various states in the state-space. Notice how the optimal control trajectory can be quite different even if the car starts from nearby states $(-0.5, 1)$ and $(-0.4, 1.2)$. This is also quite typical of difficult dynamic programming problems.

179 continuous-time system

$$\frac{1}{2} x(T)^\top Q_f x(T) + \frac{1}{2} \int_0^T x(t)^\top Q x(t) + u(t)^\top R u(t) dt.$$

180 This is a very nice setup for using the HJB equation from the previous section.

181 Let us use our intuition from the discrete-time LQR problem and say that
182 the optimal cost is quadratic in the states, namely,

$$J^*(x, t) = \frac{1}{2} x(t)^\top P(t) x(t);$$

183 notice that as usual the optimal cost-to-go is a function of the states x and the
184 time t because is the optimal cost of the continuous-time LQR problem if the
185 system starts at a state x at time t and goes on until time $T \geq t$. We will now
186 check if this J^* satisfies the HJB equation (we don't write the arguments $x(t)$,
187 $u(t)$ etc. to keep the notation clear)

$$-\partial_t J^*(x, t) = \min_{u \in U} \left\{ \frac{1}{2} (x^\top Q x + u^\top R u) + (A x + B u)^\top \partial_x J^*(x, t) \right\} \quad (6.9)$$

188 from (6.7). The minimization is over the control input that we take at time t .

189 Also notice the partial derivatives

$$\begin{aligned}\partial_x J^*(x, t) &= P(t) x. \\ \partial_t J^*(x, t) &= \frac{1}{2} x^\top \dot{P}(t) x.\end{aligned}$$

190 It is convenient in this case to see that the minimization can be performed
191 using basic calculus (just like the discrete-time LQR problem), we differentiate
192 with respect to u and set it to zero.

$$\begin{aligned}0 &= \frac{\text{d RHS of HJB}}{\text{d}u} \\ \Rightarrow u^*(t) &= -R^{-1} B^\top P(t) x(t) \\ &\equiv -K(t) x(t).\end{aligned}\tag{6.10}$$

193 where $K(t) = R^{-1} B^\top P(t)$ is the Kalman gain. The controller is again linear
194 in the states $x(t)$ and the expression for the gain is very simple in this case,
195 much simpler than discrete-time LQR. Since $R \succ 0$, we also know that $u^*(t)$
196 computed here is the global minimum. If we substitute this value of $u^*(t)$
197 back into the HJB equation we have

$$\left. \frac{\text{d}J^*}{\text{d}t} \right|_{u^*(t)} = \frac{1}{2} x^\top \{PA + A^\top P + Q - PBR^{-1}B^\top P\} x.$$

198 In order to satisfy the HJB equation, we must have that the expression above is
199 equal to $-\partial_t J^*(x, t)$. We therefore have, what is called the Continuous-time
200 Algebraic Riccati Equation (CARE), for the matrix $P(t) \in \mathbb{R}^{d \times d}$

$$-\dot{P} = PA + A^\top P + Q - PBR^{-1}B^\top P.\tag{6.11}$$

201 This is an ordinary differential equation for the matrix P . The derivative
202 $\dot{P} = \frac{dP}{dt}$ stands for differentiating every entry of P individually with time t .
203 The terminal cost is $\frac{1}{2} x(T)^\top Q_f x(T)$ which gives the boundary condition for
204 the ODE as

$$P(T) = Q_f.$$

205 Notice that the ODE for the $P(t)$ travels backwards in time.

206 Continuous-time LQR has particularly easy equations, as you can see
207 in (6.10) and (6.11) compared to those for discrete-time ((6.4) and (6.5)).
208 Special techniques have been invented for solving the Riccati equation. I
209 used the function `scipy.linalg.solve_continuous_are` to obtain Figure 6.3 using
210 the continuous-time equations; the corresponding function for solving
211 Discrete-time Algebraic Riccati Equation (DARE) which is given in (6.5)
212 is `scipy.linalg.solve_discrete_are`. The continuous-time point-of-view also
213 gives powerful connections to the Kalman filter, where you can show that the
214 Kalman filter and LQR are duals of each other: in fact the equations for the
215 Kalman filter (in continuous-time) and continuous-time LQR turn out to be
216 exactly the same after you interchange appropriate quantities (!).

