AA203: Optimal and Learning-based Control Course Notes

James Harrison*

April 13, 2019

2 Dynamic Programming and the Linear Quadratic Regulator

2.1 The Optimal Control Problem

In this section, we will outline the deterministic continuous-time optimal control problem that we will aim to solve. We will denote the state at time t as $\boldsymbol{x}(t) \in \mathbb{R}^n$, and the control as $\boldsymbol{u}(t) \in \mathbb{R}^m$. We will also occasionally write these as $\boldsymbol{x}_t$ and $\boldsymbol{u}_t$, respectively. We will write the continuous-time systems dynamics as

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t). \tag{1}$$

We will refer to a history of control input values during an interval $[t_0, t_f]$ as a control history, and we will refer to a history of state values over this interval as a state trajectory.

Different control problems may call for various constraints. For example, we may constrain a quadrotor to only fly in space not occupied by obstacles. Examples of constraints we will see are

- Initial and final conditions, $\boldsymbol{x}(t_0) = \boldsymbol{x}_0$, $\boldsymbol{x}(t_f) = \boldsymbol{x}_f$
- Trajectory constraints, $\underline{\boldsymbol{x}} \leq \boldsymbol{x}(t) \leq \bar{\boldsymbol{x}}$
- Control limits, $\underline{\boldsymbol{u}} \leq \boldsymbol{u}(t) \leq \bar{\boldsymbol{u}}$.

A state trajectory and control history that satisfy the constraints during the entire time interval $[t_0, t_f]$ are called admissible trajectories and admissible controls, respectively.

Finally, we will define the performance measure,

$$J = c_f(\boldsymbol{x}(t_f), t_f) + \int_{t_0}^{t_f} c(\boldsymbol{x}(t), \boldsymbol{u}(t), t) dt \tag{2}$$

*Contact: jharrison@stanford.edu

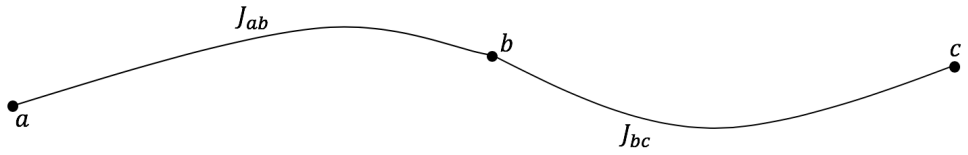

Figure 1: An optimal trajectory connecting point a to point c. There are no better (lower cost) trajectories than the sub-trajectory connecting b and c, by the principle of optimality.

where c is the instantaneous cost function, and c_f is the terminal state cost. We are now able to state the continuous-time optimal control problem. We aim to find an admissible control, $\boldsymbol{u}^*$, which causes the system (1) to follow an admissible trajectory, $\boldsymbol{x}^*$, that minimizes the performance measure given by (2). The minimizer $(\boldsymbol{x}^*, \boldsymbol{u}^*)$ is called an optimal trajectory-control pair.

Note, first of all, that this is an extremely general problem formulation. We have not fixed our system dynamics, cost function, or specific constraints. We can't, in general, guarantee the existence or uniqueness of the optimal solution.

There are two possible solution forms for the optimal control. The first, $\boldsymbol{u}^* = e(\boldsymbol{x}(t_0), t)$ is referred to as an open-loop solution. This is an input function that is applied to the system, without using feedback. Practically, such solutions usually require augmentation with a feedback controller, as small model mismatch may lead to compounding errors. The second possible solution form is a feedback policy, $\boldsymbol{u}^* = \pi(\boldsymbol{x}(t), t)$. This feedback law maps all state-time pairs to an action and thus is usually more robust to possible model mismatch. However, depending on the particular problem formulation, open-loop solutions may be easier to compute.

2.2 Dynamic Programming and the Principle of Optimality

In this chapter we will outline the principle of optimality, and the method of dynamic programming (DP), one of two main approaches to solving the optimal control problem. The second, so-called variational approaches based on Pontryagin's Maximum Principle (PMP) will be discussed in future chapters. Dynamic programming has the strong advantage of yielding a feedback policy, however, exactly solving the dynamic programming problem is infeasible for many systems. We will address special cases in which the DP problem can be solved exactly, and approximate methods that work for a wide variety of systems.

Despite having just introduced the optimal control problem in continuous time, we will be operating in discrete time here, in which we aim to minimize

$$J_f(\boldsymbol{x}_0) = c_f(\boldsymbol{x}_N) + \sum_{k=0}^{N-1} c(\boldsymbol{x}_k, \boldsymbol{u}_k, k). \tag{3}$$

We will extend the methods we develop in this chapter to continuous time in the next chapter.

