MA5232 Modeling and Numerical Simulations

Project 2

Hu Hanyang (A0245841A)

March 26, 2025

Part 1

(a)

Solution. For the LQR problem, we have

$$f(t, x(t), u(t)) = A(t)x(t) + B(t)u(t) \quad \text{and} \quad L(t, x(t), u(t)) = x^\top Q(t)x + u^\top R(t)u \tag{1}$$

hence the corresponding Hamiltonian $H : \mathbb{R} \times \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}$ is given by

$$\begin{aligned} H(t, x, p, u) &= p^\top f(t, x, u) - L(t, x, u) \\ &= p^\top \left(A(t)x + B(t)u\right) - \left(x^\top Q(t)x + u^\top R(t)u\right). \end{aligned} \tag{2}$$

□

(b)

Solution. By Pontryagin's maximum principle (PMP), the optimal control u^* and the corresponding state trajectory x^* and co-state trajectory p^* must satisfy

$$\dot{x}^*(t) = \nabla_p H(t, x^*(t), p^*(t), u^*(t)), \qquad x^*(0) = x_0 \tag{3}$$

$$\dot{p}^*(t) = -\nabla_x H(t, x^*(t), p^*(t), u^*(t)), \qquad p^*(T) = -\nabla_x \Phi(x^*(T)) \tag{4}$$

$$H(t, x^*(t), p^*(t), u^*(t)) \geq H(t, x^*(t), p^*(t), u) \qquad (\text{for all } u \in \mathbb{R}^m \text{ and a.e. } t \in [0, T]). \tag{5}$$

Specifically, for the LQR problem, from equation (2) and $\Phi(x) = x^\top Mx$, we have

$$\dot{x}^*(t) = A(t)x^*(t) + B(t)u^*(t), \qquad x^*(0) = x_0 \tag{6}$$

$$\dot{p}^*(t) = -A^\top(t)p^*(t) + 2Q(t)x^*(t), \qquad p^*(T) = -2Mx^*(T) \tag{7}$$

$$\begin{aligned} &p^*(t)^\top \left(A(t)x^*(t) + B(t)u^*(t)\right) - \left(x^*(t)^\top Q(t)x^*(t) + u^*(t)^\top R(t)u^*(t)\right) \\ &\geq p^*(t)^\top \left(A(t)x^*(t) + B(t)u\right) - \left(x^*(t)^\top Q(t)x^*(t) + u^\top R(t)u\right) \end{aligned} \tag{8}$$

$$\text{for all } u \in \mathbb{R}^m \text{ and a.e. } t \in [0,T].$$

□

(c)

Proof. From condition (8), if an optimal control u^* exists, then we have

$$p^*(t)^\top B(t)u^*(t) - u^*(t)^\top R(t)u^*(t) \geq p^*(t)^\top B(t)u - u^\top R(t)u \tag{9}$$

for all $u \in \mathbb{R}^m$ and a.e. $t \in [0,T]$, i.e. $u^*(t)$ maximizes the following quadratic function

$$h_t(u) = p^*(t)^\top B(t)u - u^\top R(t)u = (B^\top(t)p^*(t))^\top u - u^\top R(t)u \tag{10}$$

which is concave since $R(t)$ is symmetric positive definite. Consequently, we have

$$u^*(t) = \frac{1}{2}R^{-1}(t)B^\top(t)p^*(t) \tag{11}$$

since it is the unique solution of $\nabla_u h_t(u) = -2R(t)u + B^\top(t)p^*(t) = 0$.

Notice that the first-order linear system of ODEs together with the initial conditions in (7) implies that p^* is a linear function of x^*. In particular, assume we have two state trajectories x_1^*, x_2^* and the

corresponding co-state trajectories p_1^*, p_2^* satisfying condition (7), i.e.

$$\dot{p}_i^*(t) = -A^\top(t)p_i^*(t) + 2Q(t)x_i^*(t), \qquad p_i^*(T) = -2Mx_i^*(T) \tag{12}$$

for $i = 1, 2$. For any $a, b \in \mathbb{R}$, let $x^* = ax_1^* + bx_2^*$, it is easy to verify that $p^* = ap_1^* + bp_2^*$ also satisfies condition (7) since $p^*(T) = -2aMx_1^*(T) - 2bMx_2^*(T) = -2Mx^*(T)$ and

$$\begin{aligned}
\dot{p}^*(t) &= a\dot{p}_1(t) + b\dot{p}_2(t) \\
&= a[-A^\top(t)p_1^*(t) + 2Q(t)x_1^*(t)] + b[-A^\top(t)p_2^*(t) + 2Q(t)x_2^*(t)] \\
&= -A^\top(t)[ap_1^*(t) + bp_2^*(t)] + 2Q(t)[ax_1^*(t) + bx_2^*(t)] \\
&= -A^\top(t)p^*(t) + 2Q(t)x^*(t).
\end{aligned} \tag{13}$$

Furthermore, by the Picard-Lindelöf theorem[1], the co-state trajectory p^* satisfying condition (7) must be unique given a fixed (and admissible) state trajectory x^*, hence condition (7) defines a linear operator $\mathcal{L}$ such that $p^* = \mathcal{L}\{x^*\}$. □

Remark. Since x^* is fixed (subject to the dynamics) given the control (11) and the initial state x_0, we have thus obtained a linear map from the initial state x_0 to the co-state p_0 at time $t = 0$, i.e. there exists $P \in \mathbb{R}^{d\times d}$ such that $p_0 = Px_0$. Consequently, we could re-write the state equation (6) and co-state equation (7) by the following linear time-varying system with initial conditions

$$\begin{pmatrix} \dot{x}^*(t) \\ \dot{p}^*(t) \end{pmatrix} = \underbrace{\begin{pmatrix} A(t) & \frac{1}{2}B(t)R^{-1}(t)B^\top(t) \\ 2Q(t) & -A^\top(t) \end{pmatrix}}_{\mathcal{H}(t)\in\mathbb{R}^{2d\times 2d}} \begin{pmatrix} x^*(t) \\ p^*(t) \end{pmatrix}, \qquad \begin{pmatrix} x^*(0) \\ p^*(0) \end{pmatrix} = \begin{pmatrix} x_0 \\ Px_0 \end{pmatrix} \tag{14}$$

