Pontryagin Differentiable Programming: 
An End-to-End Learning and Control Framework

Wanxin Jin Zhaoran Wang Zhuoran Yang Shaoshuai Mou
Purdue University Northwestern University Princeton University Purdue University
{wanxinjin,zhaoranwang}@gmail.com zy6@princeton.edu mou@purdue.edu

Abstract

This paper develops a Pontryagin Differentiable Programming (PDP) methodology, which establishes a unified framework to solve a broad class of learning and control tasks. The PDP distinguishes from existing methods by two novel techniques: first, we differentiate through Pontryagin’s Maximum Principle, and this allows to obtain the analytical derivative of a trajectory with respect to tunable parameters within an optimal control system, enabling end-to-end learning of dynamics, policies, or/and control objective functions; and second, we propose an auxiliary control system in the backward pass of the PDP framework, and the output of this auxiliary control system is the analytical derivative of the original system’s trajectory with respect to the parameters, which can be iteratively solved using standard control tools. We investigate three learning modes of the PDP: inverse reinforcement learning, system identification, and control/planning. We demonstrate the capability of the PDP in each learning mode on different high-dimensional systems, including multi-link robot arm, 6-DoF maneuvering quadrotor, and 6-DoF rocket powered landing.

1 Introduction

Many learning tasks can find their counterpart problems in control fields. These tasks both seek to obtain unknown aspects of a decision-making system with different terminologies compared below.

Table 1: Topic connections between control and learning (details presented in Section 2)

<table>
<thead>
<tr>
<th>UNKNOWNS IN A SYSTEM</th>
<th>LEARNING METHODS</th>
<th>CONTROL METHODS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamics $x_{t+1} = f_{\theta}(x_t, u_t)$</td>
<td>Markov decision processes</td>
<td>System identification</td>
</tr>
<tr>
<td>Policy $u_t = \pi_{\theta}(t, x_t)$</td>
<td>Reinforcement learning (RL)</td>
<td>Optimal control (OC)</td>
</tr>
<tr>
<td>Control objective $J = \sum_t c_{\theta}(x_t, u_t)$</td>
<td>Inverse RL</td>
<td>Inverse OC</td>
</tr>
</tbody>
</table>

With the above connections, learning and control fields have begun to explore the complementary benefits of each other: control theory may provide abundant models and structures that allow for efficient or certificated algorithms for high-dimensional tasks, while learning enables to obtain these models from data, which are otherwise not readily attainable via classic control tools. Examples that enjoy both benefits include model-based RL [1, 2], where dynamics models are used for sample efficiency; and Koopman-operator control [3, 4], where via learning, nonlinear systems are lifted to a linear observable space to facilitate control design. Inspired by those, this paper aims to exploit the advantage of integrating learning and control and develop a unified framework that enables to solve a wide range of learning and control tasks, e.g., the challenging problems in Fig. 1.

2 Background and Related Work

Learning dynamics. This is usually referred to as system identification in control fields, which typically consider linear systems represented by transfer functions [5]. For nonlinear systems, the Koopman theory [6] provides a way to lift states to a (infinite-dimensional) linear observable space [3, 7]. In learning, dynamics is characterized by Markov decision processes and implemented using linear regression [8], observation-transition modeling [9], latent-space modeling [10], (deep) neural networks [11], Gaussian process [12], transition graphs [13], etc. Although off-the-shelf, most of these methods have to trade off between data efficiency and long-term prediction accuracy. To achieve both, physically-informed learning [14–17] injects physics laws into learning models, but they are limited to mechanical systems. Recently, a trend of work starts to use dynamical systems to explain (deep) neural networks, and some new algorithms [18–25] have been established.

This paper focuses on learning general dynamical models, encompassing either physical dynamics with unknown parameters or neural difference equations. The proposed learning framework is injected with inductive knowledge of optimal control theory to achieve efficiency and explainability.

Learning optimal policies. In learning fields, it relates to reinforcement learning (RL). Model-free RL provides a general-purpose framework to learn policies directly from interacting with environments [26–28], but usually suffers from significant data complexity. Model-based RL addresses this by first learning a dynamics model from experience and then integrating it to policy improvement [1, 12, 29–31]. The use of a model can assist to augment experience data [32, 33], perform back-propagation through time [12], or test policies before deployment. Model-based RL also faces some challenges that are not well-addressed. For example, how to efficiently leverage imperfect models [34], and how to maximize the joint benefit by combining policy learning and motion planning (trajectory optimization) [31, 35], where a policy has the advantage of execution coherence and fast deployment while the trajectory planning has the competence of adaption to unseen or future situations.

