Control of Multiplicative Noise Systems

Ali Tout, Karen Shieh and Riyya Ahmed

May 10, 2024

Abstract

In this project, we analyze numerous control systems with different multiplicative noise models of which have variables that are either chosen to be deterministic or random. Our first noise control system analyzes when just the control has noise; our second noise control system analyzes when the control, the state, and our initial state have noise; and our third noise control system analyzes when our output state has noise. We then ensure our control system in the second moment is stable under broader conditions. Finally, for our extension we focus on combining our previous results to solve for when all our systems (except for the initial state) have noise.

1 Introduction

In traditional multiplicative noise models, studies typically assume that environmental noise or disturbance is independent of the state or observation of the system itself. However, this assumption does not always hold to be true; in reality the noise emitted by a system usually has a relation with the system itself. As such, in our paper, we use Multiplicative models with state dependent variables to represent how having state dependent noise will effect our system. We solve for multiple cases of this where some noise variables are deterministic and others are random and prove at what point they will converge, along with their optimal policies.

2 Related Work

1. "The Uncertainty Threshold Principle: Some Fundamental Limitations of Optimal Decision Making under Dynamic Uncertainty" by Athans et al. focuses on the scalar linear-quadratic optimal control (which can be scaled to linear-quadratic-Gaussian and multivariable nonlinear-non-quadratratic-non-Gaussian Problems), and what happens to the infinite horizon case (when the iteration is large, say, infinity) when the parameter uncertainty is large. The model assumes that the situation one wants to analyze has state-dependent and control-dependent white noise.

   The cost function considered for the optimal control problem is given as the minimization of the standard quadratic cost function

   $$E[\sum_{t=0}^{N} Qx^2(t) + Ru^2(t)] \tag{1}$$

   where N is the horizon time (in some cases, ∞), and $Q > 0$ and $R > 0$. We assume that $x(t)$ can be measured exactly.

   The optimal feedback control policy is of the form:

   $$x(t+1) = a(t)x(t) + b(t)u(t) \qquad \forall t \geq 0 \tag{2}$$

   where the parameters $a(t)$ and $b(t)$ are Gaussian and white with known constant means $\bar{a},\ \bar{b}$ and variances $\Sigma_{aa},\ \Sigma_{bb}$ respectively.

Lastly, the control policy exists *if and only if* $m < 1$, where m (the *threshold parameter*) is defined as

$$m = \Sigma_{aa} + \bar{a}^2 - \frac{(\Sigma + ab + \bar{a}\bar{b})^2}{\Sigma_{bb} + \bar{b}^2} \tag{3}$$

If $m \geq 1$, the control policy does not exist.

In conclusion, the optimum decision rules (aka control policies) do not exist if the dynamic uncertainty is over a certain threshold.

2. "When Multiplicative Noise Stymies Control" by Ding et al. focuses on the general system S_a when having continuous multiplicative noise, and forms a control system based on the assumptions. The paper also wants to find out the largest growth factor a (which captures the growth of the system) that the system can stabilize for a given distribution on Z_n (noise), where Z_n are i.i.d. random variables which the distribution is known (in our case, the assumption is that $Z_n \sim \mathcal{N}(1, \sigma^2)$).

   Specifically, the system S_a, has the following assumptions:

$$\begin{aligned} X_0 &\sim \mathcal{N}(0,1) \\ X_{n+1} &= a \cdot X_n - U_n \\ Y_n &= Z_n \cdot X_n \end{aligned} \tag{4}$$

   where U_n is the control policy at step n.

   In the linear memoryless strategy, the control policy is

$$U_n = d^* Y_n \tag{5}$$

   where d^* is defined as

$$d^* = \frac{a}{1 + \sigma^2} \tag{6}$$

   This implies that by the linear policy, the system stabilizes when the growth of the system is bounded, i.e.

$$1 < a < a^* \tag{7}$$

   where a^* is defined as

$$a^* = \sqrt{1 + \frac{1}{\sigma^2}} \tag{8}$$

   Nonlinear policies tend to do better than linear policies when $a^* = \sqrt{1 + \frac{1}{\sigma^2}}$ and in the case when $\mathrm{E}[Z^n] = 0$ since the nonlinear policies can work with higher values of a.

   This relates to the earlier work of Athans et al. by having similar (after some math calculation, essentially the same) equation, taking the form of

$$x(t+1) = a(t)x(t) + b(t)u(t) \qquad \forall t \geq 0 \tag{9}$$

   and of the form

$$\begin{aligned} X_{n+1} &= a \cdot X_n - U_n \\ Y_n &= Z_n \cdot X_n \end{aligned} \tag{10}$$

   consisting of a linear system with a control policy that depends on Y.

3. The reading “Optimal Filtering for Discrete-Time Linear Systems With Time-Correlated Multiplicative Measurement Noises” by Wei Liu focuses on optimal control systems similar to paper 2 with now time correlated multiplicative measurement noises, meaning that our Z Random Variable used to represent noise is not independent anymore. To solve this, Z is replaced with a new term $\zeta_{\mu,k}$ which represents multiplicative noise and has it's own equation rather than just being a normal distribution with the assumptions that ω_k and v_k are uncorrelated and x_0 and $\zeta_{\mu,0}$ are uncorrelated. Using an MMSE criterion, the study finds that there is a computationally feasible algorithm for optimal filtering without even having to add an extra dimension to the data to account for the time correlation. Another advantage of the algorithm found is that it can take into account multiple time correlated multiplicative measurement noises and will still produce a feasible result while other algorithms mentioned in the paper by other studies could only take one into account. One question that remains after reading the paper “How can the proposed algorithm be adapted or extended to handle non-linear systems with time-correlated multiplicative and additive noises?”. This question arises as the current study focuses on the linear systems and extending the algorithm to non-linear scenarios could broaden its applicability.

3 Problems

1. environments/multiplicative_gaussian_noise_environment.py

```
class MultiplicativeGaussianNoiseEnvironment(Environment):
    """
    Defines the transition dynamics of the environment with
       multiplicative noise according to the equations:
    x_(t+1) = A_t x_t + B_t u_t
    y_t     = C_t x_t
    l_t     = x_(t+1)^2 + lambda * u_t
    where A_t ~ N(a, alpha^2), B_t ~ N(b, beta^2), C_t ~ N(c,
       gamma^2), and X_0 ~ N(mu, sigma^2).
    """
    def __init__(self, a: float, b: float, c: float, mu: float,
       alpha: float, beta: float, gamma: float,
                 sigma: float, lmbda: float):
        super(MultiplicativeGaussianNoiseEnvironment, self).__init__()
        self.a: float = a
        self.b: float = b
        self.c: float = c
        self.mu: float = mu
        self.alpha: float = alpha
        self.beta: float = beta
        self.gamma: float = gamma
        self.sigma: float = sigma
        self.lmbda: float = lmbda

        self.x: Tensor = self.mu + self.sigma * torch.randn(size=())

        C_0: Tensor = self.c + self.gamma * torch.randn(size=())
        self.y: Tensor = C_0 * self.x

    def step(self, u: Tensor):
        """
        Given an input u, executes the transition dynamics once.
        """
        A_t: Tensor = self.a + self.alpha * torch.randn(size=())
        B_t: Tensor = self.b + self.beta * torch.randn(size=())
        C_t: Tensor = self.c + self.gamma * torch.randn(size=())
        self.x: Tensor = A_t*self.x + B_t*u # TODO
        self.y: Tensor = self.x*C_t # TODO

    def loss(self, u: Tensor) -> Tensor:
        """
        Given an input u, computes the loss l = x^2 + lambda u^2.
        """
        return (self.x ** 2) + self.lmbda * (u ** 2)

    def reset(self):
        """
        Resets the state and observation to their initial
            distributions.
        """

        self.x: Tensor = self.mu + self.sigma * torch.randn(size=())

        C_0: Tensor = self.c + self.gamma * torch.randn(size=())
        self.y: Tensor = C_0 * self.x
```

2. (a) Note: $X_t = Y_t$

$$\begin{aligned} E[X_{t+1}^2|Y_t] &= E[(aX_t + B_tU_t)^2 \mid Y_t] \\ &= E[a^2X_t^2 + 2aX_tB_tU_t + B_t^2U_t^2 \mid Y_t] \\ &= a^2X_t^2 + 2aX_tE[B_tU_t \mid Y_t] + E[B_t^2U_t^2 \mid Y_t] \\ &= a^2Y_t^2 + 2aY_tE[B_t \mid Y_t]E[U_t \mid Y_t] + E[B_t^2 \mid Y_t]E[U_t^2 \mid Y_t] \\ &= a^2Y_t^2 + 2abY_tU_t + (b^2+\beta^2)U_t^2 \end{aligned} \tag{11}$$

(b)

$$\begin{aligned} F_t^*(Y_t) &= argmin_{U_t \in \mathbb{R}}\ E[X_{t+1}^2|Y_t] \\ &= argmin_{U_t \in \mathbb{R}}\ a^2Y_t^2 + 2abY_tU_t + (b^2+\beta^2)U_t^2 \\ &= argmin_{U_t \in \mathbb{R}}\ 2abY_tU_t + (b^2+\beta^2)U_t^2 \end{aligned} \tag{12}$$

We see that this equation is convex in U_t, which means taking the derivative will give us a global minimum.

$$\begin{aligned} \frac{d}{dU_t}2abY_tU_t + (b^2+\beta^2)U_t^2 &= 2abY_t + 2(b^2+\beta^2)U_t \\ 0 &= 2abY_t + 2(b^2+\beta^2)U_t^* \\ U_t^* &= \frac{-ab}{(b^2+\beta^2)}Y_t \end{aligned} \tag{13}$$

Hence, the control policy is

$$F_t^*(Y_t) = \frac{-ab}{(b^2+\beta^2)}Y_t \qquad \forall Y_t \in \mathbb{R} \tag{14}$$

