EECS 127/227AT Optimization Models in Engineering
Spring 2020 Optimal and Adaptive Control Project

In this project, we'll theoretically and empirically study the differences between *optimal* and *adaptive* control.

Introduction Control of dynamical systems is ubiquitous in the real world. It is used airplanes, manufacturing plants, and can even be used to model biological phenomena as well as the stock market. The aforementioned systems can always be approximated via vector-valued differential equations

$$\dot{x} = f(x, u)$$
$$y = h(x, u)$$

where f and h are vector valued functions, y is an *observation*, x is a *state* and u is known as the input to the system or the *control*. 99% of the time, the question posed by controls engineers is the following:

Given f and g, how can I design a control law $u = g(y)$ to control x to a desired state x_T?

The first equation $\dot{x} = f(x, u)$ tells us (implicitly) how the internal state of the system evolves as a function of the input. Note in real life we actually need to solve the vector-valued equation $\dot{x} = f(x, u)$ to figure out how our state x evolves in time. The second equation $y = h(x, u)$ describes the relation between our state, our control and what we can actually observe from this. As a concrete example, you may model an airplane as a 12-state system (that is $x \in \mathbb{R}^{12}$) but perhaps you can only measure the plane's speed, height, pitch, roll and yaw ($y \in \mathbb{R}^5$). If we had access to the internal state (that is $y = x$ or equivalently $h(x, u) = x$), we practically have a lens into the inner workings of the entire dynamical system governed by our function f. Note usually f is derived using the laws of physics for mechanical systems, or interpolation models for more complex phenomena. h is usually given to us or we get to design it as a controls engineer (should I put a sensor to measure the velocity? if that's too expensive, should I just measure the acceleration?). The study of all these questions and more demand an entire course on control.

The more complicated your system (the more complicated f and g are), it becomes harder to design control laws. For this reason, lots of theory and time has been devoted to understanding Linear Time Invariant (LTI) systems

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

The above is Linear because everything is a linear function of (x, u), and Time Invariant since the *state transition matrix* A is constant throughout time. For the remainder of this project, we will consider *discrete* LTI systems

$$x_{k+1} = Ax_k + Bu_k$$
$$y_k = Cx_k + Du_k$$

It is easy to go from traditional LTI systems to their discrete counterpart by rewriting the derivative as a first order difference and appropriately redefining the (A, B, C, D) matrices. Note the role f, g played before are now played by (A, B, C, D) – that is, these four matrices capture the internal physics of the system and what we can observe about the system.

Motivation Our goal (for this project and most controls problems) is to design a controller that tells us the optimal sequence of controls $u_1, u_2, \ldots, u_T$ to apply to a given system in order to control it from an initial state x_0 to a desired state x_T. We can reformulate this problem using the language of optimization. We are interested in the following two settings

(a) You know (A, B, C, D)

(b) You don't know (A, B, C, D)

The study of the first setting is often referred to as *optimal control* and the second setting is referred to as *adaptive control*. Clearly adaptive control is much harder than optimal control (how can you control something where you don't fully understand how it works?).

Outline We'll first walk through optimal control, how to formulate the LQR problem, and derive some properties/variations of it. After, we will look at some very simple adaptive control problems and see the challenges faced. Note this project comes with <u>two</u> Jupyter Notebooks; you must complete both.

Notation and Assumption All matrices are denoted using uppercase letters and all vectors are denoted using lowercase letters (differentiating scalars from vectors should be obvious from context). $A \in \mathbb{R}^{n \times n}$ is always a square matrix. Additionally, $x_k \in \mathbb{R}^n$, $u_k \in \mathbb{R}^m$, and $B \in \mathbb{R}^{n \times m}$. Unless otherwise specified, the dynamical system being studied is always

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= x_k \end{aligned}$$

that is, you always have full access to the state of the system.

