AA203: Optimal and Learning-based Control Combined Course Notes

James Harrison*

April 13, 2019

Introduction

These notes accompany the newly revised (Spring 2019) version of AA203 at Stanford. The goal of this new course is to present a unified treatment of optimal control and reinforcement learning (RL), with an emphasis on model-based reinforcement learning. The goal of the instructors is to unify the subjects as much as possible, and to concretize connections between these research communities.

How is this course different from a standard class on Optimal Control? First, we will emphasize practical computational tools for real world optimal control problems, such as model predictive control and sequential convex programming. Beyond this, the last third of the course focuses on the case in which an exact model of the system is not available. We will discuss this setting both in the online context (typically referred to as adaptive optimal control) and in the episodic context (the typical setting for reinforcement learning).

How is this course different from a standard class on Reinforcement Learning? Many classes on reinforcement learning focus primarily on the setting of discrete Markov Decision Processes (MDPs), whereas we will focus primarily on continuous MDPs. More importantly, the focus on discrete MDPs leads planning with a known model (which is typically referred to as "planning" or "control" in RL) to be relatively simple. In this course, we will spend considerably more time focusing on planning with a known model in both continuous and discrete time. Finally, the focus of this course will primarily be on model-based methods. We will touch briefly on model-free methods at the end, and combinations of model-free and model-based approaches.

*Contact: jharrison@stanford.edu

A Note on Notation

The notation and language used in the control theory and reinforcement learning communities vary substantially, as so we will state all of the notational choices we make in this section. First, optimal control problems are typically stated in terms of minimizing a cost function, whereas reinforcement learning problems aim to maximize a reward. These are mathematically identical statements, where one is simply the negation of the other. Herein, we will use the control theoretic approach of cost minimization. We write c for the cost function, f for the system dynamics, and denote the state and action at time t as $\boldsymbol{x}_t$ and $\boldsymbol{u}_t$ respectively. We write scalars as lower case letters, vectors as bold lower case letters, and matrices as upper case letters. We write a deterministic policy as $\pi(\boldsymbol{x})$, and a stochastic policy as $\pi(\boldsymbol{u} \mid \boldsymbol{x})$. We write the cost-to-go (negation of the value function) associated with policy π at time t and state $\boldsymbol{x}$ as $J_t^\pi(\boldsymbol{x})$. We will also sometimes refer to the cost-to-go as the value, but in these notes we are always referring to the expected sum of future costs. For an in-depth discussion of the notational and language differences between the artificial intelligence and control theory communities, we refer the reader to [Pow12].

For notational convenience, we will write the Hessian of a function $f(x)$, evaluated at x^*, as $\nabla^2 f(x^*)$.

Prerequisites

While these notes aim to be almost entirely self contained, familiarity with undergraduate level calculus, differential equations, and linear algebra (equivalent to CME 102 and EE 263 at Stanford) are assumed. We will briefly review nonlinear optimization in the first section of these notes, but previous experience with optimization (e.g. EE 364A) will be helpful. Finally, previous experience with machine learning (at the level of CS 229) is beneficial.

Contents

1 Nonlinear Optimization **3**

1.1 Unconstrained Nonlinear Optimization 3

1.1.1 Necessary Conditions for Optimality 3

1.1.2 Sufficient Conditions for Optimality 4

1.1.3 Special case: Convex Optimization 5

1.1.4 Computational Methods 5

1.2 Constrained Nonlinear Optimization 8

1.2.1 Equality Constrained Optimization 8

1.2.2 Inequality Constrained Optimization 10

1.3 Further Reading 11

2 Dynamic Programming and the Linear Quadratic Regulator **11**

2.1 The Optimal Control Problem 11

2.2 Dynamic Programming and the Principle of Optimality 12

2.3 Discrete LQR 14
2.3.1 LQR with (Bi)linear Cost and Affine Dynamics 16
2.3.2 LQR Tracking around a Linear Trajectory 17
2.3.3 LQR Tracking around a Nonlinear Trajectory 18
2.4 Iterative LQR and Differential Dynamic Programming 19
2.4.1 Iterative LQR 19
2.4.2 Differential Dynamic Programming 20
2.4.3 Algorithmic Details for iLQR and DDP 21
2.5 LQG 22
2.6 Further Reading 22

1 Nonlinear Optimization

In this section we discuss the generic nonlinear optimization problem that forms the basis for the rest of the material presented in this class. We write the minimization problem as

$$\min_{\boldsymbol{x}\in\mathcal{X}} \quad f(\boldsymbol{x})$$

where f is the cost function, usually assumed twice continuously differentiable, $x \in \mathbb{R}^n$ is the optimization variable, and $\mathcal{X} \subset \mathbb{R}^n$ is the constraint set. The special case in which the cost function is linear and the constraint set is specified by linear equations and/or inequalities is *linear optimization*, which we will not discuss.

1.1 Unconstrained Nonlinear Optimization

We will first address the unconstrained case, in which $\mathcal{X} = \mathbb{R}^n$. A vector $\boldsymbol{x}^*$ is said to be an unconstrained *local minimum* if there exists $\epsilon > 0$ such that $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \{\boldsymbol{x} \mid \|\boldsymbol{x} - \boldsymbol{x}^*\| \leq \epsilon\}$, and $\boldsymbol{x}^*0$ is said to be an unconstrained *global minimum* if $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ for all $x \in \mathbb{R}^n$.