(c)

$$\begin{aligned}
E(X_{t+1}^2) &= E(E(X_{t+1}^2|Y_t) \\
&= E(a^2Y_t^2 + 2abY_tU_t + (b^2+\beta^2)U_t^2) \\
&= E(a^2Y_t^2 - 2abY_t^2(\frac{ab}{b^2+\beta^2}) + (b^2+\beta^2)(\frac{ab}{b^2+\beta^2}Y_t)^2) \\
&= a^2E(X_t^2) - \frac{2a^2b^2}{b^2+\beta^2}E(X_t^2) + \frac{a^2b^2}{b^2+\beta^2}E(X_t^2) \\
&= (a^2 - \frac{2a^2b^2}{b^2+\beta^2} + \frac{a^2b^2}{b^2+\beta^2})E(X_t^2) \\
&= (a^2 - \frac{a^2b^2}{b^2+\beta^2})E(X_t^2) \\
&= (\frac{a^2b^2 + a^2\beta^2 - a^2b^2}{b^2+\beta^2})E(X_t^2) \\
&= \frac{a^2\beta^2}{b^2+\beta^2}E(X_t^2) \\
&= (\frac{a^2\beta^2}{b^2+\beta^2})^{t+1}
\end{aligned} \tag{15}$$

Which means that

$$E(X_t^2) = (\frac{a^2\beta^2}{b^2+\beta^2})^t \tag{16}$$

(d) For the above equation to converge, we know that $\frac{a^2\beta^2}{b^2+\beta^2}$ has to be ≤ 1 or else as $t \to \infty$, $E(X_t^2) \to \infty$ and hence doesn't exist M such that $E(X_t^2) \leq M$ (so it is not stable).

$$\begin{aligned}
E(X_t^2) &= (\frac{a^2\beta^2}{b^2+\beta^2})^t \\
1 &\geq \frac{a^2\beta^2}{b^2+\beta^2} \\
a^2 &\leq \frac{b^2+\beta^2}{\beta^2} \\
|a| &\leq \sqrt{1+\frac{b^2}{\beta^2}}
\end{aligned} \tag{17}$$

(e) Note: $X_t = Y_t$, and B_t is known at every timestep $t \geq 0$.

$$\begin{aligned}
E[X_{t+1}^2|Y_t] &= E[(aX_t + B_tU_t)^2 \mid Y_t] \\
&= E[a^2X_t^2 + 2aX_tB_tU_t + B_t^2U_t^2 \mid Y_t] \\
&= a^2X_t^2 + 2aX_tE[B_tU_t \mid Y_t] + E[B_t^2U_t^2 \mid Y_t] \\
&= a^2Y_t^2 + 2aY_tE[B_t \mid Y_t]E[U_t \mid Y_t] + E[B_t^2 \mid Y_t]E[U_t^2 \mid Y_t] \\
&= a^2Y_t^2 + 2aY_tB_tU_t + B_t^2U_t^2
\end{aligned} \tag{18}$$

We see that this equation is convex in U_t, which means taking the derivative will give us a global minimum.

$$\begin{aligned}
F_t^*(Y_t) &= argmin_{U_t\in\mathbb{R}}\ E[X_{t+1}^2|Y_t] \\
&= argmin_{U_t\in\mathbb{R}}\ a^2Y_t^2 + 2aY_tB_tU_t + B_t^2U_t^2 \\
&= argmin_{U_t\in\mathbb{R}}\ 2aY_tB_tU_t + B_t^2U_t^2
\end{aligned} \tag{19}$$

$$\begin{aligned}\frac{d}{dU_t} 2aY_tB_tU_t + B_t^2U_t^2 &= 2aY_tB_t + 2B_t^2U_t \\ 0 &= 2aY_tB_t + 2B_t^2U_t^* \\ U_t^* &= -\frac{a}{B_t}Y_t\end{aligned} \tag{20}$$

Hence, the greedy optimal control policy that stabilizes this system is.

$$F_t^*(Y_t) = -\frac{a}{B_t}Y_t \tag{21}$$

One can see that this is not time-invariant because the control policy depends on B_t, which changes over time. Hence, the control policy is time variant.

(f) *Note: This section of the problem is done in Jupyter Notebook,* ***main.py*** *Control Noise section, with the prompt also reference in this document.*

The optimal linear coefficient θ found in Problem 3 for the control noise system as a function of a, b, and β is listed below.

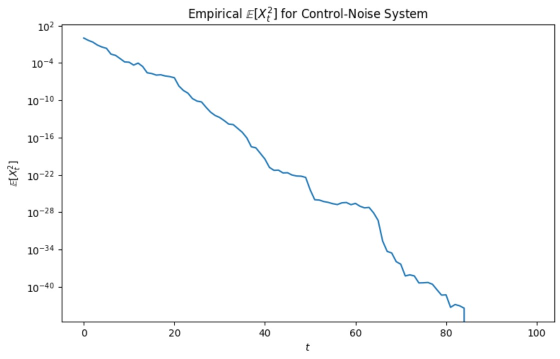
```
# TODO
theta_cn = lambda a, b, beta: -(a*b)/(b**2+beta**2)
"""Analyze the system where $a = 1$, $b = 1$, $\beta = 1$."""
a = 1
b = 1
beta = 1
env = MultiplicativeGaussianControlNoiseEnvironment(
    a = a,
    b = b,
    beta = beta,
    lmbda = 0
)
```

We empirically estimate $\mathbb{E}[X_t^2] \approx \frac{1}{N}\sum_{i=1}^{N}(X^{(i)})_t^2$, where $X_t^{(i)}$ is the state at timestep t on the i-th rollout.

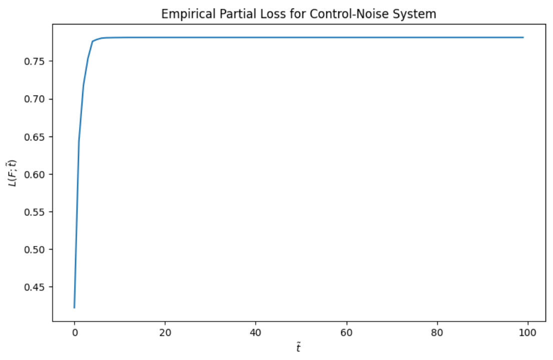
Likewise, we define the empirical loss as $\overline{L}(F) = \sum_{t=1}^{T}\mathbb{E}[X_t^2] \approx \frac{1}{N}\sum_{t=1}^{T}\sum_{i=1}^{N}\left(X_t^{(i)}\right)^2$. It is sometimes interesting to look at the 'partial' loss; that is, the loss for the first $\tilde{t}$ timesteps. Define this as $\overline{L}(F;\tilde{t}) = \frac{1}{N}\sum_{t=1}^{\tilde{t}}\sum_{i=1}^{N}\left(X_t^{(i)}\right)^2$.

```
linear_policy = Policy(FixedWeightM1P1LinearModule(theta_cn(a, b,
    beta)))
states, losses = sim_policy(100, 100, linear_policy, env)
#Figure 1
plot_empirical_average_second_moment_and_partial_loss(states,
    losses, "Control-Noise")
```

```
#Figure 2
a_list = [3, 2, 1.5, 1.4, 1.0, 0.9]
a_losses = []
for a in a_list:
    env = MultiplicativeGaussianControlNoiseEnvironment(
        a = a,
        b = b,
        beta = beta,
        lmbda = 0
    )

    linear_policy = Policy(FixedWeightM1P1LinearModule(theta_cn(a,
        b, beta)))
    states, losses = sim_policy(100, 100, linear_policy, env)
    plot_average_second_moment_trajectory(states, a)
```

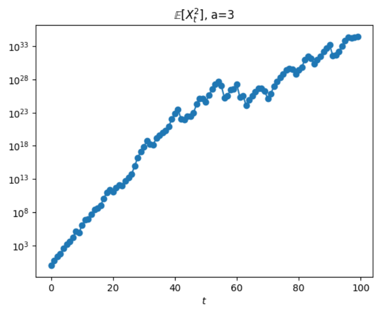


(a) Empirical Average Second Moment

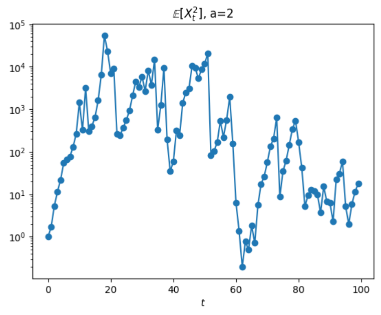


(b) Empirical Partial Loss

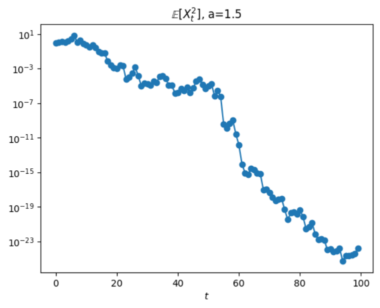
Figure 1: Empirical Average Second Moment and Partial Loss of a Control Noise System



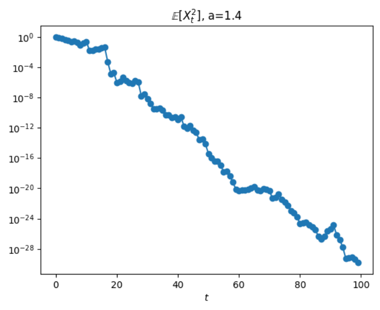
(a) $a = 3$



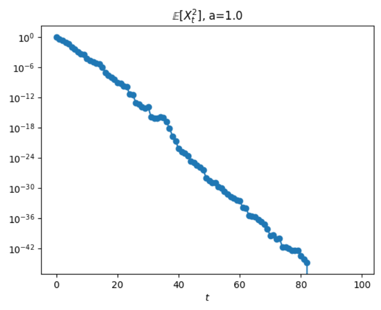
(b) $a = 2$



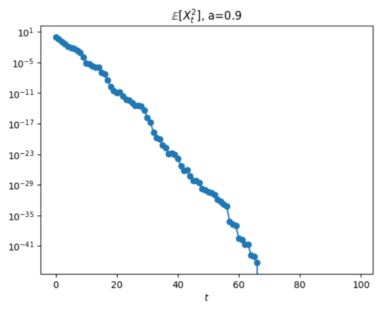
(c) $a = 1.5$



(d) $a = 1.4$



(e) $a = 1.0$



(f) $a = 0.9$

Figure 2: Different values of a

When $|a| \leq \sqrt{2}$, the system is stabilze. The plots above line up with the analysis, with a=1.4 and below stablizes after 100 iterations.

