Online Optimization of Neural Network Controls for Stochastic Differential Equations

1077710

Kellogg College

University of Oxford

A thesis submitted for the degree of

Master of Science

Trinity 2024

Acknowledgements

I would like to express my gratitude to my supervisor, Justin Sirignano, for his invaluable guidance and encouragement throughout this research.

I extend my appreciation to my teachers in the Department of Math, for their wonderful teaching and assistance.

On a personal note, I would like to thank my parents, for their endless support and belief in me.

Contents

List of Figures

Chapter 1

Introduction

1.1 Backgrounds

Stochastic optimal control is a crucial field in applied mathematics and engineering. It is vital due to its ability to address uncertainty in dynamic systems, improve decision-making, drive mathematical innovation, and impact a wide range of disciplines and applications. Such as mathematical finance. For example, in Portfolio Optimisation and Optimal Execution problems.

The problems we are facing is that, high-dimensional stochastic optimal control presents significant computational challenges. Since the optimal control must satisfy the Hamilton-Jacobi-Bellman (HJB) equation, which becomes computationally infeasible to solve in high-dimensional settings. Hence we need to implement a new online optimization method for training neural network controls for SDEs that remains computationally feasible in high dimensions. The paper Continuous-time stochastic gradient descent for optimizing over the stationary distribution of stochastic differential equations, by Wang, Ziheng and Sirignano, Justin [4] have given a successful implementation of the method which is feasible in the high dimensional cases.

1.2 Objectives of This Dissertation

In this dissertation we re-implemented those examples and conducted research beyond the original paper, such as exploring higher dimensions, adding correlations, and studying cases of non-linear processes. Additionally, we went over the principle of this given method (algorithm) and verify the principles of the original paper.

1.3 Paper Outline

The thesis is structured as follows. First we explain the main theorem of the original paper and justify it. Then we use the given method as a tenet for the different algorithms for different cases. We do the implementation from simple to complicated cases. Next, in Chapter 4 we go beyond the original research to try some new things. Then, in the last chapter, we conclude and summarize the main findings of your research, as well as suggest directions for future research based on the findings and limitations.

Chapter 2

Main Theorem

2.1 The Main Theorem

Given a multi-dimensional Ornstein-Uhlenbeck process:

$$dX_t^\theta = (g(\theta) - h(\theta)X_t^\theta)dt + dW_t, \quad X_0^\theta = x \tag{2.1}$$

where $\theta \in \mathbb{R}^l$, $g(\theta) \in \mathbb{R}^d$, $h(\theta) \in \mathbb{R}^{d\times d}$, $W_t \in \mathbb{R}^d$, $X_t^\theta \in \mathbb{R}^d$ and σ is a scalar constant. Let π_θ be the stationary distribution of X_t^θ. Our goal is to solve the optimization problem

$$\min_\theta J(\theta) = \min_\theta (\mathbb{E}_{\pi_\theta} f(Y) - \beta)^2 \tag{2.2}$$

where β is a constant. To solve (2.3), we have the main online algorithm:

$$\begin{aligned}
\frac{d\theta_t}{dt} &= -2\alpha_t \left(f\left(\bar{X}_t\right) - \beta\right) \nabla f\left(X_t\right) \tilde{X}_t \\
dX_t &= \left(g\left(\theta_t\right) - h\left(\theta_t\right) X_t\right) dt + \sigma dW_t \\
\frac{d\tilde{X}_t}{dt} &= \nabla_\theta g\left(\theta_t\right) - \nabla_\theta h\left(\theta_t\right) X_t - h\left(\theta_t\right) \tilde{X}_t \\
d\bar{X}_t &= \left(g\left(\theta_t\right) - h\left(\theta_t\right) \bar{X}_t\right) dt + \sigma d\bar{W}_t
\end{aligned} \tag{2.3}$$

where W_t and $\tilde{W}_t$ are independent Brownian motions, $\nabla_\theta g(\theta) \in \mathbb{R}^{d\times l}$, $\nabla_\theta h(\theta) \in \mathbb{R}^{d\times d\times l}$ and $\tilde{X}_t \in \mathbb{R}^{d\times l}$ is the gradient process for X_t.

Our convergence theorem will require the following assumptions.

Assumption

- $g(\theta), \nabla_\theta^i g(\theta), h(\theta)$ and $\nabla_\theta^i h(\theta)$ are uniformly bounded functions for $i = 1, 2$.

- h is symmetric and uniformly positive definite, i.e. there exists a constant $c > 0$ such that

$$\min\left\{x^\top h(\theta)x\right\} \geq c|x|^2, \quad \forall\theta \in \mathbb{R}^\ell, x \in \mathbb{R}^d$$

- $f, \nabla^i f, i = 1, 2, 3$ are polynomially bounded.

$$|f(x)| + \sum_{i=1}^{3} \left|\nabla^i f(x)\right| \leq C\left(1 + |x|^{\hat{m}}\right), \quad \forall x \in \mathbb{R}^d$$

  for some constant $C, \hat{m} > 0$.

- The learning rate α_t satisfies $\int_0^\infty \alpha_t dt = \infty, \int_0^\infty \alpha_t^2 dt < \infty, \int_0^\infty |\alpha_s'|\, ds < \infty$, and there is a $\hat{p} > 0$ such that $\lim_{t\to\infty} \alpha_t^2 t^{\frac{1}{2}+2\hat{p}} = 0$.

Under these assumptions, we are able to prove the following convergence result.

The Main Theorem

Under above assumptions and for the Ornstein-Uhlenbeck process (2.1), the algorithm (2.3) will converge to a stationary point almost surely:

$$\lim_{t\to\infty} \left|\nabla_\theta J\left(\theta_t\right)\right| \overset{\text{a.s.}}{=} 0 \tag{2.4}$$

2.2 Intuitive Inspiration

The Old Way

In the algorithm we use $\left(f\left(\bar{X}_t\right) - \beta\right)\left(\nabla f\left(X_t\right)\tilde{X}_t\right)^\top$ as a stochastic estimate for $\nabla_\theta J\left(\theta_t\right)$, where $J(\theta) = \sum_{n=1}^{N}\left(\mathbf{E}_{\pi_\theta}\left[f_n(Y)\right] - \beta_n\right)^2$, and it is used to update θ_t.

To better understand the main online algorithm (2.3), we first re-write the gradient of the objective function using the ergodicity of X_t^θ in the case of $N = 1$:

$$\begin{aligned}\nabla_\theta J(\theta) &= 2\left(\mathbf{E}_{Y\sim\pi_\theta} f(Y) - \beta\right)\nabla_\theta \mathbf{E}_{Y\sim\pi_\theta} f(Y) \\ &\overset{\text{a.s.}}{=} 2\left(\lim_{T\to\infty}\frac{1}{T}\int_0^T f\left(X_t^\theta\right)dt - \beta\right)\cdot\nabla_\theta\left(\lim_{T\to\infty}\frac{1}{T}\int_0^T f\left(X_t^\theta\right)dt\right)\end{aligned}$$

Assume the derivative and the limit can be interchanged, the gradient can be expressed as

$$\nabla_\theta J(\theta) = 2\left(\lim_{T\to\infty}\frac{1}{T}\int_0^T f\left(X_t^\theta\right)dt - \beta\right)\cdot\lim_{T\to\infty}\frac{1}{T}\int_0^T \nabla f\left(X_t^\theta\right)\nabla_\theta X_t^\theta dt$$

Define $\tilde{X}_t^\theta = \nabla_\theta X_t^\theta$ and, under mild regularity conditions for the coefficients[3][2], $\tilde{X}_t^\theta$ will satisfy

$$d\tilde{X}_t^\theta = \left(\nabla_x\mu\left(X_t^\theta,\theta\right)\tilde{X}_t^\theta + \nabla_\theta\mu\left(X_t^\theta,\theta\right)\right)dt + \left(\nabla_x\sigma\left(X_t^\theta,\theta\right)\tilde{X}_t^\theta + \nabla_\theta\sigma\left(X_t^\theta,\theta\right)\right)dW_t$$

Note that $\tilde{X}_t$ and $\tilde{X}_t^\theta$ satisfy the same equations, except θ is a fixed constant for $\tilde{X}_t^\theta$ while θ_t is updated continuously in time for $\tilde{X}_t$. Then, we have that

$$\nabla_\theta J(\theta) = 2\left(\lim_{T\to\infty}\frac{1}{T}\int_0^T f\left(X_t^\theta\right)dt - \beta\right)\cdot\lim_{T\to\infty}\frac{1}{T}\int_0^T \nabla f\left(X_t^\theta\right)\tilde{X}_t^\theta dt$$

The formula above can be used to evaluate $\nabla_\theta J(\theta)$ and thus allows for optimization via a gradient descent algorithm. However, the problem is, X_t^θ must be simulated for a large time period $[0,T]$ for each optimization iteration, which is computationally costly.

The New Way

A natural alternative is to develop a continuous-time stochastic gradient descent algorithm which updates θ using a stochastic estimate $G(\theta_t)$ for $\nabla_\theta J(\theta_t)$, where $G(\theta_t)$ asymptotically converges to an unbiased estimate for the direction of steepest descent $\nabla_\theta J(\theta_t)$. The online algorithm in the main theorem does exactly this using $G(\theta_t) = 2\left(f\left(\bar{X}_t\right)-\beta\right)\nabla f\left(X_t\right)\tilde{X}_t$ as a stochastic estimate for $\nabla_\theta J(\theta_t)$.

For large t, we expect that $\mathbf{E}\left[f\left(\bar{X}_t\right)-\beta\right] \approx \mathbf{E}_{Y\sim\pi_{\theta_t}}[f(Y)-\beta]$ and $\mathbf{E}\left[\nabla f\left(X_t\right)\tilde{X}_t\right] \approx \nabla_\theta\left(\mathbf{E}_{Y\sim\pi_{\theta_t}}[f(X)-\beta]\right)$ since θ_t is changing very slowly, this is because t becomes large due to $\lim_{t\to\infty}\alpha_t = 0$. We need to mind that even though the expectations above is fulfilled, in order to make the product of $\mathbf{E}\left[f\left(\bar{X}_t\right)-\beta\right]$ and $\mathbf{E}\left[\nabla f\left(X_t\right)\tilde{X}_t\right]$ be the estimate of $\nabla_\theta J(\theta_t)$, it is important to highlight that for random variables X and Y, it is not always true that $\mathbf{E}[XY] = \mathbf{E}X\cdot\mathbf{E}Y$ unless X and Y are independent. This is the reason why the process $\bar{X}_t$ is needed. Since $\bar{X}_t$ and X_t are driven by independent Brownian motions, we expect that $\mathbf{E}\left[2\left(f\left(\bar{X}_t\right)-\beta\right)\nabla f\left(X_t\right)\tilde{X}_t\right] \approx \nabla_\theta J(\theta_t)$ for large t due to $\bar{X}_t$ and $\left(X_t,\tilde{X}_t\right)$ becoming asymptotically independent since θ_t will be changing very slowly for large t.