The counterpart topic in control is optimal control (OC), which is more concerned with characterizing optimal trajectories in presence of dynamics models. As in RL, the main strategy for OC is based on dynamic programming, and many valued-based methods are available, such as HJB [36], differential dynamical programming (DDP) [37] (by quadratizing dynamics and value function), and iterative linear quadratic regulator (iLQR) [38] (by linearizing dynamics and quadratizing value function). The second strategy to solve OC is based on the Pontryagin’s Maximum/Minimal Principle (PMP) [39]. Derived from calculus of variations, PMP can be thought of as optimizing directly over trajectories, thus avoiding solving for value functions. Popular methods in this vein include shooting methods [40] and collocation methods [41]. However, the OC methods based on PMP are essentially open loop control and thus susceptible to model errors or disturbances in deployment. To address these, model predictive control (MPC) [42] generates controls given the system current state by repeatedly solving an OC problem over a finite prediction horizon (only the first optimal input is executed), leading to a closed-loop control form. Although MPC has dominated across many industrial applications [43], developing fast MPC implementations is still an active research direction [44].

The proposed learning framework in this work has a special mode for model-based control tasks. The method can be viewed as a complement to classic open-loop OC methods, because, although derived from PMP (trajectory optimization), the method here is to learn a feedback/closed-loop control policy. Depending on the specific policy parameterization, the method here can also be used for motion planning. All these features will provide new perspectives for model-based RL or MPC control.

Learning control objective functions. In learning, this relates to inverse reinforcement learning (IRL), whose goal is to find a control objective function to explain the given optimal demonstrations. The unknown objective function is typically parameterized as a weighted sum of features [45–47]. Strategies to learn the unknown weights include feature matching [45] (matching the feature values between demonstrations and reproduced trajectories), maximum entropy [46] (finding a trajectory distribution of maximum entropy subject to empirical feature values), and maximum margin [47] (maximizing the margin of objective values between demonstrations and reproduced trajectories). The learning update in the above IRL methods is preformed on a selected feature space by taking advantage of linearity of feature weights, and thus cannot be directly applied to learning objective functions that are nonlinear in parameters. The counterpart topic in the control field is inverse optimal control (IOC) [48–51]. With knowledge of dynamics, IOC focuses on more efficient learning paradigms. For example, by directly minimizing the violation of optimality conditions by the observed demonstration data, [48, 50–52] directly compute feature weights without repetitively solving the OC problems.
Despite the efficiency, minimizing optimality violation does not directly assure the closeness between the final reproduced trajectory and demonstrations or the closeness of their objective values.

Fundamentally different from existing IRL/IOC methods, this paper will develop a new framework that enables to learn complex control objective functions, e.g., neural objective functions, by directly minimizing the loss (e.g., the distance) between the reproduced trajectory and demonstrations.

**A unified perspective on learning dynamics/policy/control objective functions.** Consider a general decision-making system, which typically consists of aspects of dynamics, control policy, and control objective function. In a unified perspective, learning dynamics, policies, or control objective functions can be viewed as instantiations of the same learning problem but with (i) unknown parameters appearing in the system’s different aspects and (ii) the different losses. For example, in learning dynamics, a differential/difference equation is parameterized and the loss function can be defined as the prediction error between the equation’s output and target data; in learning policies, the unknown parameters are in a feedback policy and the loss function is just the control objective function; and in learning control objective functions, the control objective function is parameterized and the loss function can be the discrepancy between the reproduced trajectory and the observed demonstrations.

**Claim of contributions.** Motivated by the above, this paper develops a unified learning framework, named as PDP, that is flexible enough to be customized for different learning and control tasks and capable enough to efficiently solve high-dimensional and continuous-space problems. The proposed PDP framework borrows the idea of ‘end-to-end’ learning [53] and chooses to optimize a loss function directly with respect to the tunable parameters in the aspect(s) of a decision-making system, such as the dynamics, policy, or/and control objective function. The key contribution of the PDP is that we inject the optimal control theory as an inductive bias into the learning process to expedite the learning efficiency and explainability. Specifically, the PDP framework centers around the system’s trajectory and differentiates through PMP, and this allows us to obtain the analytical derivative of the trajectory with respect to the tunable parameters, a key quantity for end-to-end learning of (neural) dynamics, (neural) policies, and (neural) control objective functions. Furthermore, we introduce an auxiliary control system in the back pass of the PDP framework, and its output trajectory is exactly the derivative of the trajectory with respect to the parameters, which can be iteratively solved using standard control tools. In control fields, to our best knowledge, this is the first work to propose the technique of the differential PMP, and more importantly, we show that the differential PMP can be easily obtained using the introduced auxiliary control system.