Now, we will examine an interesting property of the optimal policy derived in Problem 3 (b). Let F^* denote this policy, and let $U_t^* = F_t^*(Y_{(t)})$. It's worth noting that this policy does not necessarily set the expected value of X_{t+1} to 0; that is,

$$\mathbb{E}[X_{t+1}|Y_t] = \mathbb{E}[aY_t + B_t U_t^* \mid Y_t] \neq 0$$

whenever $\beta > 0$. We say that this policy is "biased".
We could compute another ("unbiased") policy F^0 such that

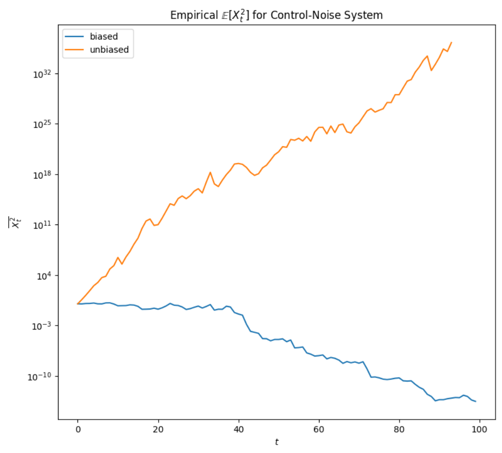
$$\mathbb{E}[aY_t + B_t U_t^0 \mid Y_t] = 0.$$

This policy is given by $U_t^0 = F_t^0(Y_{(t)}) = -\frac{a}{b}Y_t$. When $\beta \gg 0$, the difference between these policies is quite pronounced. Since this policy does not optimize the second moment, it will perform worse (in the second-moment sense) when you compare it to the optimal second-moment-minimizing policy.

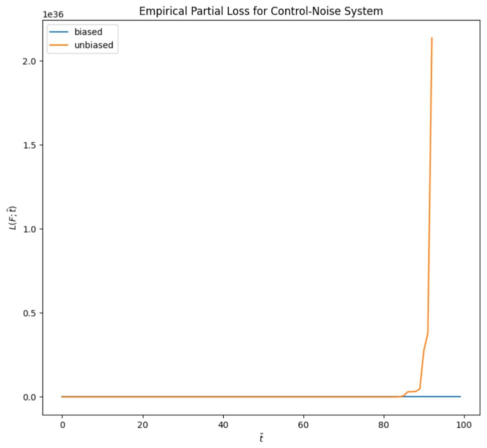
```python
# TODO
theta_0 = lambda a, b, beta: -a/b
"""Compare both policies when $a = 1.1$, $b = 1$, and $\beta =
    2$."""
a = 1.1
b = 1
beta = 2

env = MultiplicativeGaussianControlNoiseEnvironment(
    a = a,
    b = b,
    beta = beta,
    lmbda = 0
)

### U^*
linear_policy = Policy(FixedWeightM1P1LinearModule(theta_cn(a, b,
    beta)))
states_star, losses_star = sim_policy(100, 100, linear_policy, env)

### U^0
linear_policy = Policy(FixedWeightM1P1LinearModule(theta_0(a, b,
    beta)))
states_0, losses_0 = sim_policy(100, 100, linear_policy, env)
```

```python
mean_states_star = np.array(states_star).mean(axis=0)
mean_second_moments_star = np.power(states_star, 2).mean(axis=0)
mean_cumsum_losses_star =
    np.array(losses_star).cumsum(axis=1).mean(0)

mean_states_0 = np.array(states_0).mean(axis=0)
mean_second_moments_0 = np.power(states_0, 2).mean(axis=0)
mean_cumsum_losses_0 = np.array(losses_0).cumsum(axis=1).mean(0)

#Figure 3
fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(9,18))

ax[0].plot(mean_second_moments_star, label='biased')
ax[0].plot(mean_second_moments_0, label='unbiased')
```

```python
ax[0].set_yscale("log")
ax[0].set_xlabel(r'$t$')
ax[0].set_ylabel(r'$\overline{X_t^2}$')
ax[0].legend()
ax[0].set_title('Empirical␣$\mathbb{E}[X_{t}^{2}]$␣for␣
    Control-Noise␣System')

ax[1].plot(mean_cumsum_losses_star, label='biased')
ax[1].plot(mean_cumsum_losses_0, label='unbiased')
ax[1].set_xlabel(r'$\tilde{t}$')
ax[1].set_ylabel(r'$L(F;␣\tilde{t})$')
ax[1].legend()
ax[1].set_title('Empirical␣Partial␣Loss␣for␣Control-Noise␣System')
plt.show()
```



(a) Empirical Average Second Moment



(b) Empirical Partial Loss

Figure 3: Empirical Average Second Moment and Partial Loss of a Control Noise System

The empirical average state second moment of the biased policy decreases as the iterations go on, and the empirical average state second moment of the unbiased policy increases as the iterations go on. They different at about 10^50 when the iterations hit 100. The biased policy is more stable.

3. (a)

$$\begin{aligned}
E[X_{t+1}^2|Y_t] + \lambda U_t^2 &= E[(A_tX_t + B_tU_t)^2 \mid Y_t] + \lambda U_t^2 \\
&= E[A_t^2X_t^2 + 2A_tX_tB_tU_t + B_t^2U_t^2 \mid Y_t] + \lambda U_t^2 \\
&= (a^2+\alpha^2)Y_t^2 + 2Y_tU_tE[A_t]E[\ B_t] + (b^2+\beta^2)U_t^2 + \lambda U_t^2 \\
&= (a^2+\alpha^2)Y_t^2 + 2abY_tU_t + (b^2+\beta^2)U_t^2 + \lambda U_t^2
\end{aligned} \tag{22}$$

$$\begin{aligned}
F_t^*(Y_t) &= argmin_{U_t\in\mathbb{R}}\ E[X_{t+1}^2|Y_t] + \lambda U_t^2 \\
&= argmin_{U_t\in\mathbb{R}}\ (a^2+\alpha^2)Y_t^2 + 2abY_tU_t + (b^2+\beta^2)U_t^2 + \lambda U_t^2 \\
&= argmin_{U_t\in\mathbb{R}}\ 2abY_tU_t + (b^2+\beta^2)U_t^2 + \lambda U_t^2
\end{aligned} \tag{23}$$

We see that this equation is convex in U_t, which means taking the derivative will give us a global minimum.

$$\begin{aligned} \frac{d}{dU_t} 2abY_tU_t + (b^2+\beta^2)U_t^2 + \lambda U_t^2 &= 2abY_t + 2(b^2+\beta^2+\lambda)U_t \\ 0 &= 2abY_t + 2(b^2+\beta^2+\lambda)U_t^* \\ U_t^* &= \frac{-ab}{b^2+\beta^2+\lambda}Y_t \end{aligned} \tag{24}$$

Hence, the control policy is

$$F_t^*(Y_t) = \frac{-ab}{b^2+\beta^2+\lambda}Y_t \ , \ \forall t \geq 0 \tag{25}$$

$$\begin{aligned} E(X_{t+1}^2) &= E(E(X_{t+1}^2|Y_t) \\ &= E((a^2+\alpha^2)Y_t^2 + 2abY_tU_t + (b^2+\beta^2)U_t^2) \\ &= (a^2+\alpha^2)E(X_t^2) + 2ab\ (\frac{-ab}{b^2+\beta^2+\lambda^2})\ E(X_t^2) + (b^2+\beta^2)\ (\frac{-ab}{b^2+\beta^2+\lambda^2})^2\ E(X_t^2) \\ &= (a^2+\alpha^2 - 2\ \frac{a^2b^2}{b^2+\beta^2+\lambda}\ +\ \frac{(b^2+\beta^2)a^2b^2}{(b^2+\beta^2+\lambda)^2})\ E(X_t^2) \\ &= (\alpha^2 +\ \frac{a^2b^2+a^2\beta^2+a^2\lambda-2a^2b^2}{b^2+\beta^2+\lambda}\ +\frac{a^2\ (b^4+\beta^2b^2)}{(b^2+\beta^2+\lambda)^2})E(X_t^2) \\ &= (\alpha^2 +\ \frac{a^2(\beta^2+\lambda-b^2)}{b^2+\beta^2+\lambda}\ +\frac{a^2\ (b^4+\beta^2b^2)}{(b^2+\beta^2+\lambda)^2})\ E(X_t^2) \\ &= (\alpha^2 +\ \frac{a^2(\beta^2+\lambda-b^2)(\beta^2+\lambda+b^2)}{(b^2+\beta^2+\lambda)^2}\ +\frac{a^2\ (b^4+\beta^2b^2)}{(b^2+\beta^2+\lambda)^2})\ E(X_t^2) \\ &= (\alpha^2 +\ \frac{a^2[(\beta^2+\lambda)^2-b^4]}{(b^2+\beta^2+\lambda)^2}\ +\frac{a^2\ (b^4+\beta^2b^2)}{(b^2+\beta^2+\lambda)^2})\ E(X_t^2) \\ &= [\alpha^2 +\ \frac{a^2(\beta^2b^2+(\beta^2+\lambda)^2)}{(b^2+\beta^2+\lambda)^2}]^{\ t+1} \end{aligned} \tag{26}$$

Which means for $E(X_{t+1}^2)$ to converge as $t \to \infty$, the statement

$$\alpha^2 + \frac{a^2(\beta^2b^2+(\beta^2+\lambda)^2)}{(b^2+\beta^2+\lambda)^2} \leq 1 \tag{27}$$

must be true.