1. **Optimal Control** Just like how LTI systems are the most well understood control systems, arguably the most well understood optimal control problem is the Linear Quadratic Regulator (LQR). In the LQR problem, we are interested in controlling our system from a given initial state $x_0 = \bar{x}$ to a desired state $x_N = 0$ (without loss of generality, we want to control our system to the 0 state). The optimization problem we are interested in solving (known as the *discrete time, finite horizon LQR* problem) is

$$\begin{aligned} \min_{\{x_i\}_{i=1}^N, \{u_i\}_{i=0}^{N-1}} \quad & J_N(x, u) \\ \text{s.t.} \quad & x_{k+1} = Ax_k + Bu_k, \quad k = 0, \ldots, N-1 \\ & x_0 = \bar{x} \end{aligned} \tag{1}$$

where

$$J_N(x, u) = x_N^\top Q x_N + \sum_{k=0}^{N-1} x_k^\top Q x_k + u_k^\top R u_k$$

Note we explicitly write the optimization over the variables $\{x_i\}_{i=1}^N$ despite this being trivial – given optimal u_k, the optimal x_{k+1} are obviously $x_{k+1} = Ax_k + Bu_k$. Here, $Q, R \succeq 0$ specified by us, the control engineer. These matrices determine how much we penalize the magnitudes of the entries of x_k and u_k respectively. For a concrete example, consider the case when $Q = qI$ and $R = rI$ for some $q, r > 0$. Then,

$$x^\top Qx + u^\top Ru = q\|x\|_2^2 + r\|u\|_2^2$$

where the first term can be seen as a measurement of how far away we are from the origin and the second can be viewed as how much "energy" we are putting into the system. Changing Q and R will result in different control policies with different properties.

There are two popular approaches to solving (1). The first we'll cover is known as the *Discrete Algebraic Riccati Equations* (DARE). For a thorough derivation of what we present below, see `https://stanford.edu/class/ee363/lectures/dlqr.pdf`.

DARE is a two step procedure: compute a set of matrices P_k backwards in time (get P_N, then P_{N-1}, ...) and then compute the optimal control u_k forward in time. More concretely, let $P_N = Q$ and compute

$$P_k = A^T P_{k+1} A - (A^T P_{k+1} B)(R + B^T P_{k+1} B)^{-1}(B^T P_{k+1} A) + Q, \;\; k = N-1, \ldots, 0$$

and set

$$u_k^* = -(R + B^T P_{k+1} B)^{-1}(B^T P_{k+1} A)x_k, \;\; k = 0, \ldots, N-1$$

For a more direct approach, notice that (1) is simply a quadratic program. We show how we can reformulate this problem as a least-squares problem and trivially arrive at a solution. The insight is that we unroll the recursion given by the equality constraints and pass them into our objective function. More concretely, let

$$\tilde{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}, \quad \tilde{u} = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix}$$

In other words, $\tilde{x}$ is the stack of all x_k vectors and is therefore in $\mathbb{R}^{Nn}$. Similarly, $\tilde{u}$ is in $\mathbb{R}^{Nm}$. We will not include x_0 here because it is fixed and doesn't affect the optimization.

(a) Rewrite the objective as $J_N = \tilde{x}^\top \tilde{Q}\tilde{x} + \tilde{u}^\top \tilde{R}\tilde{u}$ where $\tilde{Q}$ and $\tilde{R}$ are block matrices you will define.

(b) Note now that $\tilde{x}$ is a a function of $\tilde{u}$ (for example $x_2 = Ax_1 + Bu_1$). Using the fact that $x_{k+1} = Ax_k + Bu_k$ re-write $\tilde{x}$ in terms of $\tilde{u}$ – that is, find matrices G, H such that $\tilde{x} = G\tilde{u} + Hx_0$. Using this, rewrite J_N from above so that it is only a function of $\tilde{u}$.

(c) Solve for $\tilde{u}^* = \arg\min_{\tilde{u}} \;\; J_N(\tilde{u})$.

(d) Consider a generalization of (1), the situation where we have constraints on our control (for example $\|u_k\|_\infty \leq \bar{u}$ for all k). If this is the case, we cannot solve (1) using DARE or the method above. Alternatively, we can tune Q, R appropriately to achieve a desired system response (if R is extremely large, u will be forced to be small). Work through the python notebook `lqr.ipynb` to study the differences between enforcing constraints on the optimization directly and "pseudo"-enforcing them by tuning Q, R.

2. Adaptive Control Generally speaking, adaptive control problems are orders of magnitude harder than traditional optimal control problems. There are two main ways of performing adaptive control: 1) Dual Control and 2) Non-dual control.