Thus, we expect that for large t, the stochastic sample $2\left(f\left(\bar{X}_t\right)-\beta\right)\nabla f\left(X_t\right)\tilde{X}_t$ will provide an asymptotically unbiased estimate for the direction of steepest descent $\nabla_\theta J\left(\theta_t\right)$ and $\|\nabla_\theta J\left(\theta_t\right)\|$ will converge to zero as $t\to\infty$. The above analysis is just some intuitive inspirations, what we need is to make sure this works. Hence rigorous mathematical proofs in the original paper is needed.

2.3 High Light of The Mathematical Proof

1. Setup and Assumptions

The theorem's proof relies on Assumption 2.1, which ensures certain conditions are met for the stochastic processes involved.

2. Application of Itô's Formula

The proof utilizes Itô's formula applied to a specific function $u_2(t,x,\tilde{x},\bar{x},\theta)=\alpha_t v_2(x,\tilde{x},\bar{x},\theta)$. Here, v_2 is the solution to a related partial differential equation (PDE) given in Lemma 3.6 of the original paper. Itô's formula is used to handle the stochastic process $(X_t,\tilde{X}_t,\bar{X}_t,\theta_t)$.

3. Derivation of Expressions

By applying Itô's formula, the following expression is derived for any i in the set $\{1,2,\ldots,\ell\}$:

$$\begin{aligned}
u_2^i(\sigma,X_\sigma,\tilde{X}_\sigma,\bar{X}_\sigma,\theta_\sigma)-u_2^i(\tau,X_\tau,\tilde{X}_\tau,\bar{X}_\tau,\theta_\tau)&\\
&=\int_\tau^\sigma \partial_s u_2^i(s,X_s,\tilde{X}_s,\bar{X}_s,\theta_s)\,ds\\
&\quad+\int_\tau^\sigma L_{\theta_s}^x u_2^i(s,X_s,\tilde{X}_s,\bar{X}_s,\theta_s)\,ds\\
&\quad+\int_\tau^\sigma L_{\theta_s}^{\bar{x}} u_2^i(s,X_s,\tilde{X}_s,\bar{X}_s,\theta_s)\,ds\\
&\quad+\int_\tau^\sigma \nabla_\theta u_2^i(s,X_s,\tilde{X}_s,\bar{X}_s,\theta_s)\,d\theta_s\\
&\quad+\int_\tau^\sigma \nabla_x u_2^i(s,X_s,\tilde{X}_s,\bar{X}_s,\theta_s)\,dW_s\\
&\quad+\int_\tau^\sigma \nabla_{\bar{x}} u_2^i(s,X_s,\tilde{X}_s,\bar{X}_s,\theta_s)\,d\bar{W}_s.
\end{aligned}$$

4. Convergence Analysis

The proof requires analyzing the fluctuations of parameter evolution around the direction of steepest descent. It involves proving bounds for the solutions of a new class of Poisson partial differential equations (PDEs). These bounds are then used to analyze the parameter fluctuations in the algorithm, ensuring convergence.

Conclusion

The combination of Itô's calculus, bounds on PDE solutions, and convergence analysis leads to the rigorous proof of Theorem 2.2 of the original paper, demonstrating the effectiveness of the continuous-time SGD method for optimizing over the stationary distribution of stochastic differential equations.

2.4 Theoretical Limitation

There are several theoretical limitations of the proposed algorithm:

1. Dependence on Hyper-parameters

The performance of the algorithm is sensitive to the selection of hyper-parameters such as the learning rate and mini-batch size. The learning rate must decay as $t \to \infty$, but it should not decrease too rapidly, and the initial magnitude should be sufficiently large to ensure convergence.

2. Stochastic Estimates and Noise

The gradient descent direction's estimation is stochastic and depends on the mini-batch size. A larger mini-batch size reduces the noise in the estimation but may increase computational cost. The algorithm's effectiveness in reducing this noise is crucial for its performance.

3. Convergence to Global Minima

For problems with a unique global minimizer, the algorithm converges to the optimal solution if the learning rate is chosen correctly. However, for problems with multiple global minimizers, the algorithm may converge to any one of the global minimizers, depending on the initial conditions and the learning rate schedule.

4. High-dimensional Problems

While the algorithm is designed to handle high-dimensional stochastic optimal control problems, it may face computational challenges in very high dimensions. Traditional numerical methods struggle with high-dimensional PDEs, and although the proposed algorithm aims to address this, its scalability and efficiency in extremely high dimensions remain a concern.

5. Path-dependent and Nonlinear SDEs

The algorithm's implementation and performance vary for different types of SDE models, including path-dependent and nonlinear SDEs. The theoretical guarantees provided for linear SDEs may not fully extend to these more complex models.

6. Ergodicity and Long Time Horizons

The algorithm assumes ergodicity of the data and stochastic process for long time horizons. This assumption may not hold in all practical scenarios, potentially affecting the algorithm's applicability and performance.

7. Application-specific Challenges

The algorithm is tested on applications in mathematical finance, such as parameter estimation for SDE models and stochastic optimal control. Each application presents unique challenges, and the algorithm's performance may vary based on the specific characteristics and requirements of each application.

Conclusions

These theoretical limitations highlight the importance of careful parameter tuning, the potential impact of noise in stochastic estimates, and the need for further research to extend the algorithm's applicability to more complex and higher-dimensional problems.

Chapter 3

Implementation

3.1 Simple Cases

3.1.1 One-Dimensional Ornstein-Uhlenbeck Process

To Start With

$$dX_t^\theta = \left(\theta - X_t^\theta\right) dt + dW_t \tag{3.1}$$

In this case, the main algorithm is used to learn the minimizer for $J(\theta) = (\mathbf{E}_{Y\sim\pi_\theta} Y - 2)^2$. Note that in this case we have the closed-form solution $\pi_\theta \sim N\left(\theta, \frac{1}{2}\right)$ and thus the global minimizer is $\theta^* = 2$.

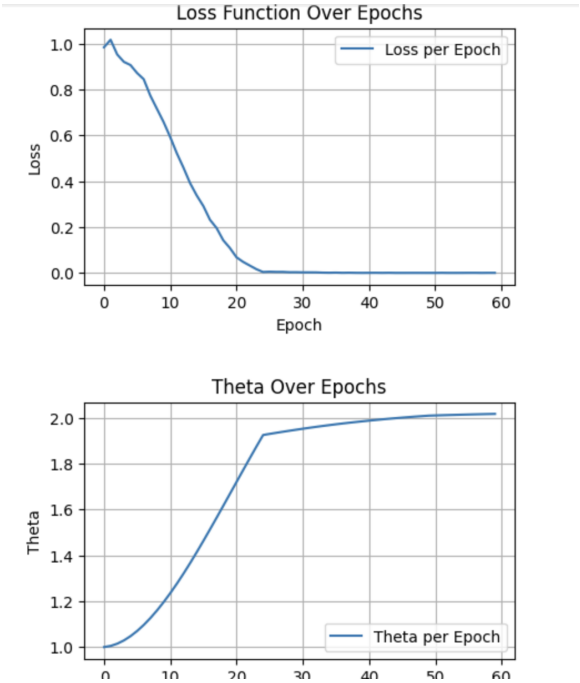


Figure 3.1: $J(\theta) = (\mathbf{E}_{Y\sim\pi_\theta} Y - 2)^2$

The result of the implementation is above. With total epochs equal to 60, final theta is $\theta = 2.016258716583252$ and final loss $J(\theta) = 0.0002$, which is consistent with the theory. To show the algorithm indeed converge in the long run. We adjust the total number of epochs to be 100. The below results shows that the algorithm indeed converge.

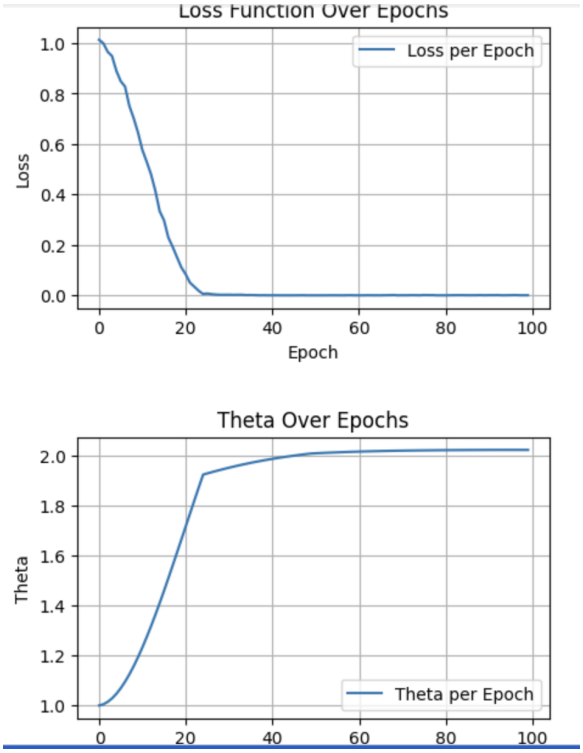


Figure 3.2: $\#epochs = 100$

Similarly, we also use the algorithm to learn the minimizer for $J(\theta) = (\mathbf{E}_{Y\sim\pi_\theta} Y^2 - 2)^2$. In this case, the two global minimizers are $\theta^* = \pm\sqrt{1.5}$.

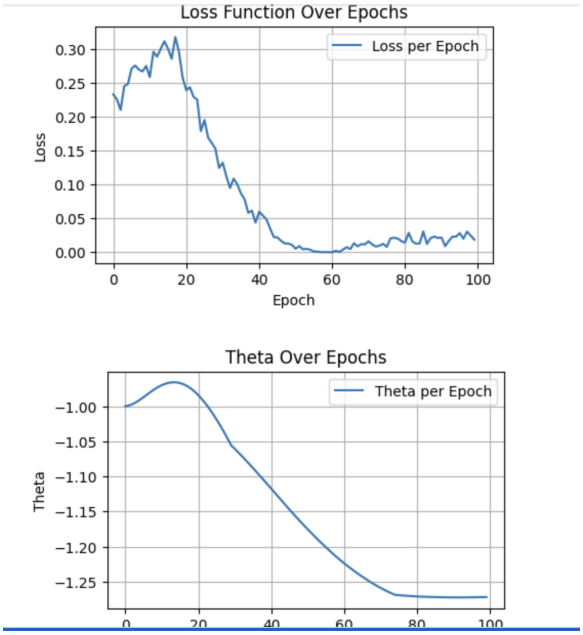


Figure 3.3: $J(\theta) = \left(\mathbf{E}_{Y\sim\pi_\theta} Y^2 - 2\right)^2$

Here the parameter trained by the online algorithm converges to a global minimizer. The global minimizer which the algorithm converges to depends on the initial value of θ_0.