The principle of optimality is as follows. Figure 1 shows a trajectory from point a to c. If the cost of the trajectory, $J_{ac} = J_{ab} + J_{bc}$, is minimal, then J_{bc} is also a minimum

cost trajectory connecting b and c. The proof of this principle, stated informally, is simple. Assume there exists an alternative trajectory connecting b and c, for which we will write the cost as $\tilde{J}_{bc}$, that achieves $\tilde{J}_{bc} < J_{bc}$. Then, we have

$$\begin{aligned} \tilde{J}_{ac} &= J_{ab} + \tilde{J}_{bc} && (4)\\ &< J_{ab} + J_{bc} && (5)\\ &= J_{ac}, && (6) \end{aligned}$$

and thus J_{ac} isn't minimal. More formally,

Theorem 2.1 (Discrete-time Principle of Optimality)**.** *Let $\pi^* = (\pi_0^*, \ldots, \pi_{N-1}^*)$ be an optimal policy. Assume state $\boldsymbol{x}_k$ is reachable. Consider the subproblem whereby we are at $\boldsymbol{x}_k$ at time k and we wish to minimize the cost-to-go from time k to time N. Then the truncated policy $(\pi_k^*, \ldots, \pi_{N-1}^*)$ is optimal for the subproblem.*

Dynamic programming, intuitively, proceeds backwards in time, first solving simpler shorter horizon problems. If we have found the optimal policy for times $k+1$ to $N-1$, along with the associated cost-to-go for each state, choosing the optimal policy for time k is a one step optimization problem. More concretely, we will assume we have dynamics of the form $\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k, k)$ with $\boldsymbol{u}_k \in \mathcal{U}(\boldsymbol{x}_k)$, and the cost given by (3). Then, dynamic programming iterates backward in time, from $N-1$ to 0, with

$$\begin{aligned} J_N(\boldsymbol{x}_N) &= c_T(\boldsymbol{x}_N) && (7)\\ J_k(\boldsymbol{x}_k) &= \min_{\boldsymbol{u}_k \in \mathcal{U}(\boldsymbol{x}_k)} \left\{ c_k(\boldsymbol{x}_k, \boldsymbol{u}_k, k) + J_{k+1}(f(\boldsymbol{x}_k, \boldsymbol{u}_k, k)) \right\}. && (8) \end{aligned}$$

Note that here we have considered only deterministic dynamical systems (there is no stochastic disturbance). Equation (8) is one form of the *Bellman equation*, one of the most important relations in optimal control.

Dynamic programming raises many practical issues if one were to attempt to apply it directly. To perform the recursion, J_{k+1} must be known for all $\boldsymbol{x}_{k+1}$ (or more precisely, all $\boldsymbol{x}_{k+1}$ that are reachable from $\boldsymbol{x}_k$). If the state space is discrete (and relatively small), this is tractable as the cost-to-go may just be maintained in tabular form. In the next subsection, we will discuss an extremely important case in continuous space in which the cost-to-go can be computed exactly for all states. However, for general systems, we can not expect to be able to compute the cost-to-go for all states. Possible approaches to make the DP approach tractable are discretizing the state space, approximating the cost-to-go (i.e. restricting the family of functions that J_{k+1} may be in), or interpolating between cost-to-go computed for a finite set of states.

2.3 Discrete LQR

An important instance in which dynamic programming can be solved analytically for continuous state-action systems is the *linear quadratic regulator* problem. We will fix the dynamics of the system to be (possibly time-varying) linear,

$$\boldsymbol{x}_{k+1} = A_k \boldsymbol{x}_k + B_k \boldsymbol{u}_k \tag{9}$$

and the cost function as quadratic

$$c(\boldsymbol{x}_k, \boldsymbol{u}_k) = \frac{1}{2}(\boldsymbol{x}_k^T Q_k \boldsymbol{x}_k + \boldsymbol{u}_k^T R_k \boldsymbol{u}_k) \tag{10}$$

$$c_N(\boldsymbol{x}_k) = \frac{1}{2}\boldsymbol{x}_k^T Q_N \boldsymbol{x}_k \tag{11}$$

where $Q_k \in \mathbb{R}^{n\times n}$ is positive semi-definite and $R_k \in \mathbb{R}^{m\times m}$ is positive definite for all $k = 0, \ldots, N$. Importantly, we assume $\boldsymbol{x}_k$ and $\boldsymbol{u}_k$ are unconstrained for all k. To perform DP recursion, we initialize

$$J_N^*(\boldsymbol{x}_N) = \frac{1}{2}\boldsymbol{x}_N^T Q_N \boldsymbol{x}_N := \frac{1}{2}\boldsymbol{x}_N^T V_N \boldsymbol{x}_N. \tag{12}$$