[1]We should verify the following conditions: (1) $g(t,p) := -A^\top(t)p + 2Q(t)x^*(t)$ is continuous in $t \in [0,T]$ since $A(t), Q(t), x^*(t)$ are all continuous; (2) for all $p_1, p_2 \in \mathbb{R}^d$ and $t \in [0,T]$, we have $\|g(t,p_1) - g(t,p_2)\| = \|-A^\top(t)(p_1 - p_2)\| \leq C\|p_1 - p_2\|$ where $C = \sup_{t\in[0,T]} \|A^\top(t)\| < \infty$ since $\|A^\top(t)\|$ is continuous on the compact set $[0,T]$

which, by writing the corresponding state transition matrix $\Phi(t)$ in block form[2], gives us

$$\begin{pmatrix} x^*(t) \\ p^*(t) \end{pmatrix} = \underbrace{\begin{pmatrix} \Phi_{11}(t) & \Phi_{12}(t) \\ \Phi_{21}(t) & \Phi_{22}(t) \end{pmatrix}}_{\Phi(t)\in\mathbb{R}^{2d\times 2d}} \begin{pmatrix} x_0 \\ Px_0 \end{pmatrix} \tag{15}$$

so that $x^*(t) = (\Phi_{11}(t) + P\Phi_{12}(t))x_0$ and

$$\begin{aligned} p^*(t) &= (\Phi_{21}(t) + P\Phi_{22}(t))x_0 \\ &= \underbrace{(\Phi_{21}(t) + P\Phi_{22}(t))(\Phi_{11}(t) + P\Phi_{12}(t))^{-1}}_{-2P(t)} x^*(t) \end{aligned} \tag{16}$$

assuming that $\Phi_{11}(t) + P\Phi_{12}(t)$ is invertible.

Notice that the LTV system (14) is similar to the system (24) which we shall discuss later.

Part 2

(a)

Proof. From the previous result, we could write

$$p^*(t) = -2P(t)x^*(t) \tag{17}$$

where $P(t) \in \mathbb{R}^{d\times d}$, which implies

$$\dot{p}^*(t) = -2P(t)\dot{x}^*(t) - 2\dot{P}(t)x^*(t) \tag{18}$$

by differentiating w.r.t. t. Furthermore, to satisfy condition (8), we could use equation (11) to obtain

$$u^*(t) = \frac{1}{2}R^{-1}(t)B^\top(t)p^*(t) = -R^{-1}(t)B^\top(t)P(t)x^*(t). \tag{19}$$

[2]The state transition matrix $\Phi(t) \in \mathbb{R}^{2d\times 2d}$ is the unique solution of $\dot{\Phi}(t) = \mathcal{H}(t)\Phi(t)$ with initial condition $\Phi(0) = I$.

If we assume that P is known and take the control u^* described by equation (19), then there is a unique solution that satisfies condition (6), which should be the corresponding state trajectory x^*.

Suppose $P(t)$ satisfies the following Ricatti differential equation (RDE)

$$\dot{P}(t) = -P(t)A(t) - A^\top(t)P(t) - Q(t) + P(t)B(t)R^{-1}(t)B^\top(t)P(t), \qquad P(T) = M. \tag{20}$$

From equations (17) and (19), we could obtain

$$\dot{P}(t)x^*(t) = -P(t)(A(t)x^*(t) + B(t)u^*(t)) + \frac{1}{2}A^\top(t)p^*(t) - Q(t)x^*(t). \tag{21}$$

Substitute equations (6) and (21) into equation (18), we have

$$\begin{aligned}
\dot{p}^*(t) &= -2P(t)\dot{x}^*(t) - 2\left[-P(t)(A(t)x^*(t) + B(t)u^*(t)) + \frac{1}{2}A^\top(t)p^*(t) - Q(t)x^*(t)\right] \\
&= -2P(t)\dot{x}^*(t) - 2\left[-P(t)\dot{x}^*(t) + \frac{1}{2}A^\top(t)p^*(t) - Q(t)x^*(t)\right] \\
&= -A^\top(t)p^*(t) + 2Q(t)x^*(t).
\end{aligned} \tag{22}$$

Furthermore, we have $p^*(T) = -2P(T)x^*(t) = -2Mx^*(t)$. That is, the corresponding co-state trajectory p^* satisfies condition (7).

Consequently, the control that is given by

$$u_t = -R^{-1}(t)B^\top(t)P(t)x_t \tag{23}$$

would satisfy conditions (6, 7, 8) of PMP if P satisfies the RDE (20). □

(b)

Proof. Suppose X, Y satisfy the following matrix differential equation of size $2d \times d$

$$\frac{d}{dt}\begin{pmatrix} X(t) \\ Y(t) \end{pmatrix} = \begin{pmatrix} A(t) & \frac{1}{2}B(t)R^{-1}(t)B^\top(t) \\ 2Q(t) & -A^\top(t) \end{pmatrix}\begin{pmatrix} X(t) \\ Y(t) \end{pmatrix}, \qquad \begin{pmatrix} X(T) \\ Y(T) \end{pmatrix} = \begin{pmatrix} I \\ -2M \end{pmatrix}. \tag{24}$$

Define $P(t) := -\frac{1}{2}Y(t)X^{-1}(t)$. Differentiate w.r.t. t, we have

$$\dot{P}(t) = \frac{d}{dt}\left(-\frac{1}{2}Y(t)X^{-1}(t)\right) = -\frac{1}{2}\left(\frac{d}{dt}Y(t)X^{-1}(t) - Y(t)X^{-1}(t)\frac{d}{dt}X(t)X^{-1}(t)\right) \tag{25}$$

from the product rule and $\frac{d}{dt}X^{-1}(t) = -X^{-1}(t)\frac{d}{dt}X(t)X^{-1}(t)$.[3] Substituting equation (24) as well as the definition $P(t) = -\frac{1}{2}Y(t)X^{-1}(t)$, the RHS of equation (25) becomes

$$\begin{aligned}
&-\frac{1}{2}\left[(2Q(t)X(t) - A^\top(t)Y(t))X^{-1}(t) - Y(t)X^{-1}(t)\left(A(t)X(t) + \frac{1}{2}B(t)R^{-1}(t)B^\top(t)Y(t)\right)X^{-1}(t)\right] \\
&= -Q(t) + \frac{1}{2}A^\top(t)Y(t)X^{-1}(t) + \frac{1}{2}Y(t)X^{-1}(t)A(t) + \frac{1}{4}Y(t)X^{-1}(t)B(t)R^{-1}(t)B^\top(t)Y(t)X^{-1}(t) \\
&= -Q(t) - A^\top(t)P(t) - P(t)A(t) + P(t)B(t)R^{-1}(t)B^\top(t)P(t) \\
&= -P(t)A(t) - A^\top(t)P(t) - Q(t) + P(t)B(t)R^{-1}(t)B^\top(t)P(t)
\end{aligned} \tag{26}$$

which equals the RHS of equation (20). Furthermore, it is clear that $P(T) = -\frac{1}{2}Y(T)X^{-1}(T) = M$.