1.1.1 Necessary Conditions for Optimality

For a differentiable cost function, we can compare the cost of a point to its neighbors by considering a small variation $\Delta\boldsymbol{x}$ from $\boldsymbol{x}^*$. By using Taylor expansions, this yields a first order cost variation

$$f(\boldsymbol{x}^* + \Delta\boldsymbol{x}) - f(\boldsymbol{x}^*) \approx \nabla f(\boldsymbol{x}^*)^T \Delta\boldsymbol{x} \tag{1}$$

and a second order cost variation

$$f(\boldsymbol{x}^* + \Delta\boldsymbol{x}) - f(\boldsymbol{x}^*) \approx \nabla f(\boldsymbol{x}^*)^T \Delta\boldsymbol{x} + \frac{1}{2}\Delta\boldsymbol{x}^T \nabla^2 f(\boldsymbol{x}^*)\Delta\boldsymbol{x}. \tag{2}$$

Setting $\Delta\boldsymbol{x}$ to be equal to positive and negative multiples of the unit coordinate vector, we have

$$\frac{\partial f(\boldsymbol{x}^*)}{\partial x_i} \geq 0 \tag{3}$$

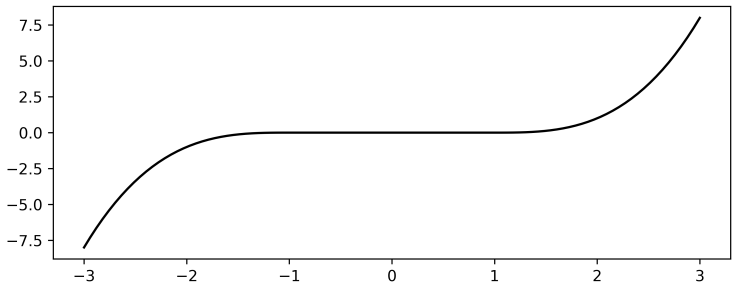


Figure 1: An example of a function for which the necessary conditions of optimality are satisfied but the sufficient conditions are not.

where x_i denotes the i'th coordinate of $\boldsymbol{x}$, and

$$\frac{\partial f(\boldsymbol{x}^*)}{\partial x_i} \leq 0 \tag{4}$$

for all i, which is only satisfied by $\nabla f(\boldsymbol{x}^*) = 0$. This is referred to as the *first order necessary condition for optimality*. Looking at the second order variation, and noting that $f(\boldsymbol{x}^*) = 0$, we expect

$$f(\boldsymbol{x}^* + \Delta\boldsymbol{x}) - f(\boldsymbol{x}^*) \geq 0 \tag{5}$$

and thus

$$\Delta\boldsymbol{x}^T \nabla^2 f(\boldsymbol{x}^*) \Delta\boldsymbol{x} \geq 0 \tag{6}$$

which implies $\nabla^2 f(\boldsymbol{x}^*)$ is positive semidefinite. This is referred to as the *second order necessary condition for optimality*. Stating these conditons formally,

Theorem 1.1 (Necessary Conditions for Optimality (NOC))**.** *Let $\boldsymbol{x}^*$ be an unconstrained local minimum of $f : \mathbb{R}^n \to \mathbb{R}$ and $f \in C^1$ in an open set S containing $\boldsymbol{x}^*$. Then,*

$$\nabla f(\boldsymbol{x}^*) = 0. \tag{7}$$

If $f \in C^2$ within S, $\nabla^2 f(\boldsymbol{x}^*)$ is positive semidefinite.

Proof. See section 1.1 of [Ber16].

1.1.2 Sufficient Conditions for Optimality

If we strengthen the second order condition to $\nabla^2 f(\boldsymbol{x}^*)$ being positive definite, we have the sufficient conditions for $\boldsymbol{x}^*$ being a local minimum. Why is the second order necessary conditions not sufficient? An example function is given in figure 1. Formally,

Theorem 1.2 (Sufficient Conditions for Optimality (SOC)). *Let $f : \mathbb{R}^n \to \mathbb{R}$ be C^2 in an open set S. Suppose a vector $\boldsymbol{x}^*$ satisfies the conditions $\nabla f(\boldsymbol{x}^*) = 0$ and $\nabla^2 f(\boldsymbol{x}^*)$ is positive definite. Then $\boldsymbol{x}^*$ is a strict unconstrained local minimum of f.*

Proof is again given in Section 1.1 of [Ber16]. There are several reasons why the optimality conditions are important. In a general nonlinear optimization setting, they can be used to filter candidates for global minima. They can be used for sensitivity analysis, in which the sensitivity of $\boldsymbol{x}^*$ to model parameters can be quantified [Ber16]. This is common in e.g. microeconomics. Finally, these conditions often provide the basis for the design and analysis of optimization algorithms.

1.1.3 Special case: Convex Optimization

A special case within nonlinear optimization is the set of *convex optimization* problems. A set $S \subset \mathbb{R}^n$ is called *convex* if

$$\alpha \boldsymbol{x} + (1-\alpha)\boldsymbol{y} \in S, \quad \forall \boldsymbol{x}, \boldsymbol{y} \in S, \forall \alpha \in [0,1]. \tag{8}$$

For S convex, a function $f : S \to \mathbb{R}$ is called convex if

$$f(\alpha \boldsymbol{x} + (1-\alpha)\boldsymbol{y}) \leq \alpha f(\boldsymbol{x}) + (1-\alpha) f(\boldsymbol{y}). \tag{9}$$

This class of problems has several important characteristics. If f is convex, then

- A local minimum of f over S is also a global minimum over S. If in addition f is strictly convex (the inequality in (9) is strict), there exists at most one global minimum of f.
- If $f \in C^1$ and convex, and the set S is open, $\nabla f(\boldsymbol{x}^*) = 0$ is a necessary and sufficient condition for a vector $\boldsymbol{x}^* \in S$ to be a global minimum over S.

Convex optimization problems have several nice properties that make them (usually) computationally efficient to solve, and the first property above gives a certificate of having obtained global optimality that is difficult or impossible to obtain in the general nonlinear optimization setting. For a thorough treatment of convex optimization theory and algorithms, see [BV04].

1.1.4 Computational Methods

In this subsection we will discuss the class of algorithms known as *gradient methods* for finding local minima in nonlinear optimization problems. These approaches, rely (roughly) on following the gradient of the function "downhill", toward the minima. More concretely, these algorithms rely on taking steps of the form

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha^k \boldsymbol{d}^k \tag{10}$$

where if $\nabla f(\boldsymbol{x}) \neq 0$, $\boldsymbol{d}^k$ is chosen so that

$$\nabla f(\boldsymbol{x})^T \boldsymbol{d}^k < 0 \tag{11}$$

and $\alpha > 0$. Typically, the step size α^k is chosen such that

$$f(\boldsymbol{x}^k + \alpha^k \boldsymbol{d}^k) < f(\boldsymbol{x}^k), \tag{12}$$

but generally, the step size and the direction of descent ($\boldsymbol{d}^k$) are tuning parameters.

