Intro to Optimal Control without Proofs

Anshuman Medhi

May 14, 2021

Contents

1 Introduction

Optimal control, as it is usually taught, is a post-graduate level field of mathematics that involves a lot of theory and proofs. For mathematical rigor, you must understand differential equations and optimization solvers. They try to solve continuous-time control problems which inherently requires a lot of complicated maths and logical thinking to understand how you get from one place to another. They try to prove properties of dynamical systems and optimization solvers to fully understand the nitty gritty details of how you are solving the problem and when some algorithm will work.

In the real world however, we almost always have to work in the discrete time world, and we have to rely on simulation or testing to understand if the controller works. Theoretical results or predictions are largely useless. Finally, we don't need to understand the rigorous mathematical proof of why a solver works. There are very few tutorials on Optimal Control that focus on someone who just wants to use these methods and algorithms, with just enough understanding to be able to debug when things go wrong.

This article attempts to strip away the theory and proofs from Optimal Control and give a working mans understanding of methods like multiple shooting and direct collocation, along with examples (with code) for implementing some of these methods in popular librarires like CasADi.

1.1 Structure of this Article

First we will cover defining the system and the problem we are trying to solve in a mathematical way. This would involve continuous time equations, so the next section will be about

2 Defining the Problem

$$\begin{aligned} \dot{x} &= f(x,u) \\ x(0) &= x_0 \\ x(t_{end}) &= x_{end} \\ C_{ineq}(x,u) &\leq 0 \\ C_{eq}(x,u) &= 0 \end{aligned}$$

$$J(x(\cdot),u(\cdot)) = \int_0^\infty g(x(t),u(t))$$

$$\underset{x,u}{\operatorname{argmin}}\, J(x,u) \tag{1}$$

Notice that all these equations are living in continuous time. Real world is a continuous time system [1]. However our controllers generally live in discrete time

[1] Lets revisit this sentence if the quantum physicists every finally prove anything about the nature of our universe, regardless the planck timescale is far below anyone computation capabilities (for now)

2.1 Trajectory Optimization

This solves a single instance of the problem, ie for some specific initial conditions. This can be thought of as optimizing an open loop plan for controlling the robot. Open-loop is not generally considered a good, robust method of controlling robots or any other kind of system that has to interect with the real world. However, by developing a closed loop controller for following the optimized trajectory plan, you may be able to get the best of both worlds.

There is also the approach of continually resolving the trajectory optimization problem, with the initial/boundary conditions being the real (measured) state of the system. This gives us feedback, brings the optimization into the control loop of the system, but it is only feasible if the problem can be solved very quickly, and more importantly predictably quickly.

2.2 Global controllers

What if we wanted to solve the problem for any and all initial conditions. Well besides the fact that however we formulate the trajectory optimization problem it should be able to be solved for any initial condition. What if there were a more efficient way, either at solve time or at run time? The ideal goal is a simple, mostly constant time algorithm that gives the optimal control action from just the current state.

The well known result is that for linear systems and quadratic cost functions we can create a linear feedback law for optimally controlling the system. This is known as the **Linear-Quadratic-Regulator**. I won't cover the derivation or even the equations for the LQR, you can look at just about any optimal control lecture notes to find this.

The way to solve the optimal control globally is to find a **Lyapunov Function**

$$L(x) = \min_u J(x(\cdot), u) \tag{2}$$

This is basically a function (of the state) that gives you the how much cost/energy/time is required to reach the goal. From this function it is easy enough to recover optimal action. As we know the gradient of a function is the direction of steepest descent. In this context that means that ∇L is a vector that describes a direction in state space that would minimize the cost left as fast as possible. So from the equation $\nabla L = f(x, u)$ we can solve for u to get the optimal action.

