Control with Multiplicative Noise

1 Introduction and Background

Traditional models in control typically assume that any environmental noise or disturbance is independent of the state or observation of the system itself. While this assumption greatly simplifies the models and their control, it is not always an accurate representation of physical phenomena. Systems with state-dependent or observation-dependent noise require different classes of models to capture this nature of the disturbance. *Multiplicative-noise models* are a favorable modeling choice for such systems. However, under the standard linear control perspective, some of these systems with state-dependent observation noise would be considered uncontrollable. In this project, you will follow some recent and classical research papers to understand the state of the art in our understanding of the control of systems with multiplicative noise. Along the way, you will solve multiple optimization problems, and learn a new technique that is commonly used to find optimal policies for sequential optimization problems called *policy gradient*.

1.1 Introduction to Control

In this project, we will use different techniques to analyze *control systems*, which are discrete-time dynamical systems that have external inputs. For concreteness and simplicity, we work in the case where everything is a scalar, though the ideas generalize to vector systems.

First, we begin by looking at discrete-time dynamical systems. Let $X_t \in \mathbb{R}$ be the *state* at time t. The state starts at $X_0 \in \mathbb{R}$ at time $t = 0$. At each timestep $t \geq 0$:

1. The system (also called the *environment*) hands us an *observation* $Y_t \in \mathbb{R}$.
2. The system uses the state X_t to choose a new state $X_{t+1} \in \mathbb{R}$.

There may be *noise* in the state and observation; thus, all the X's (including X_0) and Y's are *real-valued random variables*.

Now we look at what changes when we are able to supply external input. More precisely, we supply a so-called *control policy* $F = (F_0, F_1, F_2, \dots)$, where for each $t \geq 0$ we have that $F_t \colon \mathbb{R}^{t+1} \to \mathbb{R}$ is a real-valued and deterministic function. The system state initializes at $X_0 \in \mathbb{R}$ as before, and at each timestep $t \geq 0$:

1. The system hands us an observation $Y_t \in \mathbb{R}$.
2. Using the history of observations $Y_{(t)} \doteq (Y_0, Y_1, \dots, Y_t) \in \mathbb{R}^{t+1}$, we choose *control* $U_t = F_t(Y_{(t)}) \in \mathbb{R}$.
3. The system uses the state X_t and the input U_t to choose a new state $X_{t+1} \in \mathbb{R}$.

As before, all the X's and Y's are random variables; now the U's are also random variables.

For the purpose of the project, our goal as users is to provide a control policy F which *stabilizes* the system. One way of defining stabilization is the so-called property of *stability in the second moment*.

Definition 1 (Stability of Second Moment)
For a given control policy F, the control system $\mathcal{S}_F$ is *stable in the second moment* if and only if there exists

$M \in \mathbb{R}_{\geq 0}$ such that

$$\mathbb{E}\left[X_t^2\right] \leq M, \qquad \forall t \geq 0. \tag{1}$$

Let $\lambda \geq 0$ be a regularization parameter. In order to find an *optimal* control policy, i.e., one which stabilizes the system as efficiently as possible, we seek to minimize the loss

$$L(F) \doteq \mathbb{E}\left[\sum_{t=0}^{\infty}\{X_{t+1}^2 + \lambda U_t^2\}\right], \tag{2}$$

over control policies F.

Tackling the problem of designing optimal control policies is extremely hard in full generality. Indeed, as stated, the above optimal control optimization problem is *infinite dimensional*! This is because the optimization variables are actually functions $F_t \colon \mathbb{R}^{t+1} \to \mathbb{R}$ for each time-step $t \geq 0$.

In this project, we discuss a particularly challenging control system and explore simplifications we can make to make the optimal control problem tractable and solvable – first by hand, then by the *policy gradient* algorithm.

1.2 Multiplicative Noise Control

In this project we analyze the following control system. Let $a, b, c, \mu \in \mathbb{R}$ and $\alpha, \beta, \gamma, \sigma \geq 0$ be constants.

Definition 2 (Multiplicative Noise Control System)
The multiplicative noise control system is given as

$$\begin{aligned} X_{t+1} &= A_t X_t + B_t U_t \\ Y_t &= C_t X_t, \end{aligned} \qquad \forall t \geq 0, \tag{3}$$

where for each $t \geq 0$:

- X_t is the state at time t, Y_t is the observation at time t, and $U_t = F_t(Y_{(t)})$ is the control at time t.
- X_t, Y_t, A_t, B_t, C_t are real-valued random variables, where:
  - All A's, B's, C's, and X_0 are independent from each other.
  - We have
    $$\begin{aligned} \mathbb{E}[A_t] &= a, \quad \mathrm{Var}(A_t) = \alpha^2, \quad \forall t \geq 0 \\ \mathbb{E}[B_t] &= b, \quad \mathrm{Var}(B_t) = \beta^2, \quad \forall t \geq 0 \\ \mathbb{E}[C_t] &= c, \quad \mathrm{Var}(C_t) = \gamma^2, \quad \forall t \geq 0 \\ \mathbb{E}[X_0] &= \mu, \quad \mathrm{Var}(X_0) = \sigma^2. \end{aligned} \tag{4}$$
    From this one can show that
    $$\mathbb{E}\left[A_t^2\right] = a^2 + \alpha^2, \quad \mathbb{E}\left[B_t^2\right] = b^2 + \beta^2, \quad \mathbb{E}\left[C_t^2\right] = c^2 + \gamma^2, \quad \forall t \geq 0 \quad \text{and} \quad \mathbb{E}\left[X_0^2\right] = \mu^2 + \sigma^2. \tag{5}$$
  - Each B_t is not deterministically zero (that is b and β are not both zero). This means that our control U_t has an impact on the state update and thus the overall system.
  - Similarly, each C_t is not deterministically zero (that is c and γ are not both zero). This means that our observations Y_t are not just 0, and in fact have some information about the state.

1.3 Project Statement

In this project we will consider three variants of the multiplicative control model introduced in Definition 2:

- The control noise model in which we only consider randomness in the control due to randomness of B_t.
- The state and control noise model in which we additionally consider randomness in the state due to the randomness of A_t.
- The observation noise model in which we only consider randomness in the output due to the randomness of C_t.

For each of these models we aim to answer two main questions. The first is a question of existence of an optimal feedback control policy that stabilizes the system. That is for which combination of parameters is the system stabilizable in the second moment. The second question is about finding the optimal policy if it exists. We will start by answering these questions in the setting where the system parameters are known. Then, we consider answering these questions in the more realistic setting of unknown system parameters for which we will introduce the *policy gradient* algorithm as a useful tool.

You will answer these questions through a guided review of some recent and classical results published in the literature. Along the way, you will analytically solve multiple optimization problems, and perform empirical evaluations of some policies. We recommend that all problems are done in order.

2 Problems

1. Overview of Relevant Literature

In this problem, you will read two research papers that study the optimal control of multiplicative noise systems. The aim is to gain a better understanding of the topics discussed in this project and to get insights into the state-of-the-art development in our understanding of control systems with multiplicative noise. You will summarize the main results and findings of each paper and answer a few questions about them.

(a) Read and summarize the main results in the paper "The uncertainty threshold principle: Some fundamental limitations of optimal decision making under dynamic uncertainty" by Athans et al. [1]. In your summary include answers to the following questions

i. Describe the multiplicative noise model studied in the paper. What are the assumptions made about the noise?

ii. What is the cost function considered for the optimal control problem?

iii. What is the form of the optimal feedback control policy for the studied model?

iv. Under what condition does such optimal feedback control policy exist?

(b) Read and summarize the main results in the paper "When multiplicative noise stymies control" by Ding et al. [2]. In your summary include answers to the following questions

i. What is the main question the paper is answering?

ii. Describe the multiplicative noise model studied in the paper. What are the assumptions made about the noise?

iii. Under what conditions is the system stabilizable by a linear policy?

iv. What results do the paper state about non-linear policies?

v. How does this work relate to the earlier work of Athans et al. that you reviewed in part (a)?

2. Environment Implementation

Before attempting this problem, please read the introduction to the codebase in Section 4.1.

In later parts, we will empirically test our policies on various systems. To do this, we will be using the environment classes in `environments/multiplicative_gaussian_noise_environment.py`. Using the system defined in Definition 2, implement the `MultiplicativeGaussianNoiseEnvironment` class in `environments/multiplicative_gaussian_noise_environment.py`.

3. Control Noise: Derivations

In this problem we consider a simplified version of the multiplicative noise system where the noise is only on the input. More formally, we consider a system that satisfies the following assumptions:

Definition 3 (Control Noise System)

The control noise system is a variant of the general multiplicative noise system such that

- $A_t = a$ deterministically (that is a can take any arbitrary value and $\alpha = 0$)
- B_t is a random variable with $\mathbb{E}[B_t] = b$ and $\mathrm{Var}(B_t) = \beta^2$ where b and β can take any arbitrary values
- $C_t = 1$ deterministically (that is $c = 1$ and $\gamma = 0$)
- $X_0 = 1$ deterministically (that is $\mu = 1$ and $\sigma = 0$)

These assumptions combined result in the following system

$$\begin{aligned} X_{t+1} &= aX_t + B_t U_t \\ Y_t &= X_t, \end{aligned} \qquad \forall t \geq 0. \tag{6}$$

We also assume that our regularization parameter $\lambda = 0$, so the loss we attempt to minimize is $L(F) = \mathbb{E}\left[\sum_{t=0}^{\infty} X_{t+1}^2\right]$. This system was studied several times, including by Gravell et al. [3] and Ranade et al. [4].