(b) With $a = a,\ b = 1,\ 0.5,\ \beta = 0.5,\ \lambda = 1$

$$\begin{aligned} 1 &\geq \alpha^2 + \frac{a^2\ (\beta^2b^2+(\beta^2+\lambda)^2)}{(b^2+\beta^2+\lambda)^2} \\ &= 0.25 + \frac{a^2(0.25+1.5^2)}{(2.25)^2} \\ a^2 &\leq \frac{(1-0.25)(2.25)^2}{(0.25+1.5)^2} \\ &= 1.2397959 \\ |a| &\leq 1.11346 \end{aligned} \tag{28}$$

Which means for the system to stabilize, $|a|$ has to be less than or equal to 1.11346.

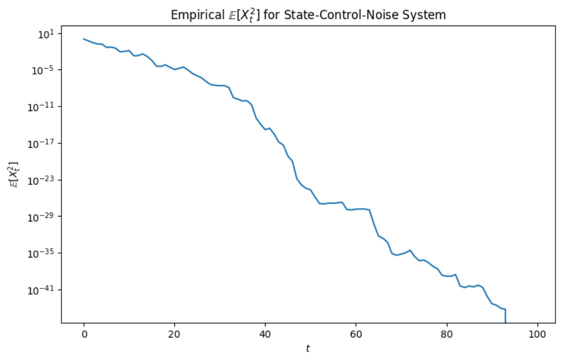
The following is from Jupyter Notebook, ***`main.py`*** *State-Control Noise section.*

The optimal linear coefficient θ found in Problem 4 for the state-control noise system as a function of a, b, α, β, and λ is listed below.

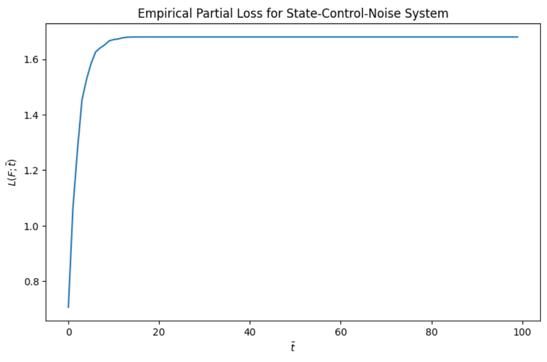
```python
# TODO
theta_scn = lambda a, b, alpha, beta, lmbda:
    -(a*b)/(b**2+beta**2+lmbda)
"""Analyze the system where $a = 1$, $b = 1$, $\alpha = 0.5$,
    $\beta = 0.5$, $\lambda = 1$."""
a = 1
b = 1
alpha = 0.5
beta = 0.5
lmbda = 1

env = MultiplicativeGaussianStateControlNoiseEnvironment(
    a = a,
    b = b,
    alpha = alpha,
    beta = beta,
    lmbda = lmbda
)

linear_policy = Policy(FixedWeightM1P1LinearModule(theta_scn(a, b,
    alpha, beta, lmbda)))
states, losses = sim_policy(100, 100, linear_policy, env)

#Figure 4
plot_empirical_average_second_moment_and_partial_loss(states,
    losses, "State-Control-Noise")
```

As we did in the case of Control Noise, for a variety of a (and with $b = 1$, $\alpha = 0.5$, $\beta = 0.5$, and $\lambda = 1$ as above), we rerun the above simulation

```python
#Figure 5
a_list = [3.0, 2.0, 1.4, 1.3, 1.0, 0.9]

a_losses = []

for a in a_list:
    env = MultiplicativeGaussianStateControlNoiseEnvironment(
        a = a,
        b = b,
        alpha = alpha,
        beta = beta,
        lmbda = lmbda
    )

    linear_policy =
        Policy(FixedWeightM1P1LinearModule(theta_scn(a, b, alpha,
        beta, lmbda)))
    states, losses = sim_policy(100, 100, linear_policy, env)
    plot_average_second_moment_trajectory(states, a)
```

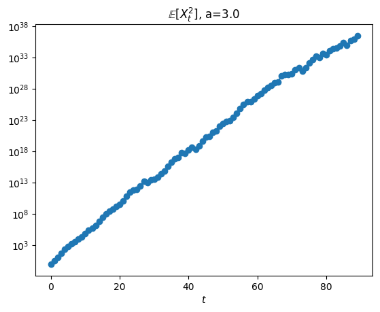


(a) Empirical Average Second Moment

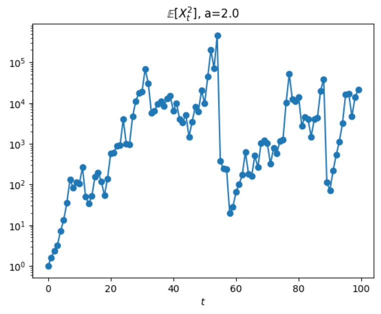


(b) Empirical Partial Loss

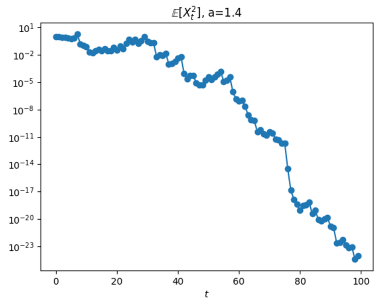
Figure 4: Empirical Average Second Moment and Partial Loss of a State-Control Noise System



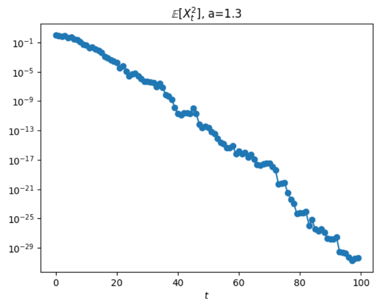
(a) $a = 3$



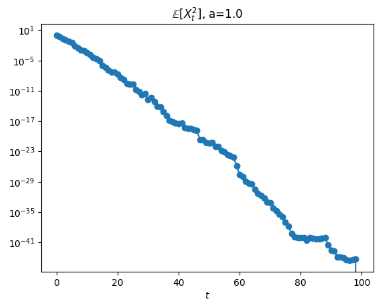
(b) $a = 2$



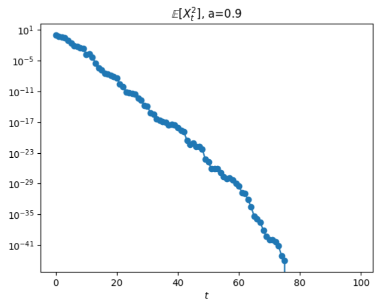
(c) $a = 1.4$



(d) $a = 1.3$



(e) $a = 1.0$



(f) $a = 0.9$

Figure 5: Different values of a

4. (a) Proof by Induction

Base Case: $t = 0$

$$\begin{aligned} E(X_0^2) &= (a^2 + 2ac\theta + (c^2 + \gamma^2)\theta^2)^0 \\ &= 1 \end{aligned} \tag{29}$$

Induction Hypothesis: If $\forall t \geq 0, U_t = F_t(Y_t) = \theta Y_t$, then

$$E(X_t^2) = (a^2 + 2ac\theta + (c^2 + \gamma^2)\theta^2)^t, \qquad \forall t \geq 0 \tag{30}$$

Induction Step:
Note: $U_t = \theta Y_t = \theta C_t X_t$

$$\begin{aligned} E(X_{t+1}^2) &= E((aX_t + U_t)^2) \\ &= E(a^2 X_t^2 + 2aX_t U_t + U_t^2) \\ &= a^2 E(X_t^2) + 2a\theta E(X_t Y_t) + \theta^2 E(Y_t^2) \\ &= a^2 E(X_t^2) + 2a\theta E(C_t X_t^2) + \theta^2 E(C_t^2 X_t^2) \\ &= a^2 E(X_t^2) + 2ac\theta E(X_t)^2 + \theta^2 (c^2 + \gamma^2) E(X_t^2) \\ &= (a^2 + 2ac\theta + (c^2 + \gamma^2)\theta^2) E(X_t^2) \\ &= [(a^2 + 2ac\theta + (c^2 + \gamma^2)\theta^2)][(a^2 + 2ac\theta + (c^2 + \gamma^2)\theta^2)^t] \\ &= (a^2 + 2ac\theta + (c^2 + \gamma^2)\theta^2)^{t+1} \end{aligned} \tag{31}$$

$\square$

(b)

$$\begin{aligned} \theta^* &= argmin_{\theta \in \mathbb{R},\ U_t = \theta Y_t}\ E[X_{t+1}^2] \\ &= argmin_{\theta \in \mathbb{R},\ U_t = \theta Y_t}\ (a^2 + 2ac\theta + (c^2 + \gamma^2)\theta^2)^{t+1} \\ &= argmin_{\theta \in \mathbb{R},\ U_t = \theta Y_t}\ a^2 + 2ac\theta + (c^2 + \gamma^2)\theta^2 \end{aligned} \tag{32}$$

We see that this equation is convex in θ, which means taking the derivative will give us a global minimum.

$$\begin{aligned} \frac{d}{d\theta} a^2 + 2ac\theta + (c^2 + \gamma^2)\theta^2 &= 2ac + 2(c^2 + \gamma^2)\theta \\ 0 &= 2ac + 2(c^2 + \gamma^2)\theta^* \\ \theta^* &= -\frac{ac}{c^2 + \gamma^2} \end{aligned} \tag{33}$$

Hence, the control policy is

$$F_t^*(Y_t) = -\frac{ac}{c^2 + \gamma^2} Y_t, \qquad \forall Y_t \in \mathbb{R} \tag{34}$$

This is similar to the optimal control policy derived in Problem 2. Instead of b and β, we have c and γ since now $B_t = 1$ (not random) and C_t is random (as opposed to Problem 2).