We will look at the general class of descent directions of the form

$$\boldsymbol{d}^k = -D^k \nabla f(\boldsymbol{x}^k) \tag{13}$$

where $D^k > 0$ (note that this guarantees $\nabla f(\boldsymbol{x}^k)^T \boldsymbol{d}^k < 0$).

Steepest descent, $D^k = I$. The simplest choice of descent direction is directly following the gradient, and ignoring second order function information. In practice, this often leads to slow convergence (figure 2a) and possible oscillation (figure 2b).

Newton's Method, $D^k = (\nabla^2 f(\boldsymbol{x}^k))^{-1}$. The underlying idea of this approach is to at each iteration, minimize the quadratic approximation of f around $\boldsymbol{x}^k$,

$$f^k(\boldsymbol{x}) = f(\boldsymbol{x}^k) + \nabla f(\boldsymbol{x}^k)^T(\boldsymbol{x} - \boldsymbol{x}^k) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^k)^T \nabla^2 f(\boldsymbol{x}^k)(\boldsymbol{x} - \boldsymbol{x}^k). \tag{14}$$

Setting the derivative of this to zero, we obtain

$$\nabla f(\boldsymbol{x}^k) + \nabla^2 f(\boldsymbol{x}^k)(\boldsymbol{x} - \boldsymbol{x}^k) = 0 \tag{15}$$

and thus, by setting $\boldsymbol{x}^{k+1}$ to be the $\boldsymbol{x}$ that satisfies the above, we get the

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - (\nabla^2 f(\boldsymbol{x}^k))^{-1} \nabla f(\boldsymbol{x}^k) \tag{16}$$

or more generally,

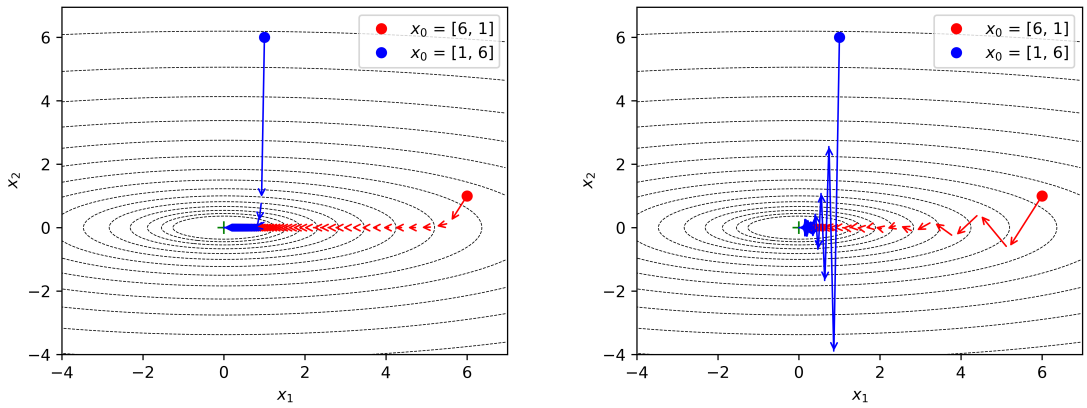
$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \alpha(\nabla^2 f(\boldsymbol{x}^k))^{-1} \nabla f(\boldsymbol{x}^k). \tag{17}$$

Note that this update is only valid for $\nabla^2 f(\boldsymbol{x}^k) > 0$. When this condition doesn't hold, $\boldsymbol{x}^{k+1}$ is not a minimizer of the second order approximation (as a result of the SOCs). See figure 2d for an example where Newton's method converges in one step, as a result of the cost function being quadratic.

Diagonally scaled steepest descent, $D^k = \mathbf{diag}(d_1^k, \ldots, d_n^k)$. Have $d_i^k > 0 \forall i$. A popular choice is

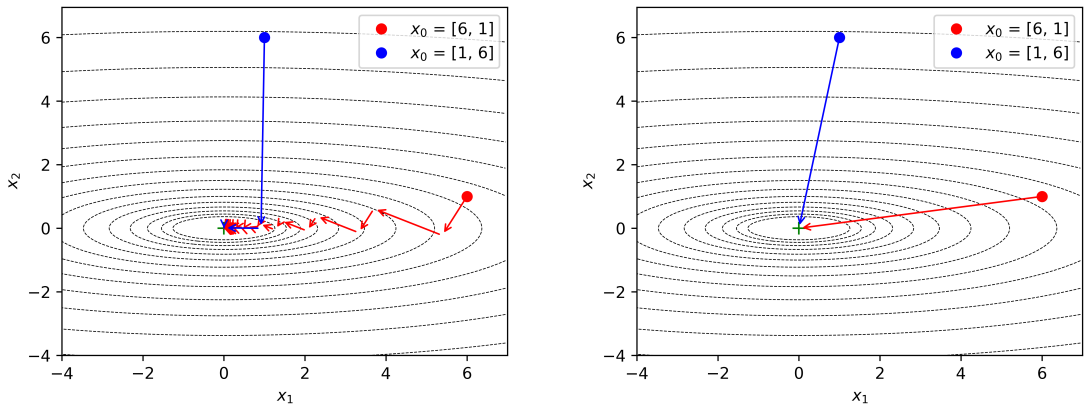
$$d_i^k = \left(\frac{\partial^2 f(\boldsymbol{x}^k)}{\partial x_i^2} \right)^{-1} \tag{18}$$

which is a diagonal approximation of the Hessian.



(a) Steepest descent, small fixed step size.

(b) Steepest descent, large fixed step size.



(c) Steepest descent, step size chosen via line search.

(d) Newton's method. Note that the method converges in one step.

Figure 2: Comparison of steepest descent methods with various step sizes, and Newton's method, on the same quadratic cost function.

Modified Newton's method, $D^k = (\nabla^2 f(\boldsymbol{x}^0))^{-1}$**.** Requires $\nabla^2 f(\boldsymbol{x}^0) > 0$. For cases in which one expects $\nabla^2 f(\boldsymbol{x}^0) \approx \nabla^2 f(\boldsymbol{x}^k)$, this removes having to compute the Hessian at each step.