Concretely, our goal in this problem is to find all values of a for which the system is stabilizable in the second moment. When a is such that the system is stabilizable in the second moment, we also want to find an optimal policy $F^\star$.

One may show using the technique of *stochastic dynamic programming*[1] that the optimal so-called *greedy memory*-1 policy is optimal overall. In other words, suppose that our policy F says that at every step, we should use *only* the most recent observation Y_t in order to choose the U_t which minimizes the immediate cost, i.e., the state second moment $\mathbb{E}\left[X_{t+1}^2 \mid Y_t\right]$. Then this policy is optimal, in the sense that it minimizes the loss $L(F)$ and stabilizes the widest range of a.

Fortunately, determining such a control policy turns out to be tractable; we will do so now.

(a) First, let us determine what the state second moment $\mathbb{E}\left[X_{t+1}^2 \mid Y_t\right]$ is. Fix $t \geq 0$. Show that

$$\mathbb{E}\left[X_{t+1}^2 \mid Y_t\right] = a^2 Y_t^2 + 2abU_tY_t + (b^2 + \beta^2)U_t^2. \tag{7}$$

(b) Now, we will determine what exactly the optimal greedy memory-1 policy $F^\star$ is. Fix $t \geq 0$. Define $F_t^\star \colon \mathbb{R}^{t+1} \to \mathbb{R}$ by

$$F_t^\star(Y_{(t)}) \doteq \operatorname*{argmin}_{U_t \in \mathbb{R}} \mathbb{E}\left[X_{t+1}^2 \mid Y_t\right]. \tag{8}$$

Show that

$$F_t^\star(Y_{(t)}) \doteq -\frac{ab}{b^2 + \beta^2} Y_t, \qquad \forall Y_t \in \mathbb{R} \tag{9}$$

so that $F_t^\star$ is a *linear* function of *only* Y_t, and the strategy is the *same* regardless of the value of t. This optimal policy $F^\star$ is therefore called a *linear period-1* (or *linear time-invariant*) policy.

[1] A reference can be found in Bertsekas' book *Dynamic Programming and Optimal Control* [5].

(c) With the optimal control $U_t = F_t^\star(Y_{(t)})$, where $F_t^\star$ was given in part (b), show that

$$\mathbb{E}\left[X_t^2\right] = \left(\frac{a^2\beta^2}{b^2+\beta^2}\right)^t, \qquad \forall t \geq 0. \tag{10}$$

HINT: Refer to Section 4.2 for useful probability identities.

(d) With the optimal control $U_t = F_t^\star(Y_{(t)})$, where $F_t^\star$ was given in part (b), and using the result from part (c), show that the system is stable in the second moment if and only if

$$|a| \leq \sqrt{1+\frac{b^2}{\beta^2}}. \tag{11}$$

(e) Now consider a system similar to the one described in Definition 3 but with B_t being a deterministic and time varying parameter. Assume that B_t is known at every timestep $t \geq 0$. Derive the greedy optimal control policy that stabilizes this system. Is it a time-invariant policy?

(f) In `main.ipynb`, complete the Control Noise section (refer to Section 4.1 for information about the codebase).

4. State and Control Noise: Derivations

In this problem, we consider a more general version of the system considered in the previous problem, where the noise is on the state as well as on the control. More formally, we consider a system that satisfies the following assumptions:

Definition 4 (State and Control Noise System)

The control noise system is a variant of the general multiplicative noise system such that

- A_t is a random variable with $\mathbb{E}[A_t] = a$ and $\mathrm{Var}(A_t) = \alpha^2$ where a and α can take any arbitrary values
- B_t is a random variable with $\mathbb{E}[B_t] = b$ and $\mathrm{Var}(B_t) = \beta^2$ where b and β can take any arbitrary values
- $C_t = 1$ deterministically (that is $c = 1$ and $\gamma = 0$)
- X_0 is a random variable with $\mathbb{E}[X_0] = \mu$ and $\mathrm{Var}(X_0) = \sigma^2$ where μ and σ can take any arbitrary values

These assumptions combined result in the following system

$$\begin{aligned} X_{t+1} &= A_t X_t + B_t U_t \\ Y_t &= X_t, \end{aligned} \qquad \forall t \geq 0. \tag{12}$$

We attempt to minimize the loss $L(F) = \mathbb{E}\left[\sum_{t=0}^{\infty}\{X_{t+1}^2 + \lambda U_t^2\}\right]$. Similarly to the previous problem, one can show using dynamic programming that there is a memory-1 greedy optimal control policy. Namely, the policy F which says that at every step we should use only Y_t in order to choose U_t which minimizes the immediate cost $\mathbb{E}\left[X_{t+1}^2 \mid Y_t\right] + \lambda U_t^2$ is overall optimal.