More General OU Processes

$$dX_t^\theta = \left(\theta^1 - \theta^2 X_t^\theta\right) dt + dW_t \tag{3.2}$$

In this case, $\theta = (\theta^1, \theta^2)$.

The objective function $J(\theta) = \left(\mathbf{E}_{Y\sim\pi_\theta} Y^2 - 2\right)^2$. Algorithm with batch size N is as follows:

$$\begin{aligned}
d\theta_t^1 &= -4\alpha_t \left(\frac{1}{N}\sum_{i=1}^N \left(\bar{X}_t^{(i)}\right)^2 - 2\right) \cdot \left(\frac{1}{N}\sum_{i=1}^N X_t^{(i)} \tilde{X}_t^{1,(i)}\right) dt \\
d\theta_t^2 &= -4\alpha_t \left(\frac{1}{N}\sum_{i=1}^N \left(\bar{X}_t^{(i)}\right)^2 - 2\right) \cdot \left(\frac{1}{N}\sum_{i=1}^N X_t^{(i)} \tilde{X}_t^{2,(i)}\right) dt \\
dX_t^{(i)} &= \left(\theta_t^1 - \theta_t^2 X_t^{(i)}\right) dt + dW_t^i \\
d\tilde{X}_t^{1,(i)} &= \left(1 - \theta_t^2 \tilde{X}_t^{1,(i)}\right) dt \\
d\tilde{X}_t^{2,(i)} &= \left(-X_t^{(i)} - \theta_t^2 \tilde{X}_t^{2,(i)}\right) dt \\
d\bar{X}_t^{(i)} &= \left(\theta_t^1 - \theta_t^2 \bar{X}_t^{(i)}\right) dt + d\bar{W}_t^i
\end{aligned} \tag{3.3}$$

for $i = 1, 2, \cdots, N$.

In practice, we choose the batch size $N = 10000$, so the training can be more stable. The below figure shows the result of this two parameter case.

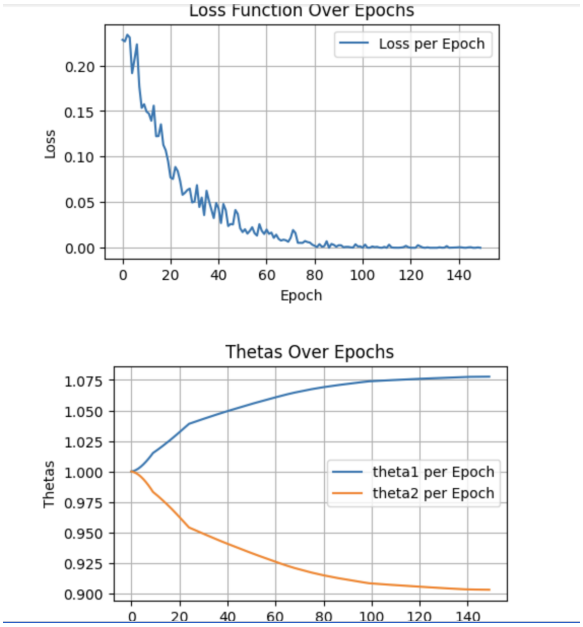


Figure 3.4: Two Parameter case

Final theta1 is tensor(1.0776), final theta2 equals to tensor(0.9033), and final $J(\theta)$ is $4.1243e-06$ which is small.

NOTE: Computation of $E_{\pi_\theta}[Y^2]$

To compute $E_{\pi_\theta}[Y^2]$, it is obvious we have to natural methods. The first method is as follows.

```
import torch
import matplotlib.pyplot as plt


# Simulate the OU process of e.g. 4.1 two parameter case
# X_simu = torch.randn(N, requires_grad=False)
batch_size = 10000
X_simu = torch.ones(batch_size)
dt_simu = 0.1
number_of_step = 10000
# theta1 = torch.randn(1)
```

```
# theta2 = torch.randn(1)
theta1 = 1.075
theta2 = 0.9

for i in range(1, number_of_step):
    dW = torch.randn(batch_size) * torch.sqrt(torch.tensor(dt_simu))
    X_simu = X_simu + (theta1 - theta2 * X_simu) * dt_simu + dW

print(torch.mean(X_simu**2))
```

We have another method. This is using the ergodic theorem.

$$\lim_{t\to\infty} \frac{1}{t} \int_0^t f\left(X_s^\theta\right) ds = \mathbf{E}_{Y\sim\pi_\theta} f(Y) \quad \text{a.s. ,}$$

and the implementation is below.

```
import torch

# Simulate the OU process
batch_size = 10000
X_simu = torch.ones(batch_size)
dt_simu = 0.01
number_of_step = 100000
# theta1 = torch.randn(1)
# theta2 = torch.randn(1)
theta1 = 1.075
theta2 = 0.9
estimator = 0

for i in range(1, number_of_step):
    dW = torch.randn(batch_size) * torch.sqrt(torch.tensor(dt_simu))
    delta_X_simu = (theta1 - theta2 * X_simu) * dt_simu + dW
    X_simu = X_simu + delta_X_simu
    estimator += (1/(number_of_step * dt_simu)) * X_simu**2 * dt_simu

print(torch.mean(estimator))
```

We conclude that both methods have the same result. The only thing to consider is the so called cost of implementation. After consideration we choose to use the first method since it is cheaper.

3.1.2 One-Dimensional Nonlinear Process

We now use the online algorithm to optimize over the stationary distribution of a one-dimensional nonlinear process

$$dX_t^\theta = \left(\theta - X_t^\theta - \left(X_t^\theta\right)^3\right) dt + dW_t \tag{3.4}$$

Note that the format of the drift term is not the same as what is required in the main theorem. But we can still use the main algorithm to learn the minimizer of $J(\theta) = \left(\mathbf{E}_{Y\sim\pi_\theta} Y^2 - 2\right)^2$. The following mini-batch algorithm is used:

$$\begin{aligned}
d\theta_t &= -4\alpha_t \left(\frac{1}{N}\sum_{i=1}^N \left(\bar{X}_t^{(i)}\right)^2 - 2\right) \cdot \left(\frac{1}{N}\sum_{i=1}^N X_t^{(i)} \tilde{X}_t^{(i)}\right) dt \\
dX_t^{(i)} &= \left(\theta_t - X_t^{(i)} - \left(X_t^{(i)}\right)^3\right) dt + dW_t^{(i)} \\
d\tilde{X}_t^{(i)} &= \left(1 - \tilde{X}_t^{(i)} - 3\left(X_t^{(i)}\right)^2 \tilde{X}_t^{(i)}\right) dt \\
d\bar{X}_t^{(i)} &= \left(\theta_t - \bar{X}_t^{(i)} - \left(\bar{X}_t^{(i)}\right)^3\right) dt + d\bar{W}_t^{(i)}
\end{aligned}$$

for $i = 1, 2, \cdots, N$.

Figure 3.5 below shows the convergence of the parameter θ_t. And it shows the loss function decays to zero, i.e. the global minimum, very quickly. The final theta is 4.351974964141846, and final J is tensor(0.0014, device='cuda:0'), which is in a acceptable range. Hence we see the interesting thing: Even though the requirement is not totally fulfilled in some problems, it is possible that our algorithm can be adapted to those problem.

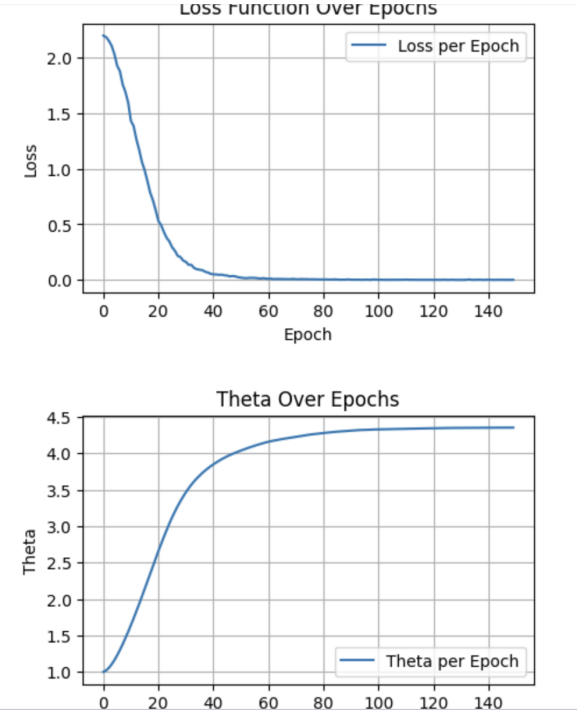


Figure 3.5: Non linear case

3.1.3 Optimizing over the Drift and Volatility Coefficients

Linear Case

We now optimize over the drift and volatility functions of the process

$$dX_t^\theta = \left(\mu - X_t^\theta\right) dt + \sigma dW_t \tag{4.11}$$

with parameters $\theta = (\mu, \sigma)$.

Since the format perfectly satisfies the requirement, our online algorithm can be used to learn the minimizer of $J(\theta) = (\mathbf{E}_{Y\sim\pi_\theta} Y^2 - 2)^2$. The variant is written below:

$$
\begin{aligned}
d\mu_t &= -4\alpha_t \left(\frac{1}{N} \sum_{i=1}^{N} \left(\bar{X}_t^{(i)} \right)^2 - 2 \right) \cdot \left(\frac{1}{N} \sum_{i=1}^{N} X_t^{(i)} \tilde{X}_t^{1,(i)} \right) dt \\
d\sigma_t &= -4\alpha_t \left(\frac{1}{N} \sum_{i=1}^{N} \left(\bar{X}_t^{(i)} \right)^2 - 2 \right) \cdot \left(\frac{1}{N} \sum_{i=1}^{N} X_t^{(i)} \tilde{X}_t^{2,(i)} \right) dt \\
dX_t^i &= \left(\mu_t - X_t^{(i)} \right) dt + \sigma_t dW_t^{(i)} \\
d\tilde{X}_t^{1,(i)} &= \left(1 - \tilde{X}_t^{1,(i)} \right) dt \\
d\tilde{X}_t^{2,(i)} &= -\tilde{X}_t^{2,(i)} dt + dW_t^{(i)} \\
d\bar{X}_t^{(i)} &= \left(\mu_t - \bar{X}_t^{(i)} \right) dt + \sigma_t d\bar{W}_t^{(i)}
\end{aligned}
$$

for $i = 1, 2, \cdots, N$.