### 3 Problem Formulation

We begin with formulating a base problem and then discuss how to accommodate the base problem to specific applications. Consider a class of optimal control systems $\Sigma(\theta)$, which is parameterized by a tunable $\theta \in \mathbb{R}^r$ in both dynamics and control (cost) objective function:

\[
\Sigma(\theta): \begin{align*}
\text{dynamics:} & \quad x_{t+1} = f(x_t, u_t, \theta) \quad \text{with given } x_0, \\
\text{control objective:} & \quad J(\theta) = \sum_{t=0}^{T-1} c_t(x_t, u_t, \theta) + h(x_T, \theta). \tag{1}
\end{align*}
\]

Here, $x_t \in \mathbb{R}^n$ is the system state; $u_t \in \mathbb{R}^m$ is the control input; $f : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^r \mapsto \mathbb{R}^n$ is the dynamics model, which is assumed to be twice-differentiable; $t = 0, 1, \ldots, T$ is the time step with $T$ being the time horizon; and $J(\theta)$ is the control objective function with $c_t : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^r \mapsto \mathbb{R}$ and $h : \mathbb{R}^n \times \mathbb{R}^r \mapsto \mathbb{R}$ denoting the stage/running and final costs, respectively, both of which are twice-differentiable. For a choice of $\theta$, $\Sigma(\theta)$ will produce a trajectory of state-inputs:

\[
\xi_\theta = \{x_{0:T}, u_{0:T-1}\} \in \arg \min_{\{x_{0:T}, u_{0:T-1}\}} J(\theta) \quad \text{subject to} \quad x_{t+1} = f(x_t, u_t, \theta) \text{ for all } t \text{ given } x_0, \tag{2}
\]

that is, $\xi_\theta$ optimizes $J(\theta)$ subject to the dynamics constraint $f(\theta)$. For many applications (we will show next), one evaluates the above $\xi_\theta$ using a scalar-valued differentiable loss $L(\xi_\theta, \theta)$. Then, the **problem of interest** is to tune the parameter $\theta$, such that $\xi_\theta$ has the minimal loss:

\[
\min_{\theta} L(\xi_\theta, \theta) \quad \text{subject to} \quad \xi_\theta \text{ is in (2)}. \tag{3}
\]

Under the above base formulation, for a specific learning or control task, one only needs to accordingly change precise details of $\Sigma(\theta)$ and define a specific loss function $L(\xi_\theta, \theta)$, as we discuss below.
IRL/IOC Mode. Suppose that we are given optimal demonstrations \( \xi^d = \{ x^d_0, u^d_0 \} \) of an expert optimal control system. We seek to learn the expert’s dynamics and control objective function from \( \xi^d \). To this end, we use \( \Sigma(\theta) \) in (1) to represent the expert, and define the loss in (3) as

\[
L(\xi_\theta, \theta) = l(\xi_\theta, \xi^d),
\]

where \( l \) is a scalar function that penalizes the inconsistency of \( \xi_\theta \) with \( \xi^d \), e.g., \( l(\xi_\theta, \xi^d) = \| \xi_\theta - \xi^d \|^2 \).

By solving (3) with (4), we can obtain a \( \Sigma(\theta^*) \) whose trajectory is consistent with the observed demonstrations. It should be noted that even if the demonstrations \( \xi^d \) significantly deviate from the optimal ones, the above formulation still finds the ‘best’ control objective function (and dynamics) within the parameterized set \( \Sigma(\theta) \) such that its reproduced \( \xi_\theta \) in (2) has the minimal distance to \( \xi^d \).