Consequently, $P(t) = -\frac{1}{2}Y(t)X^{-1}(t)$ satisfies the RDE (20). □

Part 3

(a)

Solution. Let $V : [0,T] \times \mathbb{R}^d \to \mathbb{R}$ denote the value function corresponding to the LQR problem. The Hamilton-Jacobi-Bellman (HJB) equation that V should satisfy is

$$\partial_t V(t,x) + \inf_{u\in\mathbb{R}^m}\{\underbrace{(x^\top Q(t)x + u^\top R(t)u)}_{L(t,u,x)} + [\partial_x V(t,x)]^\top \underbrace{(A(t)x + B(t)u)}_{f(t,u,x)}\} = 0 \tag{27}$$

for all $(t,x) \in (0,T) \times \mathbb{R}^d$. Furthermore, $V(T,x) = \Phi(x) = x^\top Mx$. □

[3] Since $0 = \frac{dI}{dt} = \frac{d}{dt}(X(t)X^{-1}(t)) = X(t)\left(\frac{d}{dt}X^{-1}(t)\right) + \left(\frac{d}{dt}X(t)\right)X^{-1}(t)$.

(b)

Proof. Notice that the following quadratic function

$$g_t(u) = u^\top R(t)u + [\partial_x V(t,x)]^\top B(t)u \tag{28}$$

is convex since $R(t)$ is symmetric positive definite, hence it has a unique minimum $u^* \in \mathbb{R}^m$ such that

$$\begin{aligned} \nabla_u g_t(u) &= 2R(t)u + B^\top(t)[\partial_x V(t,x)] = 0 \\ \implies \qquad u^* &= -\frac{1}{2}R^{-1}(t)B^\top(t)[\partial_x V(t,x)]. \end{aligned} \tag{29}$$

Consequently, we have

$$\begin{aligned} g_t(u^*) &= \frac{1}{4}[\partial_x V(t,x)]^\top B(t)R^{-1}(t)B^\top(t)[\partial_x V(t,x)] - \frac{1}{2}[\partial_x V(t,x)]^\top B(t)R^{-1}(t)B^\top(t)[\partial_x V(t,x)] \\ &= -\frac{1}{4}[\partial_x V(t,x)]^\top B(t)R^{-1}(t)B^\top(t)[\partial_x V(t,x)] \end{aligned} \tag{30}$$

which implies

$$\begin{aligned} &\inf_{u\in\mathbb{R}^m}\{(\underbrace{x^\top Q(t)x + u^\top R(t)u}_{L(t,u,x)}) + [\partial_x V(t,x)]^\top(\underbrace{A(t)x + B(t)u}_{f(t,u,x)})\} \\ &= x^\top Q(t)x + (u^*)^\top R(t)u^* + [\partial_x V(t,x)]^\top A(t)x + g_t(u^*) \\ &= x^\top Q(t)x + [\partial_x V(t,x)]^\top A(t)x - \frac{1}{4}[\partial_x V(t,x)]^\top B(t)R^{-1}(t)B^\top(t)[\partial_x V(t,x)]. \end{aligned} \tag{31}$$

To conclude, the HJB equation (27) could be simplified to

$$\begin{aligned} -\partial_t V(t,x) &= x^\top Q(t)x + [\partial_x V(t,x)]^\top A(t)x \\ &\quad - \frac{1}{4}[\partial_x V(t,x)]^\top B(t)R^{-1}(t)B^\top(t)[\partial_x V(t,x)] \\ V(T,x) &= x^\top Mx. \end{aligned} \tag{32}$$

□

(c)

Proof. Using the ansatz $V(t,x) = x^\top P(t)x$, the simplified HJB equation (32) becomes

$$\begin{aligned} -x^\top \dot{P}(t)x &= x^\top Q(t)x + 2x^\top P^\top(t)A(t)x \\ &\quad - x^\top P^\top(t)B(t)R^{-1}(t)B^\top(t)P(t)x \\ P(T) &= M. \end{aligned} \tag{33}$$

We would like to show that the solution of the RDE (20) is symmetric. Suppose $P(t)$ satisfies the RDE (20), then we have

$$\begin{aligned} \dot{P}^\top(t) &= (\dot{P}(t))^\top = (-P(t)A(t) - A^\top(t)P(t) - Q(t) + P(t)B(t)R^{-1}(t)B^\top(t)P(t))^\top \\ &= -A^\top(t)P^\top(t) - P^\top(t)A(t) - Q(t) + P^\top(t)B(t)R^{-1}(t)B^\top(t)P^\top(t) \\ P^\top(T) &= M^\top = M \end{aligned} \tag{34}$$

since $Q(t), R(t)$ and M are symmetric. That is to say, $P^\top(t)$ also satisfies the RDE (20). Using the fact that the RDE (20) admits a unique solution, we have $P(t) = P^\top(t)$, i.e. $P(t)$ must be symmetric.

Consequently, substituting the RDE (20) and using symmetricities of $A(t)$ and $P(t)$, we have

$$\begin{aligned} -x^\top \dot{P}(t)x &= -x^\top \Big(-P(t)A(t) - A^\top(t)P(t) - Q(t) + P(t)B(t)R^{-1}(t)B^\top(t)P(t)\Big)\, x \\ &= x^\top Q(t)x + 2x^\top P^\top(t)A(t)x - x^\top P^\top(t)B(t)R^{-1}(t)B^\top(t)P(t)x \\ P(T) &= M \end{aligned} \tag{35}$$

which shows that the value function $V(t,x)$ we considered solves the HJB equation (27). □

Part 4

(a)

Solution. We use $z(t) \in \mathbb{R}^2$ to denote the state vector, then we have

$$z(t) = \begin{pmatrix} x(t) \\ v(t) \end{pmatrix},\ A(t) = \begin{pmatrix} 0 & 1 \\ 0 & -\alpha(t) \end{pmatrix},\ B(t) = \begin{pmatrix} 0 \\ 1 \end{pmatrix},\ Q(t) = 0_{2,2},\ R(t) = \lambda,\ M = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}. \tag{36}$$