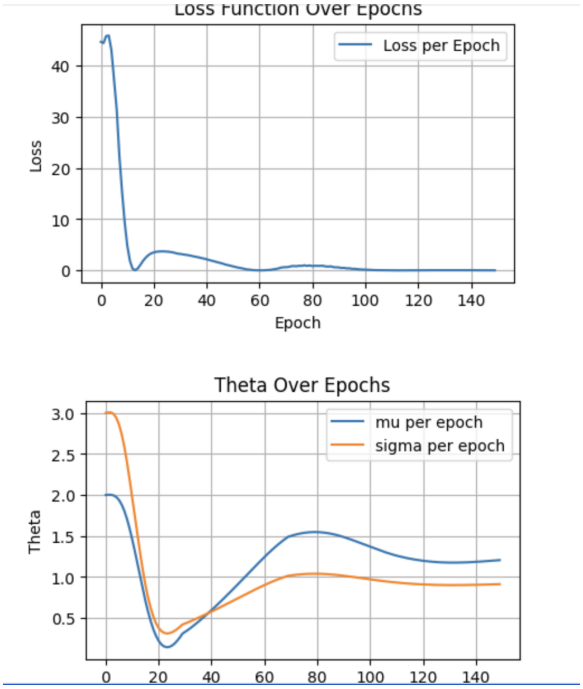


Figure 3.6: Drift and Volatility Linear

In Figure 3.6, the trained parameters μ_t, σ_t converge and the objective function $J(\theta_t) \to 0$ very quickly. And we have results Final mu: 1.2057600021362305; Final sigma: 0.9127710461616516; Final J: 0.011372823268175125.

Notice that there weird things happened. We know $\mu = 1.0$ and $\sigma = 1.5$ is also a set of parameters which can make loss $J(\theta)$ small. This is shown by the output below:

Figure 3.7: Drift and Volatility Linear

Hence in this special case there are multiple local minimums.

Non Linear Case

We also implement the online algorithm for the nonlinear process to see the ability to adapt.

$$dX_t^\theta = \left(\mu - \left(X_t^\theta\right)^3\right) dt + \sigma X_t^\theta dW_t \tag{3.5}$$

where $\theta = (\mu, \sigma)$ are the parameters and the objective function is $J(\theta) = \left(\mathbf{E}_{Y\sim\pi_\theta} Y^2 - 10\right)^2$. The mini-batch algorithm 3.3 now becomes:

$$\begin{aligned}
d\mu_t &= -4\alpha_t \left(\frac{1}{N}\sum_{i=1}^N \left(\bar{X}_t^{(i)}\right)^2 - 2\right) \cdot \left(\frac{1}{N}\sum_{i=1}^N X_t^{(i)} \tilde{X}_t^{1,(i)}\right) dt \\
d\sigma_t &= -4\alpha_t \left(\frac{1}{N}\sum_{i=1}^N \left(\bar{X}_t^{(i)}\right)^2 - 2\right) \cdot \left(\frac{1}{N}\sum_{i=1}^N X_t^{(i)} \tilde{X}_t^{2,(i)}\right) dt \\
dX_t^i &= \left(\mu_t - \left(X_t^{(i)}\right)^3\right) dt + \sigma_t X_t^{(i)} dW_t^{(i)} \\
d\tilde{X}_t^{1,(i)} &= \left(1 - 3\left(X_t^{(i)}\right)^2 \tilde{X}_t^{1,(i)}\right) dt + \sigma_t \tilde{X}_t^{1,(i)} dW_t^{(i)} \\
d\tilde{X}_t^{2,(i)} &= -3\left(X_t^{(i)}\right)^2 \tilde{X}_t^{2,(i)} dt + \left(X_t^{(i)} + \sigma_t \tilde{X}_t^{2,(i)}\right) dW_t^{(i)} \\
d\bar{X}_t^{(i)} &= \left(\mu_t - \left(\bar{X}_t^{(i)}\right)^3\right) dt + \sigma_t \bar{X}_t^{(i)} d\bar{W}_t^{(i)}
\end{aligned}$$

for $i = 1, 2, \cdots, N$.

By calibration (use the final parameters to simulate process and check the loss) we can find the original paper has some typos in this part. And the results I get are described further below.

In Figure 3.8, the trained parameters μ_t, σ_t converge and it also shows that the objective function $J(\theta_t) \to 0$ very quickly. Hence we proved that the algorithm can adapt this case.

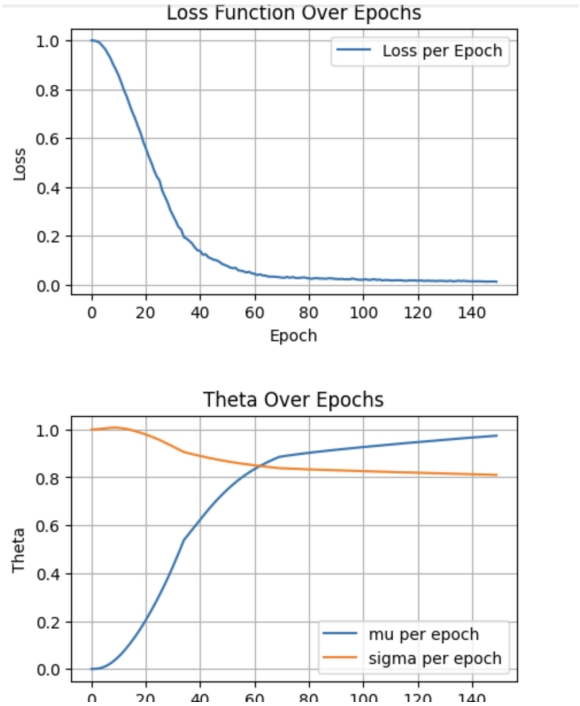


Figure 3.8: Drift and Volatility Non Linear

3.1.4 Multi-Dimensional Ornstein-Uhlenbeck Process

Independent OU Process

We next consider a simple multi-dimensional Ornstein-Uhlenbeck process, it consists of m independent copies of an OU process. For the parameter $\theta = (\theta^1, \theta^2) \in R^{2m}$, let the m-dimensional Ornstein-Uhlenbeck process be

$$dX_t^\theta = \left(\theta^1 - \theta^2 \odot X_t^\theta\right) dt + dW_t \tag{3.6}$$

where $X_t^\theta \in R^m, W_t \in R^m$, and $\odot$ is an element-wise product.

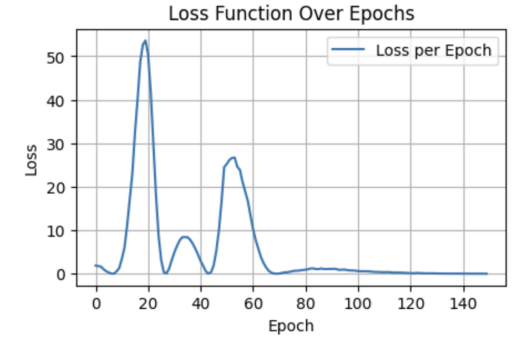
The objective function is

$$J(\theta) := \left(\sum_{k=1}^{m} \mathbf{E}_{Y \sim \pi_\theta} |Y_k|^2 - 2m \right)^2 \tag{3.7}$$

The online algorithm becomes:

$$
\begin{aligned}
d\theta_t^1 &= -4\alpha_t \left(\left| \bar{X}_t \right|^2 - 2 \right) X_t \odot \tilde{X}_t^1 dt \\
d\theta_t^2 &= -4\alpha_t \left(\left| \bar{X}_t \right|^2 - 2 \right) X_t \odot \tilde{X}_t^2 dt \\
dX_t &= \left(\theta_t^1 - \theta_t^2 \odot X_t \right) dt + dW_t^i \\
d\tilde{X}_t^1 &= \left(1 - \theta_t^2 \odot \tilde{X}_t^1 \right) dt \\
d\tilde{X}_t^2 &= \left(-X_t - \theta_t^2 \odot \tilde{X}_t^2 \right) dt \\
d\bar{X}_t &= \left(\theta_t^1 - \theta_t^2 \odot \bar{X}_t \right) dt + d\bar{W}_t^i
\end{aligned}
$$

We implement the algorithm for $m = 3$ and $m = 10$. In Figures 3.9 and 3.10 , the objective functions $J(\theta_t) \to 0$ as t becomes large.




```
Final theta1: tensor([1.1053, 1.1097, 1.1102], device='cuda:0')
Final theta2: tensor([0.9271, 0.9198, 0.9248], device='cuda:0')
Final J: tensor(3.2166e-05, device='cuda:0')
```

Figure 3.9: Independent m=3

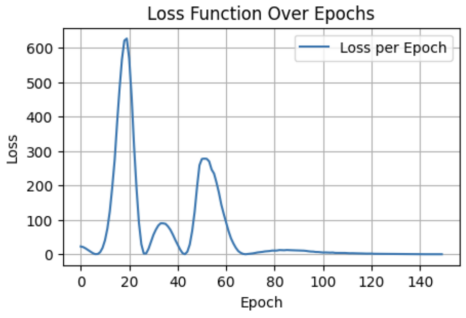


Figure 3.10: Independent m=10

It is worth mentioning that the final thetas values and the near-zero final loss suggest effective convergence. And despite the higher initial values, the loss converges

effectively, reaching near-zero levels after epoch 80, showcasing the robustness of the optimization process even with increased dimensionality. Overall, the gradient descent algorithm is working effectively, successfully minimizing the loss and converging to optimal parameter values despite initial instabilities.

Correlated OU Process

For the parameters $\theta = (\mu, \sigma)$ with $\mu \in \mathbb{R}^m, \sigma \in \mathbb{R}^{m\times m}$, let the m-dimensional process X_t^θ satisfy

$$dX_t^\theta = \left(\mu - X_t^\theta\right) dt + \sigma dW_t$$

where $W_t \in \mathbb{R}^m$.

Let $X_t^{\theta,i}$ denote the i-th element of X_t^θ and define $\tilde{X}_t^\mu$ and $\tilde{X}_t^\sigma$ as the Jacobian matrices of X_t^θ with respect to μ and σ :

$$\begin{aligned}
\tilde{X}_t^\mu &= \nabla_\mu X_t^\theta \in \mathbb{R}^{m\times m}, \quad \tilde{X}_t^{\mu,i} = \nabla_\mu X_t^{\theta,i} \in \mathbb{R}^m \\
\tilde{X}_t^\sigma &= \nabla_\sigma X_t^\theta \in \mathbb{R}^{m\times m\times m}, \quad \tilde{X}_t^{\sigma,i} = \nabla_\sigma X_t^{\theta,i} \in \mathbb{R}^{m\times m}
\end{aligned}$$

for $i \in \{1, 2, \cdots, m\}$.

$$dX_t^{\theta,i} = \left(\mu_i - X_t^{\theta,i}\right) dt + \sum_j \sigma_{i,j} dW_t^j$$

Now in this case the algorithm 2.3 becomes

$$\begin{aligned}
d\mu_t &= -4\alpha_t \left(\left|\bar{X}_t\right|^2 - 2m\right) \left(\sum_{k=1}^m X_t^k \tilde{X}_t^{\mu,k}\right) dt \\
d\lambda_t &= -4\alpha_t \left(\left|\bar{X}_t\right|^2 - 2m\right) \left(\sum_{k=1}^m X_t^k \tilde{X}_t^{\lambda,k}\right) dt \\
dX_t &= \left(\mu_t - X_t\right) dt + \sigma_t dW_t \\
d\bar{X}_t &= \left(\mu_t - \bar{X}_t\right) dt + \sigma_t d\bar{W}_t \\
d\tilde{X}_t^\mu &= \left(I_m - \tilde{X}_t^\mu\right) dt \\
d\tilde{X}_t^{\sigma,i} &= -\tilde{X}_t^{\sigma,i} dt + D_i\left(dW_t\right), \quad i \in \{1, \cdots, m\}
\end{aligned}$$

where I_m is the $m \times m$ identity matrix and where $D_i\left(dW_t\right)$ is a $m \times m$ matrix with all elements equal to 0 except i-th column being dW_t. We examine the algorithm's performance for dimensions $m = 3, 10$.