SysID Mode. Suppose that we are given data \( \xi^o = \{ x^o_0, u^o_0 \} \) collected from, say, a physical system (here, unlike \( \xi^d, \xi^o \) is not necessarily optimal), and we wish to identify the system’s dynamics. Here, \( u_{0:T-1} \) are usually externally supplied to ensure the physical system is of persistent excitation [54]. In order for \( \Sigma(\theta) \) in (1) to only represent dynamics (as we do not care about its internal control law), we set \( J(\theta) = 0 \). Then, \( \xi_\theta \) in (2) accepts any \( u^o_{0:T-1} = u_{0:T-1} \) as it always optimizes \( J(\theta) = 0 \). In other words, by setting \( J(\theta) = 0, \Sigma(\theta) \) in (1) now only represent a class of dynamics models:

\[
\Sigma(\theta) : \text{dynamics: } x_{t+1} = f(x_t, u_t, \theta) \quad \text{with } x_0 \text{ and } u^o_{0:T-1} = u_{0:T-1}. \tag{5}
\]

Now, \( \Sigma(\theta) \) produces \( \xi_\theta = \{ x^\theta_0, u^\theta_0 \} \) subject to (5). To use (3) for identifying \( \theta \), we define

\[
L(\xi_\theta, \theta) = l(\xi_\theta, \xi^o),
\]

where \( l \) is to quantify the prediction error between \( \xi^o \) and \( \xi_\theta \) under the same inputs \( u_{0:T-1} \).

Control/Planning Mode. Consider a system with its dynamics learned in the above SysID. We want to obtain a feedback controller or trajectory such that the system achieves a performance of minimizing a given cost function. To that end, we specialize \( \Sigma(\theta) \) in (1) as follows: first, set \( f \) as the learned dynamics and \( J(\theta) = 0 \); and second, through a close-loop link, we connect the input \( u_t \) and state \( x_t \) via a parameterized policy block \( u_t = u(t, x_t, \theta) \) (reminder: unlike SysID Mode with \( u_t \) supplied externally, the inputs here are from a policy via a feedback). \( \Sigma(\theta) \) now becomes

\[
\Sigma(\theta) : \text{dynamics: } x_{t+1} = f(x_t, u_t) \quad \text{with } x_0, \quad \text{control policy: } u_t = u(t, x_t, \theta). \tag{7}
\]

Now, \( \Sigma(\theta) \) produces a trajectory \( \xi_\theta = \{ x^\theta_0, u^\theta_0 \} \) subject to (7). We set the loss in (3) as

\[
L(\xi_\theta, \theta) = \sum_{t=0}^{T-1} l(x^\theta_t, u^\theta_t) + l_f(x^\theta_T), \tag{8}
\]

where \( l \) and \( l_f \) are the stage and final costs, respectively. Then, (3) is an optimal control or planning problem: if \( u_t = u(t, x_t, \theta) \) (i.e., feedback policy explicitly depends on \( x_t \)), (3) is a close-loop optimal control problem; otherwise if \( u_t = u(t, \theta) \) (e.g., polynomial parameterization), (3) is an open-loop motion planning problem. This mode can also be used as a component to solve (1) in IRL/IOC Mode.

4 An End-to-End Learning Framework

To solve the generic problem in (3), the idea of end-to-end learning [53] seeks to optimize the loss \( L(\xi_\theta, \theta) \) directly with respect to the tunable parameter \( \theta \), by applying the gradient descent

\[
\theta_{k+1} = \theta_k - \eta_k \frac{\partial L}{\partial \theta_k} \bigg|_{\theta_k} \quad \text{with } \frac{\partial L}{\partial \theta_k} \bigg|_{\theta_k} = \frac{\partial L}{\partial \xi_\theta} \bigg|_{\xi_\theta} \frac{\partial \xi_\theta}{\partial \theta} \bigg|_{\theta_k}. \tag{9}
\]

Here, \( k = 0, 1, \cdots \) is the iteration index; \( \frac{\partial L}{\partial \theta} \bigg|_{\theta_k} \) is the gradient of the loss with respect to \( \theta \) evaluated at \( \theta_k \); and \( \eta_k \) is the learning rate. From (9), we can draw a learning architecture in Fig. 2. Each update of \( \theta \) consists of a forward pass, where at \( \xi_\theta \), the corresponding trajectory \( \xi_\theta \) is solved from \( \Sigma(\theta_k) \) and the loss is computed, and a backward pass, where \( \frac{\partial L}{\partial \xi_\theta} \) and \( \frac{\partial L}{\partial \theta} \) are computed.