217 **Infinite-horizon LQR** Just like the infinite-horizon HJB equation has $\partial_t J^*(x, t) =$
 218 0, if we have an infinite-horizon LQR problem, the cost matrix P should not
 219 be a function of time

$$\dot{P} = 0.$$

220 The continuous-time algebraic Riccati equation in (6.11) now becomes

$$0 = PA + A^\top P + Q - PBR^{-1}B^\top P.$$

221 with the cost-to-go being given by $J^*(x) = \frac{1}{2}x^\top Px$.

222 6.3 Stochastic LQR

223 We will next look at a very powerful result. Say we have a stochastic linear
 224 dynamical system

$$\dot{x}(t) = Ax(t) + Bu(t) + B_\epsilon \epsilon(t); x(0) \text{ is given}$$

225 where $\epsilon(t)$ is standard Gaussian noise $\epsilon(t) \sim N(0, I)$ that is uncorrelated
 226 in time and would like to find a control sequence $\{u(t) : t \in [0, T]\}$ that
 227 minimizes a quadratic run-time and terminal cost

$$\mathbb{E}_{\epsilon(t):t \in [0, T]} \left[\frac{1}{2}x(T)^\top Q_f x(T) + \frac{1}{2} \int_0^T x(t)^\top Q x(t) + u(t)^\top R u(t) dt \right].$$

228 over a finite-horizon T . Notice that since the system is stochastic now, we
 229 should minimize the expected value of the cost over all possible realizations of
 230 the noise $\{\epsilon(t) : t \in [0, T]\}$. This is a very challenging problem, conceptually
 231 it is the equivalent of dynamic programming for an MDP with an infinite
 232 number of states $x(t) \in \mathbb{R}^d$ and an infinite number of controls $u(t) \in \mathbb{R}^m$.

233 However, it turns out that the optimal controller that we should pick in this
 234 case is also given by the standard LQR problem

$$u^*(t) = -R^{-1}B^\top P(t) x(t)$$

$$\text{with } -\dot{P} = PA + A^\top P + Q - PBR^{-1}B^\top P; P(T) = Q_f.$$

235 We will not do the proof (it is easy but tedious, you can try to show it by
 236 writing the HJB equation for the stochastic LQR problem). This is a very
 237 surprising result because it says that even if the dynamical system had noise,
 238 the optimal control we should pick is exactly the same as the control we would
 239 have picked had the system been deterministic. It is a special property of the
 240 LQR problem and not true for other dynamical systems (nonlinear ones, or
 241 ones with non-Gaussian noise) or other costs.

242 We know that the control $u^*(t)$ is the same as the deterministic case. Is
 243 the cost-to-go $J^*(x, t)$ also the same? If you think about this, the cost-to-go
 244 in the stochastic case has to be a bit larger than the deterministic case because
 245 the noise $\epsilon(t)$ is always going to non-zero when we run the system, the LQR
 246 cost $J^*(x_0, 0) = \frac{1}{2}x_0^\top P(0)x_0$ is, after all, only the cost of the deterministic
 247 problem. It turns out that the cost for the stochastic LQR case for an initial