In Figures 3.11, the objective function $J(\theta_t)$ does not converge to zero, which indicates there may be issues in the algorithm or the choice of hyper-parameters.

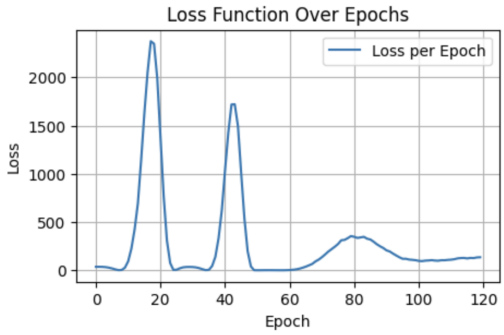
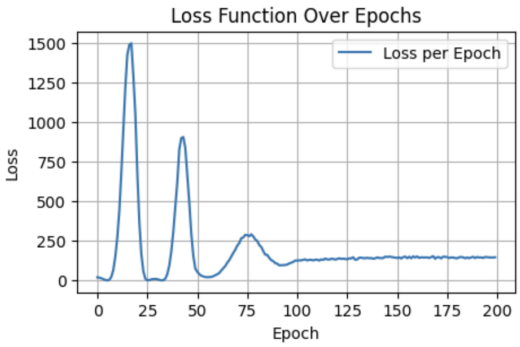


Figure 3.11: Correlated m=3

While the algorithm shows promising signs of effectiveness, addressing the identified instabilities and refining the training process will be essential to achieving more reliable and optimal performance. Hence we refined the code by tuning the learning rate. Unfortunately the loss did not converge to 0 even after the refine.




```
Final mu: tensor([0.7447, 0.7694, 0.7118], device='cuda:0')
Final sigma: tensor([[3.2132, 0.0000, 0.0000],
        [0.0000, 3.2349, 0.0000],
        [0.0000, 0.0000, 3.2303]], device='cuda:0')
Final J: tensor(146.1420, device='cuda:0')
```

Figure 3.12: Correlated m=3 refined

And for higher dimensions, $m = 10$, the algorithm loss dose not converge to 0.

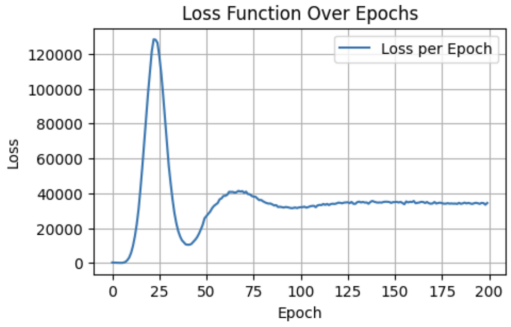


Figure 3.13: Correlated m=10 refined

Notice that our mathematical proof only ensures that $\lim_{t\to\infty} |\nabla_\theta J(\theta_t)| \overset{\text{a.s.}}{=} 0$, while it dose not guaranty the loss to converge to 0.

3.1.5 Path-dependent SDE

We consider the path-dependent SDE

$$dX_t^\theta = \left(\theta - X_t^\theta - \frac{1}{t}\int_0^t X_s^\theta ds\right) dt + dW_t \tag{3.8}$$

where $X_t^\theta, W_t \in \mathbb{R}$.

Although path-dependent SDEs are not directly addressed by this article's convergence theory, the numerical example in this subsection suggests that the online forward propagation algorithm can also be applied to path-dependent stochastic processes.

For this numerical example, the objective function is $J(\theta) = (\mathbf{E}_{Y\sim\pi_\theta} Y - 2)^2$. The SDE 3.8 does not fit the problem described in 2.1. However, our algorithm still can find the global optimum. Which shows the ability of adapt. Now the online algorithm 2.3 is:

$$\begin{aligned}
d\theta_t &= -4\alpha_t \left(\bar{X}_t - 2\right) \tilde{X}_t dt \\
dX_t &= \left(\theta_t - X_t - \frac{1}{t}\int_0^t X_s ds\right) dt + dW_t \\
d\tilde{X}_t &= \left(1 - \tilde{X}_t - \frac{1}{t}\int_0^t \tilde{X}_s ds\right) dt \\
d\bar{X}_t &= \left(\theta_t - \bar{X}_t - \frac{1}{t}\int_0^t \bar{X}_s ds\right) dt + d\bar{W}_t
\end{aligned}$$

In Figure 3.14 , the trained parameter converges. The objective function $J(\theta_t)$ is approximated using a time-average. And the objective function $J(\theta_t)$ converges to 0 very quickly.

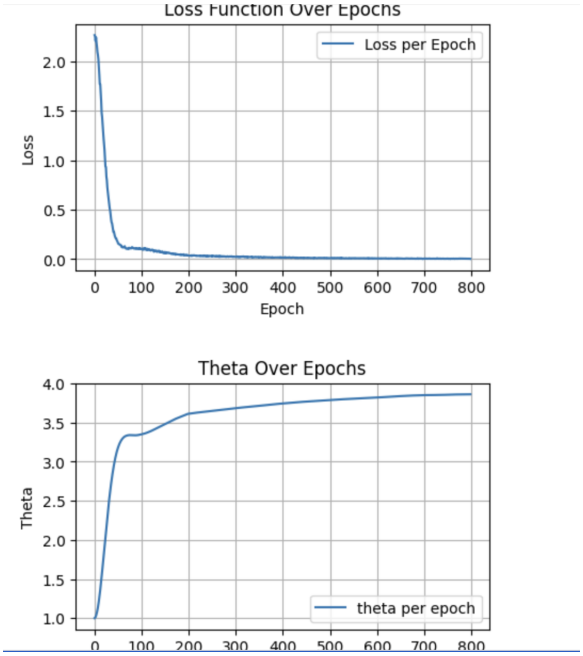


Figure 3.14: Path Dependent

3.2 Linear–quadratic Regulator

The online optimization algorithm can be employed to solve stochastic optimal control problems, including high-dimensional scenarios where traditional numerical methods, such as solving the Hamilton-Jacobi-Bellman (HJB) equation with finite difference methods, are computationally expensive or impractical. As a numerical example, we consider the classic Linear Quadratic Regulator (LQR) problem, which has numerous financial applications such as optimal execution[1]. Let $\{X_t\}_{t\geq 0}$ be the state process governed by the stochastic differential equation (SDE):

$$dX_t = (AX_t + BU_t)\,dt + \sigma\,dW_t \tag{3.9}$$

where $X_0 = x_0$, $X_t \in \mathbb{R}^n$, and matrices $A, \sigma \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$.

Here, $\{W_t\}_{t\geq 0}$ is an $\mathbb{R}^n$-valued standard Wiener process, and $\{U_t\}_{t\geq 0} \in \mathbb{R}^m$ represents the control. The objective is to learn a control process U_t to minimize the ergodic cost functional for system below:

$$J(U) = \lim_{T\to\infty} \frac{1}{T} \int_0^T \left(X_t^{\mathrm{T}} Q X_t + U_t^{\mathrm{T}} R U_t\right)\, dt \tag{3.10}$$

where Q and R are positive definite matrices. It is well-known that the optimal control is given by: $U = -R^{-1}B^\top KX$, where K is the unique solution of the following algebraic Riccati equation (ARE):

$$A^\top K + KA - KBR^{-1}B^\top K + Q = 0 \tag{3.11}$$

In order to evaluate the accuracy of our algorithm in solving stochastic optimal control problems, we numerically implement it for several high-dimensional stochastic LQR problems. The LQR problem is selected because it has a closed-form solution, even in high dimensions, allowing us to assess the accuracy of our algorithm. We present a series of numerical examples where the online optimization algorithm learns parametric controls for various LQR problems. The parametric control is either a linear function or a neural network.

3.2.1 One-dimensional Linear Control

As a first step, we implement the online optimization algorithm for the one-dimensional case with a linear control function. For simplicity, we assume that $A = -1, B = \sigma = Q = R = 1$ for 3.9, then we get:

$$\begin{aligned} dX_t^\theta &= \left(-X_t^\theta + \theta X_t^\theta\right) dt + dW_t \\ J(\theta) &= \lim_{T\to\infty} \frac{1}{T} \int_0^T \left(1+\theta^2\right) \left(X_t^\theta\right)^2 dt \end{aligned}$$

The system becomes

$$\begin{aligned} d\theta_t &= -\alpha_t \left[\frac{1}{N}\sum_{i=1}^N \left(2\theta_t \left(X_t^{(i)}\right)^2 + 2\left(1+\theta_t^2\right) X_t^{(i)} \tilde{X}_t^{(i)}\right)\right] dt \\ dX_t^{(i)} &= \left(\theta_t - 1\right) X_t^{(i)} dt + dW_t^{(i)} \\ d\tilde{X}_t^{(i)} &= \left(X_t^{(i)} + \left(\theta_t - 1\right) \tilde{X}_t^{(i)}\right) dt \end{aligned}$$

with $i = 1, 2, \cdots, N$.

Solving the ARE yields the optimal control $\theta^* = -0.41421$. Here we can directly use loss to measure the performance. Figure 3.15 below shows that the parameter θ_t trained with the online optimization algorithm converges to θ^*.

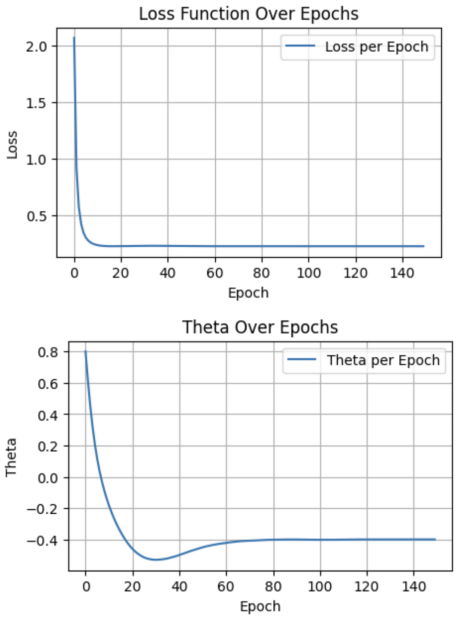


Figure 3.15: LQR 1-D

3.2.2 Multi-dimensional Linear Control

We next solve a multi-dimensional LQR problem with a linear control function. For simplicity, we assume that $m = n, A = -I_n, B = \sigma = I_n$ in 3.9 where I_n is n dimensional identity matrix. That is,

$$dX_t^\theta = \left(-X_t^\theta + \theta X_t^\theta\right) dt + dW_t \tag{3.12}$$

$$J(\theta) = \lim_{T\to\infty} \frac{1}{T} \int_0^T \left(X_t^\theta\right)^\top \left(Q + \theta^\top R\theta\right) X_t^\theta dt \tag{3.13}$$

where $\theta \in \mathbb{R}^{n\times n}$.