Then, applying (8), we have

$$J_{N-1}^*(\boldsymbol{x}_{N-1}) = \frac{1}{2}\min_{\boldsymbol{u}_{N-1}\in\mathbb{R}^m} \left\{\boldsymbol{x}_{N-1}^T Q_{N-1}\boldsymbol{x}_{N-1} + \boldsymbol{u}_{N-1}^T R_{N-1}\boldsymbol{u}_{N-1} + \boldsymbol{x}_N^T V_N \boldsymbol{x}_N\right\} \tag{13}$$

which, applying the dynamics,

$$\begin{aligned} J_{N-1}^*(\boldsymbol{x}_{N-1}) = \frac{1}{2}\min_{\boldsymbol{u}_{N-1}\in\mathbb{R}^m} \{&\boldsymbol{x}_{N-1}^T Q_{N-1}\boldsymbol{x}_{N-1} + \boldsymbol{u}_{N-1}^T R_{N-1}\boldsymbol{u}_{N-1} \\ &+ (A_{N-1}\boldsymbol{x}_{N-1} + B_{N-1}\boldsymbol{u}_{N-1})^T V_N (A_{N-1}\boldsymbol{x}_{N-1} + B_{N-1}\boldsymbol{u}_{N-1})\}. \end{aligned} \tag{14}$$

Rearranging, we have

$$\begin{aligned} J_{N-1}^*(\boldsymbol{x}_{N-1}) = \frac{1}{2}\min_{\boldsymbol{u}_{N-1}\in\mathbb{R}^m} \{&\boldsymbol{x}_{N-1}^T (Q_{N-1} + A_{N-1}^T V_N A_{N-1})\boldsymbol{x}_{N-1} \\ &+ \boldsymbol{u}_{N-1}^T (R_{N-1} + B_{N-1}^T V_N B_{N-1})\boldsymbol{u}_{N-1} \\ &+ 2\boldsymbol{u}_{N-1}^T (B_{N-1}^T V_N A_{N-1})\boldsymbol{x}_{N-1}\}. \end{aligned} \tag{15}$$

Note that this optimization problem is convex in $\boldsymbol{u}_{N-1}$ as $R_{N-1} + B_{N-1}^T V_N B_{N-1} > 0$. Therefore, any local minima is a global minima, and therefore we can simply apply the first order optimality conditions. Differentiating,

$$\frac{\partial J_{N-1}^*(\boldsymbol{x}_{N-1})}{\partial \boldsymbol{u}_{N-1}} = (R_{N-1} + B_{N-1}^T V_N B_{N-1})\boldsymbol{u}_{N-1} + (B_{N-1}^T V_N A_{N-1})\boldsymbol{x}_{N-1} \tag{16}$$

and setting this to zero yields

$$\boldsymbol{u}_{N-1}^* = -(R_{N-1} + B_{N-1}^T V_N B_{N-1})^{-1}(B_{N-1}^T V_N A_{N-1})\boldsymbol{x}_{N-1} \tag{17}$$

which we write

$$\boldsymbol{u}_{N-1}^* = L_{N-1}\boldsymbol{x}_{N-1} \tag{18}$$

which is a time-varying linear feedback policy. Plugging this feedback policy into (24),

$$\begin{aligned} J_{N-1}^*(\boldsymbol{x}_{N-1}) =&\boldsymbol{x}_{N-1}^T (Q_{N-1} + L_{N-1}^T R_{N-1} L_{N-1} \\ &+ (A_{N-1} + B_{N-1}L_{N-1})^T V_N (A_{N-1} + B_{N-1}L_{N-1}))\boldsymbol{x}_{N-1}. \end{aligned} \tag{19}$$

Critically, this implies that the cost-to-go is always a positive semi-definite quadratic function of the state. Because the optimal policy is always linear, and the optimal cost-to-go is always quadratic, the DP recursion may be recursively performed backward in time and the minimization may be performed analytically.

Following the same procedure, we can write the DP recursion for the discrete-time LQR controller:

1. $V_N = Q_N$
2. $L_k = -(R_k + B_k^T V_{k+1} B_k)^{-1}(B_k^T V_{k+1} A_k)$
3. $V_k = Q_k + L_k^T R_k L_k + (A_k + B_k L_k)^T V_{k+1}(A_k + B_k L_k)$
4. $\boldsymbol{u}_k^* = L_k \boldsymbol{x}_k$
5. $J_k^*(\boldsymbol{x}_k) = \frac{1}{2}\boldsymbol{x}_k^T V_k \boldsymbol{x}_k$

There are several implications of this recurrence relation. First, even if A, B, Q, R are all constant (not time-varying), the policy is still time-varying. Why is this the case? Control effort invested early in the problem will yield dividends over the remaining length of the horizon, in terms of lower state cost for all future time steps. However, as the remaining length of the episode becomes shorter, this tradeoff is increasingly imbalanced, and the control effort will decrease. However, for a linear time-invariant system, if (A, B) is controllable, the feedback gain L_k approach a constant as the episode length approaches infinity. This time-invariant policy is practical for long horizon control problems, and may be approximately computed by running the DP recurrence relation until approximate convergence.