In addition to choosing the descent direction, there also exist a variety of methods to choose the step size α. A computationally intensive but efficient (in terms of the number of steps taken) is using a minimization rule of the form

$$\alpha^k = \text{argmin}_{\alpha \geq 0} f(\boldsymbol{x}^k + \alpha \boldsymbol{d}^k) \tag{19}$$

which is usually solved via line search (figure 2c). Alternative approaches include a limited minimization rule, in which you constrain $\alpha^k \in [0, s]$ during the line search, or simpler approach such as a constant step size (which may not guarantee convergence), or a diminishing

scheduled step size. In this last case, schedules are typically chosen such that $\alpha^k \to 0$ as $k \to \infty$, while $\sum_{k=0}^{\infty} \alpha^k = +\infty$.

1.2 Constrained Nonlinear Optimization

In this section we will address the general constrained nonlinear optimization problem,

$$\min_{\boldsymbol{x} \in \mathcal{X}} \quad f(\boldsymbol{x})$$

which may equivalently be written

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{x} \in \mathcal{X} \end{aligned}$$

where the set $\mathcal{X}$ is usually specified in terms of equality and inequality constraints. To operate within this problem structure, we will develop a set of optimality conditions involving auxiliary variables called *Lagrange multipliers*.

1.2.1 Equality Constrained Optimization

We will first look at optimization with equality constraints of the form

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & h_i(\boldsymbol{x}) = 0, \quad i = 1, \ldots, m \end{aligned}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $h_i : \mathbb{R}^n \to \mathbb{R}$ are C^1. We will write $\boldsymbol{h} = [h_1, \ldots, h_m]^T$. For a given local minimum $\boldsymbol{x}^*$, there exist scalars $\lambda_1, \ldots, \lambda_m$ called Lagrange multipliers such that

$$\nabla f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i \nabla h_i(\boldsymbol{x}^*) = 0. \tag{20}$$

There are several possible interpretations for Lagrange multipliers. First, note that the cost gradient $f(\boldsymbol{x}^*)$ is in the subspace spanned by the constraint gradients at $\boldsymbol{x}^*$. Equivalently, $f(\boldsymbol{x}^*)$ is orthogonal to the subspace of first order feasible variations

$$V(\boldsymbol{x}^*) = \{\Delta \boldsymbol{x} \mid \nabla h_i(\boldsymbol{x}^*)^T \Delta \boldsymbol{x}, i = 1, \ldots, m\}. \tag{21}$$

This subspace is the space of variations $\Delta \boldsymbol{x}$ for which $\boldsymbol{x} = \boldsymbol{x}^* + \Delta \boldsymbol{x}$ satisfies the constraint $\boldsymbol{h}(\boldsymbol{x}) = 0$ up to first order. Therefore, at a local minimum, the first order cost variation $\nabla f(\boldsymbol{x}^*)^T \Delta \boldsymbol{x}$ is zero for all variations $\Delta \boldsymbol{x}$ in this space.

Given this informal understanding, we may now precisely state the necessary conditions for optimality in constrained optimization.

Theorem 1.3 (NOC for equality constrained optimization)**.** *Let $\boldsymbol{x}^*$ be a local minimum of f subject to $\boldsymbol{h}(\boldsymbol{x}) = 0$, and assume that the constraint gradients $\nabla h_1(\boldsymbol{x}^*), \ldots, \nabla h_m(\boldsymbol{x}^*)$ are*

linearly independent. Then there exists a unique vector $\boldsymbol{\lambda}^* = [\lambda_1^*, \ldots, \lambda_m^*]^T$ *called a Lagrange multiplier vector, such that*

$$\nabla f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i \nabla h_i(\boldsymbol{x}^*) = 0. \tag{22}$$

If in addition f *and* $\boldsymbol{h}$ *are* C^2*, we have*

$$\boldsymbol{y}^T(\nabla^2 f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i \nabla^2 h_i(\boldsymbol{x}^*))\boldsymbol{y} \geq 0, \quad \forall \boldsymbol{y} \in V(\boldsymbol{x}^*) \tag{23}$$

where

$$V(\boldsymbol{x}^*) = \{\boldsymbol{y} \mid \nabla h_i(\boldsymbol{x}^*)^T \boldsymbol{y} = 0, i = 0, \ldots, m\}. \tag{24}$$

Proof. See [Ber16] Section 3.1.1 and 3.1.2. □

We will sketch two possible proofs for the NOC for equality constrained optimization.

Penalty approach. This approach relies on adding a to the cost function a large penalty term for constraint violation. This is the same approach that will be used in proving the necessary conditions for inequality constrained optimization, and is the basis of a variety of practical numerical algorithms.

Elimination approach. This approach views the constraints as a system of m equations with n unknowns, for which m variables can be expressed in terms of the remaining $m - n$ variables. This reduces the problem to an unconstrained optimization problem.

Note that in theorem 1.3, we assumed the gradients of the constraint functions were linearly independent. A feasible vector for which this holds is called *regular*. If this condition is violated, a Lagrange multiplier for a local minimum may not exist.

For convenience, we will write the necessary conditions in terms of the Lagrangian function $L : \mathbb{R}^{m+n} \to \mathbb{R}$,

$$L(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i h_i(\boldsymbol{x}). \tag{25}$$

This function allows the NOC conditions to be succinctly stated as

$$\nabla_{\boldsymbol{x}} L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = 0 \tag{26}$$

$$\nabla_{\boldsymbol{\lambda}} L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = 0 \tag{27}$$

$$\boldsymbol{y}^T \nabla^2_{\boldsymbol{xx}} L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)\boldsymbol{y} \geq 0, \quad \forall \boldsymbol{y} \in V(\boldsymbol{x}^*). \tag{28}$$

which form a system of $n + m$ equations with $n + m$ unknowns. Given this notation, we can state the sufficient conditions.

Theorem 1.4 (SOC for equality constrained optimization)**.** *Assume that f and $\boldsymbol{h}$ are C^2 and let $\boldsymbol{x}^* \in \mathbb{R}^n$ and $\boldsymbol{\lambda}^* \in \mathbb{R}^m$ satisfy*

$$\nabla_{\boldsymbol{x}} L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = 0 \tag{29}$$

$$\nabla_{\boldsymbol{\lambda}} L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = 0 \tag{30}$$

$$\boldsymbol{y}^T \nabla^2_{\boldsymbol{xx}} L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) \boldsymbol{y} > 0, \quad \forall \boldsymbol{y} \neq 0, \boldsymbol{y} \in V(\boldsymbol{x}^*). \tag{31}$$

Proof. See [Ber16] Section 3.2. □

Note that the SOC does not include regularity of $\boldsymbol{x}^*$.