Substitute into the RDE (20) and use the fact that $P_{12}(t) = P_{21}(t)$ as shown in Part 3(c), we have

$$\begin{aligned} \begin{pmatrix} \dot{P}_{11}(t) & \dot{P}_{12}(t) \\ \dot{P}_{21}(t) & \dot{P}_{22}(t) \end{pmatrix} &= -P(t)A(t) - A^\top(t)P(t) - Q(t) + P(t)B(t)R^{-1}(t)B^\top(t)P(t) \\ &= -\begin{pmatrix} 0 & P_{11}(t) - \alpha(t)P_{12}(t) \\ 0 & P_{21}(t) - \alpha(t)P_{22}(t) \end{pmatrix} - \begin{pmatrix} 0 & 0 \\ P_{11}(t) - \alpha(t)P_{21}(t) & P_{12}(t) - \alpha(t)P_{22}(t) \end{pmatrix} \\ &\quad + \frac{1}{\lambda}\begin{pmatrix} P_{12}(t)P_{21}(t) & P_{12}(t)P_{22}(t) \\ P_{22}(t)P_{21}(t) & P_{22}(t)^2 \end{pmatrix} \\ &= \begin{pmatrix} \frac{1}{\lambda}P_{12}(t)^2 & -P_{11}(t) + \alpha(t)P_{12}(t) + \frac{1}{\lambda}P_{12}(t)P_{22}(t) \\ \cdots & -2P_{12}(t) + 2\alpha(t)P_{22}(t) + \frac{1}{\lambda}P_{22}(t)^2 \end{pmatrix}. \end{aligned} \tag{37}$$

which could be re-written (together with the condition $P(T) = M$) as

$$\begin{pmatrix} \dot{P}_{11}(t) \\ \dot{P}_{12}(t) \\ \dot{P}_{22}(t) \end{pmatrix} = \begin{pmatrix} \frac{1}{\lambda}P_{12}(t)^2 \\ -P_{11}(t) + \alpha(t)P_{12}(t) + \frac{1}{\lambda}P_{12}(t)P_{22}(t) \\ -2P_{12}(t) + 2\alpha(t)P_{22}(t) + \frac{1}{\lambda}P_{22}(t)^2 \end{pmatrix}, \qquad \begin{pmatrix} P_{11}(T) \\ P_{12}(T) \\ P_{22}(T) \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}. \tag{38}$$

Once we use a numerical ODE solver[4] to solve equation (38), we may substitute the relation $u^*(t) = -R^{-1}(t)B^\top(t)P(t)z^*(t)$ to the state equation $z^*(t) = A(t)z^*(t) + B(t)u^*(t)$ with initial condition $z^*(0) = (1, 0)$. The resulting system could then be solved forward in time using the same numerical ODE solver. To enhance computational efficiency (although negligible in this case), we avoid storing the full backward solution $P(t)$. Instead, only $P(0)$ is retained, and an augmented forward system (combining the state equation and equation (38)) is passed to the numerical solver.

[4]We use the implementation of the explicit Runge-Kutta method of order 5(4) available in the SciPy library.

Alternatively, we may use Neural ODE to solve this optimal control problem, which is essentially a variant of the method of successive approximation. In each training step, we first compute $z(T)$ based on the current MLP-based control $u_\theta(\cdot)$, then solve the following augmented ODE backward in time

$$\begin{pmatrix} \dot{z}(t) \\ \dot{p}(t) \\ \dot{w}(t) \end{pmatrix} = \begin{pmatrix} \nabla_p H(t, z(t), p(t), u_\theta(t)) \\ -\nabla_x H(t, z(t), p(t), u_\theta(t)) \\ -\nabla_\theta H(t, z(t), p(t), u_\theta(t)) \end{pmatrix}, \qquad \begin{pmatrix} z(T) \\ p(T) \\ w(T) \end{pmatrix} = \begin{pmatrix} z(T) \\ -2Mz(T) \\ 0 \end{pmatrix} \tag{39}$$

where $H(t, z(t), p(t), u_\theta(t))$ is the Hamiltonian.[5] Subsequently, we apply the parameter update

$$\theta_{\text{next}} \leftarrow \theta + \eta \cdot w(0) \tag{40}$$

with learning rate $\eta > 0$. Notice that $w(0) = \int_{t_0}^{T} \nabla_\theta H(t, z(t), p(t), u_\theta(t))dt$. A "justification" of this Neural ODE algorithm, from the perspective of Pontryagin's maximum principle, is deferred to the end of this project. The comparison of the RDE solution and the Neural ODE solution is presented in Figure 1, tested on $\alpha(t) = \sin(10t)$ and $\alpha(t) = t^2$ with $\lambda = 1$.

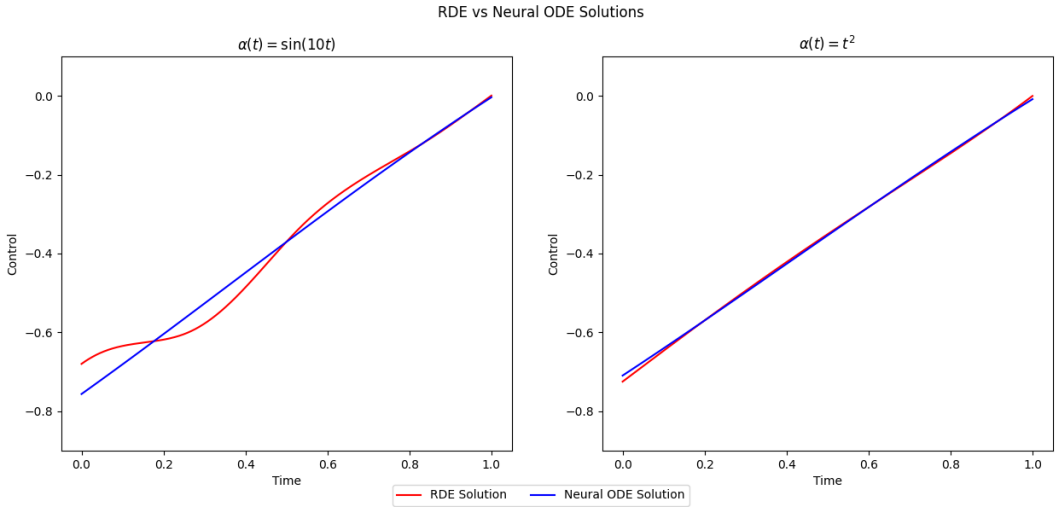

Figure 1: Comparison of the control solved by RDE and Neural ODE on the LQR problem. For $\alpha(t) = \sin(10t)$, the Neural ODE solution has a large discrepancy with the RDE solution.

□

[5] We use auto differentiation available in the PyTorch library to compute $\nabla_\theta H$.