2.3.1 LQR with (Bi)linear Cost and Affine Dynamics

In the previous subsection, we have derived the common formulation of the LQR controller. In this subsection, we will derive the discrete time LQR controller for a more general system with bilinear/linear terms in the cost and affine terms in the dynamics. This derivation will be the basis of algorithms we will build up in the following subsections. More concretely, we consider systems with stage-wise cost

$$c(\boldsymbol{x}_k, \boldsymbol{u}_k) = \frac{1}{2}\boldsymbol{x}_k^T Q_k \boldsymbol{x}_k + \frac{1}{2}\boldsymbol{u}_k^T R_k \boldsymbol{u}_k + \boldsymbol{u}_k^T H_k \boldsymbol{x}_k + \boldsymbol{q}_k^T \boldsymbol{x}_k + \boldsymbol{r}_k \boldsymbol{u}_k + q_k, \tag{20}$$

terminal cost

$$c_N(\boldsymbol{x}_k) = \frac{1}{2}\boldsymbol{x}_k^T Q_N \boldsymbol{x}_k + \boldsymbol{q}_N^T \boldsymbol{x}_k + q_N, \tag{21}$$

and dynamics

$$\boldsymbol{x}_{k+1} = A_k \boldsymbol{x}_k + B_k \boldsymbol{u}_k + d_k. \tag{22}$$

The cost-to-go will take the form

$$J_k(\boldsymbol{x}_k) = \frac{1}{2}\boldsymbol{x}_k^T V_k \boldsymbol{x}_k + \boldsymbol{v}_k^T \boldsymbol{x}_k + v_k. \tag{23}$$

Repeating our approach from the last subsection, we have

$$J_k^*(\boldsymbol{x}_k) = \min_{\boldsymbol{u}_k \in \mathbb{R}^m} \{\frac{1}{2}\boldsymbol{x}_k^T Q_k \boldsymbol{x}_k + \frac{1}{2}\boldsymbol{u}_k^T R_k \boldsymbol{u}_k + \boldsymbol{u}_k^T H_k \boldsymbol{x}_k + \boldsymbol{q}_k^T \boldsymbol{x}_k + \boldsymbol{r}_k \boldsymbol{u}_k + q_k \tag{24}$$
$$+ \frac{1}{2}(A_k \boldsymbol{x}_k + B_k \boldsymbol{u}_k + \boldsymbol{d}_k)^T V_{k+1}(A_k \boldsymbol{x}_k + B_k \boldsymbol{u}_k + \boldsymbol{d}_k)$$
$$+ \boldsymbol{v}_{k+1}^T(A_k \boldsymbol{x}_k + B_k \boldsymbol{u}_k + \boldsymbol{d_k}) + v_{k+1}\}.$$

Rearranging, have

$$J_k^*(\boldsymbol{x}_k) = \min_{\boldsymbol{u}_k \in \mathbb{R}^m} \{\frac{1}{2}\boldsymbol{x}_k^T(Q_k + A_k^T V_{k+1} A_k)\boldsymbol{x}_k + \frac{1}{2}\boldsymbol{u}_k^T(R_k + B_k^T V_{k+1} B_k)\boldsymbol{u}_k \tag{25}$$
$$+ \boldsymbol{u}_k^T(H_k + B_k^T V_{k+1} A_k)^T \boldsymbol{x}_k + (\boldsymbol{q}_k + A_k^T V_{K+1} \boldsymbol{d}_k + A_k^T \boldsymbol{v}_{k+1})^T \boldsymbol{x}_k$$
$$+ (\boldsymbol{r}_k + B_k^T V_{k+1} \boldsymbol{d} + B_k^T \boldsymbol{v}_{k+1})\boldsymbol{u}_k + (v_{k+1} + \frac{1}{2}\boldsymbol{d}_k^T V_{k+1} \boldsymbol{d}_k + \boldsymbol{v}_{k+1}^T \boldsymbol{d}_k)\}.$$

Solving this minimization problem, we see that our optimal controller takes the form

$$\boldsymbol{u}_k^* = \boldsymbol{l}_k + L_k \boldsymbol{x}_k. \tag{26}$$

We will define the following useful terms which will be used throughout the remainder of this section

$$S_{\boldsymbol{u},k} = \boldsymbol{r}_k + B_k^T \boldsymbol{v}_{k+1} + B_k^T V_{k+1} \boldsymbol{d}_k \tag{27}$$
$$S_{\boldsymbol{uu},k} = R_k + B_k^T V_{k+1} B_k \tag{28}$$
$$S_{\boldsymbol{ux},k} = H_k + B_k^T V_{k+1} A_k. \tag{29}$$

Given this notation, all necessary terms can be computed via the following relations