(a) Using the same approach as Problem 3., show that an optimal control policy is

$$F_t^\star(Y_{(t)}) = -\frac{ab}{b^2 + \beta^2 + \lambda} Y_t, \qquad \forall t \geq 0, \tag{13}$$

and that the system is stabilizable in the second moment if and only if

$$\alpha^2 + \frac{a^2(b^2\beta^2 + (\beta^2 + \lambda)^2)}{(b^2 + \beta^2 + \lambda)^2} \leq 1. \tag{14}$$

(b) In `main.ipynb`, complete the State-Control Noise section (refer to Section 4.1 for information about the codebase).

5. Observation Noise: Derivations

Now, we will consider the complementary problem to the state and control noise system. That is, we will consider a system where the noise is only on the observation. Formally, we consider a system that satisfies the following assumptions:

Definition 5 (Observation Noise System)
The observation noise system is a variant of the general multiplicative noise system such that

- $A_t = a$ deterministically (that is a can take any arbitrary value and $\alpha = 0$)
- $B_t = 1$ deterministically (that is $b = 1$ and $\beta = 0$)
- C_t is a random variable with $\mathbb{E}[C_t] = c$ and $\mathrm{Var}(C_t) = \gamma^2$ where c and γ can take any arbitrary values
- $X_0 = 1$ deterministically (that is $\mu = 1$ and $\sigma = 0$)

These assumptions combined result in the following system

$$\begin{aligned} X_{t+1} &= aX_t + U_t \\ Y_t &= C_t X_t, \end{aligned} \qquad \forall t \geq 0. \tag{15}$$

We also assume that our regularization parameter $\lambda = 0$, so the loss we attempt to minimize is $L(F) = \mathbb{E}\left[\sum_{t=0}^{\infty} X_{t+1}^2\right]$.

This system was studied several times, including by Gravell et al. [3] and Subramanian et al. [6]. In fact, designing the optimal control for this system is still an open problem!

Similarly to previous problems, in this problem we will derive the best linear memory-1 period-1 greedy policy for the observation noise system. We will also compare the performance of this policy on stabilizing the observation noise system to its performance in stabilizing the control noise system. Finally, we will comment on the optimality of the linear memory-1 period-1 greedy policy on this system.

(a) Show by induction on t that, if U is a linear memory-1 period-1 greedy control policy, i.e., if for all t we have $U_t = F_t(Y_t) = \theta Y_t$, then

$$\mathbb{E}\left[X_t^2\right] = (a^2 + 2ac\theta + (c^2 + \gamma^2)\theta^2)^t, \qquad \forall t \geq 0. \tag{16}$$

HINT: Write X_{t+1}^2 in terms of θ, X_t, and C_t

(b) Suppose that $U_t = F_t(Y_{(t)}) = \theta Y_t$ for all $t \geq 0$, and fix a particular t. Define

$$\theta^\star \doteq \operatorname*{argmin}_{\substack{\theta \in \mathbb{R} \\ U_t = \theta Y_t}} \mathbb{E}\left[X_{t+1}^2\right]. \tag{17}$$

Show that

$$\theta^\star = -\frac{ac}{c^2 + \gamma^2}. \tag{18}$$

This result shows that the optimal linear memory-1 period-1 greedy control policy is

$$F_t^\star(Y_{(t)}) = -\frac{ac}{c^2 + \gamma^2} Y_t. \tag{19}$$

Contrast this to the optimal control policy $F_t^\star(Y_{(t)}) = -\dfrac{ab}{b^2 + \beta^2} Y_t$ derived in Problem 3..

(c) With the control $U_t = F_t^\star(Y_{(t)})$, where F_t was given in part (b), show that

$$\mathbb{E}\left[X_t^2\right] = \left(\frac{a^2\gamma^2}{c^2+\gamma^2}\right)^t, \qquad \forall t \geq 0. \tag{20}$$

(d) With the control $U_t = F_t^\star(Y_{(t)})$, where F_t was given in part (b), and using the result from part (c), show that the system is stable in the second moment if and only if

$$|a| \leq \sqrt{1+\frac{c^2}{\gamma^2}}. \tag{21}$$

(e) In `main.ipynb`, complete the Observation Noise section (refer to Section 4.1 for information about the codebase).

(f) Now, suppose $c = 0$ and $\gamma = 1$. The condition from part (d) indicates that if $|a| \leq 1$ then the system is stabilizable in the second moment using the linear memory-1 period-1 greedy policy from part (b). Empirically verify that with $a = 1.01$ and this control policy, the system is not stable in the second moment.