(c)

$$\begin{aligned} E(X_{t+1}^2) &= (a^2 + 2ac\theta + (c^2 + \gamma^2)\theta^2)^{t+1} \\ &= (a^2 + 2ac(\frac{-ac}{c^2 + \gamma^2}) + (c^2 + \gamma^2)(\frac{-ac}{c^2 + \gamma^2})^2)^{t+1} \\ &= (a^2 - \frac{2a^2c^2}{c^2 + \gamma^2} + \frac{a^2c^2}{c^2 + \gamma^2})^{t+1} \\ &= (\frac{a^2c^2 + a^2\gamma^2 - 2a^2c^2 + a^2c^2}{c^2 + \gamma^2})^{t+1} \\ &= (\frac{a^2\gamma^2}{c^2 + \gamma^2})^{t+1} \end{aligned} \tag{35}$$

Which means that

$$E(X_t^2) = (\frac{a^2\gamma^2}{c^2 + \gamma^2})^t, \qquad \forall t \geq 0 \tag{36}$$

(d) For the above equation to converge, we know that $\frac{a^2\gamma^2}{c^2+\gamma^2}$ has to be ≤ 1 or else as $t \to \infty$, $E(X_t^2) \to \infty$ and hence doesn't exist M such that $E(X_t^2) \leq M$ (so it is not stable).

$$\begin{aligned} E(X_t^2) &= (\frac{a^2\gamma^2}{c^2+\gamma^2})^t, \qquad \forall t \geq 0 \\ 1 &\geq \frac{a^2\gamma^2}{c^2+\gamma^2} \\ a^2 &\leq \frac{c^2+\gamma^2}{\gamma^2} \\ |a| &\leq \sqrt{1+\frac{c^2}{\gamma^2}} \end{aligned} \tag{37}$$

(e) *Note: This section of the problem is done in Jupyter Notebook, **main.py** Observation Noise section, with the prompt also reference in this document.*

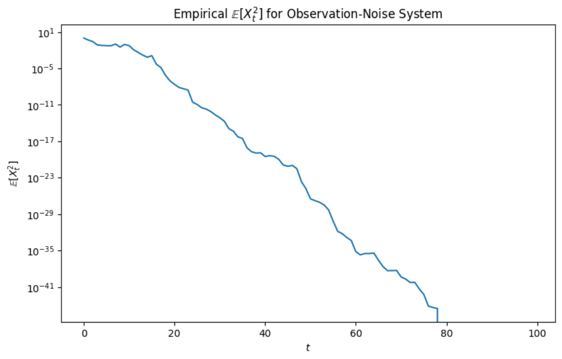
The optimal linear coefficient θ found in Problem 5 for the control noise system as a function of a, c, and γ is listed below.

```
# TODO
theta_on = lambda a, c, gamma: -(a*c)/(c**2+gamma**2)

"""Analyze the system where $a = 1$, $c = 1$, and $\gamma = 1$."""
a = 1
c = 1
gamma = 1
env = MultiplicativeGaussianObservationNoiseEnvironment(
    a = a,
    c = c,
    gamma = gamma,
    lmbda = 0
)

linear_policy = Policy(FixedWeightM1P1LinearModule(theta_on(a, c,
    gamma)))
states, losses = sim_policy(100, 100, linear_policy, env)

#Figure 6
plot_empirical_average_second_moment_and_partial_loss(states,
    losses, "Observation-Noise")
```

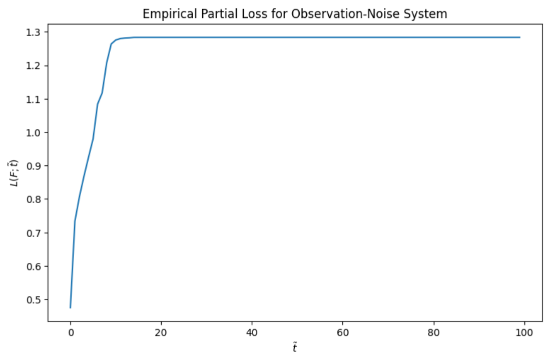
The performance is similar compared to the performance of the greedy linear policy in the beginning of the notebook.

As we did in the case of Control and State-Control Noise systems, for a variety of a, $c = 1$, $\gamma = 1$ (as above), we reran the above simulation and save $\overline{L}(F^*)$.

```
#Figure 7
a_list = [3, 2, 1.5, 1.4, 1.0, 0.9]
a_losses = []
for a in a_list:
    env = MultiplicativeGaussianObservationNoiseEnvironment(
        a = a,
        c = c,
        gamma = gamma,
        lmbda = 0
    )
    linear_policy = Policy(FixedWeightM1P1LinearModule(theta_on(a,
       c, gamma)))
    states, losses = sim_policy(100, 100, linear_policy, env)
    plot_average_second_moment_trajectory(states, a)
```

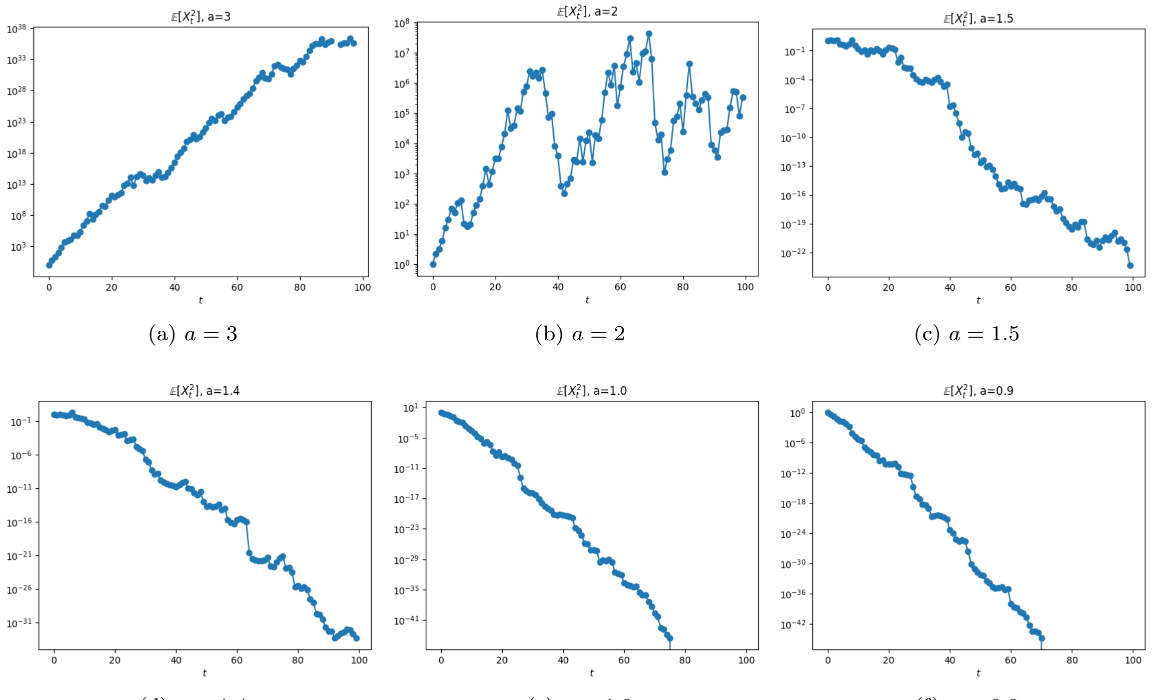


(a) Empirical Average Second Moment



(b) Empirical Partial Loss

Figure 6: Empirical Average Second Moment and Partial Loss of an Observation Noise System



(a) $a = 3$ (b) $a = 2$ (c) $a = 1.5$

(d) $a = 1.4$ (e) $a = 1.0$ (f) $a = 0.9$

Figure 7: Different values of a

(f) *Note: This section of the problem is done in Jupyter Notebook,* `main.py` *Period-1 vs Period-2 Observation Noise section, with the prompt also reference in this document.*

We implemented the memory-1 period-2 policy defined in Problem 4 (f) as

$$F'_t(Y_{(t)}) = \begin{cases} \frac{1}{2} + \frac{2}{5}|Y_t| & \text{if } t \text{ is even} \\ -\frac{1}{2} - \frac{1}{2}|Y_t| & \text{if } t \text{ is odd.} \end{cases} \tag{38}$$

```python
class FPrimeModule(torch.nn.Module):
    def forward(self, y_history: List[Tensor]):  # List[()] -> ()
        """
        Given the most recent observation, returns a control
           F_t(Y_t) = 0.5 + 0.4 * |Y_t| when t is even
        and -0.5 - 0.5 * |Y_t| when t is odd.
        """
        t = len(y_history)

        # TODO
        if t%2 ==1:
            return -1/2 -1/2*np.abs((y_history[-1]))
        return 1/2+2/5*np.abs((y_history[-1]))
```

Case: $a = 1$, $c = 0$, $\gamma = 1$

```python
a = 1
c = 0
gamma = 1
env = MultiplicativeGaussianObservationNoiseEnvironment(
    a = a,
    c = c,
    gamma = gamma,
    lmbda = 0
)
### F^*
linear_policy = Policy(FixedWeightM1P1LinearModule(theta_on(a, c,
   gamma)))
states_star, losses_star = sim_policy(100, 100, linear_policy, env)
### F^'
p2_policy = Policy(FPrimeModule())
states_prime, losses_prime = sim_policy(100, 100, p2_policy, env)
```

```python
mean_second_moments_star = np.power(states_star, 2).mean(axis=0)
mean_cumsum_losses_star =
   np.array(losses_star).cumsum(axis=1).mean(0)

mean_second_moments_prime = np.power(states_prime, 2).mean(axis=0)
mean_cumsum_losses_prime =
   np.array(losses_prime).cumsum(axis=1).mean(0)

#Figure 8
fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(9,12))

ax[0].plot(mean_second_moments_star, label=r"$F^*$")
ax[0].plot(mean_second_moments_prime, label=r"$F'$")
ax[0].set_xlabel(r'$t$')
ax[0].set_ylabel(r'$\mathbb{E}[X_t^2]$')
```

```
ax[0].set_yscale('log')
ax[0].legend()
ax[0].set_title('Empirical Average Second Moment for 
    Observation-Noise System')

ax[1].plot(mean_cumsum_losses_star, label=r"$F^*$")
ax[1].plot(mean_cumsum_losses_prime, label=r"$F'$")
ax[1].set_xlabel(r'$\tilde{t}$')
ax[1].set_ylabel(r'$L(U; \tilde{t})$')
ax[1].legend()
ax[1].set_title('Partial Empirical Loss for Observation-Noise 
    System')
plt.show()
```

When $a = 1$, $c = 0$, $\gamma = 1$, F^* is not the overall optimal policy for the observation noise system since F' has a better performamce in both the empirical average second moment and partial empirical loss.