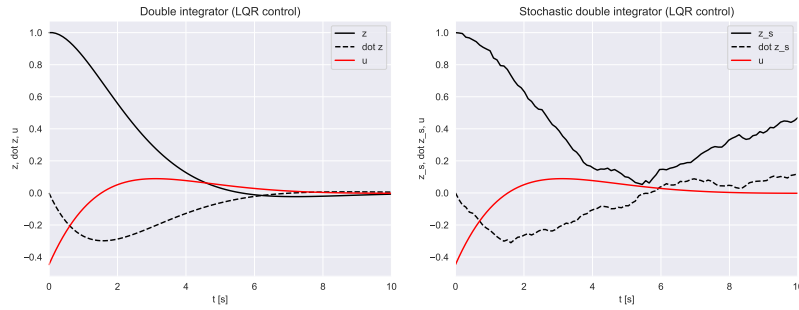


Figure 6.6: Comparison of the state trajectories of deterministic LQR and stochastic LQR problem with $B_\epsilon = [0.1, 0.1]$. The left panel is the same as that in Figure 6.3. The control input is the same in both cases but notice that the states in the plot on the right need not converge to the equilibrium due to noise. The cost of the trajectory will also be higher for the stochastic LQR case due to this. The total cost is $J^*(x_0) = 32.5$ for the deterministic case (32.24 for the quadratic state-cost and 0.26 for the control cost). The total cost $J^*(x_0)$ is much higher for the stochastic case, it is 81.62 (81.36 for the quadratic state cost and 0.26 for the control cost).

248 state x_0 is

$$\begin{aligned}
 J^*(x_0, 0) &= \mathbf{E}_{\epsilon(t):t \in [0, T]} \left[\frac{1}{2} x(T)^\top Q_f x(T) + \frac{1}{2} \int_0^T \dots dt \right] \\
 &= \frac{1}{2} x_0^\top P(0) x_0 + \frac{1}{2} \int_0^T \text{tr}(P(t) B_\epsilon B_\epsilon^\top) dt.
 \end{aligned}$$

249 The first term is the same as that of the deterministic LQR problem. The
 250 second term is the penalty we incur for having a stochastic dynamical system.
 251 This is the minimal cost achievable for stochastic LQR but it is not the same
 252 as that of the deterministic LQR.

253 6.4 Linear Quadratic Gaussian (LQG)

254 Our development in the previous sections and the previous chapter was based
 255 on a Markov Decision Process, i.e., we know the state $x(t)$ at each instant in
 256 time t even if this state $x(t)$ changes stochastically. We said that the optimal
 257 control for the linear dynamics is still $u^*(t) = -K(t) x(t)$. What should one
 258 do if we cannot observe the state exactly?

259 Imagine a “continuous-time” form the observation equation in the Kalman
 260 filter where we receive observations of the form

$$y(t) = Cx(t) + D\nu.$$

261 where $\nu \sim N(0, I)$ is standard Gaussian noise that corrupts our observations
 262 y . If we extrapolate the definitions of the Kalman filter mean and covariance
 263 to this continuous-time setting, we can write the KF as follows. We know that
 264 the Kalman filter is the optimal estimate of the state given all past observations,

265 so it computes

$$\mu(t) = \mathbb{E}_{\epsilon(s), \nu(s): s \in [0, t]} [x(t) | y(s) : s \in [0, t]].$$

266 There exists a “continuous-time version” of the Kalman filter (which was
267 actually invented first), called the Kalman-Bucy filter. If the covariance of the
268 estimate is

$$\Sigma(t) = \mathbb{E}_{\epsilon(s), \nu(s): s \in [0, t]} [x(t) x(t)^\top | y(s) : s \in [0, t]],$$

269 the Kalman-Bucy filter updates $\mu(t)$, $\Sigma(t)$ using the differential equation

$$\begin{aligned} \frac{d}{dt} \mu(t) &= Ax(t) + Bu(t) + K(t) (y(t) - C\mu(t)) \\ \frac{d}{dt} \Sigma(t) &= A\Sigma(t) + \Sigma(t)A^\top + B_\epsilon B_\epsilon^\top - K(t)DD^\top K(t)^\top \end{aligned} \quad (6.12)$$

where $K(t) = \Sigma(t) C^\top (DD^\top)^{-1}$.