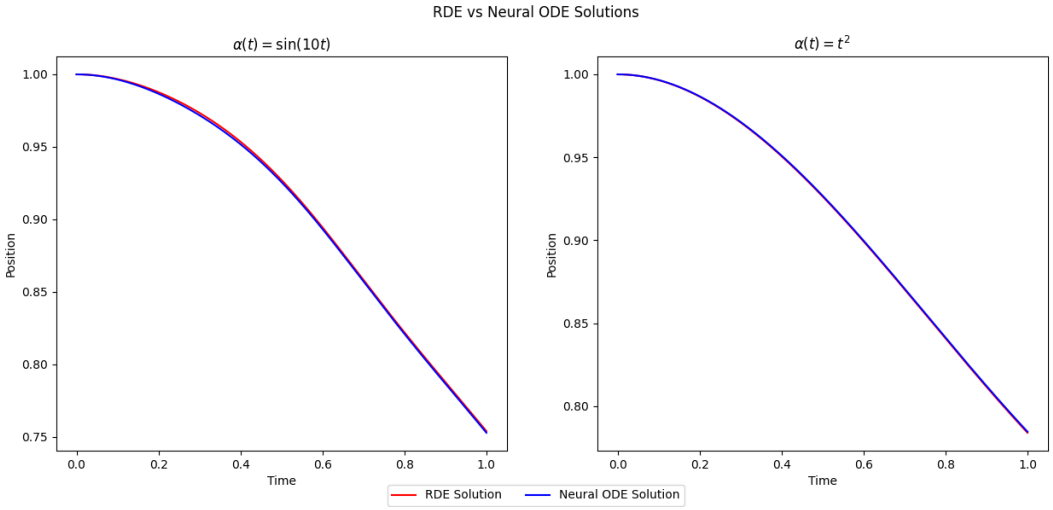

Figure 2: Comparison of the position trajectory solved by RDE and Neural ODE on the LQR problem. Despite the discrepancy in the controls in Figure 1, the resulting position trajectories are similar.

RDE Implementation

```python
import numpy as np
from scipy.integrate import solve_ivp

def rde_solver(L, alpha):
    def rde_func(t, y, *args):
        L, alpha = args
        dydt = np.zeros_like(y)
        dydt[0] = 1/L * y[1]**2
        dydt[1] = -y[0] + alpha(t) * y[1] + 1/L * y[1] * y[2]
        dydt[2] = -2 * y[1] + 2 * alpha(t) * y[2] + 1/L * y[2]**2
        return dydt

    def dynamics_func(t, y, *args):
        L, alpha = args
        A = np.array([[0, 1], [0, -alpha(t)]])
        B = np.array([[0], [1]])
        P = np.array([[y[2], y[3]], [y[3], y[4]]])
        z = np.array([y[0], y[1]])
```

```
        u = -1/L * B.T @ P @ z
        dzdt = A @ z + B @ u
        dydt = np.zeros_like(y)
        dydt[0:2] = dzdt
        dydt[2:5] = np.zeros(3)
        dydt[2] = 1/L * y[3]**2
        dydt[3] = -y[2] + alpha(t) * y[3] + 1/L * y[3] * y[4]
        dydt[4] = -2 * y[3] + 2 * alpha(t) * y[4] + 1/L * y[4]**2
        return dydt

    params = (L, alpha)
    P_T = [1.0, 0.0, 0.0]  # Initial state vector
    t_span = (1.0, 0.0)  # Solve the ODE backward in time

    backward_solution = solve_ivp(
        fun=rde_func,
        t_span=t_span,
        y0=P_T,
        args=params,
        method='RK45',  # Explicit Runge-Kutta method of order 5(4)
        dense_output=False,
        t_eval=[0.0]  # Only evaluate at t=0
    )

    P_0 = backward_solution.y.flatten()
    z_0 = np.array([1.0, 0.0])
    augmented_z_0 = np.concatenate((z_0, P_0), axis=0)
    t_span = (0.0, 1.0)
    t_eval = np.linspace(t_span[0], t_span[1], 1000)

    # Solve the augmented forward dynamics
    forward_solution = solve_ivp(
        fun=dynamics_func,
```

```
        t_span=t_span,
        y0=augmented_z_0,
        args=params,
        method='RK45',
        dense_output=True,
        t_eval=t_eval
    )

    t = forward_solution.t
    augmented_z = forward_solution.y

    # Compute the optimal control u(t) = -1/L * B^T(t)P(t)z(t)
    B = np.array([[0], [1]])
    u = np.zeros(len(t))
    for i in range(len(t)):
        P_i = np.array(
            [
                [augmented_z[2, i], augmented_z[3, I]],
                [augmented_z[3, i], augmented_z[4, I]]
            ]
        )
        z_i = augmented_z[0:2, i]
        u[i] = -1/L * B.T @ P_i @ z_i

    return (t, u)
```