Note from the plots above that the optimal linear memory-1 period-1 policy is in fact $F^*(Y_t) = 0$ when $C = 0$. Hence, $X_t = a^t \cdot X_0$ so when $|a| > 1$, this policy will necessarily not stabilize in the second moment. It remains unanswered, however, whether F' can stabilize a system with $|a| > 1$.

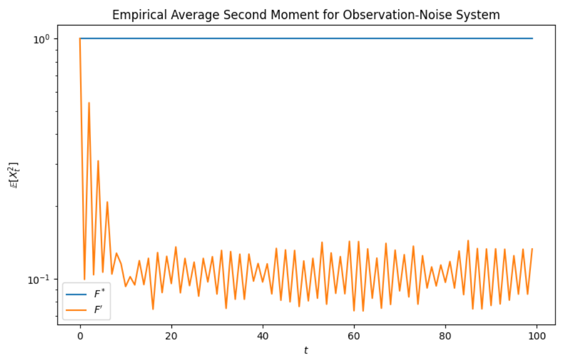
Case: $a = 1.01$, $c = 0$, $\gamma = 1$

```
a = 1.01
c = 0
gamma = 1
env = MultiplicativeGaussianObservationNoiseEnvironment(
    a = a,
    c = c,
    gamma = gamma,
    lmbda = 0
)
### F^*
linear_policy = Policy(FixedWeightM1P1LinearModule(theta_on(a, c,
    gamma)))
states_star, losses_star = sim_policy(100, 100, linear_policy, env)

### F^'
p2_policy = Policy(FPrimeModule())
states_prime, losses_prime = sim_policy(100, 100, p2_policy, env)
```

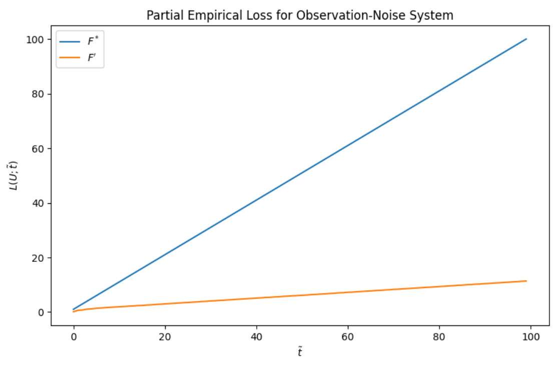
```
mean_second_moments_star = np.power(states_star, 2).mean(axis=0)
mean_cumsum_losses_star =
    np.array(losses_star).cumsum(axis=1).mean(0)

mean_second_moments_prime = np.power(states_prime, 2).mean(axis=0)
mean_cumsum_losses_prime =
    np.array(losses_prime).cumsum(axis=1).mean(0)

#Figure 9
fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(9,12))

ax[0].plot(mean_second_moments_star, label=r"$F^*$")
```

```
ax[0].plot(mean_second_moments_prime, label=r"$F'$")
ax[0].set_xlabel(r'$t$')
ax[0].set_ylabel(r'$\mathbb{E}[X_t^2]$')
ax[0].set_yscale('log')
ax[0].legend()
ax[0].set_title('Empirical␣$\mathbb{E}[X_t^2]$␣for␣
    Observation-Noise␣System')

ax[1].plot(mean_cumsum_losses_star, label=r"$F^*$")
ax[1].plot(mean_cumsum_losses_prime, label=r"$F'$")
ax[1].set_xlabel(r'$\tilde{t}$')
ax[1].set_ylabel(r'$L(F);␣\tilde{t})$')
ax[1].legend()
ax[1].set_title('Empirical␣Partial␣Loss␣for␣Observation-Noise␣
    System')
plt.show()
```

By inspection, F' seems to stabilize the system, as contrast to F^*.

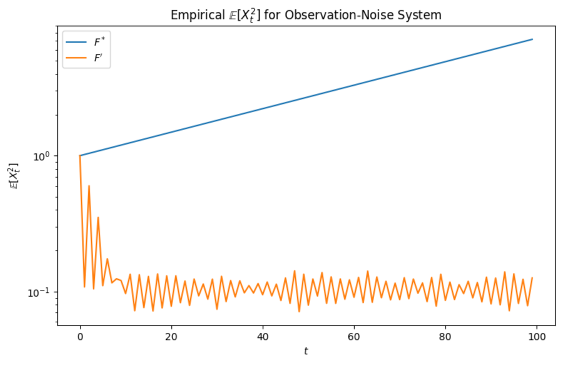


(a) Empirical Average Second Moment

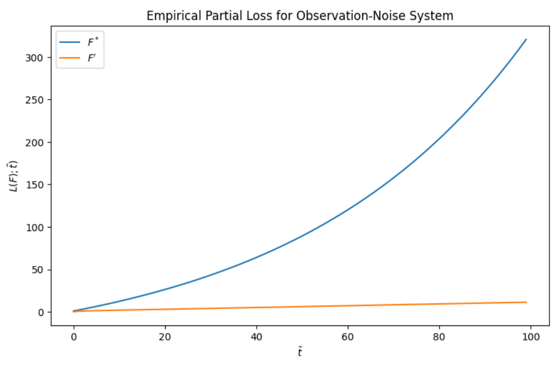


(b) Empirical Partial Loss

Figure 8: Empirical Average Second Moment and Partial Loss of an Observation Noise System



(a) Empirical Average Second Moment



(b) Empirical Partial Loss

Figure 9: Empirical Average Second Moment and Partial Loss of an Observation Noise System

5. (a) Assume that $U_t = F_t(Y_{(t)};\theta) = \theta Y_t$

$$\begin{aligned}
\nabla_\theta \log(\pi_\theta(\tilde{U}_t|Y_t)) &= \nabla_\theta \log(\frac{1}{\sqrt{2\pi\omega}} e^{-\frac{1}{2}(\frac{\tilde{u}_t - \theta Y_t}{\omega})^2}) \\
&= \nabla_\theta(-\frac{1}{2}(\frac{\tilde{u}_t - \theta Y_t}{\omega})^2) \log(\frac{1}{\sqrt{2\pi\omega}}) \\
&= ((\frac{\tilde{u}_t - \theta Y_t}{\omega})\frac{Y_t}{\omega}) \log(\frac{1}{\sqrt{2\pi\omega}}) \\
&= (\frac{\tilde{u}_t - \theta Y_t}{\omega^2}) Y_t \log(\frac{1}{\sqrt{2\pi\omega}}) \qquad \forall t \geq 0
\end{aligned} \tag{39}$$

6. (a) *The following is from Jupyter Notebook, `policies/our_policy_modules.py`.*

```python
import typing

import torch.nn as nn
from torch import Tensor
import torch
```

M1P1

```python
class M1P1ControlPolicy(nn.Module):
    """
    Implements the policy function F(Y_(t)) = theta * Y_t.
    """
    def forward(self, y_history: typing.List[Tensor]):  # List[()]
        -> ()
        y_t: Tensor = y_history[-1]  # ()
        u_t: Tensor = self.theta * y_t  # ()
        return u_t

class FixedWeightM1P1ControlPolicy(M1P1ControlPolicy):
    """
    Implements the policy function F(Y_(t)) = theta * Y_t, for
        theta a parameter whose value is fixed at initialization.
    """
    def __init__(self, theta: float):
        super(FixedWeightM1P1ControlPolicy, self).__init__()
        self.theta: nn.Parameter =
            nn.Parameter(data=torch.tensor(theta),
            requires_grad=False)

class LearnableWeightM1P1ControlPolicy(M1P1ControlPolicy):
    """
    Implements the policy function F(Y_(t)) = theta * Y_t, for
        theta a parameter whose value is trainable.
    Initializes theta_0 = 0.
    """
    def __init__(self):
        super(LearnableWeightM1P1ControlPolicy, self).__init__()
        self.theta: nn.Parameter =
            nn.Parameter(data=torch.zeros(size=()),
            requires_grad=True)
```

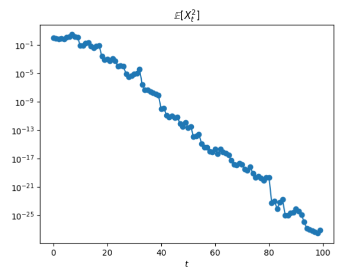
M2P1

```
class M2P1ControlPolicy(nn.Module):
    """
    Implements the policy function:
        F(Y_(t)) = theta_0 + theta_1 * Y_t for t=0
        F(Y_(t)) = theta_0 + theta_1 * Y_t + theta_2 * Y_t-1 for t>=1
    """
    def forward(self, y_history: typing.List[Tensor]):  # List[()] -> ()

        if len(y_history)==1:
            y_t: Tensor = y_history[-1]
            theta_0: Tensor = self.theta[0]
            theta_1: Tensor = self.theta[1]
            u_t: Tensor = theta_0 + theta_1*y_t
        else:
            y_t: Tensor = y_history[-1]
            y_t_1: Tensor = y_history[-2]
            theta_0: Tensor = self.theta[0]
            theta_1: Tensor = self.theta[1]
            theta_2: Tensor = self.theta[2]
            u_t: Tensor = theta_0 + theta_1*y_t + theta_2*y_t_1

        return u_t

class FixedWeightM2P1ControlPolicy(M2P1ControlPolicy):
    """
    Implements the policy function
        F(Y_(t)) = theta_0 + theta_1 * Y_t for t=0
        F(Y_(t)) = theta_0 + theta_1 * Y_t + theta_2 * Y_t-1 for t>=1
    for theta a parameter whose value is fixed at initialization.
    """
    def __init__(self, theta: float):
        super(FixedWeightM2P1ControlPolicy, self).__init__()
        self.theta: nn.Parameter = nn.Parameter(data=torch.tensor(theta), requires_grad=False)

class LearnableWeightM2P1ControlPolicy(M2P1ControlPolicy):
    """
    Implements the policy function
        F(Y_(t)) = theta_0 + theta_1 * Y_t for t=0
        F(Y_(t)) = theta_0 + theta_1 * Y_t + theta_2 * Y_t-1 for t>=1
    for theta a parameter whose value is trainable.
    Initializes theta_0 = 0.
    """
    def __init__(self):
        super(LearnableWeightM2P1ControlPolicy, self).__init__()
        self.theta: nn.Parameter = nn.Parameter(data=torch.zeros(3), requires_grad=True)
```