270 This equation is very close to the Kalman filter equations you saw in Chapter
271 3. In particular, notice the close similarity of the expression for the Kalman
272 gain $K(t)$ with the Kalman gain of the LQR problem. You can read more at
273 https://en.wikipedia.org/wiki/Kalman_filter.

Linear Quadratic Gaussian (LQG) It turns out that we can plug in the Kalman filter estimate $\mu(t)$ of the state $x(t)$ in order to compute optimal control for LQR if we know the state only through observations $y(t)$

$$u^*(t) = -K(t) \mu(t). \quad (6.13)$$

It is almost as if, we can blindly run a Kalman Filter in parallel with the deterministic LQR controller and get the optimal control for the stochastic LQR problem even if we did not observe the state of the system exactly. This method is called Linear Quadratic Gaussian (LQG).

This is a very powerful and surprising result. It is only true for linear dynamical systems with linear observations, Gaussian noise in both the dynamics and the observations and quadratic run-time and terminal costs. It is not true in other cases. However, it is so elegant and useful that it inspires essentially all other methods that control a dynamical system using observations from sensors.

274 **Certainty equivalence** For instance, even if we are using a particle filter to
275 estimate the state of the system, we usually use the mean of the state estimate
276 at time t given by $\mu(t)$ “as if” it were the true state of the system. Even if we
277 were using some other feedback control $u(x)$ different than the LQR control
278 (say feedback linearization), we usually plug in this estimate $\mu(t)$ in place of
279 $x(t)$. Doing so is called “certainty equivalence” in control theory/robotics,
280 which is a word borrowed from finance where one takes decisions (controls)
281 directly using the estimate of the state (say stock price) while fully knowing

❗ As we discussed while introducing stochastic dynamical systems, there are various mathematical technicalities associated with conditioning on a continuous-time signal $\{y(s) : s \in [0, t]\}$. To be precise mathematicians define what is called a “filtration” $\mathcal{Y}(t)$ which is the union of the Borel σ -fields constructed using increasing subsets of the set $\{y(s) : s \in [0, t]\}$. Let us not worry about this here.

282 the the stock price will change in the future stochastically.

283 **6.4.1 (Optional material) The duality between the Kalman** 284 **Filter and LQR**

285 We can re-write the covariance in (6.12) using the identity

$$\frac{d}{dt} (\Sigma(t)^{-1}) = \Sigma(t)^{-1} \dot{\Sigma}(t) \Sigma(t)^{-1}$$

286 to get

$$\dot{S} = C^\top (DD^\top)^{-1} C - A^\top S - SA - SB_w B_w^\top S \quad (6.14)$$

287 where we have defined $S := \Sigma^{-1}$.

288 Notice that the two equations, updates to the LQR cost matrix in (6.11)

$$-\dot{P} = PA + A^\top P + Q - PBR^{-1}B^\top P$$

289 look quite similar to this equation. In fact, they are identical and you can
290 substitute the following.

LQR	Kalman-Bucy filter
P	Σ^{-1}
A	$-A$
$BR^{-1}B$	$B_w B_w^\top$
Q	$C^\top (DD^\top)^{-1} C$
t	$T - t$

292 Let us analyze this equivalence. Notice that the inverse of the Kalman
293 filter covariance is like the cost matrix of LQR. This is conceptually easy to
294 understand, our figure of merit for filtering is the inverse covariance matrix
295 (smaller the better) and our figure of merit for the LQR problem is the cost
296 matrix P (smaller the better). Similarly, smaller the LQR cost, better the
297 controller. The “dynamics” of the Kalman filter is the reverse of the dynamics
298 of the LQR problem, this shows that the P matrix is updated backwards in
299 time while the covariance Σ is updated forwards in time. The next identity