Let $X_t^{\theta,i}$ denote the i-th element of X_t^θ and define

$$\tilde{X}_t^\theta = \nabla_\theta X_t^\theta, \quad \tilde{X}_t^{\theta,i} = \nabla_\theta X_t^{\theta,i}, \forall i \in \{1, 2, \cdots, n\}$$

$\tilde{X}_t^\theta$ has dimensions $n \times n \times n$ and $\tilde{X}_t^{\theta,i}$ has dimensions $n \times n$. Note that when we are training over a mini-batch of size N, $\tilde{X}_t^\theta$ has dimensions $N \times n \times n \times n$.

We first discuss the methods necessary for the computationally efficient simulation of the gradient $\nabla_\theta X_t^\theta$. The state process from 3.12 satisfies

$$dX_t^{\theta,i} = \left(-X_t^{\theta,i} + \sum_{j=1}^{n} \theta_{i,j} X_t^{\theta,j}\right) dt + dW_t^i$$

and therefore

$$d\tilde{X}_t^{\theta,i} = \left(-\tilde{X}_t^{\theta,i} + \sum_{j=1}^{n} \theta_{i,j} \tilde{X}_t^{\theta,j} + D_i\left(X_t^{\theta}\right) \right) dt$$

where $D_i\left(X_t^{\theta}\right)$ is an $n \times n$ matrix whose elements are all zeros except for the i-th row, which has values X_t^{θ}. The gradient of the objective function in 3.13 is:

$$\begin{aligned}
\nabla_\theta \left[\left(X_t^{\theta}\right)^\top \left(Q + \theta^\top R \theta\right) X_t^{\theta} \right] &= \sum_{i,j} \nabla_\theta \left(\delta_{i,j} + \sum_{k=1}^{n} \theta_{k,i} \theta_{k,j} \right) X_t^{\theta,i} X_t^{\theta,j} \\
&\quad + 2 \sum_{i,j} \nabla_\theta X_t^{\theta,i} \left(q_{i,j} + \theta_{:,i}^\top R \theta_{:,j} \right) X_t^{\theta,j} \\
&= \sum_{i,j} \left((R\theta)_{:,j} 1_{\{i=n\}} + (R\theta)_{:,i} 1_{\{j=n\}} \right) X_t^{\theta,i} X_t^{\theta,j} \\
&\quad + 2 \sum_{i,j} \tilde{X}_t^{\theta,i} \left(q_{i,j} + \theta_{:,i}^\top R \theta_{:,j} \right) X_t^{\theta,j}
\end{aligned}$$

In multi-dimensional case, the performance can not be measured by the loss function $J(\theta)$ any more, since the numerical computation of loss J is unstable in most cases. This may because errors show up while computing the integral $\int_0^T \left(X_t^{\theta}\right)^\top \left(Q + \theta^\top R \theta\right) X_t^{\theta} dt$ using numerical discretization. Without mathematical proof, I reckon "the Riemann Sum" in this case is only in the realm of stability when the time interval dt is very small.

Hence we use a different way. The error metrics are defined as:

$$\begin{aligned}
\text{Ave Error} &= \frac{\sum_{i,j=1}^{n} \left| \theta_{t,i,j} - \theta_{i,j}^* \right|}{\sum_{i,j=1}^{n} \left| \theta_{i,j}^* \right|} \\
\text{Max Error} &= \frac{\max_{i,j \in \{1,2,\cdots,n\}} \left| \theta_{t,i,j} - \theta_{i,j}^* \right|}{\frac{1}{n^2} \sum_{i,j=1}^{n} \left| \theta_{i,j}^* \right|} \\
\text{Cost Error} &= \frac{\left| J\left(\theta_T\right) - J\left(\theta^*\right) \right|}{\left| J\left(\theta^*\right) \right|}
\end{aligned}$$

where θ^* is the optimal control and θ_t is the parameter during training. $J\left(\theta_T\right)$ and $J\left(\theta^*\right)$ denote the objective function $J(\theta)$ in 3.13 with the parameters θ_T and θ^*, respectively.

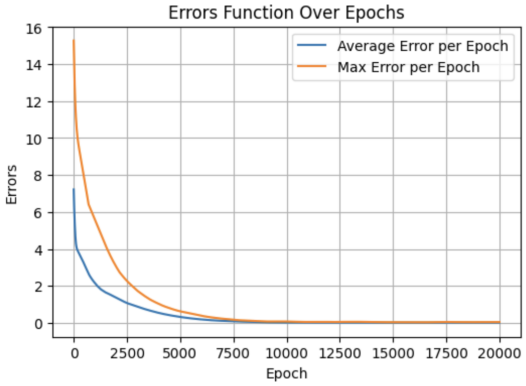


Figure 3.16: LQR Multi-D n=5

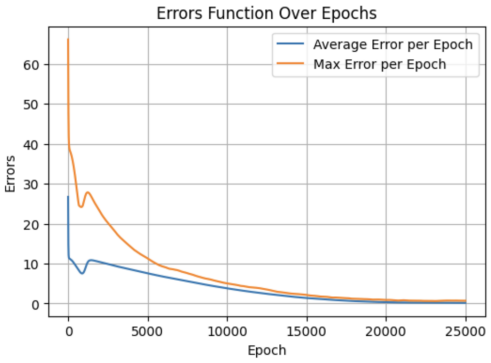


Figure 3.17: LQR Multi-D n=20

3.3 Neural Network Control

3.3.1 One-dimensional Neural Network Control

We will now train a single-layer neural network control using the online optimization algorithm. The state process is:

$$dX_t^\theta = \left(-X_t^\theta + f_\theta\left(X_t^\theta\right)\right) dt + dW_t$$

where the control $f_\theta(\cdot)$ is a single-layer neural network $f_\theta(x) = \sum_{i=1}^m c^i \sigma\left(w^i x + b^i\right)$, with parameters $\theta = (c^i, w^i, b_i)_{i=1}^m$. The objective function is

$$J(\theta) = \lim_{T\to\infty} \frac{1}{T} \int_0^T \left(X_t^\theta\right)^2 + \left(f_\theta\left(X_t^\theta\right)\right)^2 dt \tag{3.14}$$

Define the gradient of X_t with respect to the parameters as:

$$\tilde{X}_t^w = \nabla_w X_t^\theta \in \mathbb{R}^m, \quad \tilde{X}_t^b = \nabla_b X_t^\theta \in \mathbb{R}^m, \quad \tilde{X}_t^c = \nabla_c X_t^\theta \in \mathbb{R}^m$$

The coupled system 3.3 becomes

$$\begin{aligned}
dw_t &= -\alpha_t \left(2X_t \tilde{X}_t^w + 2f_{\theta_t}(X_t) \left(c_t \odot \sigma'(w_t X_t + b_t) X_t + f'_{\theta_t}(X_t) \tilde{X}_t^w\right)\right) dt \\
db_t &= -\alpha_t \left(2X_t \tilde{X}_t^b + 2f_{\theta_t}(X_t) \left(c_t \odot \sigma'(W_t X_t + B_t) + f'_{\theta_t}(X_t) \tilde{X}_t^b\right)\right) dt \\
dc_t &= -\alpha_t \left(2X_t \tilde{X}_t^c + 2f_{\theta_t}(X_t) \left(\sigma(w_t X_t + b_t) + f'_{\theta_t}(X_t) \tilde{X}_t^c\right)\right) dt \\
dX_t &= \left(-X_t + f_{\theta_t}(X_t)\right) dt + dW_t \\
d\tilde{X}_t^w &= \left(-\tilde{X}_t^w + c_t \odot \sigma'\left(w_t X_t^i + b_t\right) X_t + f'_{\theta_t}(X_t) \tilde{X}_t^w\right) dt \\
d\tilde{X}_t^b &= \left(-\tilde{X}_t^b + c_t \odot \sigma'\left(w_t X_t^i + b_t\right) + f'_{\theta_t}(X_t) \tilde{X}_t^b\right) dt \\
d\tilde{X}_t^c &= \left(-\tilde{X}_t^c + \sigma(w_t X_t + b_t) + f'_{\theta_t}(X_t) \tilde{X}_t^c\right) dt \\
d\bar{X}_t &= \left(-\bar{X}_t + f_{\theta_t}\left(\bar{X}_t\right)\right) dt + d\bar{W}_t
\end{aligned}$$

In the original paper we use error metrics to measure the performance of the algorithm. But here in the 1-dimensional case, we can directly use the loss function (3.14) to measure the performance. The training result for 1 dimensional LQR with network network control is presented in Figure 3.18. We can observe that $J(\theta)$ is decreasing while final J is 0.008304406888782978, which shows the performance is good. And since the code becomes complicated we also check whether the gradient the algorithm gives is consistent with the gradient the finite difference gives. The result is shown in figure 3.19. It shows that those gradients are consistent.

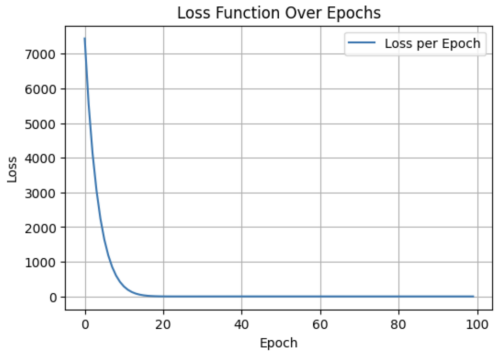


Figure 3.18: NN control 1-D

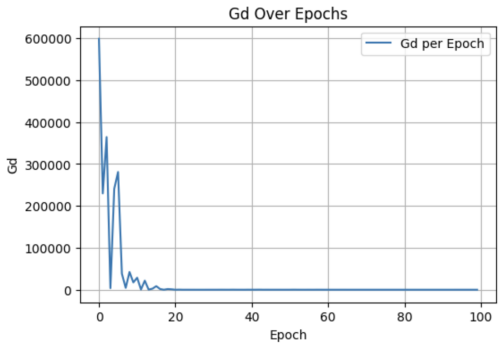


Figure 3.19: NN control 1-D Gradient Check

3.3.2 Multi-dimensional Neural Network Control

We now optimize a single-layer neural network control for a high-dimensional state process:

$$dX_t^\theta = \left(-X_t^\theta + f_\theta\left(X_t^\theta\right)\right) dt + dW_t \tag{3.15}$$

$$J(\theta) = \lim_{T\to\infty} \frac{1}{T} \int_0^T \left(X_t^\theta\right)^\top Q X_t^\theta + \left(f_\theta\left(X_t^\theta\right)\right)^\top R f_\theta\left(X_t^\theta\right) dt \tag{3.16}$$

where $X_t^\theta \in \mathbb{R}^n$ and the single-layer neural network with m hidden units is:

$$f_\theta(x) = c\sigma(wx + b)$$

where $w \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m$, and $c \in \mathbb{R}^{n\times m}$.