Now, define the policy F' by

$$F_t'(Y_{(t)}) = \begin{cases} \frac{1}{2} + \frac{2}{5}\left|Y_t\right| & t \text{ is even,} \\ -\frac{1}{2} - \frac{1}{2}\left|Y_t\right| & t \text{ is odd.} \end{cases} \tag{22}$$

(This policy was obtained from Figure 3 of [6])

In `main.ipynb`, complete the Period-1 vs Period-2 Observation Noise section. Based on the result of this section, comment on the optimality of the linear memory-1 period-1 greedy control policy derived in part (b). Is it the overall optimal policy for the observation noise systems?

6. Introduction to Policy Gradient

Using optimal control policies derived by hand, such as in Problem **3.**, Problem **4.**, and Problem **5.**, provably ensures that the control system is stable in the second moment under broad conditions. However, implementing the optimal control requires knowledge of the environment, namely the parameters $a, b, c, \alpha, \beta, \gamma$. At face value, this is an unrealistic assumption; most of the time we only have noisy estimates, at most, for these parameters. Thus, we introduce the *policy gradient* method to learn a control policy from data without having full knowledge of the environment.

The policy gradient method is conceptually very similar to gradient descent. It is an iterative procedure where at each iteration we estimate the gradient of the cost function in Equation (2) and then use the gradient to update the control policy.

One problem we have is that we cannot run gradient descent on control policies, since we are directly optimizing over functions. The solution to this is a rather common idea; we parameterize our control policy $F = (F_0, F_1, \dots)$ by some parameter θ, so it would be written as $F(\theta)$, where the policy at time t is $F_t(\cdot;\theta)$. As an example, if we had a linear memory-1 period-1 policy $F_t(Y_{(t)};\theta) = \theta Y_t$, then the parameter θ would be a scalar. As another example in this setting, if we had a neural network policy $F_t(Y_{(t)};\theta) = \theta_2 q(\theta_1 Y_t)$, where $q\colon \mathbb{R} \to \mathbb{R}$ were some nonlinear function (such as the very popular ReLU $q(x) = \max\{x, 0\}$), then $\theta = [\theta_1, \theta_2]$. Overall, instead of taking a gradient step over F, we just take a gradient step over θ.

Writing down our first attempt at an algorithm, we have the following description:

Algorithm 1 Our first try at policy gradient.

```
function PolicyGradientFirstAttempt
    Initialize at some parameter θ_0
    for i ∈ {0, ..., M − 1} do                                   ▷ Using M training iterations
        Estimate ∇_θ L(θ_i) ≐ ∇_θ E[ Σ_{t=0}^{∞} {X_{t+1}^2 + λU_t^2} ] from samples
        θ_{i+1} ← θ_i − η∇_θ L(θ_i)
    end for
    return F(θ)
end function
```

Everything seems fine so far except for how to estimate the quantity

$$\nabla_\theta L(\theta) = \nabla_\theta \mathbb{E}\left[\sum_{t=0}^{\infty}(X_{t+1}^2 + \lambda U_t^2)\right]. \tag{23}$$

Statistics theory tells us that one may estimate the expected value by sampling many, say N, system trajectories from the environment using the policy $F(\theta)$, and then averaging over them. There are three issues with this approach:

- The sum goes to ∞, so an infinitely long trajectory is needed; thus sampling even a single trajectory requires an infinite amount of data! The fix is to estimate the sum up to some large time index T, so we only need to collect the first T steps of a given trajectory.
- If we don't know the true value of X_t, which can happen in systems with observation noise, we cannot compute even one term of the sum. Instead, we model the term of the sum corresponding to time t, i.e.,

$X_{t+1}^2 + \lambda U_t^2$ as the *loss* collected at time t, which we denote as ℓ_t. We assume that even if we do not know X_t, the term ℓ_t is handed to us by the system. This helps our ideas generalize to handle systems with observation noise.

- Another issue with this approach is that as stated, the only randomness in each trajectory is the initial condition X_0. Thus, many sampled trajectories will look similar to each other, and this greatly decreases the efficiency of our sampling proceedure, along with making the optimization much more difficult. Thus, we perturb each control U_t with random noise; that is, instead of inputting the control $U_t = F_t(Y_{(t)}; \theta)$, we input the control $\widetilde{U}_t = U_t + W_t = F_t(Y_{(t)}; \theta) + W_t$, where $W_t \sim \mathcal{N}(0, \omega^2)$ is random zero-mean Gaussian noise.