1. $V_N = Q_N$; $\boldsymbol{v}_N = \boldsymbol{q}_N$; $v_N = q_N$
2. $$L_k = -S_{\boldsymbol{uu},k}^{-1} S_{\boldsymbol{ux},k} \tag{30}$$
$$\boldsymbol{l}_k = -S_{\boldsymbol{uu},k}^{-1} S_{\boldsymbol{u},k} \tag{31}$$
3. $$V_k = Q_k + A_k^T V_{k+1} A_k - L_k^T S_{\boldsymbol{uu},k} L_k \tag{32}$$
$$\boldsymbol{v}_k = \boldsymbol{q}_k + A_k^T(\boldsymbol{v}_{k+1} + V_{k+1}\boldsymbol{d}_k) + S_{\boldsymbol{ux},k}\boldsymbol{l}_k + L_k^T(S_{\boldsymbol{u},k} + S_{\boldsymbol{uu},k}\boldsymbol{l}_k) \tag{33}$$
$$v_k = v_{k+1} + q_k + \boldsymbol{d}_k^T \boldsymbol{v}_{k+1} + \frac{1}{2}\boldsymbol{d}_k^T V_{k+1}\boldsymbol{d}_k + \boldsymbol{l}_k^T(S_{\boldsymbol{u},k} + \frac{1}{2}S_{\boldsymbol{uu},k}\boldsymbol{l}_k) \tag{34}$$
4. $\boldsymbol{u}_k^* = \boldsymbol{l}_k + L_k \boldsymbol{x}_k$
5. $J_k(\boldsymbol{x}_k) = \frac{1}{2}\boldsymbol{x}_k^T V_k \boldsymbol{x}_k + \boldsymbol{v}_k^T \boldsymbol{x}_k + v_k$.

Note that in the following subsections (specifically in our discussion of differential dynamic programming) we will introduce more convenient (and compact) notation.

2.3.2 LQR Tracking around a Linear Trajectory

In the previous subsections we have considered the generic linear quadratic control problem, in which we want to regulate to a fixed point, and deviations from this point are penalized. In this section, we will address the case in which we want to track a pre-specified trajectory. Let us assume (for now) that we have been given a nominal trajectory of the form $(\bar{\boldsymbol{x}}_0, \dots, \bar{\boldsymbol{x}}_N)$ and $(\bar{\boldsymbol{u}}_0, \dots, \bar{\boldsymbol{u}}_{N-1})$. We will also assume that this trajectory satisfies our given dynamics, such that

$$\bar{\boldsymbol{x}}_{k+1} = A_k \bar{\boldsymbol{x}}_k + B_k \bar{\boldsymbol{u}}_k + \boldsymbol{d}_k, \ \forall k = 0, \dots, N-1. \tag{35}$$

Then, we can rewrite our dynamics in terms of deviations from the nominal trajectory,

$$\delta \boldsymbol{x}_k = \boldsymbol{x}_k - \bar{\boldsymbol{x}}_k \tag{36}$$

$$\delta \boldsymbol{u}_k = \boldsymbol{u}_k - \bar{\boldsymbol{u}}_k. \tag{37}$$

Rewriting, we have

$$\delta \boldsymbol{x}_{k+1} = A_k \delta \boldsymbol{x}_k + B_k \delta \boldsymbol{u}_k. \tag{38}$$

Thus, tracking the nominal trajectory reduces to driving the state deviation, $\delta \boldsymbol{x}_k$, to zero. Note that solving this problem requires rewriting the original cost function in terms of the deviations $\delta \boldsymbol{x}_k, \delta \boldsymbol{u}_k$.

2.3.3 LQR Tracking around a Nonlinear Trajectory

Despite LQR being an extremely powerful approach to optimal control, it suffers from a handful of limitations. First and foremost, it assumes the dynamics are (possibly time-varying) linear, and the cost function is quadratic. While most systems are in fact nonlinear, a typical approach to designing feedback controllers is to linearize around some operating point. This is an effective method for designing regulators, which aim to control the system to some particular state. If, in contrast, we wish to track a trajectory, we must instead linearize around this trajectory. We will assume we are given a nominal trajectory which satisfies the nonlinear dynamics, such that

$$\bar{\boldsymbol{x}}_{k+1} = f(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k), \ \forall k = 0, \dots, N-1. \tag{39}$$

Given this, we can linearize our system at each timestep by Taylor expanding,

$$\boldsymbol{x}_{k+1} \approx f(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) + \underbrace{\frac{\partial f}{\partial \boldsymbol{x}}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)}_{A_k}(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k) + \underbrace{\frac{\partial f}{\partial \boldsymbol{u}}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)}_{B_k}(\boldsymbol{u}_k - \bar{\boldsymbol{u}}_k). \tag{40}$$

which allows us to again rewrite the system in terms of deviations, to get

$$\delta \boldsymbol{x}_{k+1} = A_k \delta \boldsymbol{x}_k + B_k \delta \boldsymbol{u}_k \tag{41}$$

which is linear in $\delta \boldsymbol{x}_k, \delta \boldsymbol{u}_k$. Note that design of systems of this type often require careful design and analysis, as deviating from the nominal trajectory results in the loss of accuracy of the local model linearization.