Neural ODE Implementation

```
import torch
import numpy as np
from tqdm import tqdm

class MLP(torch.nn.Module):
```

```
    def __init__(self, input_size, hidden_size, output_size, zero_init=False):
        super(MLP, self).__init__()
        self.fc1 = torch.nn.Linear(input_size, hidden_size)
        self.fc2 = torch.nn.Linear(hidden_size, output_size)

        # Initialize weights
        if zero_init:
            torch.nn.init.zeros_(self.fc1.weight)
            torch.nn.init.zeros_(self.fc1.bias)
            torch.nn.init.zeros_(self.fc2.weight)
            torch.nn.init.zeros_(self.fc2.bias)
        else:
            torch.nn.init.xavier_uniform_(self.fc1.weight)
            torch.nn.init.zeros_(self.fc1.bias)
            torch.nn.init.xavier_uniform_(self.fc2.weight)
            torch.nn.init.zeros_(self.fc2.bias)

        self.act_fn = torch.nn.Tanh()

    def forward(self, x):
        x = self.act_fn(self.fc1(x))
        x = self.fc2(x)
        return x

@torch.no_grad()
def state_eqn(t, z, u, alpha):
    A = torch.tensor([[0., 1.], [0., -alpha(t)]])
    B = torch.tensor([[0.], [1.]])
    f = A @ z + B @ u
    return f

@torch.no_grad()
def costate_eqn(t, p, alpha):
```

```
    A = torch.tensor([[0., 1.], [0., -alpha(t)]])
    return -A.T @ p

def hamiltonian(t, z, p, u, *args):
    L, alpha = args
    A = torch.tensor([[0., 1.], [0., -alpha(t)]])
    B = torch.tensor([[0.], [1.]])
    f = A @ z + B @ u
    H = p @ f - L * u ** 2
    return H

def update_w(w, t, z, p, model, L, alpha, h):
    model.zero_grad(set_to_none=True)

    u = model(torch.tensor([t]))
    H = -hamiltonian(t, z, p, u, L, alpha) # for computing $-\nabla_\theta H$
    H.backward() # backpropagation

    with torch.no_grad():
        w.fc1.weight -= h * model.fc1.weight.grad
        w.fc1.bias -= h * model.fc1.bias.grad
        w.fc2.weight -= h * model.fc2.weight.grad
        w.fc2.bias -= h * model.fc2.bias.grad

    return w


def neural_ode_solver(
        L, alpha, num_iters=500, lr=0.05, decay_rate=0.999, time_iterval=0.05,
        rde_init=True, pre_train_iters=5000, return_init=False
    ):
    hidden_dim = 10
    model = MLP(1, hidden_dim, 1)
```

```
72
73      if rde_init:
74          # Use RDE solution as initial guess
75          from rde_solver import rde_solver
76          t_rde, u_rde = rde_solver(L, alpha)
77          t, u = torch.tensor(t_rde).reshape(-1, 1).float(), torch.tensor(u_rde).
    reshape(-1, 1).float()
78
79          # Fit a neural network to the control trajectory
80          progress_bar = tqdm(range(pre_train_iters), desc="Fitting RDE Solution")
81          optimizer = torch.optim.Adam(model.parameters(), lr=0.1)
82          for _ in progress_bar:
83              optimizer.zero_grad()
84
85              loss = torch.mean((model(t) - u) ** 2)
86
87              loss.backward()
88              optimizer.step()
89
90              progress_bar.set_postfix({'Loss': loss.item()})
91
92      # add random noise to the model parameters
93      with torch.no_grad():
94          sigma = 0.1
95          model.fc1.weight += sigma * torch.randn_like(model.fc1.weight)
96          model.fc1.bias += sigma * torch.randn_like(model.fc1.bias)
97          model.fc2.weight += sigma * torch.randn_like(model.fc2.weight)
98          model.fc2.bias += sigma * torch.randn_like(model.fc2.bias)
99
100     if return_init:
101         t = torch.linspace(0, 1, 100).reshape(-1, 1)
102         u_init = model(t).detach().numpy()
103
```

```
    progress_bar = tqdm(range(num_iters), desc="Training Neural ODE")
    for _ in progress_bar:
        with torch.no_grad():
            t_0, T, h = 0.0, 1.0, time_iterval
            z = torch.tensor([1.0, 0.0])
            for t in torch.arange(t_0, T, h):
                u = model(torch.tensor([t]))
                z = z + h * state_eqn(t, z, u, alpha)
        z_T = z.detach()

        M = torch.tensor([[1.0, 0.0], [0.0, 0.0]])
        augmented_z = {
            'z': z_T,
            'p': -2 * M @ z_T,
            'w': MLP(1, hidden_dim, 1, zero_init=True)
        }
        for t in torch.arange(T, t_0, -h):
            new_z = augmented_z['z'] - h * state_eqn(t, augmented_z['z'], model(torch
    .tensor([t])).detach(), alpha)
            new_p = augmented_z['p'] - h * costate_eqn(t, augmented_z['p'], alpha)
            new_w = update_w(augmented_z['w'], t, augmented_z['z'], augmented_z['p'],
     model, L, alpha, h)
            augmented_z.update({
                'z': new_z,
                'p': new_p,
                'w': new_w
            })

        w_0 = augmented_z['w'] # extract gradient and update model
        with torch.no_grad():
            model.fc1.weight += lr * w_0.fc1.weight
            model.fc1.bias += lr * w_0.fc1.bias
            model.fc2.weight += lr * w_0.fc2.weight
```

```
            model.fc2.bias += lr * w_0.fc2.bias

            grad_norm = torch.norm(w_0.fc1.weight) + torch.norm(w_0.fc1.bias) + torch
    .norm(w_0.fc2.weight) + torch.norm(w_0.fc2.bias)

        lr *= decay_rate # learning rate decay

        progress_bar.set_postfix({'Grad Norm': grad_norm.item()})

    t = torch.linspace(0, 1, 100).reshape(-1, 1)
    u = model(t).detach().numpy()

    if return_init:
        return t.numpy(), u, u_init
    else:
        return t.numpy(), u
```

Remark. The full code is available at `https://github.com/hanyang-hu/neural-ode-exp`.

(b)

Discussion. In this simple example, the RDE solution is not only more reliable[6] but also significantly more efficient, as it requires only one forward pass and one backward pass through the numerical solver. In contrast, each gradient update step for the Neural ODE (with a total of 1000 steps) necessitates both a forward pass and a backward pass. Nevertheless, the Neural ODE could be applied in a more general setting, whereas the RDE method is limited to simpler problems, such as the LQR. Moreover, an advantage of the Neural ODE over other variants of the method of successive approximation (MSA) is that it requires constant memory, whereas conventional MSAs need to maintain $\{u(t^{(0)}), \dots, u(t^{(n)})\}$ for different timesteps.

It is important to observe that the Neural ODE with random initialization converges to a solution

[6] We can infer this from the previous theoretical discussions. Furthermore, we performed a sanity check of the RDE method for the case where $\alpha(t) \equiv 0$ at the end of this project.

that is noticeably different from that of the RDE method when $\alpha(t) = \sin(10t)$ in Figure 1.[7] We hypothesize that this discrepancy arises because the Neural ODE becomes trapped in a local optimum. To test this hypothesis, we pre-trained the MLP-based control u_θ to fit the RDE solution and then randomly perturbed the pre-trained parameters to obtain an initialization. The converged results of Neural ODE indeed match more with those of the RDE method, as shown in Figure 3.