  The expectation is now over the randomness in X_0 and W_t; this leads to a wider range of sampled control policies and greatly helps the optimization overall. We denote the probability density of $\widetilde{U}_t$ given $Y_{(t)}$ with parameter θ as $\pi_\theta(\widetilde{U}_t \mid Y_{(t)})$.

Suppose we collect triples $(Y_t^j, \widetilde{U}_t^j, \ell_t^j)$ for all $t \in \{0, \ldots, T\}$ and $j \in \{1, \ldots, N\}$. Our final gradient approximation is

$$\nabla_\theta L(\theta) = \nabla_\theta \mathbb{E}\left[\sum_{t=0}^{\infty} \ell_t\right] \tag{24}$$

$$\approx \nabla_\theta \mathbb{E}\left[\sum_{t=0}^{T} \ell_t\right] \tag{25}$$

$$\approx \mathbb{E}\left[\left(\sum_{t=0}^{T} \nabla_\theta \log\left(\pi_\theta(\widetilde{U}_t \mid Y_{(t)})\right)\right)\left(\sum_{t=0}^{T} \ell_t\right)\right] \tag{26}$$

$$\approx \frac{1}{N}\sum_{j=1}^{N}\left(\sum_{t=0}^{T} \nabla_\theta \log\left(\pi_\theta(\widetilde{U}_t^j \mid Y_{(t)}^j)\right)\right)\left(\sum_{t=0}^{T} \ell_t^j\right). \tag{27}$$

Thus, we present our complete policy gradient approach, which is called the REINFORCE algorithm in the reinforcement learning literature. For completeness, we include the mechanism we use to sample trajectories from the environment.

Algorithm 2 Our policy gradient algorithm.

function POLICYGRADIENT
  Initialize at some parameter θ_0
  for $i \in \{0, \dots, M-1\}$ **do**
    for $j \in \{1, \dots, N\}$ **do**
      Begin new trajectory.
      for $t \in \{0, \dots, T\}$ **do**
        Collect observation Y_t^j from environment.
        Sample $W_t^j \sim \mathcal{N}(0, \omega^2)$.
        Hand control input $\widetilde{U}_t^j \doteq F_t(Y_{(t)}^j; \theta_i) + W_t^j$ to environment and collect loss ℓ_t^j.
      end for
    end for
    Approximate $\nabla_\theta L(\theta_i) \approx \dfrac{1}{N} \sum_{j=1}^{N} \left(\sum_{t=0}^{T} \nabla_\theta \log\left(\pi_{\theta_i}(\widetilde{U}_t^j \mid Y_{(t)}^j)\right) \right) \left(\sum_{t=0}^{T} \ell_t^j \right)$
    $\theta_{i+1} \leftarrow \theta_i - \eta \nabla_\theta L(\theta_i)$
  end for
  return $F(\theta_M)$
end function

(a) Suppose that $U_t = F_t(Y_{(t)}; \theta) = \theta Y_t$ for $\theta \in \mathbb{R}$. Find the gradient

$$\nabla_\theta \log\left(\pi_\theta(\widetilde{U}_t \mid Y_t)\right) \qquad \forall t \geq 0. \tag{28}$$

7. Input Noise and Observation Noise: Policy Gradient

In this problem, we will use the policy gradient algorithm to find the best linear memory-1 period-1 control policy for the control noise system of Problem **3.** and the observation noise system of Problem **5.** in the setting of unknown system parameters. We will also consider other classes of policies and evaluate and compare their performance on these two systems. Through this analysis, we aim to empirically support the conclusion that the optimal greedy linear memory-1 period-1 control policy is:

- The optimal control policy overall, in the input-noise setting of Problem **3.**.
- *Not* the optimal control policy overall, in the observation-noise setting of Problem **5.**.

More specifically, we consider the following three classes of policies:

- The class of linear memory-1 period-1 control policies:

$$F_t(Y_{(t)};\theta) \doteq \theta_0 Y_t. \tag{29}$$

- The class of affine memory-2 period-1 control policies:

$$F_t(Y_{(t)};\theta) \doteq \begin{cases} \theta_0 + \theta_1 Y_t, & t = 0 \\ \theta_0 + \theta_1 Y_t + \theta_2 Y_{t-1}, & t \geq 1. \end{cases} \tag{30}$$

- The class of affine memory-1 period-2 control policies:

$$F_t(Y_{(t)};\theta) \doteq \begin{cases} \theta_0 + \theta_1 Y_t, & t \text{ is even} \\ \theta_2 + \theta_3 Y_t, & t \text{ is odd.} \end{cases} \tag{31}$$

Each of these policies manifests as a PyTorch `nn.Module` in the provided sample code (refer to Section 4.1 for information about the codebase).