In the forward pass, \( \xi_\theta \) is obtained by solving an optimal control problem in \( \Sigma(\theta) \) using any available OC methods, such as iLQR or Control/Planning Mode, (note that in SysID or Control/Planning modes, it is reduced to integrating difference equations (5) or (7)). In backward pass, \( \frac{\partial L}{\partial \xi_\theta} \) and \( \frac{\partial L}{\partial \theta} \) are easily obtained from the loss function \( L(\xi_\theta, \theta) \). The main challenge, however, is to solve \( \frac{\partial L}{\partial \theta} \), i.e., the derivative of a trajectory with respect to the parameters in the system. Next, we will analytically solve \( \frac{\partial L}{\partial \theta} \) by proposing two techniques: differential PMP and auxiliary control system.
5 Key Contributions: Differential PMP & Auxiliary Control System

We first recall the discrete-time Pontryagin’s Maximum/Minimum Principle (PMP) [39] (a derivation of discrete-time PMP is given in Appendix C). For the optimal control system \( \Sigma(\theta) \) in (1) with a fixed \( \theta \), PMP describes a set of optimality conditions which the trajectory \( \xi_\theta = \{x^\theta_{0:T}, u^\theta_{0:T-1}\} \) in (2) must satisfy. To introduce these conditions, we first define the following Hamiltonian,
\[
H_t = c_t(x_t, u_t; \theta) + f(x_t, u_t; \theta)'\lambda_{t+1},
\]
where \( \lambda_t \in \mathbb{R}^n (t = 1, 2, \cdots, T) \) is called the costate variable, which can be also thought of as the Lagrange multipliers for the dynamics constraints. According to PMP, there exists a sequence of costates \( \lambda^\theta_{0:T} \), which together with the optimal trajectory \( \xi_\theta = \{x^\theta_{0:T}, u^\theta_{0:T-1}\} \) satisfy

- dynamics equation: \( x^\theta_{t+1} = \frac{\partial H_t}{\partial \lambda^{\theta}_{t+1}} = f(x^\theta_t, u^\theta_t; \theta), \)
- costate equation: \( \lambda^\theta_t = \frac{\partial H_t}{\partial x^\theta_t} = \frac{\partial c_t}{\partial x^\theta_t} + \frac{\partial f}{\partial x^\theta_t} \lambda^\theta_{t+1}, \)
- input equation: \( 0 = \frac{\partial H_t}{\partial u^\theta_t} = \frac{\partial c_t}{\partial u^\theta_t} + \frac{\partial f}{\partial u^\theta_t} \lambda^\theta_{t+1}, \)
- boundary conditions: \( \lambda^\theta_0 = \frac{\partial h}{\partial x^\theta_0}, \quad x_0 = x_0. \)

For notation simplicity, \( \frac{\partial g}{\partial x_t} \) means the derivative of function \( g(x) \) with respect to \( x \) evaluated at \( x_t \).

5.1 Differential PMP

To begin, recall that our goal (in Section 4) is to obtain \( \frac{\partial \xi_\theta}{\partial \theta} \), that is,
\[
\frac{\partial \xi_\theta}{\partial \theta} = \left\{ \frac{\partial x^\theta_{0:T}}{\partial \theta}, \frac{\partial u^\theta_{0:T-1}}{\partial \theta} \right\}. \tag{12}\]

To this end, we are motivated to differentiate the PMP conditions in (11) on both sides with respect to \( \theta \). This leads to the following differential PMP:

- differential dynamics equation: \( \frac{\partial x^\theta_{t+1}}{\partial \theta} = F_t \frac{\partial x^\theta_t}{\partial \theta} + G_t \frac{\partial u^\theta_t}{\partial \theta} + E_t, \)
- differential costate equation: \( \frac{\partial \lambda^\theta_t}{\partial \theta} = H^{tx}_t \frac{\partial x^\theta_t}{\partial \theta} + H^{tu}_t \frac{\partial u^\theta_t}{\partial \theta} + F_t' \frac{\partial \lambda^\theta_{t+1}}{\partial \theta} + H^{\tau c}_t, \)
- differential input equation: \( 0 = H^{du}_t \frac{\partial x^\theta_t}{\partial \theta} + H^{du u}_t \frac{\partial u^\theta_t}{\partial \theta} + G_t' \frac{\partial \lambda^\theta_{t+1}}{\partial \theta} + H^{\tau uc}_t, \)
- differential boundary conditions: \( \frac{\partial \lambda^\theta_0}{\partial \theta} = H^{\tau x}_0 \frac{\partial x^\theta_0}{\partial \theta} + H^{\tau uc}_x, \quad \frac{\partial x^\theta_0}{\partial \theta} = \frac{\partial x_0}{\partial \theta} = 0. \)