$$BR^{-1}B^\top = B_w B_w^\top$$

300 is very interesting. Imagine a situation where we have a fully-actuated system
301 with $B = I$ and B_w being a diagonal matrix. This identity suggests that
302 larger the control cost R_{ii} of a particular actuator i , lower is the noise of using
303 that actuator $(B_w)_{ii}$, and vice-versa. This is how muscles in your body have
304 evolved: muscles that are cheap to use (low R) are also very noisy in what they
305 do whereas muscles that are expensive to use (large R) which are typically
306 the biggest muscles in the body are also the least noisy and most precise. You
307 can read more about this in the paper titled “General duality between optimal
308 control and estimation” by Emanuel Todorov. The next identity

$$Q = C^\top (DD^\top)^{-1} C$$

is related to the quadratic state-cost in LQR. Imagine the situation where both Q, D are diagonal matrices. If the noise in the measurements D_{ii} is large, this is equivalent to the state-cost matrix Q_{ii} being small; roughly there is no way we can achieve a low state-cost $x^\top Q x$ in our system that consists of LQR and a Kalman filter (this combination is known as Linear Quadratic Gaussian LQG as saw before) if there is lots of noise in the state measurements. The final identity

$$t = T - t$$

is the observation that we have made many times before: dynamic programming travels backwards in time and the Kalman filter travels forwards in time.

6.5 Iterative LQR (iLQR)

This section is analogous to the section on the Extended Kalman Filter. We will study how to solve optimal control problems for a nonlinear dynamical system

$$\dot{x} = f(x, u); x(0) = x_0 \text{ is given.}$$

We will consider a deterministic continuous-time dynamical system, the modifications to following section that one would make if the system is discrete-time, or stochastic, are straightforward and follow the same strategy. First consider the problem where the run-time and terminal costs are quadratic

$$\frac{1}{2}x(T)^\top Q_f x(T) + \frac{1}{2} \int_0^T x(t)^\top Q x(t) + u(t)^\top R u(t) dt.$$

Receding horizon control and Model Predictive Control (MPC) One easy way to solve the dynamic programming problem, i.e., find a control trajectory of the *nonlinear* system that minimizes this cost functional, approximately, is by linearizing the system about the initial state x_0 and some reference control u_0 (this can usually be zero). Let the linear system be

$$\dot{z} = A_{x_0, u_0} z + B_{x_0, u_0} v; z(0) = 0; \quad (6.15)$$

where $A_{x_0, u_0} = \left. \frac{df}{dx} \right|_{x=x_0, u=u_0}$ and $B_{x_0, u_0} = \left. \frac{df}{du} \right|_{x=x_0, u=u_0}$ are the Jacobians of the nonlinear function $f(x, u)$ with respect to the state and control respectively. The state of the linearized dynamics is

$$z := x - x_0, \text{ and } v := u - u_0,$$

We have emphasized the fact that the matrices $A_{x_0, u_0}, B_{x_0, u_0}$ depend upon the reference state and control using the subscript. Given the above linear system, we can find a control sequence $u^*(\cdot)$ that minimizes the cost functional using the standard LQR formulation. Notice now that even we computed this control trajectory using the approximate linear system, it can certainly be *executed* on the nonlinear system, i.e., at run-time we will simply set $u \equiv u^*(z)$.

The linearized dynamics in (6.15) is potentially going to be very different from the nonlinear system. The two are close in the neighborhood of x_0 (and

343 u_0) but as the system evolves using our control input to move further away
 344 from x_0 , the linearized model no longer is a faithful approximation of the
 345 nonlinear model. A reasonable way to fix matters is to linearize about another
 346 point, say the state and control after $t = 1$ seconds, x_1, u_1 to get a new system

$$\dot{z} = A_{x_1, u_1} z + B_{x_1, u_1} v; z(0) = 0$$

347 and take the LQR-optimal control corresponding to this system for the next
 348 second.