(a) For each policy above, implement it in a `nn.Module` within `policies/our_policy_modules.py`.
HINT: Look at `policies/m1p1_linear_policy_modules.py` *as an example of how to implement such modules.*

(b) Change `driver.py` to train each policy on the control noise system described in Problem **3.** with $b = 1$ and $\beta = 1$ and test various values of a. Visualize the results and comment on which class of policies tends to do well? What level of a can each policy stabilize? Include the provided visualizations in your project writeup.

(c) Change `driver.py` to train each policy on the observation noise system described in Problem **5.** with $c = 1$ and $\gamma = 1$ and test various values of a. Visualize the results and comment on which class of policies tends to do well? What level of a can each policy stabilize? Include the provided visualizations in your project writeup.

3 Rubric

Here's what the rubric looks like:

- To get a B: read and summarize the papers in Problem 1, correctly solve Problems 2–5 including the attached Jupyter notebook, and make an honest attempt at problems 6 and 7.
- To get a B+/A-: read and summarize the papers in Problem 1, correctly solve Problems 2–7 including the attached Jupyter notebook and coding requirements.
- To get a A: complete all problems (1-7) as stated above, plus a complete implementation one of the extensions below.

To get an A, you should complete one of the following extensions. You should include your extension(s) in a separate report that you attach to your project writeup. Exceptional projects that go above and beyond may receive extra credit at our discretion.

- Propose a novel class of policies that is different from the three classes studied in Problem **7.**. You can explore function classes beyond the affine relations considered earlier. You may also consider policies that use higher moments and policies which have more complicated time-dependence than just alternating strategies. You may refer to [6] for inspiration. Remember that finding the optimal control policy of the observation noise system is still an open problem. So try to design a policy that beats the performance of the policies you implemented in Problem **7.** for that system.

  Similar to what you did in Problem **7.**, implement this policy in a `nn.Module`. Then use the `driver.py` code to train your policy on the same observation noise system in part (c) of problem **7.**. Compare the performance of your policy to the three policies you trained previously on each system.

  HINT: Training the parameters of a policy is generally a difficult (non-convex) optimization problem. If you are struggling to learn good parameters for your proposed policy, you can discuss this with one of the course staff during the project office hours for possible suggestions on how to improve the training processes.

- One potential approach towards overcoming the difficulty of training a control policy and improving the convergence is *overparameterization*. The idea behind overparameterization is that the difficulty of the training process and its convergence can be improved by representing the policy with a function with a very large number of parameters.[2] In this project extension, you will utilize this idea by representing the control policy as a neural network with at least one hidden layer such that it has much more parameters than the policies investigated in Problem **7.**. You can test different network architectures and vary the number of parameters.

  Similar to what you did in Problem **7.**, you can implement this policy in a `nn.Module` and use the `driver.py` code to train it. Then train your policy on the same observation noise system in part (c) of problem **7.**. Do a detailed empirical analysis of the performance of your overparameterized policy and compare it with the performance of the ones you trained in Problem **7.**.

- A recent (not yet published) study proposed a new control policy for the observation noise model that outperforms all the policy classes we consider in this project as well as the ones described in [6]. This policy chooses the

[2] You will probably notice that, when your policy has a small number of parameters, increasing the number of parameters makes training the policy harder. When you have a large number of parameters, however, the training is *not* impossibly hard, but is actually easier than with a small number of parameters. Understanding the exact interplay between overparameterization and optimization is an area of active research.

control U_t at every timestep based on the *maximum a-posteriori* (MAP) estimator of the state at that timestep given all previous information about the system $Y_{(t)}$. More formally, let $f_x(x_t = \bar{x}|Y_{(t)})$ be the conditional density of the state x_t given the observations $Y_{(t)}$. Then the control input at time $t \geq 0$ is selected as

$$U_t = -\operatorname*{argmax}_{\bar{x}} f_x(x_t = \bar{x}|Y_{(t)}). \tag{32}$$

Although this control strategy is based on a simple idea, it beats the state of the art control policies developed for observation noise models.

In this project extension, you will consider the uniform distribution noise model (rather than the normal distribution used throughout this project) and you will implement the MAP control policy and empirically test its performance. If you choose to do this extension, you will have to meet with one of the course staff during the project office hours to get access to the paper which derives the density function $f_x(x_t = \bar{x}|Y_{(t)})$ for the aforementioned noise model.

- You may propose your own extension of similar level of interest and difficulty to the options above. You have to get your proposed idea approved by the course staff.

3.1 Deliverables

Your submission should be in the form of one PDF with the following parts:

1. Your project writeup.
2. A PDF printout of the completed Jupyter notebook `main.ipynb`.
3. A report of any project extension you choose to complete. Your report should follow the template available on the course website under "Projects". The report must have a minimum of 1000 words and should include the following sections:
    - Introduction section: includes literature review of any relevant papers you summarized and identifies open problems in the understanding of multiplicative noise systems.
    - Methodology section: includes description of the methodology you follow in the project extension you chose to implement. Make sure you include description of the noise model you use and any parameters you use in the algorithm or in training.
    - Results section: summarizes the results you obtained from the project extension you implement. You may use some of the visualizations from your project writeup for performance comparison.
4. A post-mortem survey about your project experience.