In designing this tracking system, a second question now occurs: how do we choose our cost function? One possible option is arbitrary choice of Q and R by the system designer. This has the advantage of being easily customizable to change system behavior, and we can guarantee the necessary conditions on these matrices. A second option, if we are given some arbitrary (possibly non-quadratic) cost function c, is to locally quadratize the cost function. Writing

$$c_k := c(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) \tag{42}$$

$$c_{i,k} := \frac{\partial c}{\partial i}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) \tag{43}$$

$$c_{ij,k} := \frac{\partial^2 c}{\partial i \partial j}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) \tag{44}$$

we can second order Taylor expand our cost function around our nominal trajectory

$$c(\delta \boldsymbol{x}_k, \delta \boldsymbol{u}_k) \approx \frac{1}{2} \begin{bmatrix} 1 \\ \delta \boldsymbol{x}_k \\ \delta \boldsymbol{u}_k \end{bmatrix}^T \begin{bmatrix} 2c_k & c_{\boldsymbol{x},k}^T & c_{\boldsymbol{u},k}^T \\ c_{\boldsymbol{x},k} & c_{\boldsymbol{xx},k} & c_{\boldsymbol{ux},k}^T \\ c_{\boldsymbol{x},k} & c_{\boldsymbol{ux},k} & c_{\boldsymbol{uu},k} \end{bmatrix} \begin{bmatrix} 1 \\ \delta \boldsymbol{x}_k \\ \delta \boldsymbol{u}_k \end{bmatrix}. \tag{45}$$

Here $c_{\boldsymbol{xx},k}$ and $c_{\boldsymbol{uu},k}$ replace Q_k and R_k from the previous section, respectively. There are two primary concerns with this approach to choosing the cost function. First, we require the quadratic form in (45) to be positive semi-definite and $c_{\boldsymbol{uu},k}$ to be positive definite, for all k. Second, we have an implicit cost that we would like to stay close to the nominal trajectory to ensure our linearized model does not become inaccurate. As a result of this implicit cost, we may wish to tune the cost terms to yield tracking that is better suited to the nonlinear model that we are tracking.

2.4 Iterative LQR and Differential Dynamic Programming

2.4.1 Iterative LQR

We have addressed the case in which we wish to track a given trajectory with LQR. A natural question, now, is whether we can use LQR to improve on this nominal trajectory? Iterative LQR augments tracking LQR with a forward pass in which the nominal trajectory is updated. As a consequence, it can be used to improve trajectories and in most cases, can be used as a practical trajectory generation and control algorithm for nonlinear systems. We will define the following useful terms

$$Q_k = c_k + v_{k+1} \tag{46}$$

$$Q_{\boldsymbol{x},k} = c_{\boldsymbol{x},k} + f_{\boldsymbol{x},k}^T \boldsymbol{v}_{k+1} \tag{47}$$

$$Q_{\boldsymbol{u},k} = c_{\boldsymbol{u},k} + f_{\boldsymbol{u},k}^T \boldsymbol{v}_{k+1} \tag{48}$$

$$Q_{\boldsymbol{xx},k} = c_{\boldsymbol{xx},k} + f_{\boldsymbol{x},k}^T V_{k+1} f_{\boldsymbol{x},k} \tag{49}$$

$$Q_{\boldsymbol{uu},k} = c_{\boldsymbol{uu},k} + f_{\boldsymbol{u},k}^T V_{k+1} f_{\boldsymbol{u},k} \tag{50}$$

$$Q_{\boldsymbol{ux},k} = c_{\boldsymbol{ux},k} + f_{\boldsymbol{u},k}^T V_{k+1} f_{\boldsymbol{x},k} \tag{51}$$

Algorithm 1 iLQR

Require: Nominal control sequence, $(\bar{\boldsymbol{u}}_0, \ldots, \bar{\boldsymbol{u}}_{N-1})$

$\delta \boldsymbol{u}_k = 0$ for all k

while not converged **do**

Forward pass:

  Compute nominal trajectory $\bar{\boldsymbol{x}}_{k+1} = f(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k + \delta \boldsymbol{u}_k)$ and set $\bar{\boldsymbol{u}}_k \leftarrow \bar{\boldsymbol{u}}_k + \delta \boldsymbol{u}_k$

Backward pass:

  Compute Q terms around $(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)$ for all k via (46 – 51)

  Update feedback law via (53 – 54)

  Update value approximation via (55 – 57)

end while

Compute control law $\pi_k(\boldsymbol{x}_k) = \bar{\boldsymbol{u}}_k + \boldsymbol{l}_k + L_k(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k)$

return $\{\pi_k\}_{k=0}^{N-1}$

where $f_{\boldsymbol{x},k} = A_k$ and $f_{\boldsymbol{u},k} = B_k$. In this form, the optimal control perturbation is

$$\delta \boldsymbol{u}_k^* = \boldsymbol{l}_k + L_k \delta \boldsymbol{x}_k \tag{52}$$

where

$$\boldsymbol{l}_k = -Q_{\boldsymbol{uu},k}^{-1} Q_{\boldsymbol{u},k} \tag{53}$$

$$L_k = -Q_{\boldsymbol{uu},k}^{-1} Q_{\boldsymbol{ux},k}. \tag{54}$$

Finally, the local backward recursion can be completed by updating the value function terms via

$$V_k = Q_k - \frac{1}{2} \boldsymbol{l}_k^T Q_{\boldsymbol{uu},k} \boldsymbol{l}_k \tag{55}$$

$$V_{\boldsymbol{x}_k} = Q_{\boldsymbol{x},k} - L_k^T Q_{\boldsymbol{uu},k} \boldsymbol{l}_k \tag{56}$$

$$V_{\boldsymbol{xx},k} = Q_{\boldsymbol{xx},k} - L_k^T Q_{\boldsymbol{uu},k} L_k. \tag{57}$$