As in the below algorithm , define

$$\begin{aligned}
\tilde{X}_t^w &= \nabla_w X_t^\theta \in \mathbb{R}^{n\times m\times n}, \quad \tilde{X}_t^{w,i} = \nabla_w X_t^{\theta,i} \in \mathbb{R}^{m\times n} \\
\tilde{X}_t^b &= \nabla_b X_t^\theta \in \mathbb{R}^{n\times m}, \quad \tilde{X}_t^{b,i} = \nabla_b X_t^{\theta,i} \in \mathbb{R}^m \\
\tilde{X}_t^c &= \nabla_c X_t^\theta \in \mathbb{R}^{n\times n\times m}, \quad \tilde{X}_t^{c,i} = \nabla_c X_t^{\theta,i} \in \mathbb{R}^{n\times m}
\end{aligned}$$

for $i = 1, 2, \cdots, n$.

The online algorithm 2.3 becomes:

$$
\begin{aligned}
dw_t &= -\alpha_t \left[\nabla_w \left(f_{\theta_t}(X_t)^\top R f_{\theta_t}(X_t) \right) + \sum_{i=1}^{n} \frac{\partial}{\partial x_i} \left((X_t)^\top Q X_t + f_{\theta_t}(X_t)^\top R f_{\theta_t}(X_t) \right) \tilde{X}_t^{w,i} \right] dt \\
db_t &= -\alpha_t \left[\nabla_b \left(f_{\theta_t}(X_t)^\top R f_{\theta_t}(X_t) \right) + \sum_{i=1}^{n} \frac{\partial}{\partial x_i} \left((X_t)^\top Q X_t + f_{\theta_t}(X_t)^\top R f_{\theta_t}(X_t) \right) \tilde{X}_t^{b,i} \right] dt \\
dc_t &= -\alpha_t \left[\nabla_c \left(f_{\theta_t}(X_t)^\top R f_{\theta_t}(X_t) \right) + \sum_{i=1}^{n} \frac{\partial}{\partial x_i} \left((X_t)^\top Q X_t + f_{\theta_t}(X_t)^\top R f_{\theta_t}(X_t) \right) \tilde{X}_t^{c,i} \right] dt \\
dX_t &= \left(-X_t + f_{\theta_t}(X_t) \right) dt + dW_t \\
d\tilde{X}_t^{w,i} &= \left(-\tilde{X}_t^{w,i} + \sum_k c_{t,i,k} \sigma' \left(w_t X_t + b_t \right)_k \left(\sum_\ell w_{t,k,\ell} \tilde{X}_t^{w,\ell} \right) + \left(c_{t,i,:} \right)^\top \odot \sigma' \left(w_t X_t + b_t \right) (X_t)^\top \right) dt \\
d\tilde{X}_t^{b,i} &= \left(-\tilde{X}_t^{b,i} + \sum_k c_{t,i,k} \sigma' \left(w_t X_t + b_t \right)_k \left(\sum_\ell w_{t,k,\ell} \tilde{X}_t^{b,\ell} \right) + \left(c_{t,i,:} \right)^\top \odot \sigma' \left(w_t X_t + b_t \right) \right) dt \\
d\tilde{X}_t^{c,i} &= \left(-\tilde{X}_t^{c,i} + \sum_k c_{t,i,k} \sigma' \left(w_t X_t + b_t \right)_k \left(\sum_\ell w_{t,k,\ell} \tilde{X}_t^{c,\ell} \right) + D_i \left(\sigma \left(w_t X_t + b_t \right) \right) \right) dt \\
d\bar{X}_t &= \left(-\bar{X}_t + f_{\theta_t} \left(\bar{X}_t \right) \right) dt + d\bar{W}_t
\end{aligned}
$$

for $i = 1, 2, \cdots, N$. And $C_{t,i,:} \in \mathbb{R}^n$ denotes the i-th row of the matrix C_t and $D_i(X_t)$ is an $n \times n$ matrix whose elements are all zeros except for the i-th row, which has the vector value $\sigma(w_t X_t + b_t)$

In the multi-dimensional case, the performance can no longer be measured by the loss function $J(\theta)$, as the numerical computation of the loss J is often unstable. This instability arises from errors that occur during the numerical integration of $\int_0^T \left(X_t^\theta \right)^\top Q X_t^\theta + \left(f_\theta \left(X_t^\theta \right) \right)^\top R f_\theta \left(X_t^\theta \right) dt$ using numerical discretization. Although a rigorous mathematical proof is not provided, it is presumed that the "Riemann sum" in this context is stable only when the time interval dt is very small. But a tiny dt will cause the computation really expensive and we have no knowledge about how small the dt should be.

Hence we use the method of error metrics. The error metrics are defined as following:

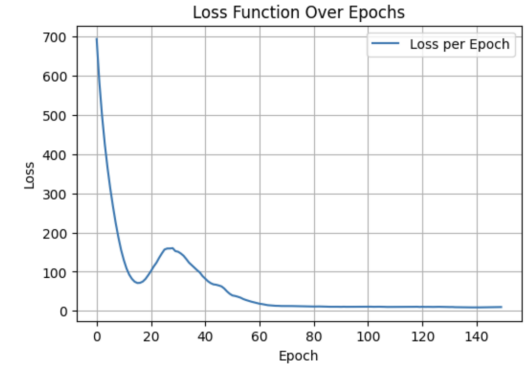
$$
\text{Ave Error } = \frac{\sum_{i=1}^n \| f_{\theta_t}(X^i) - \theta^* X^i \|}{\sum_{i=1}^n \| \theta^* X^i \|} \tag{3.17}
$$

$$
\text{Max Error } = \frac{\max_{i \in \{1,2,\cdots,n\}} \| f_{\theta_t}(X^i) - \theta^* X^i \|}{\sum_{i=1}^n \| \theta^* X^i \|} \tag{3.18}
$$

$$
\text{Cost Error } = \frac{| J(\theta_T) - J(\theta^*) |}{| C^* |} \tag{3.19}
$$

where θ^* is the optimal control and θ_t is the trained parameter. $J(\theta_T)$ and $J(\theta^*)$ denote the objective function $J(\theta)$ in 3.16 with the parameters θ_T and θ^*, respectively.

The numerical results for training the neural network SDE control with the online optimization algorithm are presented in Figure 3.20 and Figure 3.21. In Figure 3.20 we use the loss as the standard while in 3.21 we used error matrices. In general, the trained neural network control performs well, even in high dimensions.




```
Final J: tensor(9.7759, device='cuda:0', grad_fn=<DivBackward0>)
```

Figure 3.20: NN control Multi-D n=5

Chapter 4

Futher Research

4.1 Implementation Method Innovation

4.1.1 Old Way

The old way in the original paper is kind of complicated and not well described and heavily depends on the case itself. Hence it is important to give a "general law" which is applicable in the general case.

4.1.2 New Way

The new way of implementing the algorithm is mainly about using the broadcasting in PyTorch and torch.sum as tools to replace torch.matmul or torch.einsum (einsum notation). For example the update equation in multi-dimensional LQR case:

$$\begin{aligned}\nabla_\theta \left[\left(X_t^\theta\right)^\top \left(Q + \theta^\top R \theta\right) X_t^\theta \right] &= \sum_{i,j} \nabla_\theta \left(\delta_{i,j} + \sum_{k=1}^{n} \theta_{k,i}\theta_{k,j} \right) X_t^{\theta,i} X_t^{\theta,j} \\ &\quad + 2 \sum_{i,j} \nabla_\theta X_t^{\theta,i} \left(q_{i,j} + \theta_{:,i}^\top R \theta_{:,j} \right) X_t^{\theta,j} \\ &= \sum_{i,j} \left((R\theta)_{:,j} 1_{\{i=n\}} + (R\theta)_{:,i} 1_{\{j=n\}} \right) X_t^{\theta,i} X_t^{\theta,j} \\ &\quad + 2 \sum_{i,j} \tilde{X}_t^{\theta,i} \left(q_{i,j} + \theta_{:,i}^\top R \theta_{:,j} \right) X_t^{\theta,j}\end{aligned}$$

For the first term $\sum_{i,j} \left((R\theta)_{:,j} 1_{\{i=n\}} + (R\theta)_{:,i} 1_{\{j=n\}} \right) X_t^{\theta,i} X_t^{\theta,j}$. First use for-loop build $\left((R\theta)_{:,j} 1_{\{i=n\}} + (R\theta)_{:,i} 1_{\{j=n\}} \right)$ which is a n by n matrix.

```
D = torch.zeros(n, n, n, n, device=device)
for k in range(dim):
    for l in range(dim):
```

```
D[k, l, :, k] += Rtheta[:, l]
D[k, l, :, l] += Rtheta[:, k]
```

And notice here $X_t^{\theta,i}$ and $X_t^{\theta,j}$ are n dimensional tensor, i.e. without the batch dimension. Since it is obtained after the torch.mean on the batch dimension. Our thoughts can be conclude as the equation below:

$$\left((R\theta)_{:,j}1_{\{i=n\}} + (R\theta)_{:,i}1_{\{j=n\}}\right) \to (n,n) \xrightarrow{.unsqueeze()} (1,1,n,n) = M \tag{4.1}$$

$$X_t^{\theta,i} \to (n) \xrightarrow{.unsqueeze()} (n,1,1,1) = X_1 \tag{4.2}$$

$$X_t^{\theta,j} \to (n) \xrightarrow{.unsqueeze()} (1,n,1,1) = X_2 \tag{4.3}$$

where the index of the 4 dimensional tensors is (i,j,k,l).

Then with the tool broadcast in PyTorch, we get $M * X_1 * X_2$, the final step is to sum with respect to dimension i and j, which is torch.sum(torch.sum(, dim=1), dim=0).

For the second term $2\sum_{i,j} \tilde{X}_t^{\theta,i}\left(q_{i,j} + \theta_{:,i}^\top R\theta_{:,j}\right) X_t^{\theta,j}$. The implement idea is described below:

$$\tilde{X}_t^{\theta} \to (n,n,n) \xrightarrow{unsqueeze} (n,1,n,n) = X_{tilde} \tag{4.4}$$

$$\left(q_{i,j} + \theta_{:,i}^\top R\theta_{:,j}\right) \to (n,n) \xrightarrow{unsqueeze} (n,n,1,1) = Middle \tag{4.5}$$

$$X_t^{\theta} \to (n) \xrightarrow{unsqueeze} (1,n,1,1) = X_3 \tag{4.6}$$

where the index of the 4-d tensor is (i,j,k,l).