1.2.2 Inequality Constrained Optimization

We will now address the general case, including inequality constraints,

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & h_i(\boldsymbol{x}) = 0, \quad i = 0, \dots, m \\ & g_j(\boldsymbol{x}) \leq 0, \quad j = 0, \dots, r \end{aligned}$$

where f, h_i, g_i are C^1. The key intuition for the case of inequality constraints is based on realizing that for any feasible point, some subset of the constraints will be active (for which $g_j(\boldsymbol{x}) = 0$), while the complement of this set will be inactive. We define the active set of inequality constraints, which we denote

$$A(\boldsymbol{x}) = \{j \mid g_j(\boldsymbol{x} = 0\}. \tag{32}$$

A constraint is active at $\boldsymbol{x}$ if it is in $A(\boldsymbol{x})$, otherwise it is inactive. Note that if $\boldsymbol{x}^*$i is a local minimum of the inequality constrained problem, then $\boldsymbol{x}^*$ is a local minimum of the identical problem with the inactive constraints removed. Moreover, at this local minimum, the constraints may be treated as equality constraints. Thus, if $\boldsymbol{x}^*$ is regular, there exists Lagrange multipliers $\lambda_1^*, \dots, \lambda_m^*$ and $\mu_j^*, j \in A(\boldsymbol{x}^*)$ such that

$$\nabla f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i \nabla h_i(\boldsymbol{x}^*) + \sum_{j \in A(\boldsymbol{x}^*)} \mu_j^* \nabla g_j(\boldsymbol{x}^*) = 0. \tag{33}$$

We will define the Lagrangian

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i h_i(\boldsymbol{x}) + \sum_{j=1}^{r} \mu_j^* g_j(\boldsymbol{x}^*), \tag{34}$$

which we will use to state the necessary and sufficient conditions.

Theorem 1.5 (Karush-Kuhn-Tucker NOC)**.** *Let $\boldsymbol{x}^*$ be a local minimum for the inquality constrained problem where f, h_i, g_j are C^1 and assume $\boldsymbol{x}^*$ is regular (equality and active inequality constraint gradients are linearly independent). Then, there exists unique Lagrange multiplier vectors $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$ such that*

$$\nabla_{\boldsymbol{x}} L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = 0 \tag{35}$$

$$\boldsymbol{\mu} \geq 0 \tag{36}$$

$$\mu_j^* = 0, \quad \forall j \notin A(\boldsymbol{x}^*) \tag{37}$$

If in addition, $f, \boldsymbol{h}, \boldsymbol{g}$ are C^2, we have

$$\boldsymbol{y}^T \nabla^2_{\boldsymbol{xx}} L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \boldsymbol{y} \geq 0 \tag{38}$$

for all $\boldsymbol{y}$ such that

$$\nabla h_i(\boldsymbol{x}^*)^T \boldsymbol{y} = 0, \quad i = 1, \ldots, m \tag{39}$$

$$\nabla g_j(\boldsymbol{x}^*)^T \boldsymbol{y} = 0, \quad j \in A(\boldsymbol{x}^*) \tag{40}$$

Proof. See [Ber16] Section 3.3.1. □

The SOC are obtained similarly to the equality constrained case.

1.3 Further Reading

In this section we have addressed the necessary and sufficient conditions for constrained and unconstrained nonlinear optimization. This section is based heavily on [Ber16], and we refer the reader to this book for further details. We have avoided discussing linear programming, which is itself a large topic of study, about which many books have been written (we refer the reader to [BT97] as a good reference on the subject).

Convex optimization has become a powerful and widespread tool in modern optimal control. While we have only addressed it briefly here, [BV04] offers a fairly comprehensive treatment of the theory and practice of convex optimization. For a succinct overview with a focus on machine learning, we refer the reader to [Kol08].

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

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t). \tag{41}$$

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,

$$J = c_f(\boldsymbol{x}(t_f), t_f) + \int_{t_0}^{t_f} c(\boldsymbol{x}(t), \boldsymbol{u}(t), t)dt \tag{42}$$

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 (41) to follow an admissible trajectory, $\boldsymbol{x}^*$, that minimizes the performance measure given by (42). 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.

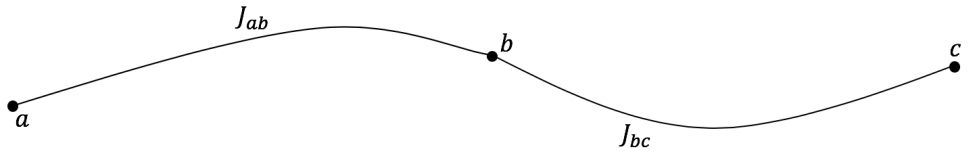


Figure 3: An optimal trajectory connecting point a to point c. There are no better (lower cost) trajectories than the sub-trajectory connecting b and c, by the principle of optimality.

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

$$J_f(\boldsymbol{x}_0) = c_f(\boldsymbol{x}_N) + \sum_{k=0}^{N-1} c(\boldsymbol{x}_k, \boldsymbol{u}_k, k). \tag{43}$$

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 3 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

$$\begin{aligned} \tilde{J}_{ac} &= J_{ab} + \tilde{J}_{bc} && (44)\\ &< J_{ab} + J_{bc} && (45)\\ &= J_{ac}, && (46) \end{aligned}$$

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 (43). Then, dynamic programming iterates backward in time, from $N-1$ to 0, with

$$\begin{aligned} J_N(\boldsymbol{x}_N) &= c_T(\boldsymbol{x}_N) && (47)\\ J_k(\boldsymbol{x}_k) &= \min_{\boldsymbol{u}_k \in \mathcal{U}(\boldsymbol{x}_k)} \left\{ c_k(\boldsymbol{x}_k, \boldsymbol{u}_k, k) + J_{k+1}(f(\boldsymbol{x}_k, \boldsymbol{u}_k, k)) \right\}. && (48) \end{aligned}$$