349 The above methodology is called “receding horizon control”. The idea is
 350 that we compute the optimal control trajectory $u^*(\cdot)$ using an approximation
 351 of the original system and recompute this control every few seconds when our
 352 approximation is unlikely to be accurate. This is a very popular technique to
 353 implement optimal controllers in typical applications. The concept of using an
 354 approximate model (almost invariably, a linear model with LQR cost) to plan
 355 for the near-term future and resolving the problem in receding horizon fashion
 356 once the system is at the end of this short time-horizon is called “Model
 357 Predictive Control”.

358 MPC is, perhaps, the second most common control algorithm implemented
 359 in the world. It is responsible for running most complex engineering systems
 360 that you can think of—power grids, oil refineries, chemical plants, rockets,
 361 aircrafts etc. Essentially, one never implements LQR directly, it is always im-
 362 plemented inside an MPC. For instance, in autonomous driving, the trajectory
 363 that the vehicle plans for traveling between two points A and B depends upon
 364 the current locations of the other cars/pedestrians in its vicinity, and potentially
 365 some prediction model of where they will be in the future. As the vehicle
 366 starts moving along this trajectory, the rest of the world evolves around it and
 367 we recompute the optimal trajectory to take into account the actual locations
 368 of the cars/pedestrians in the future.

🔗 Can you guess what is *the* most common control algorithm in the world?

369 6.5.1 Iterative LQR (iLQR)

370 Now let us consider the situation when in addition to a nonlinear system,

$$\dot{x} = f(x, u); x(0) = x_0,$$

371 the run-time and terminal cost is also nonlinear

$$q_f(x(T)) + \int_0^T q(x(t), u(t)) dt.$$

372 We can solve the dynamic programming problem in this case approximately
 373 using the following iterative algorithm.

374 Assume that we are given an initial control trajectory $u^{(0)}(\cdot) = \{u^{(0)}(t) : t \in [0, T]\}$.
 375 Let $x^{(0)}(\cdot)$ be the state trajectory that corresponds to taking this control on
 376 the nonlinear system, with of course $x^{(0)}(0) = x_0$. At each iteration k , the
 377 Iterative LQR algorithm performs the following steps.

378 **Step 1** Linearize the nonlinear system about the state trajectory $x^{(k)}(\cdot)$ and

379 $u^{(k)}(\cdot)$ using

$$z(t) := x(t) - x^{(k)}(t), \text{ and } v(t) := u(t) - u^{(k)}(t)$$

380 to get a new system

$$\dot{z} = A^{(k)}(t)z + B^{(k)}(t)v; z(0) = 0$$

381 where

$$A^{(k)}(t) = \left. \frac{df}{dx} \right|_{x(t)=x^{(k)}(t), u(t)=u^{(k)}(t)}$$

$$B^{(k)}(t) = \left. \frac{df}{du} \right|_{x(t)=x^{(k)}(t), u(t)=u^{(k)}(t)}$$

382 and compute the Taylor series approximation of the nonlinear cost up to the
383 second order

$$q_f(x(T)) \approx \text{constant} + z(T)^\top \left. \frac{dq_f}{dx} \right|_{x(T)=x^{(k)}(T)}$$

$$+ z(t)^\top \left. \frac{d^2 q_f}{dx^2} \right|_{x(T)=x^{(k)}(T)} z(t),$$

384

$$q(x, u, t) \approx \text{constant} + z(t)^\top \underbrace{\left. \frac{dq}{dx} \right|_{x(t)=x^{(k)}(t), u(t)=u^{(k)}(t)}}_{\text{affine term}}$$

$$+ v(t)^\top \underbrace{\left. \frac{dq}{du} \right|_{x(t)=x^{(k)}(t), u(t)=u^{(k)}(t)}}_{\text{affine term}}$$

$$+ z(t)^\top \underbrace{\left. \frac{d^2 q}{dx^2} \right|_{x(t)=x^{(k)}(t), u(t)=u^{(k)}(t)}}_{\equiv Q} z(t)$$

$$+ v(t)^\top \underbrace{\left. \frac{d^2 q}{du^2} \right|_{x(t)=x^{(k)}(t), u(t)=u^{(k)}(t)}}_{\equiv R} v(t).$$