In addition to the PDF submission, please do submit any code or Jupyter notebook you write for any project extension.

4 Appendix

4.1 Introduction to Codebase

Alongside the written portion of the project, you will write some code to empirically test the policies you analytically derive as well as run policy gradient to learn new policies. Fortunately, we have provided an extensive skeleton that already implements the majority of the code you will need. To ease the learning curve, here we provide a short description of the codebase structure, italicizing the files that you need to edit. You should still read the provided code – even for parts that don't require your edits – to understand the logic. Doing so will make debugging significantly easier.

In this project, we use PyTorch, which can be thought of as similar to NumPy but with a slightly different API and an automatic differentiation module. The latter will eventually help us implement policy gradient.

- `environments`
  - `base_environment.py` – the parent environment class, which determines the methods that need to be implemented to define your own environments.
  - *`multiplicative_gaussian_noise_environment.py`* – defines the multiplicative noise control system (Definition 2) in the case of Gaussian noise. Also defines various special cases (control noise, state and control noise, observation noise) that will be discussed in the problem set.
- `infrastructure`
  - `pytorch_utils.py` - miscellaneous code to deal with PyTorch tensors.
  - `visualization.py` - code to produce plots, i.e., to visualize policy performance.
- `policies`
  - `base_policy.py` - defines the parent policy class, which contains the methods required to get an action using the policy. Also defines a stochastic policy class for use during policy gradient, which contains the methods required to get a noisy action and update the policy during policy gradient training.
  - `additive_gaussian_policy.py` - defines the stochastic policy class which adds Gaussian noise to each pure action to get a noisy action.
  - `m1p1_linear_policy_modules.py` - defines the class of linear memory-1 period-1 policy modules (defined later), as an example of how to define a policy module.
  - *`our_policy_modules.py`* - empty code which you will write in Problem 6.
- `agent.py` - defines an agent which serves as a wrapper for the policy class.
- `control_engine.py` - defines the training and evaluation loops.
- *`driver.py`* - sample code which trains and evaluates a policy. Change as needed.
- `main.ipynb` - main notebook that accompanies the written problems 2 – 4.

4.2 Probability Identities

You will need the following probability identities for this project; they should all be familiar from any introductory probability course.

Let X, Y be random variables, which may be correlated or dependent, and let $\alpha, \beta \in \mathbb{R}$ be constants.

- Linearity of expectation:
$$\mathbb{E}[\alpha X + \beta Y] = \alpha \mathbb{E}[X] + \beta \mathbb{E}[Y]. \tag{33}$$

- Tower rule:
$$\mathbb{E}[\mathbb{E}[X \mid Y]] = \mathbb{E}[X]. \tag{34}$$

- Independence: if X and Y are independent, then
$$\mathbb{E}[f(X)g(Y)] = \mathbb{E}[f(X)]\,\mathbb{E}[g(Y)] \qquad \text{for all functions } f, g\colon \mathbb{R} \to \mathbb{R}, \tag{35}$$
and
$$\mathbb{E}[f(X) \mid Y] = \mathbb{E}[f(X)] \qquad \text{for all functions } f\colon \mathbb{R} \to \mathbb{R}. \tag{36}$$

References

[1] M. Athans, R. Ku, and S. Gershwin, "The uncertainty threshold principle: Some fundamental limitations of optimal decision making under dynamic uncertainty," *IEEE Transactions on Automatic Control*, vol. 22, no. 3, pp. 491–495, 1977.

[2] J. Ding, Y. Peres, G. Ranade, and A. Zhai, "When multiplicative noise stymies control," *The Annals of Applied Probability*, vol. 29, no. 4, pp. 1963–1992, 2019.

[3] B. Gravell, P. M. Esfahani, and T. Summers, "Learning optimal controllers for linear systems with multiplicative noise via policy gradient," *IEEE Transactions on Automatic Control*, vol. 66, no. 11, pp. 5283–5298, 2020.

[4] G. Ranade and A. Sahai, "Control capacity," *IEEE Transactions on Information Theory*, vol. 65, no. 1, pp. 235–254, 2018.

[5] D. Bertsekas, *Dynamic programming and optimal control: Volume I*. Athena scientific, 2012, vol. 1.

[6] V. Subramanian, M. Won, and G. Ranade, "Learning a neural-network controller for a multiplicative observation noise system," in *2020 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2020, pp. 2849–2854.