Note that here we have considered only deterministic dynamical systems (there is no stochastic disturbance). Equation (48) 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,

$$\boldsymbol{x}_{k+1} = A_k \boldsymbol{x}_k + B_k \boldsymbol{u}_k \tag{49}$$

and the cost function as quadratic

$$c(\boldsymbol{x}_k, \boldsymbol{u}_k) = \frac{1}{2}(\boldsymbol{x}_k^T Q_k \boldsymbol{x}_k + \boldsymbol{u}_k^T R_k \boldsymbol{u}_k) \tag{50}$$

$$c_N(\boldsymbol{x}_k) = \frac{1}{2}\boldsymbol{x}_k^T Q_N \boldsymbol{x}_k \tag{51}$$

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

$$J_N^*(\boldsymbol{x}_N) = \frac{1}{2}\boldsymbol{x}_N^T Q_N \boldsymbol{x}_N := \frac{1}{2}\boldsymbol{x}_N^T V_N \boldsymbol{x}_N. \tag{52}$$

Then, applying (48), we have

$$J_{N-1}^*(\boldsymbol{x}_{N-1}) = \frac{1}{2}\min_{\boldsymbol{u}_{N-1}\in\mathbb{R}^m} \left\{\boldsymbol{x}_{N-1}^T Q_{N-1}\boldsymbol{x}_{N-1} + \boldsymbol{u}_{N-1}^T R_{N-1}\boldsymbol{u}_{N-1} + \boldsymbol{x}_N^T V_N \boldsymbol{x}_N\right\} \tag{53}$$

which, applying the dynamics,

$$\begin{aligned} J_{N-1}^*(\boldsymbol{x}_{N-1}) = \frac{1}{2}\min_{\boldsymbol{u}_{N-1}\in\mathbb{R}^m} \{&\boldsymbol{x}_{N-1}^T Q_{N-1}\boldsymbol{x}_{N-1} + \boldsymbol{u}_{N-1}^T R_{N-1}\boldsymbol{u}_{N-1} \\ &+ (A_{N-1}\boldsymbol{x}_{N-1} + B_{N-1}\boldsymbol{u}_{N-1})^T V_N (A_{N-1}\boldsymbol{x}_{N-1} + B_{N-1}\boldsymbol{u}_{N-1})\}. \end{aligned} \tag{54}$$

Rearranging, we have

$$\begin{aligned} J_{N-1}^*(\boldsymbol{x}_{N-1}) = \frac{1}{2}\min_{\boldsymbol{u}_{N-1}\in\mathbb{R}^m} \{&\boldsymbol{x}_{N-1}^T (Q_{N-1} + A_{N-1}^T V_N A_{N-1})\boldsymbol{x}_{N-1} \\ &+ \boldsymbol{u}_{N-1}^T (R_{N-1} + B_{N-1}^T V_N B_{N-1})\boldsymbol{u}_{N-1} \\ &+ 2\boldsymbol{u}_{N-1}^T (B_{N-1}^T V_N A_{N-1})\boldsymbol{x}_{N-1}\}. \end{aligned} \tag{55}$$

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,

$$\frac{\partial J_{N-1}^*(\boldsymbol{x}_{N-1})}{\partial \boldsymbol{u}_{N-1}} = (R_{N-1} + B_{N-1}^T V_N B_{N-1})\boldsymbol{u}_{N-1} + (B_{N-1}^T V_N A_{N-1})\boldsymbol{x}_{N-1} \tag{56}$$

and setting this to zero yields

$$\boldsymbol{u}_{N-1}^* = -(R_{N-1} + B_{N-1}^T V_N B_{N-1})^{-1}(B_{N-1}^T V_N A_{N-1})\boldsymbol{x}_{N-1} \tag{57}$$

which we write

$$\boldsymbol{u}_{N-1}^* = L_{N-1}\boldsymbol{x}_{N-1} \tag{58}$$

which is a time-varying linear feedback policy. Plugging this feedback policy into (64),

$$\begin{aligned} J_{N-1}^*(\boldsymbol{x}_{N-1}) =& \boldsymbol{x}_{N-1}^T(Q_{N-1} + L_{N-1}^T R_{N-1} L_{N-1} \\ &+ (A_{N-1} + B_{N-1}L_{N-1})^T V_N (A_{N-1} + B_{N-1}L_{N-1}))\boldsymbol{x}_{N-1}. \end{aligned} \tag{59}$$

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

$$c(\boldsymbol{x}_k, \boldsymbol{u}_k) = \frac{1}{2}\boldsymbol{x}_k^T Q_k \boldsymbol{x}_k + \frac{1}{2}\boldsymbol{u}_k^T R_k \boldsymbol{u}_k + \boldsymbol{u}_k^T H_k \boldsymbol{x}_k + \boldsymbol{q}_k^T \boldsymbol{x}_k + \boldsymbol{r}_k \boldsymbol{u}_k + q_k, \tag{60}$$

terminal cost

$$c_N(\boldsymbol{x}_k) = \frac{1}{2}\boldsymbol{x}_k^T Q_N \boldsymbol{x}_k + \boldsymbol{q}_N^T \boldsymbol{x}_k + q_N, \tag{61}$$

and dynamics

$$\boldsymbol{x}_{k+1} = A_k \boldsymbol{x}_k + B_k \boldsymbol{u}_k + d_k. \tag{62}$$

The cost-to-go will take the form

$$J_k(\boldsymbol{x}_k) = \frac{1}{2}\boldsymbol{x}_k^T V_k \boldsymbol{x}_k + \boldsymbol{v}_k^T \boldsymbol{x}_k + v_k. \tag{63}$$

Repeating our approach from the last subsection, we have

$$\begin{aligned} J_k^*(\boldsymbol{x}_k) = \min_{\boldsymbol{u}_k \in \mathbb{R}^m} \{ & \frac{1}{2}\boldsymbol{x}_k^T Q_k \boldsymbol{x}_k + \frac{1}{2}\boldsymbol{u}_k^T R_k \boldsymbol{u}_k + \boldsymbol{u}_k^T H_k \boldsymbol{x}_k + \boldsymbol{q}_k^T \boldsymbol{x}_k + \boldsymbol{r}_k \boldsymbol{u}_k + q_k \\ & + \frac{1}{2}(A_k \boldsymbol{x}_k + B_k \boldsymbol{u}_k + \boldsymbol{d}_k)^T V_{k+1}(A_k \boldsymbol{x}_k + B_k \boldsymbol{u}_k + \boldsymbol{d}_k) \\ & + \boldsymbol{v}_{k+1}^T (A_k \boldsymbol{x}_k + B_k \boldsymbol{u}_k + \boldsymbol{d_k}) + v_{k+1} \}. \end{aligned} \tag{64}$$