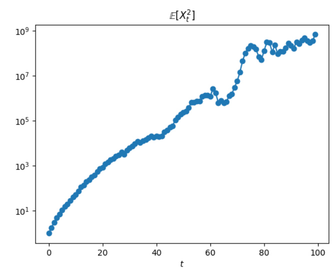
M1P2

```
class M1P2ControlPolicy(nn.Module):
    """
    Implements the policy function
        F(Y_(t)) = theta_0 + theta_1 * Y_t for t is even
        F(Y_(t)) = theta_2 + theta_3 * Y_t for t is odd
    """
    def forward(self, y_history: typing.List[Tensor]):  # List[()]
       -> ()
        y_t: Tensor = y_history[-1]  # ()

        if len(y_history)%2==1: #if y_t is even
            theta_0: Tensor = self.theta[0]
            theta_1: Tensor = self.theta[1]
            u_t: Tensor =  theta_0 + theta_1*y_t
        else:
            theta_2: Tensor = self.theta[2]
            theta_3: Tensor = self.theta[3]
            u_t: Tensor = theta_2 + theta_3*y_t

        return u_t

class FixedWeightM1P2ControlPolicy(M1P2ControlPolicy):
    """
    Implements the policy function
        F(Y_(t)) = theta_0 + theta_1 * Y_t for t is even
        F(Y_(t)) = theta_2 + theta_3 * Y_t for t is odd
    for theta a parameter whose value is fixed at initialization.
    """
    def __init__(self, theta: float):
        super(FixedWeightM1P2ControlPolicy, self).__init__()
        self.theta: nn.Parameter =
           nn.Parameter(data=torch.tensor(theta),
           requires_grad=False)

class LearnableWeightM1P2ControlPolicy(M1P2ControlPolicy):
    """
    Implements the policy function
        F(Y_(t)) = theta_0 + theta_1 * Y_t for t is even
        F(Y_(t)) = theta_2 + theta_3 * Y_t for t is odd
    for theta a parameter whose value is trainable.
    Initializes theta_0 = 0.
    """
    def __init__(self):
        super(LearnableWeightM1P2ControlPolicy, self).__init__()
        self.theta: nn.Parameter =
           nn.Parameter(data=torch.zeros(4), requires_grad=True)
```

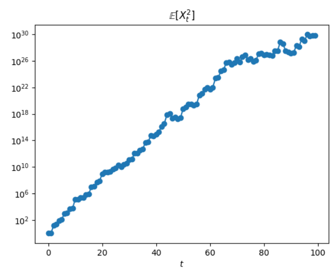
(b) Visualizing all three policy results, we can see that M1P1 tends to do best and get closest to converging compared to M2P1 and M1P2. M1P1 converges at around 1.4, M1P2 converges at around 1.1, and M2P1 converges at around 1.5.



(a) M1P1 at a =1.4



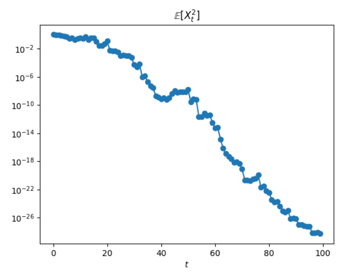
(b) M2P1 at a =1.4



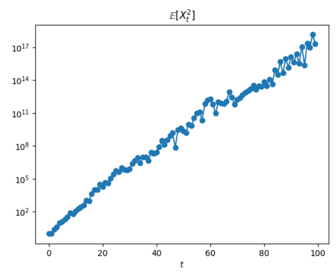
(c) M1P2 at a =1.4

Figure 10: Train policies on control noise system

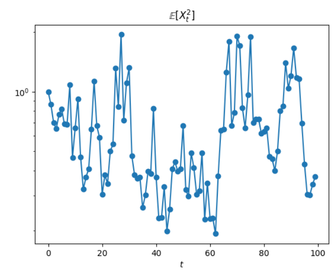
(c) Visualizing all three policy results, we can see that M1P1 again tends to do best and gets closest to converging compared to M2P1 and M1P2. M1P1 converges at around 1.4, M1P2 converges at around 1.2, and M2P1 converges at around 1.2.



(a) M1P1 at a =1.4



(b) M2P1 at a =1.4



(c) M1P2 at a =1.4

Figure 11: Train policies on observation noise system

4 Extensions

4.1 Main Idea

In the previous sections, we discussed what would happen when specific noise variables became random variables that we would have to take into account instead of being deterministic or 1. However in reality, we will most likely not know any of these noise variables and as such it would be useful to understand how optimal control policy does when Random Variables A, B, and C are random in:

$$X_{t+1} = A_t X_t + B_t U_t \tag{40}$$

$$Y_t = C_t X_t \tag{41}$$

4.2 Methodology

Let's now prove all of the

In our case, A_t, B_t, and C_t are random variables, while X_0 is deterministic.

$$\begin{aligned} E[A_t] = a,\ Var[A_t] = \alpha^2 \\ E[B_t] = b,\ Var[B_t] = \beta^2 \\ E[C_t] = c,\ Var[C_t] = \gamma^2 \\ X_0 = 1 \text{ deterministic } (\mu = 1,\ \sigma = 0) \end{aligned} \tag{42}$$

$$\begin{aligned}
E[X_{t+1}^2] &= E[(A_t X_t + \beta_t U_t)^2] \\
&= E[A_t^2 X_t^2 \ + 2A_t\beta_t X_t U_t \ + \beta_t^2 U_t^2] \\
&= E[A_t^2]E[X_t^2] + 2E[A_t]E[B_t]E[X_t U_t] + E[\beta_t^2]E[U_t^2] \\
&= (a^2+\alpha^2)\ E[X_t^2] + 2ab\theta E[X_t Y_t] + (\beta^2+b^2)\ \theta^2 E[Y_t^2] \\
&= (a^2+\alpha^2)\ E[X_t^2] + 2abc\theta E[X_t^2] + (\beta^2+b^2)\ \theta^2\ (c^2+\gamma^2)E[X_t^2] \\
&= E[X_t^2](a^2+\alpha^2+2abc\ \theta + (\beta^2+b^2)\ \theta^2(c^2+\gamma^2))
\end{aligned} \tag{43}$$

Proof by Induction

Want to prove :

$$E[X_t^2] = (a^2+\alpha^2+2abc\ \theta + (\beta^2+b^2)\ \theta^2(c^2+\gamma^2))^t \tag{44}$$

Base Case: $t = 0$: If $t = 0, E[X_0] = 1$ which is the same thing as $X_0 = 1$

Inductive Hypothesis: Assume we know

$$E[X_t^2] = (a^2+\alpha^2+2abc\ \theta + (\beta^2+b^2)\ \theta^2(c^2+\gamma^2))^t \tag{45}$$

Inductive Step: (Prove for it)

$$\begin{aligned}
E[X_{t+1}^2] &= E[X_t^2](a^2+\alpha^2+2abc\ \theta + (\beta^2+b^2)\ \theta^2(c^2+\gamma^2)) \\
&= (a^2+\alpha^2+2abc\ \theta + (\beta^2+b^2)\ \theta^2(c^2+\gamma^2))^{t+1} \\
&= (z(\theta))^{t+1}
\end{aligned} \tag{46}$$

Define $\theta^* = argmin E[X_{t+1}^2]$

$$\begin{aligned}
\frac{\partial}{\partial\theta}\ z(\theta) &= 2abc + 2(\beta^2+b^2)\ \theta\ (c^2+\gamma^2) \\
0 &= 2\ (\beta^2+b^2)\ (c+\gamma^2)
\end{aligned} \tag{47}$$

All the terms are positive when squared and as such the function must be PSD and therefore convex so we can do:

$$\begin{aligned}
\frac{\partial}{\partial\theta}\ z(\theta) &= 2abc + 2(\beta^2+b^2)\ \theta\ (c^2+\gamma^2) \\
2abc &= 2\ (\beta^2+b^2)\theta\ (c+\gamma^2) \\
\theta^* &= \frac{abc}{(\beta^2+b^2)(c^2+\gamma^2)}
\end{aligned} \tag{48}$$