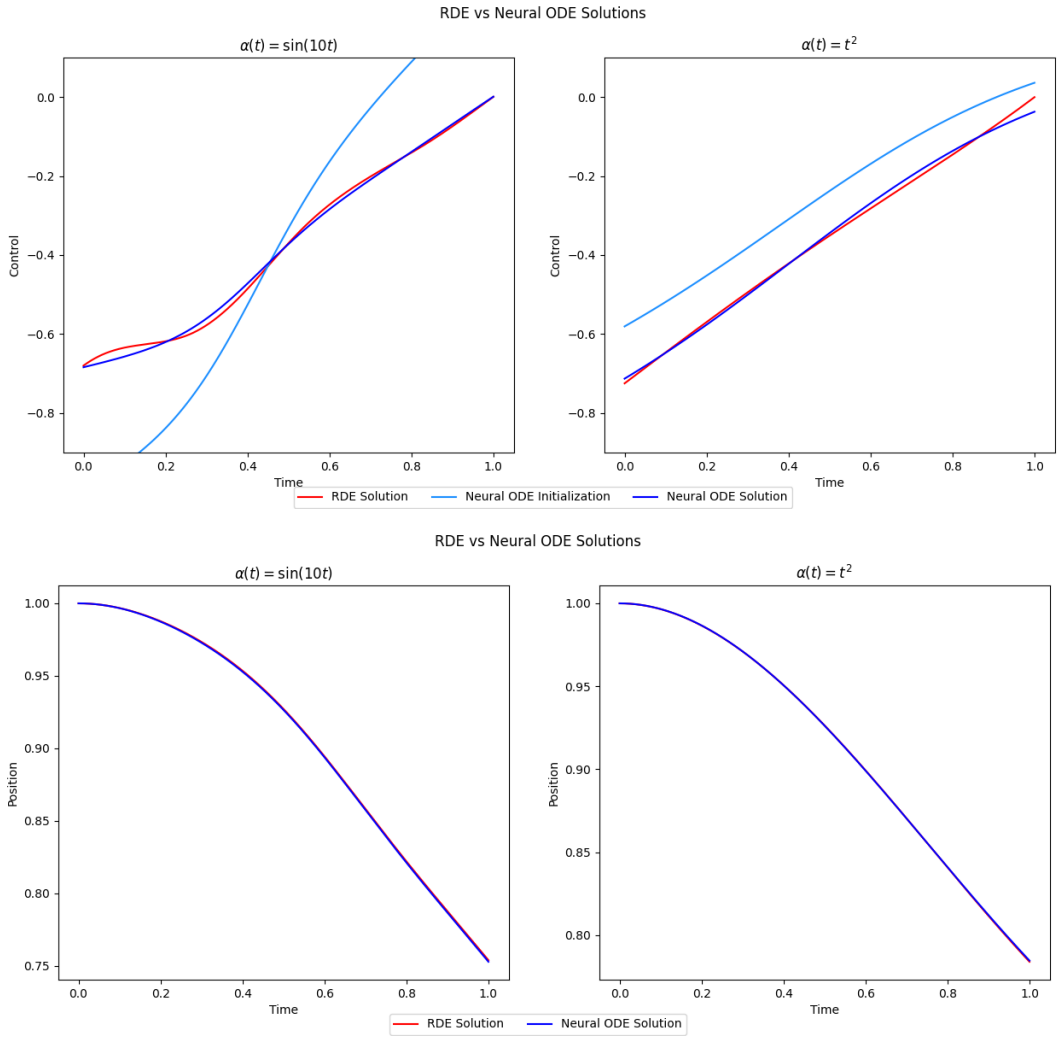

Figure 3: Comparison of the control and position trajectory solved by RDE and Neural ODE (with initialization) on the LQR problem. The solutions of the Neural ODE are indeed closer to those of the RDE method compared to Figure 1 and Figure 2.

[7] However, we could see in Figure 2 that the $x(t)$ trajectories are similar, despite having noticeably different controls.

We should notice that Neural ODE is essentially based on a relaxed version of the PMP, which does not necessarily guarantee optimality.[8] This suggests that we may want to use better optimizers (e.g., momentum method or Nesterov accelerated gradient) or better initialization when applying Neural ODE to avoid sticking in local optima. For example, we may want to use a simple control trajectory that is sufficiently close to the optimal trajectory (albeit potentially non-optimal) to learn the initialization parameters, then use Neural ODE to fine-tune the result.

Additionally, physics-informed neural networks (PINNs) could be another deep learning approach for solving optimal control problems. In this framework, one can train a PINN to learn a value function that satisfies the HJB equation and subsequently extract the optimal control $u^*(\cdot)$. However, the term

$$\inf_{u\in\mathbb{R}^m}\left\{L(t,u,x)+\left[\partial_x V(t,x)\right]^\top f(t,u,x)\right\} \tag{41}$$

may be intractable for automatic differentiation, which necessitates learning a parameterized control u_θ that approximates u^*, and subsequently approximating the infimum (41) with

$$L(t,u_\theta(t),x)+\left[\partial_x V(t,x)\right]^\top f(t,u_\theta(t),x), \tag{42}$$

similar to the treatments in deep reinforcement learning. □

[8] Although an improvement lower bound should hold under certain technical conditions, particularly when the distance between the current and updated parameters is small.

Neural ODE and Optimal Control

We attempt to interpret the adjoint sensitivities algorithm used in Neural ODEs in the context of optimal control. In particular, consider the Mayer problem for simplicity

$$\begin{aligned} &\inf_u J[u] = \Phi(x(t_1)) \\ &\text{s.t. } \dot{x}(t) = f(t, x(t), u(t)), \qquad t \in [t_0, t_1], \qquad x(t_0) = x_0. \end{aligned} \tag{43}$$

Assume that the control $u_\theta(\cdot)$ is parameterized by $\theta \in \Theta$ in the parameter space Θ (e.g., $u_\theta(\cdot)$ is an MLP), then we could re-formulate the original problem as

$$\begin{aligned} &\inf_{\theta \in \Theta} J(\theta) = \Phi(x(t_1)) \\ &\text{s.t. } \dot{x}(t) = g(t, x(t), \theta(t)), \qquad t \in [t_0, t_1], \qquad x(t_0) = x_0. \end{aligned} \tag{44}$$

where $\theta(t) \equiv \theta$ is a constant and $g(t, x(t), \theta(t)) = f(t, x(t), u_\theta(t))$. By the adjoint sensitivities algorithm, the following gradient update rule with learning rate η should be applied

$$\theta_{\text{next}} \leftarrow \theta + \eta \nabla_\theta J(\theta) \tag{45}$$

where

$$\begin{aligned} &\nabla_\theta J(\theta) = \int_{t_0}^{t_1} p(t)^\top \nabla_\theta g(t, x(t), \theta) dt \\ \text{s.t.} \quad &\dot{x}(t) = \nabla_p H(t, x(t), p(t), \theta), \qquad x(0) = x_0 \\ &\dot{p}(t) = -\nabla_x H(t, x(t), p(t), \theta), \qquad p(T) = -\nabla_x \Phi(x(T)) \end{aligned} \tag{46}$$

with $H(t, x, p, \theta) = p^\top g(t, x, \theta)$. We may observe that this is equivalent to a variant of the method of successive approximation (MSA) using the steepest ascent.

By the Pontryagin's maximum principle, the optimal parameter θ^* satisfies

$$\begin{aligned}
&\dot{x}^*(t) = \nabla_p H(t, x^*(t), p^*(t), \theta^*), \qquad x^*(0) = x_0 \\
&\dot{p}^*(t) = -\nabla_x H(t, x^*(t), p^*(t), \theta^*), \qquad p^*(T) = -\nabla_x \Phi(x^*(T)) \\
&H(t, x^*(t), p^*(t), \theta^*) \geq H(t, x^*(t), p^*(t), \theta) \qquad (\text{for all } \theta \in \Theta \text{ and a.e. } t \in [0, T]).
\end{aligned} \tag{47}$$

Notice that the last condition necessarily requires

$$\int_{t_0}^{t_1} H(t, x^*(t), p^*(t), \theta^*)dt \geq \int_{t_0}^{t_1} H(t, x^*(t), p^*(t), \theta)dt \qquad (\text{for all } \theta \in \Theta). \tag{48}$$

Consequently, it makes sense to use the following gradient update rule with learning rate η

$$\begin{aligned}
&\theta_{\text{next}} \leftarrow \theta + \eta \nabla_\theta \int_{t_0}^{t_1} H(t, x(t), p(t), \theta)dt \\
\text{s.t.} \quad &\dot{x}(t) = \nabla_p H(t, x(t), p(t), \theta), \qquad x(0) = x_0 \\
&\dot{p}(t) = -\nabla_x H(t, x(t), p(t), \theta), \qquad p(T) = -\nabla_x \Phi(x(T))
\end{aligned} \tag{49}$$

in an iterative manner similar to the MSA.