So far, we have simply derived an alternative method for performing a quadratic approximation of the DP recursion around some nominal trajectory. The iterative LQR (iLQR) algorithm differs by introducing a forward pass that updates the trajectory that is being tracked. The algorithm alternates between forward passes, in which the control policy is applied to the nonlinear dynamics, and backward passes in which the cost function and dynamics are linearized around the new nominal trajectory, and the quadratic approximation of the value, as well as the new control law, is computed. The iterative LQR algorithm is outlined in Algorithm 1. Critically, note that this algorithm returns both a nominal trajectory, in terms of the $\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k$, as well as a feedback policy that stabilizes around this trajectory.

2.4.2 Differential Dynamic Programming

Iterative LQR performs trajectory optimization by first linearizing the dynamics and quadratizing the cost function, and then performing the dynamic programming recursion to compute

optimal controls. While this linearization/quadratization approach is sufficient for approximating the Bellman equation such that it may be solved analytically, an alternative approach is to directly approximate the Bellman equation. *Differential dynamic programming* (DDP) directly builds a quadratic approximation of the right hand side of the Bellman equation (as opposed to first approximating the dynamics and the cost function), which may then be solved analytically. We will first define the change in the value of J_k under a perturbation $\delta \boldsymbol{x}_k, \delta \boldsymbol{u}_k$,

$$Q(\delta \boldsymbol{x}_k, \delta \boldsymbol{u}_k) := c(\bar{\boldsymbol{x}}_k + \delta \boldsymbol{x}_k, \bar{\boldsymbol{u}}_k + \delta \boldsymbol{u}_k) + J_{k+1}(f(\bar{\boldsymbol{x}}_k + \delta \boldsymbol{x}_k, \bar{\boldsymbol{u}}_k + \delta \boldsymbol{u}_k)). \tag{58}$$

Note that Q here is different from the Q matrix in Section 2.3. Using the same notation as in (42), we can write the quadratic expansion of (58) as

$$Q(\delta \boldsymbol{x}_k, \delta \boldsymbol{u}_k) \approx \frac{1}{2} \begin{bmatrix} 1 \\ \delta \boldsymbol{x}_k \\ \delta \boldsymbol{u}_k \end{bmatrix}^T \begin{bmatrix} 2Q_k & Q_{\boldsymbol{x},k}^T & Q_{\boldsymbol{u},k}^T \\ Q_{\boldsymbol{x},k} & Q_{\boldsymbol{x}\boldsymbol{x},k} & Q_{\boldsymbol{u}\boldsymbol{x},k}^T \\ Q_{\boldsymbol{x},k} & Q_{\boldsymbol{u}\boldsymbol{x},k} & Q_{\boldsymbol{u}\boldsymbol{u},k} \end{bmatrix} \begin{bmatrix} 1 \\ \delta \boldsymbol{x}_k \\ \delta \boldsymbol{u}_k \end{bmatrix} \tag{59}$$

where

$$Q_k = c_k + v_{k+1} \tag{60}$$
$$Q_{\boldsymbol{x},k} = c_{\boldsymbol{x},k} + f_{\boldsymbol{x},k}^T \boldsymbol{v}_{k+1} \tag{61}$$
$$Q_{\boldsymbol{u},k} = c_{\boldsymbol{u},k} + f_{\boldsymbol{u},k}^T \boldsymbol{v}_{k+1} \tag{62}$$
$$Q_{\boldsymbol{x}\boldsymbol{x},k} = c_{\boldsymbol{x}\boldsymbol{x},k} + f_{\boldsymbol{x},k}^T V_{k+1} f_{\boldsymbol{x},k} + \boldsymbol{v}_{k+1} \cdot f_{\boldsymbol{x}\boldsymbol{x},k} \tag{63}$$
$$Q_{\boldsymbol{u}\boldsymbol{u},k} = c_{\boldsymbol{u}\boldsymbol{u},k} + f_{\boldsymbol{u},k}^T V_{k+1} f_{\boldsymbol{u},k} + \boldsymbol{v}_{k+1} \cdot f_{\boldsymbol{u}\boldsymbol{u},k} \tag{64}$$
$$Q_{\boldsymbol{u}\boldsymbol{x},k} = c_{\boldsymbol{u}\boldsymbol{x},k} + f_{\boldsymbol{u},k}^T V_{k+1} f_{\boldsymbol{x},k} + \boldsymbol{v}_{k+1} \cdot f_{\boldsymbol{u}\boldsymbol{x},k}. \tag{65}$$

Note that these terms differ only from iLQR via the last term in (63 – 65), which are second order approximation of the dynamics. Note that the dot notation denotes tensor contraction.