Then with the tool broadcast in PyTorch, we get $X_{tilde} * Middle * X_3$, the final step is to sum with respect to dimension i and j, which is torch.sum(torch.sum(, dim=1), dim=0). Mind the 2 in the front of the 2nd term.

The above discussions shows the main idea of this new implementation method. With the idea in mind, we can deal with more complicated case. Another example is also in the update equation in multi-dimensional LQR case:

$$d\tilde{X}_t^{\theta,i} = \left(-\tilde{X}_t^{\theta,i} + \sum_{j=1}^{n} \theta_{i,j}\tilde{X}_t^{\theta,j} + D_i\left(X_t^{\theta}\right)\right) dt \tag{4.7}$$

where $D_i\left(X_t^{\theta}\right)$ is an $n \times n$ matrix whose elements are all zeros except for the i-th row, which has values X_t^{θ}.

To deal with $\sum_{j=1}^{n} \theta_{i,j}\tilde{X}_t^{\theta,j}$ same thing is "repeated" below:

$$\tilde{X}_t^\theta \to (N, n, n, n) \xrightarrow{unsqueeze} (N, 1, n, n, n) \tag{4.8}$$

$$\theta \to (n, n) \xrightarrow{unsqueeze} (1, n, n, 1, 1) \tag{4.9}$$

where the 5-d tensor is indexed by $(batch, i, j, a, b)$.

Then we sum with respect to dimension j, and get a 4-d tensor $(batch, i, a, b)$

To avoid using for-loops, the first question here is how to build D as a hyper-cube shown below. Our method is:

1. Our idea is to first produce an eye(n) matrix in dimension (i, b).
2. Then unsqueeze it to get (1, n, 1, n) with index $(batch, i, a, b)$.
3. broadcast the above 4-d tensor with unsqueezed X_t^θ, which has 4-dimensions $(N, 1, n, 1)$ with index $(batch, i, a, b)$.

This provide us with the hyper-cube D.

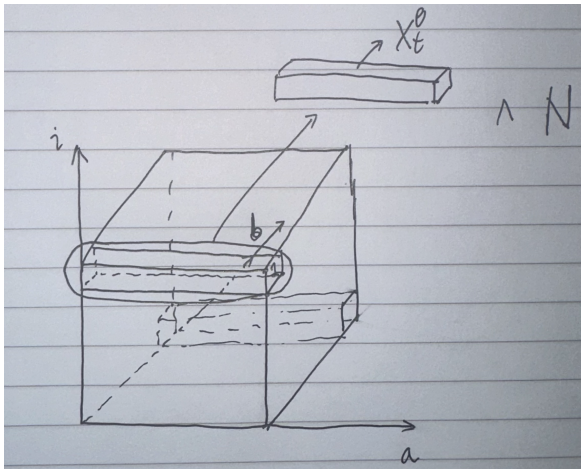


Figure 4.1: Hyper Cube

The final step of the implementation is to sum the $-X_t^\theta \sum_{j=1}^n \theta_{i,j} \tilde{X}_t^{\theta,j}$ and D.

4.1.3 Advantages

Using the new implementation idea, we can deal with multiplications between hyper-matrix as well as multiplications between tensors with different dimensions. The main advantages are as follows. The new method is universal in the sense that it applies no matter what dimensions the tensors are. And the user does not need

to remember the names and signatures of all the different functions in PyTorch for calculating dot products, outer products, transposes and matrix-vector or matrix-matrix multiplications. And the method is so natural that once you understand the equations you can implement those equations using the given idea.

4.2 Higher Dimensions

4.2.1 LQR

Since the higher the dimension is the better GPU with higher RAM to train the model. Even H100 (which is the best) only got 80GB RAM, hence the only way is to lower the batch size, which will cause our output noisy. This is a trade off. After taking those factors into account, we try the case which dimension equals to $dim = 50$.

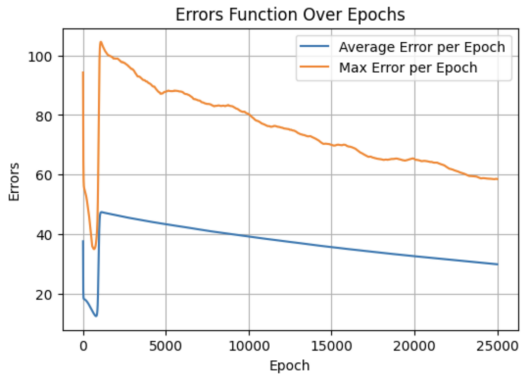


Figure 4.2: LQR n=50

Illustrating the changes in average and maximum errors across 25,000 epochs during a 4-hour runtime, the plot indicates that while the model has achieved a considerable reduction in average error, resulting in overall improved performance, the presence of slow convergence suggests a larger batch size and a longer run time will be beneficial. It is worth mentioning there is a trade-off between computational time and computational cost. With sufficient computation resources such as GPU RAM and runtime, the algorithm can perform well.

4.2.2 Neural Network Control

After taking factors (RAM, batch size, dimensions) into account, we try the case which dimension equals to $dim =?$.

Figure 4.3: NN n=30

4.2.3 Conclusions

4.3 Correlated OU Processes

For the parameters $\theta = (\mu, \sigma)$ with $\mu \in \mathbb{R}^m, \sigma \in \mathbb{R}^{m\times m}$, let the m-dimensional process X_t^θ satisfy

$$dX_t^\theta = \left(\mu - X_t^\theta\right) dt + \sigma dW_t \tag{4.10}$$

where $W_t \in \mathbb{R}^m$.

In the section 3.1.4 Multi-Dimensional Ornstein-Uhlenbeck Process, we pick σ to be an identity matrix or a matrix similar to an identity matrix. Now we try to add some more general correlation into the case.

```
# Create an identity matrix
identity_matrix = torch.eye(m, m, device='cuda')
# Add small random perturbations
perturbations = torch.randn(m, m, device='cuda') * 0.01
# Sigma
sigma = identity_matrix + perturbations
```

now the algorithm is the same as in section 3.1.4:

$$\begin{aligned}
d\mu_t &= -4\alpha_t \left(\left|\bar{X}_t\right|^2 - 2m\right) \left(\sum_{k=1}^{m} X_t^k \tilde{X}_t^{\mu,k}\right) dt \\
d\sigma_t &= -4\alpha_t \left(\left|\bar{X}_t\right|^2 - 2m\right) \left(\sum_{k=1}^{m} X_t^k \tilde{X}_t^{\sigma,k}\right) dt \\
dX_t &= \left(\mu_t - X_t\right) dt + \sigma_t dW_t \\
d\bar{X}_t &= \left(\mu_t - \bar{X}_t\right) dt + \sigma_t d\bar{W}_t \\
d\tilde{X}_t^\mu &= \left(I_m - \tilde{X}_t^\mu\right) dt \\
d\tilde{X}_t^{\sigma,i} &= -\tilde{X}_t^{\sigma,i} dt + D_i\left(dW_t\right), \quad i \in \{1, \cdots, m\}
\end{aligned}$$

where I_m is the $m \times m$ identity matrix and where $D_i\left(dW_t\right)$ is a $m \times m$ matrix with all elements equal to 0 except i-th column being dW_t.

For dimensions $m = 10$. In Figures 4.4, we observes though the algorithm converge the performance is not so good. But it is worth mentioning that for some σ the loss converges to a smaller value. Hence we reckon that for correlated cases, the correlation can heavily influence the performance.

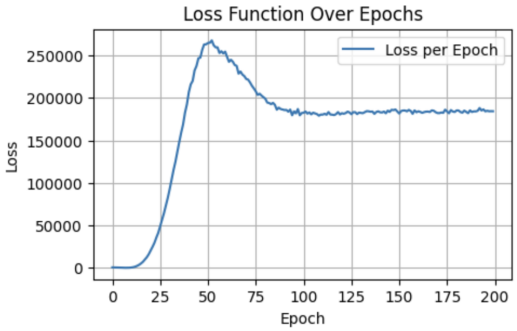


Figure 4.4: Correlated dim=10 (1)

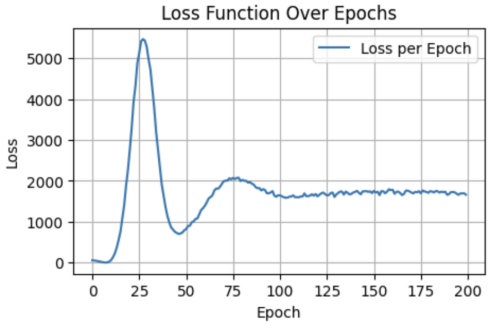


Figure 4.5: Correlated dim=10 (2)

Chapter 5

Conclusions

5.1 Summary of Main Findings

5.2 Future Research

Correlative cases and non-linear processes are the main blue oceans of this given algorithm. While discovering the abilities of the original algorithm, we find that it not only works with the OU process, but also work out with the more general cases. Hence it is very natural to ask whether the algorithm can be applied to the correlative cases and non-linear processes.

To begin with, correlative cases involve scenarios where the stochastic processes are interdependent, introducing complexities that are not present in uncorrelated cases. The ability of the algorithm to handle such inter-dependencies could vastly broaden its applicability. In particular, understanding how the algorithm manages correlations between different state variables and control processes is crucial for its implementation in real-world systems where such correlations are common.

Non-linear processes, on the other hand, pose a different set of challenges. These processes often involve non-linear dynamics in the state evolution, making the control problem significantly more complex. The algorithm's performance in non-linear settings needs thorough investigation, particularly in terms of stability, convergence, and computational feasibility. The extension to non-linear processes would demonstrate the robustness and versatility of the algorithm, potentially enabling its use in a wider array of applications, including those with inherently non-linear behaviors such as financial markets, engineering systems, and biological processes.

Also, we are interested in the mathematical proof of the more general cases. A rigorous mathematical proof of the algorithm's effectiveness in more general cases is highly desirable. Such a proof would provide theoretical underpinnings for the

observed empirical performance, ensuring that the algorithm's application is grounded in solid mathematical principles. This involves proving the stability and convergence properties of the algorithm under general conditions, as well as establishing error bounds for the numerical solutions obtained.

In summary, the extension of this algorithm to correlative cases and non-linear processes represents a promising area of research. Understanding its capabilities in these contexts and providing mathematical proof of its effectiveness would not only enhance its theoretical foundation but also significantly expand its practical applicability.

Bibliography

[1] Brian DO Anderson and John B Moore. *Optimal control: linear quadratic methods*. Courier Corporation, 2007.

[2] Michael Röckner, Xiaobin Sun, and Yingchao Xie. Strong convergence order for slow-fast mckean-vlasov stochastic differential equations, 2019.

[3] Ziheng Wang and Justin Sirignano. A forward propagation algorithm for online optimization of nonlinear stochastic differential equations, 2022.

[4] Ziheng Wang and Justin Sirignano. Continuous-time stochastic gradient descent for optimizing over the stationary distribution of stochastic differential equations, 2023.