Interchanging integration and differentiation, we have

$$\begin{aligned}
\nabla_\theta \int_{t_0}^{t_1} H(t, x(t), p(t), \theta)dt &= \int_{t_0}^{t_1} \nabla_\theta H(t, x(t), p(t), \theta)dt \\
&= \int_{t_0}^{t_1} \nabla_\theta (p(t)^\top g(t, x(t), \theta))dt \\
&= \int_{t_0}^{t_1} p(t)^\top \nabla_\theta g(t, x(t), \theta)dt
\end{aligned} \tag{50}$$

which is identical to the gradient of $J(\theta)$ in equation (46).

Remark. Alternatively, consider the MSA with the following gradient update rule

$$\theta_{\text{next}}^{(i)} \leftarrow \theta + (t_1 - t_0)\eta \cdot \nabla_\theta H(t^{(i)}, x(t^{(i)}), p(t^{(i)}), \theta) \tag{51}$$

at a finite number of timesteps ($i = 1, \ldots, n$). Since we want θ_{next} to be a constant, we may take the average of the different updated parameters $\theta_{\text{next}}^{(0)}, \ldots, \theta_{\text{next}}^{(n)}$ to obtain

$$\begin{aligned}
\theta_{\text{next}} &\leftarrow \frac{1}{n}\sum_{i=1}^{n} \theta_{\text{next}}^{(i)} \\
&= \theta + (t_1 - t_0)\eta \cdot \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta H(t^{(i)}, x(t^{(i)}), p(t^{(i)}), \theta) \\
&\approx \theta + \eta \nabla_\theta \int_{t_0}^{t_1} H(t, x(t), p(t), \theta) dt.
\end{aligned} \tag{52}$$

Remark. Furthermore, from the lecture note, let $\theta, \theta' \in \Theta$, suppose x_θ and p_θ are the state and costate trajectories corresponding to u_θ, define

$$S(\theta') := \int_{t_0}^{t_1} H(t, x_\theta(t), p_\theta(t), \theta') dt \tag{53}$$

for any $\theta' \in \Theta$, then the following bound holds under some technical assumptions

$$J(\theta') - J(\theta) \leq -(S(\theta') - S(\theta)) + C\|\theta - \theta'\|^2 \tag{54}$$

for a constant C, which again justifies maximizing $S(\cdot)$ via gradient ascent (potentially with tricks such as gradient-clipping).

Sanity Check for the RDE Solver

We first attempted to find a closed-form solution to this LQR problem when there is no resistance, i.e., $\alpha(t) \equiv 0$. From the linear ODE (24), we have

$$\frac{d}{dt}\begin{pmatrix} X(t) \\ Y(t) \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2\lambda} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} X(t) \\ Y(t) \end{pmatrix}, \qquad \begin{pmatrix} X(T) \\ Y(T) \end{pmatrix} = \begin{pmatrix} I \\ -2M \end{pmatrix}. \tag{55}$$

Notice that

$$\begin{pmatrix} \dot{Y}_{11}(t) \\ \dot{Y}_{12}(t) \\ \dot{Y}_{21}(t) \\ \dot{Y}_{22}(t) \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix} \begin{pmatrix} Y_{11}(t) \\ Y_{12}(t) \\ Y_{21}(t) \\ Y_{22}(t) \end{pmatrix}, \qquad \begin{pmatrix} Y_{11}(T) \\ Y_{12}(T) \\ Y_{21}(T) \\ Y_{22}(T) \end{pmatrix} = \begin{pmatrix} -2 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{56}$$

which is solved by

$$Y(t) = \begin{pmatrix} -2 & 0 \\ 2(t-T) & 0 \end{pmatrix} \tag{57}$$

Substitute back to (55), we have

$$\begin{pmatrix} \dot{X}_{11}(t) \\ \dot{X}_{12}(t) \\ \dot{X}_{21}(t) \\ \dot{X}_{22}(t) \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} X_{11}(t) \\ X_{12}(t) \\ X_{21}(t) \\ X_{22}(t) \end{pmatrix} + \frac{1}{\lambda} \begin{pmatrix} 0 \\ 0 \\ t-T \\ 0 \end{pmatrix}, \qquad \begin{pmatrix} X_{11}(T) \\ X_{12}(T) \\ X_{21}(T) \\ X_{22}(T) \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{58}$$

which is solved by

$$X(t) = \begin{pmatrix} \frac{6\lambda + (t-T)^3}{6\lambda} & t-T \\ \frac{(t-T)^2}{2\lambda} & 1 \end{pmatrix} \tag{59}$$

Therefore, we have

$$P(t) = -\frac{1}{2}Y(t)X^{-1}(t) = \frac{3\lambda}{3\lambda - (t-T)^3}\begin{pmatrix} 1 & T-t \\ T-t & (T-t)^2 \end{pmatrix}. \tag{60}$$

We may use the closed-form solution (60) as a sanity check for our numerical solver based on the RDE (20). For $\lambda = 1/3$, the result is shown in Figure 4, the RDE result is almost indistinguishable from the ground truth in equation (60).

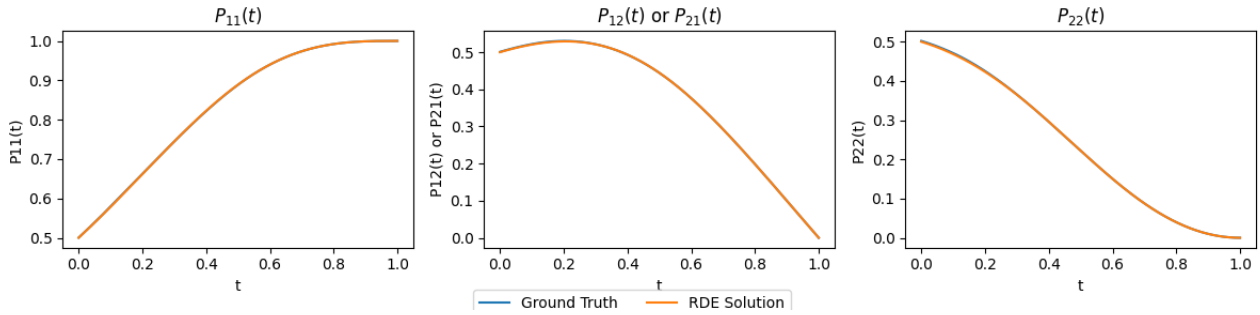

Figure 4: Sanity check with $\lambda = 1/3$.