The control policy is therefore

$$F_t^*(Y_{(t)}) = -\frac{abc}{(\beta^2+b^2)(c^2+\gamma^2)}\ Y_t \tag{49}$$

To find the value of a^* allowed, we compute the following:

$$\begin{aligned} E[X_t^2] &= (\alpha^2 + a^2 + 2abc\theta \ + (\beta^2 + b^2)\ \theta^2\ (c^2+\gamma^2))^t \\ &= (\alpha^2 + a^2 + 2abc\frac{-abc}{(\beta^2+b^2)\ (c^2+\gamma^2)}\ + (\beta^2+b^2)\ (\frac{-abc}{(\beta^2+b^2)\ (c^2+\gamma^2)})^2\ (c^2+\gamma^2))^t \\ &= (\alpha^2 + a^2 + 2(\frac{-a^2b^2c^2}{(\beta^2+b^2)(c^2+\gamma^2)})\ + (\frac{a^2b^2c^2}{(\beta^2+b^2)\ (c^2+\gamma^2)}))^t \\ &= (\alpha^2 + a^2 - \frac{a^2b^2c^2}{(\beta^2+b^2)\ (c^2+\gamma^2)})^t \\ &= (\alpha^2 + \frac{a^2\ (\beta^2+b^2)(c^2+\gamma^2) - a^2b^2c^2}{(\beta^2+b^2)\ (c^2+\gamma^2)})^t \\ &= (\alpha^2 + \frac{a^2\ (\beta^2c^2+\beta^2\gamma^2+b^2\gamma^2)}{(\beta^2+b^2)\ (c^2+\gamma^2)})^t \end{aligned} \tag{50}$$

$$\begin{aligned} 1 &= (\alpha^2 + \frac{a^2\ (\beta^2c^2+\beta^2\gamma^2+b^2\gamma^2)}{(\beta^2+b^2)\ (c^2+\gamma^2)})^t \\ (1-\alpha^2)\ (\beta^2+b^2)\ (c^2+\gamma^2) &= a^2\ (\beta^2c^2+\beta^2\gamma^2+b^2\gamma^2) \\ |a| &\le \sqrt{(1-\alpha^2)\frac{(\beta^2+b^2)\ (c^2+\gamma^2)}{(\beta^2c^2+\beta^2\gamma^2+b^2\gamma^2)}} \end{aligned} \tag{51}$$

Here are the code and graphs related to the work of extension.

```python
class MultiplicativeGaussianObservationNoiseEnvironment_all_random
(MultiplicativeGaussianNoiseEnvironment):
    """
    Defines the transition dynamics of the environment with
       multiplicative observation noise according to the equations:
    x_(t+1) = A_t*x_t + B_t*U_t
    y_t     = C_t*X_t
    l_t     = x_(t+1)^2 + lambda * u_t
    where C_t ~ N(c, gamma^2) and X_0 = 1.
    """
    def __init__(self, a: float, b: float, c: float, alpha: float, gamma:
       float, beta: float, lmbda: float):
        super(MultiplicativeGaussianObservationNoiseEnvironment_all_random,
            self).__init__(
             a=a, b=b, c=c, mu=1.0, alpha=alpha, beta=beta, gamma=gamma,
                 sigma=0.0, lmbda=lmbda
        )
__all__ = [
    "MultiplicativeGaussianNoiseEnvironment",
       "MultiplicativeGaussianControlNoiseEnvironment",
    "MultiplicativeGaussianStateControlNoiseEnvironment",
       "MultiplicativeGaussianObservationNoiseEnvironment",
    "MultiplicativeGaussianObservationNoiseEnvironment_all_random"
]
```

```python
theta_all_random = lambda a, b, c, alpha, gamma, beta: -(a*b*c)/((b**2 +
   beta**2)*(c**2+gamma**2))

"""Analyze the system where $a = 1$, $b = 1$,, $c=1$, $\alpha = 0.5$,
   $\gamma = 1$, $\beta = 1$."""
```

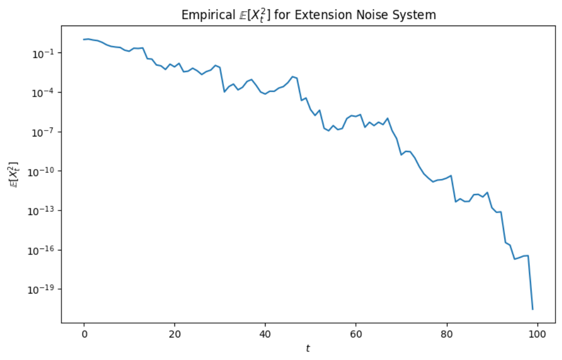
```
a = 1
b = 1
c = 1
alpha = 0.5
gamma = 1
beta = 1

env = MultiplicativeGaussianObservationNoiseEnvironment_all_random(
    a = a,
    b = b,
    c = c,
    alpha = alpha,
    gamma = gamma,
    beta = beta,
    lmbda = 0
)

### F^*
linear_policy = Policy(FixedWeightM1P1LinearModule(theta_all_random(a, b,
   c, alpha, gamma, beta)))
states_all, losses_all = sim_policy(100, 100, linear_policy, env)

#Figure 12
plot_empirical_average_second_moment_and_partial_loss(states_all,
   losses_all, "Control-Noise")
```

The performance of the control policy seems fairly good with a control noise system using A, B, and C as random variables. It has a higher average empirical second moment and partial loss than the cases where we removed one of these random variables and made it deterministic; however, this is expected as there are more uncertainties in our system.

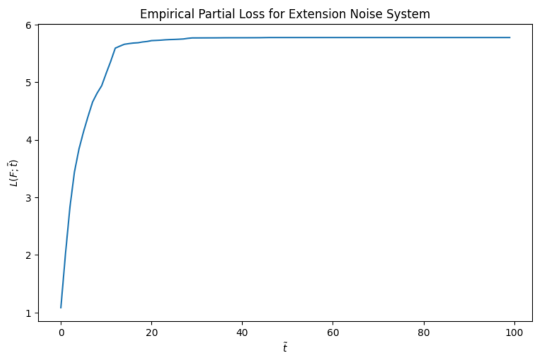
As we did in the other systems, now, we graph our system with a variety of a listed below, reran the above simulation, and save $\overline{L}(F^*)$.

```
#Figure 13
a_list = [2, 1.5, 1.4, 1.3, 1.2, 1.0, 0.9]

a_losses = []

for a in a_list:
    env = MultiplicativeGaussianObservationNoiseEnvironment_all_random(
        a = a,
        b = b,
        c = c,
        alpha = alpha,
        gamma = gamma,
        beta = beta,
        lmbda = 0
    )

    linear_policy_all =
       Policy(FixedWeightM1P1LinearModule(theta_all_random(a, b, c,
       alpha, gamma, beta)))
    states_all, losses = sim_policy(100, 100, linear_policy_all, env)
    plot_average_second_moment_trajectory(states_all, a)
```

Values of a that are stabilized by our analysis is when $a \leq 1$ as shown by the calculation and in the graphs

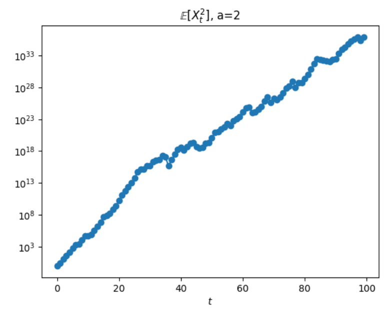


(a) Empirical Average Second Moment

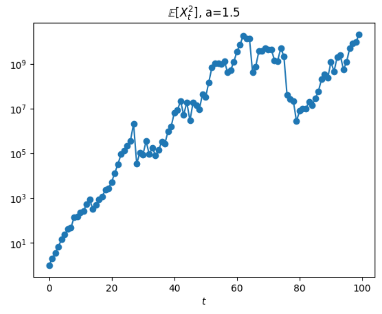


(b) Empirical Partial Loss

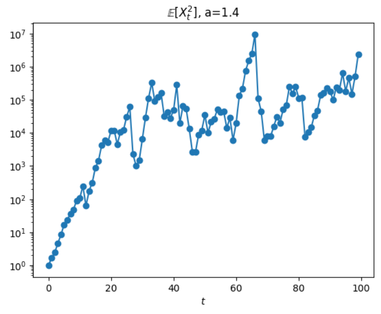
Figure 12: Empirical Average Second Moment and Partial Loss of the Extension Noise System



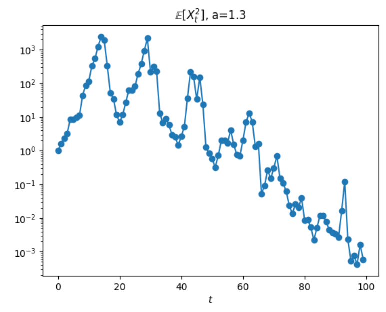
(a) $a = 2$



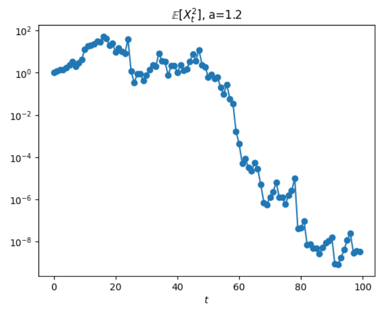
(b) $a = 1.5$



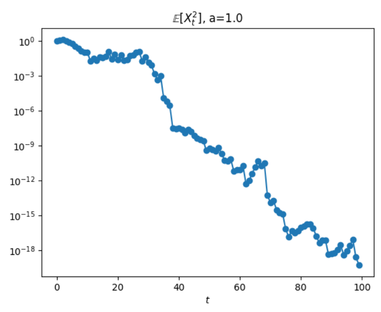
(c) $a = 1.4$



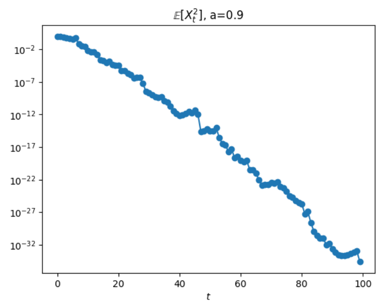
(d) $a = 1.3$



(e) $a = 1.2$



(f) $a = 1.0$



(g) $a = 0.9$

Figure 13: Different values of a

4.3 Results

Hence, if the system wants to be stabilized when all A, B, and C are random variables, the growth of the system has to be bounded, i.e.

$$1 < a < a^* \tag{52}$$

where a^* is defined as

$$a^* \leq \sqrt{(1-\alpha^2)\frac{(\beta^2+b^2)\,(c^2+\gamma^2)}{(\beta^2c^2+\beta^2\gamma^2+b^2\gamma^2)}} \tag{53}$$

5 Contributions

Karen, Ali, and Riyya read the papers and worked together to solve all the problems while cross-checking each other's work and making sure everyone reached the same conclusion. Ali and Riyya took on the role of finding an extension question to work on, while Karen took on the role of typesetting LaTex as well as checking over the extension result.