Rearranging, have

$$\begin{aligned} J_k^*(\boldsymbol{x}_k) = \min_{\boldsymbol{u}_k \in \mathbb{R}^m} \{ & \frac{1}{2}\boldsymbol{x}_k^T (Q_k + A_k^T V_{k+1} A_k) \boldsymbol{x}_k + \frac{1}{2}\boldsymbol{u}_k^T (R_k + B_k^T V_{k+1} B_k) \boldsymbol{u}_k \\ & + \boldsymbol{u}_k^T (H_k + B_k^T V_{k+1} A_k)^T \boldsymbol{x}_k + (\boldsymbol{q}_k + A_k^T V_{K+1} \boldsymbol{d}_k + A_k^T \boldsymbol{v}_{k+1})^T \boldsymbol{x}_k \\ & + (\boldsymbol{r}_k + B_k^T V_{k+1} \boldsymbol{d} + B_k^T \boldsymbol{v}_{k+1}) \boldsymbol{u}_k + (v_{k+1} + \frac{1}{2}\boldsymbol{d}_k^T V_{k+1} \boldsymbol{d}_k + \boldsymbol{v}_{k+1}^T \boldsymbol{d}_k) \}. \end{aligned} \tag{65}$$

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

$$\boldsymbol{u}_k^* = \boldsymbol{l}_k + L_k \boldsymbol{x}_k. \tag{66}$$

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

$$S_{\boldsymbol{u},k} = \boldsymbol{r}_k + B_k^T \boldsymbol{v}_{k+1} + B_k^T V_{k+1} \boldsymbol{d}_k \tag{67}$$

$$S_{\boldsymbol{uu},k} = R_k + B_k^T V_{k+1} B_k \tag{68}$$

$$S_{\boldsymbol{ux},k} = H_k + B_k^T V_{k+1} A_k. \tag{69}$$

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.

$$L_k = -S_{\boldsymbol{uu},k}^{-1} S_{\boldsymbol{ux},k} \tag{70}$$
$$\boldsymbol{l}_k = -S_{\boldsymbol{uu},k}^{-1} S_{\boldsymbol{u},k} \tag{71}$$

3.

$$V_k = Q_k + A_k^T V_{k+1} A_k - L_k^T S_{\boldsymbol{uu},k} L_k \tag{72}$$
$$\boldsymbol{v}_k = \boldsymbol{q}_k + A_k^T (\boldsymbol{v}_{k+1} + V_{k+1} \boldsymbol{d}_k) + S_{\boldsymbol{ux},k} \boldsymbol{l}_k + L_k^T (S_{\boldsymbol{u},k} + S_{\boldsymbol{uu},k} \boldsymbol{l}_k) \tag{73}$$
$$v_k = v_{k+1} + q_k + \boldsymbol{d}_k^T \boldsymbol{v}_{k+1} + \frac{1}{2} \boldsymbol{d}_k^T V_{k+1} \boldsymbol{d}_k + \boldsymbol{l}_k^T (S_{\boldsymbol{u},k} + \frac{1}{2} S_{\boldsymbol{uu},k} \boldsymbol{l}_k) \tag{74}$$

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

$$\bar{\boldsymbol{x}}_{k+1} = A_k \bar{\boldsymbol{x}}_k + B_k \bar{\boldsymbol{u}}_k + \boldsymbol{d}_k, \ \forall k = 0, \dots, N-1. \tag{75}$$

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

$$\delta \boldsymbol{x}_k = \boldsymbol{x}_k - \bar{\boldsymbol{x}}_k \tag{76}$$
$$\delta \boldsymbol{u}_k = \boldsymbol{u}_k - \bar{\boldsymbol{u}}_k. \tag{77}$$

Rewriting, we have

$$\delta \boldsymbol{x}_{k+1} = A_k \delta \boldsymbol{x}_k + B_k \delta \boldsymbol{u}_k. \tag{78}$$

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

$$\bar{\boldsymbol{x}}_{k+1} = f(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k), \ \forall k = 0, \dots, N-1. \tag{79}$$

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

$$\boldsymbol{x}_{k+1} \approx f(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) + \underbrace{\frac{\partial f}{\partial \boldsymbol{x}}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)}_{A_k}(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k) + \underbrace{\frac{\partial f}{\partial \boldsymbol{u}}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)}_{B_k}(\boldsymbol{u}_k - \bar{\boldsymbol{u}}_k). \tag{80}$$

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

$$\delta \boldsymbol{x}_{k+1} = A_k \delta \boldsymbol{x}_k + B_k \delta \boldsymbol{u}_k \tag{81}$$

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

$$c_k := c(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) \tag{82}$$

$$c_{i,k} := \frac{\partial c}{\partial i}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) \tag{83}$$

$$c_{ij,k} := \frac{\partial^2 c}{\partial i \partial j}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k) \tag{84}$$

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

$$c(\delta \boldsymbol{x}_k, \delta \boldsymbol{u}_k) \approx \frac{1}{2} \begin{bmatrix} 1 \\ \delta \boldsymbol{x}_k \\ \delta \boldsymbol{u}_k \end{bmatrix}^T \begin{bmatrix} 2c_k & c_{\boldsymbol{x},k}^T & c_{\boldsymbol{u},k}^T \\ c_{\boldsymbol{x},k} & c_{\boldsymbol{xx},k} & c_{\boldsymbol{ux},k}^T \\ c_{\boldsymbol{x},k} & c_{\boldsymbol{ux},k} & c_{\boldsymbol{uu},k} \end{bmatrix} \begin{bmatrix} 1 \\ \delta \boldsymbol{x}_k \\ \delta \boldsymbol{u}_k \end{bmatrix}. \tag{85}$$