Given this, we can partially minimize this quadratic form over the control deviation,

$$\delta \boldsymbol{u}_k^* = \text{argmin}_{\delta \boldsymbol{u}} Q(\delta \boldsymbol{x}_k, \delta \boldsymbol{u}) = \boldsymbol{l}_k + L_k \delta \boldsymbol{x}_k \tag{66}$$

where

$$\boldsymbol{l}_k = -Q_{\boldsymbol{u}\boldsymbol{u},k}^{-1} Q_{\boldsymbol{u},k} \tag{67}$$
$$L_k = -Q_{\boldsymbol{u}\boldsymbol{u},k}^{-1} Q_{\boldsymbol{u}\boldsymbol{x},k}. \tag{68}$$

The DDP algorithm is identical to Algorithm 1, just with the alternative definitions for $Q_{\boldsymbol{x}\boldsymbol{x},k}, Q_{\boldsymbol{u}\boldsymbol{u},k}$ and $Q_{\boldsymbol{u}\boldsymbol{x},k}$. The main philosophical difference between iLQR and DDP is that iLQR first approximates the dynamics and cost, and then solves the Bellman equation directly, whereas DDP directly approximates the Bellman equation. While DDP yields a more accurate approximation, computing the second order dynamics terms is expensive in practice. Practically, iLQR is sufficient for most applications.

2.4.3 Algorithmic Details for iLQR and DDP

Algorithm 1 leaves out several details that would be critical for implementing the algorithm. First, what convergence criteria should we use? In [TL05], the authors stop when the update to the nominal control action sequence is sufficiently small. In [LK14], the authors iterate until the cost of the trajectory (with some additional penalty terms) increases. Finally, a variety of convergence criteria are based on expected trajectory improvement, computed via line search [JM70, TET12]. In the forward pass, standard iLQR computes an updated nominal control sequence via $\bar{\boldsymbol{u}}_k \leftarrow \bar{\boldsymbol{u}}_k + \boldsymbol{l_k} + L_k \delta \boldsymbol{x}_k$. Instead we can weight $\boldsymbol{l_k}$ with a scalar $\alpha \in [0, 1]$ for which we perform line search. This results in increased stability (as with standard line search for step size determination in nonlinear optimization) and possibly faster convergence. When α is close to zero, or alternative conditions (such as expected improvement being small) are met, we terminate. For a further discussion of this approach, we refer the reader to [TET12], which also features a discussion of step size determination in the DDP literature.

Iterative LQR and DDP rely on minimizing a second order approximation of the cost-to-go perturbation. However, we do not have any guarantees on the convexity of $Q(\delta \boldsymbol{x}_k, \delta \boldsymbol{u}_k)$ for arbitrary cost functions. Note that DDP is performing a Newton step [LS92] (iLQR is performing a Newton step with an approximation of the Hessian) via decomposing the optimization problem over controls into N smaller optimization problems. As such, standard approaches from Newton methods for regularization have been applied, such as replacing $Q_{\boldsymbol{uu},k}$ with $Q_{\boldsymbol{uu},k} + \mu I$, which is convex for sufficiently large μ. Alternative approaches have been explored in [TET12, TMT14], based on regularizing the quadratic term in the approximate cost-to-go.

Both iLQR and DDP are local methods. Full dynamic programming approaches yield globally optimal feedback policies. In contrast, iLQR and DDP yield nominal trajectories and local stabilizing controllers. However, these local controllers are often sufficient for tracking the trajectory. As they are local method, choice of initial control sequence is important, and poor choice may result in poor convergence. Additionally, we have not considered constraints on either state or action in the derivation of iLQR or DDP. This is currently an active area of research [XLH17, TMT14, GB17].

2.5 LQG

2.6 Further Reading

References

[GB17] Markus Giftthaler and Jonas Buchli. A projection approach to equality constrained iterative linear quadratic optimal control. In *IEEE-RAS International Conference on Humanoid Robotics (Humanoids)*, 2017.

[JM70] David Jacobson and David Mayne. *Differential Dynamic Programming*. Elsevier, 1970.

[LK14] Sergey Levine and Vladlen Koltun. Learning complex neural network policies with trajectory optimization. In *International Conference on Machine Learning (ICML)*, 2014.

[LS92] Li-zhi Liao and Christine A Shoemaker. Advantages of differential dynamic programming over newton's method for discrete-time optimal control problems. Technical report, Cornell University, 1992.

[TET12] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[TL05] Emanuel Todorov and Weiwei Li. A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conference (ACC)*, 2005.

[TMT14] Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-limited differential dynamic programming. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[XLH17] Zhaoming Xie, C Karen Liu, and Kris Hauser. Differential dynamic programming with nonlinear constraints. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.