This is obviously locally optimal, if we go in this direction, the amount of time/energy/cost left will be minimized. Is this definitely globally optimal? is it possible that although we are making progress the goal as fast as possible right now, that a short term less profitable action would end up taking less time/cost/energy in total? Suffice to say no, to understand this you must understand **Pontryagin's Maximum Principle**

The problem with this is either you solve it in continuous time, which is only analytically possible for certain classes of problems, and requires some creative thought to find what the Lyapunov function would look like. Or you solve it in discrete time, which mostly comes down to having a large Look Up Table of precomputed optimal costs for every possible boundary condition. This is expensive in terms of computational time as well as memory. The field of reinforcement learning is in some sense trying to solve this global controller problem by representing the Lyapunov function as a deep neural network or some other huge parameterized function, and then 'learning' the specific parameters according to the system model,

Trajectory optimization as a method is general for any class of system, but each solution applies to only one set of boundary conditions. On the other hand, Lyapunov Functions are general for any set of boundary conditions, but usually apply to only one class of systems.

3 Just enough theory about optimization solvers

In the next section we will convert from optimizing in continuous time, over the uncountably infinitely space of all possible functions, to optimizing over a discrete number of decision variables. But lets take a moment to talk about how this kind of problem can be solved. There are many optimization algorithms that have different properties and work for different kinds of problems.

At its most general we have nonlinear optimization or nonlinear programming. This works for any kind of problem, but it also means we can say little about how well we can really solve the problem. We can't be sure that we will find the global optimal solution, or any so-

lution at all. Within this we have two major classes of optimization solvers: gradient-free/derivative-free optimizers and the opposite, which doesnt really have a name (gradient-full? gradient-based?). The most effective way we have for solving optimization problems uses the gradients (or hessians, the second derivative). This is because the gradient is the direction of steepest descent, this makes it a good direction to search for the next guess of the optimal solution. Different solvers have different specifics but they all use the gradient in a similar way, as a search direction.

Gradient-free solvers come into play when the derivative is not known or hard to compute. Historically methods that used gradient required you to derive the gradient by had and code it into a function manually, with all the error-prone nature that that suggests. However the field of automatic or algorithmic differentiation has made strides in recent years, mostly for machine learning applications in order to use gradient descent methods even with completely arbitrary and huge neural networks. Optimal Control problems are another good application of AD, and many general purpose optimal control toolkits make use of this technique.

The next important class of optimization problems is of Convex Optimization. These are the set of optimization problems where the objectives are convex functions. Among many things this means we can solve the problems faster, be guaranteed of making progress, and most importantly the global minimum is the only local minimum. If you're system admits being posed as a convex optimization problem it is usually a good idea to do so.

A nonlinear optimization solver requires some initial guess, and only guarantees you'll find **a** locally optimal solution in the neighbourhood of the initial guess. Convex Optimizations be solved entirely from scratch and guarantee a globally optimal solution. If you want to use a nonlinear solver you'll want to think of a good way to find an initial guess, maybe even analyze the problem to find a way to always fall into the globally optimal solution.

3.1 Convex Optimization problems

You should read **Convex Optimization by Stephen Boyd**.

Within convex optimization you have different classes of problems with different levels of 'difficulty', although as a user of blackbox optimization libraries they are all about the same to us.

	Objective
Linear Programming	Affine
Quadratic Programming	Symmetric Positive Definite
Second Order Cone Programming	Symmetric Positive Definite
Semidefinite Programming	Linear

4 Discretizing the problem

Fundamentally the approach is to use any method for solving differential equations to turn the continuous time integral in J into a expression involving the values of $x(t)$ and $u(t)$ at only a few discrete points in time.

This is necessarily an approximation and so a large part of theory is understanding the kinds and bounds on the error introduced by different approximations. But we can skip over that and just use some rules of thumb for deciding on how to discretize.

While we are technically discretizing a optimization problem, the optimization problem consists of a integration (cost function) and a differential equation solution. These are the parts that we are really trying to discretize, so we take clues from how to discretize and solve integration problems and differential equations, which are actually highly related.

Lets focus on the differential equation problem first: we would like to solve for $x(t)$ (assuming we have $u(t)$). Because the starting point and ending point are generally constrained we have a boundary value problem. The first class of methods for solving boundary value problems are called the shooting methods. These lift initial value problem solvers (essentially integrators for differential equations) into solving the boundary value problem. The other class of methods is the Collocation methods, which assume a structure/parameterization of the solution and then solving for the specific solution/parameters.

4.1 Shooting Methods

4.2 Collocation Methods