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 (85) 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

$$Q_k = c_k + v_{k+1} \tag{86}$$
$$Q_{\boldsymbol{x},k} = c_{\boldsymbol{x},k} + f_{\boldsymbol{x},k}^T \boldsymbol{v}_{k+1} \tag{87}$$
$$Q_{\boldsymbol{u},k} = c_{\boldsymbol{u},k} + f_{\boldsymbol{u},k}^T \boldsymbol{v}_{k+1} \tag{88}$$
$$Q_{\boldsymbol{xx},k} = c_{\boldsymbol{xx},k} + f_{\boldsymbol{x},k}^T V_{k+1} f_{\boldsymbol{x},k} \tag{89}$$
$$Q_{\boldsymbol{uu},k} = c_{\boldsymbol{uu},k} + f_{\boldsymbol{u},k}^T V_{k+1} f_{\boldsymbol{u},k} \tag{90}$$
$$Q_{\boldsymbol{ux},k} = c_{\boldsymbol{ux},k} + f_{\boldsymbol{u},k}^T V_{k+1} f_{\boldsymbol{x},k} \tag{91}$$

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

$$\delta \boldsymbol{u}_k^* = \boldsymbol{l}_k + L_k \delta \boldsymbol{x}_k \tag{92}$$

where

$$\boldsymbol{l}_k = -Q_{\boldsymbol{uu},k}^{-1} Q_{\boldsymbol{u},k} \tag{93}$$
$$L_k = -Q_{\boldsymbol{uu},k}^{-1} Q_{\boldsymbol{ux},k}. \tag{94}$$

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

$$V_k = Q_k - \frac{1}{2} \boldsymbol{l}_k^T Q_{\boldsymbol{uu},k} \boldsymbol{l}_k \tag{95}$$
$$V_{\boldsymbol{x}_k} = Q_{\boldsymbol{x},k} - L_k^T Q_{\boldsymbol{uu},k} \boldsymbol{l}_k \tag{96}$$
$$V_{\boldsymbol{xx},k} = Q_{\boldsymbol{xx},k} - L_k^T Q_{\boldsymbol{uu},k} L_k. \tag{97}$$

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 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 (86 – 91)

  Update feedback law via (93 – 94)

  Update value approximation via (95 – 97)

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

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 $\delta \boldsymbol{x}_k, \delta \boldsymbol{u}_k$,

$$Q(\delta \boldsymbol{x}_k, \delta \boldsymbol{u}_k) := c(\bar{\boldsymbol{x}}_k + \delta \boldsymbol{x}_k, \bar{\boldsymbol{u}}_k + \delta \boldsymbol{u}_k) + J_{k+1}(f(\bar{\boldsymbol{x}}_k + \delta \boldsymbol{x}_k, \bar{\boldsymbol{u}}_k + \delta \boldsymbol{u}_k)). \tag{98}$$

Note that Q here is different from the Q matrix in Section 2.3. Using the same notation as in (82), we can write the quadratic expansion of (98) as

$$Q(\delta \boldsymbol{x}_k, \delta \boldsymbol{u}_k) \approx \frac{1}{2} \begin{bmatrix} 1 \\ \delta \boldsymbol{x}_k \\ \delta \boldsymbol{u}_k \end{bmatrix}^T \begin{bmatrix} 2Q_k & Q_{\boldsymbol{x},k}^T & Q_{\boldsymbol{u},k}^T \\ Q_{\boldsymbol{x},k} & Q_{\boldsymbol{x}\boldsymbol{x},k} & Q_{\boldsymbol{u}\boldsymbol{x},k}^T \\ Q_{\boldsymbol{x},k} & Q_{\boldsymbol{u}\boldsymbol{x},k} & Q_{\boldsymbol{u}\boldsymbol{u},k} \end{bmatrix} \begin{bmatrix} 1 \\ \delta \boldsymbol{x}_k \\ \delta \boldsymbol{u}_k \end{bmatrix} \tag{99}$$

where

$$Q_k = c_k + v_{k+1} \tag{100}$$
$$Q_{\boldsymbol{x},k} = c_{\boldsymbol{x},k} + f_{\boldsymbol{x},k}^T \boldsymbol{v}_{k+1} \tag{101}$$
$$Q_{\boldsymbol{u},k} = c_{\boldsymbol{u},k} + f_{\boldsymbol{u},k}^T \boldsymbol{v}_{k+1} \tag{102}$$
$$Q_{\boldsymbol{xx},k} = c_{\boldsymbol{xx},k} + f_{\boldsymbol{x},k}^T V_{k+1} f_{\boldsymbol{x},k} + \boldsymbol{v}_{k+1} \cdot f_{\boldsymbol{xx},k} \tag{103}$$
$$Q_{\boldsymbol{uu},k} = c_{\boldsymbol{uu},k} + f_{\boldsymbol{u},k}^T V_{k+1} f_{\boldsymbol{u},k} + \boldsymbol{v}_{k+1} \cdot f_{\boldsymbol{uu},k} \tag{104}$$
$$Q_{\boldsymbol{ux},k} = c_{\boldsymbol{ux},k} + f_{\boldsymbol{u},k}^T V_{k+1} f_{\boldsymbol{x},k} + \boldsymbol{v}_{k+1} \cdot f_{\boldsymbol{ux},k}. \tag{105}$$

Note that these terms differ only from iLQR via the last term in (103 – 105), 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,

$$\delta \boldsymbol{u}_k^* = \text{argmin}_{\delta \boldsymbol{u}} Q(\delta \boldsymbol{x}_k, \delta \boldsymbol{u}) = \boldsymbol{l}_k + L_k \delta \boldsymbol{x}_k \tag{106}$$

where

$$\boldsymbol{l}_k = -Q_{\boldsymbol{uu},k}^{-1} Q_{\boldsymbol{u},k} \tag{107}$$
$$L_k = -Q_{\boldsymbol{uu},k}^{-1} Q_{\boldsymbol{ux},k}. \tag{108}$$

The DDP algorithm is identical to Algorithm 1, just with the alternative definitions for $Q_{\boldsymbol{xx},k}, Q_{\boldsymbol{uu},k}$ and $Q_{\boldsymbol{ux},k}$. 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

[Ber16] Dimitri P Bertsekas. *Nonlinear programming*. Athena Scientific, 2016.

[BT97] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*. Athena Scientific, 1997.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[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.

[Kol08] Zico Kolter. Convex optimization overview. *CS 229 Lecture Notes*, 2008.

[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.

[Pow12] Warren B Powell. AI, OR and control theory: A rosetta stone for stochastic optimization. 2012.

[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.