385 This is an LQR problem with run-time cost that depends on time (like our
386 discrete-time LQR formulation, the continuous-time formulation simply has
387 Q, R to be functions of time t in the Riccati equation) and which also has
388 terms that are affine in the state and control in addition to the usual quadratic
389 cost terms.

390 **Step 2** Solve the above linearized problem using standard LQR formulation to
391 get the new control trajectory

$$u^{(k+1)}(t) := u^{(k)}(t) - Kz(t).$$

392 Simulate the *nonlinear* system using the control $u^{(k+1)}(\cdot)$ to get the new state
393 trajectory $x^{(k+1)}(\cdot)$.

394 Some important comments to remember about the iLQR algorithm.

395 1. There are many ways to pick the initial control trajectory $u^{(0)}(\cdot)$, e.g.,

🔗 How will you solve for the optimal controller for a linear dynamics for the cost

$$\int_0^T \left(q^\top x + \frac{1}{2} x^\top Q x \right) dt,$$

i.e., when in addition the quadratic cost, we also have an affine term?

- 396 using a spline to get an arbitrary control sequence, using a spline to
 397 interpolate the states to get a trajectory $x^{(0)}(\cdot)$ and then back-calculate
 398 the control trajectory, using the LQR solution based on the lineariza-
 399 tion about the initial state, feedback linearization/differential flatness
 400 (https://en.wikipedia.org/wiki/Feedback_linearization) etc.
- 401 2. The iLQR algorithm is an approximate solution to dynamic program-
 402 ming for nonlinear system with general, nonlinear run-time and terminal
 403 costs. This is because the the algorithm uses a linearization about the
 404 previous state and control trajectory to compute the new control trajec-
 405 tory. iLQR is not guaranteed to find the optimal solution of dynamic
 406 programming, although in practice with good implementations, it works
 407 excellently.
 - 408 3. We can think of iLQR as an algorithm to track a given state trajectory
 409 $x^g(t)$ by setting

$$q_f = 0, \text{ and } q(x, u) = \|x^g(t) - x(t)\|^2.$$

410 This is often how iLQR is typically used in practice, e.g., to make
 411 an autonomous race car closely follow the racing line (see the paper
 412 “BayesRace: Learning to race autonomously using prior experience”
 413 <https://arxiv.org/abs/2005.04755> and <https://www.youtube.com/watch?v=dgIpf0Lg8Ek>
 414 for a clever application of using MPC to track a challenging race line),
 415 or to make a drone follow a given desired trajectory
 416 (<https://www.youtube.com/watch?v=QREeZvHg0IQ>).

417 **Differential Dynamic Programming (DDP)** is a suite of techniques that is
 418 a more powerful version of iterated LQR. Instead of linearizing the dynamics
 419 and taking a second order Taylor approximation of the cost, DDP takes a
 420 second order approximation of the Bellman equation directly. The two are
 421 not the same; DDP is the more correct version of iLQR but is much more
 422 challenging computationally.

423 Broadly speaking, iLQR and DDP are used to perform control for some of
 424 the most sophisticated robots today, you can see an interesting discussion of
 425 the trajectory planning of some of the DARPA Humanoid Robotics Challenge
 426 at <https://www.cs.cmu.edu/~cga/drc/atlas-control>. Techniques like feedback
 427 linearization work excellently for drones where we do not really care for opti-
 428 mal cost (see “Minimum snap trajectory generation and control for quadrotors”
 429 <https://ieeexplore.ieee.org/document/5980409>) while LQR and its variants are
 430 still heavily utilized for satellites in space.