Dual control refers to the idea of probing a system with control inputs (but not too aggressively) in order to learn how the systems behaves so that we can identify it. There is a clear trade-off here: if we give an unknown system too aggresive of a control it may cause the system to go unstable, but if we do not excite the system enough (or in the extreme case where we do not provide any control) we cannot learn anything about the system.

Non-dual control instead estimates the parameters of the system and takes these as fixed, true values. Then using these values, a control law is designed for the system. Using the new inputs and seeing new outputs of the system, we update our model parameters and iterate. This is known as non-dual control since we do not purposely probe the system in order to learn about it. This type of control has its own problems since it can lead to identifiability issues; it is plausible that we may not identify the parameters properly and controlling the system will lead us to learning erroneous parameters or prevent us from learning the system at all.

In general, people are more interested in finding non-optimal dual controllers. However, the problem of adaptive control remains extremely challenging and problem specific. Recent work in system-level synthesis is a type of non-dual adaptive control that introduces machine learning concepts and online optimization in order to learn the system parameters and stabilize the system.

For this worksheet, let's stick to the basics. Assume we are always given x_0 – a starting state. Consider the single-input single-output (SISO) system

$$x_{k+1} = x_k + bu_k + w_k$$

where $b \sim \mathcal{N}(\bar{b}, \sigma_b^2)$ and $w_k \sim \mathcal{N}(0, \sigma_w^2)$ and we have a quadratic terminal cost $\mathbb{E}[x_1^2]$. We are interested in minimizing the terminal cost but we do not know the system parameters.

(a) Solve $\min_{u_0} \mathbb{E}[x_1^2]$ where the expectation is taking with respect to all the randomness in the system. Discuss what happens to u_0 as σ_b^2 increase and interpret your result.

(b) Now we separate the control and system identification process and see what ensues for a SISO system. Consider the SISO system

$$x_{k+1} = ax_k + bu_k + w_k$$

and the quadratic cost $\mathbb{E}[\sum_{k=1}^{N} x_k^2]$. Assume (a, b) are known. Compute u_k^* for all k.

(c) Now assume (a, b) are unknown and consider the feedback law $u_k = \gamma x_k$ for some γ. Explain how using $u_k = \gamma x_k$ makes the system unidentifiable – here, unidentifiable means there exists multiple pairs (a, b) such that we would not be able to distinguish between them. (Hint: consider the closed-loop system).

(d) Show that modifying the control law to $u_k = \gamma x_k + \delta_k$ makes the system identifiable.

(e) Now let's consider a multiple-input multiple-output system (MIMO)

$$x_{k+1} = Ax_k + Bu_k$$
$$y_k = x_k$$

where $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times m}$. Suppose we've observed $\{y_i\}_{i=1}^{T}$, $\{u_i\}_{i=0}^{T-1}$ for some T. We can construct estimates $(\hat{A}, \hat{B})$ of (A, B) using a least-squares loss

$$(\hat{A}, \hat{B}) = \arg\min_{A,B} \sum_{k=0}^{T-1} \|y_{k+1} - Ay_k - Bu_k\|_2^2$$

Show how to rewrite the above as

$$(\hat{A}, \hat{B}) = \arg\min_{A,B} \|Y\text{vec}(A^\top) + U\text{vec}(B^\top) - Z\|_2^2$$

for matrices Y, U, Z that you will define. Here $\text{vec}(A)$ is the operation that takes in a matrix and stacks its columns on top of each other. For example if

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 7 \end{bmatrix} \implies \text{vec}(A^\top) = \begin{bmatrix} 1 \\ 3 \\ 2 \\ 7 \end{bmatrix}$$

(f) Compute $\text{vec}(A^{\top^*})$ and $\text{vec}(B^{\top^*})$

(g) Note sometimes the least squares problem does not have a unique solution. In this case, how can you modify the objective in order to guarantee there exists a unique solution? (Hint: regularize)

(h) Consider a case where we want to perform adaptive control on a MIMO system instead of a SISO system. We've already seen in parts (d) and (e) one way of performing system identification. Work through the python notebook `ac.ipynb` on how to implement a naive dual control strategy and study how it performs on different systems.