Here, to simplify notations and distinguish knowns and unknowns, the coefficient matrices in the above differential PMP (13) are defined as follows:

\[
F_t = \frac{\partial f}{\partial x^\theta_t}, \quad G_t = \frac{\partial f}{\partial u^\theta_t}, \quad H^{tx}_t = \frac{\partial^2 f}{\partial x^\theta_t \partial x^\theta_t}, \quad H^{tu}_t = \frac{\partial^2 f}{\partial x^\theta_t \partial u^\theta_t}, \quad H^{\tau c}_t = \frac{\partial^2 f}{\partial u^\theta_t \partial \lambda^\theta_{t+1}}, \quad H^{\tau uc}_t = \frac{\partial^2 f}{\partial u^\theta_t \partial u^\theta_t}, \tag{14a}\]

\[
E_t = \frac{\partial f}{\partial \theta}, \quad H^{\tau x}_0 = \frac{\partial^2 f}{\partial x^\theta_0 \partial x^\theta_0}, \quad H^{\tau uc}_x = \frac{\partial^2 f}{\partial u^\theta_t \partial x^\theta_T}, \quad H^{\tau uc}_\theta = \frac{\partial^2 f}{\partial x^\theta_T \partial x^\theta_T}, \quad H^{\tau uc}_\theta = \frac{\partial^2 f}{\partial u^\theta_t \partial x^\theta_T}, \tag{14b}\]

where we use \( \frac{\partial^2 g}{\partial x_i \partial x_j} \) to denote the second-order derivative of a function \( g(x, u) \) evaluated at \( (x_i, u_i) \).

Since the trajectory \( \xi_\theta = \{x^\theta_{0:T}, u^\theta_{0:T-1}\} \) is obtained in the forward pass (recall Fig. 2), all matrices
in (14) are thus known (note that the computation of these matrices also requires $X^T_0$. This can be obtained by iteratively solving (11b) and (11d) given $\xi_0$). From the differential PMP in (13), we note that to obtain $\frac{\partial \xi_0}{\partial \theta}$ in (12), it is sufficient to compute the unknowns \(\left\{\frac{\partial x^0_0}{\partial \theta}, \frac{\partial x^0_{T-1}}{\partial \theta}, \frac{\partial \lambda^0_T}{\partial \theta}\right\}\) in (13). Next we will show how these unknowns are elegantly solved by introducing a new system.

5.2 Auxiliary Control System

One important observation to the differential PMP in (13) is that it shares a similar structure to the original PMP in (11); so it can be viewed as a new set of PMP equations corresponding to an ‘oracle control optimal system’ whose the ‘optimal trajectory’ is exactly (12). This motivates us to ‘unearth’ this oracle optimal control system, because by doing so, (12) can be obtained from this oracle system by an OC solver. To this end, we first define the new ‘state’ and ‘control’ (matrix) variables:

\[
X_t = \frac{\partial x_t}{\partial \theta} \in \mathbb{R}^{n \times r}, \quad U_t = \frac{\partial u_t}{\partial \theta} \in \mathbb{R}^{m \times r},
\]

respectively. Then, we ‘artificially’ define the following auxiliary control system $\Sigma(\xi_\theta)$:

\[
\Sigma(\xi_\theta) : \quad \text{dynamics:} \quad X_{t+1} = F_t X_t + G_t U_t + E_t \quad \text{with} \quad X_0 = 0,
\]

control objective: \[
\bar{J} = \text{Tr} \sum_{t=0}^{T-1} \left( \frac{1}{2} X_t' H_t^{xx} X_t + \frac{1}{2} U_t' H_t^{uu} U_t + \frac{1}{2} \partial_x^0 X_t' H_t^{xx} \partial_x^0 + \frac{1}{2} \partial_u^0 U_t' H_t^{uu} \partial_u^0 \right)
\]

\[
+ \text{Tr} \left( \frac{1}{2} X_T' H_T^{xx} U_T + \frac{1}{2} \partial_x^0 X_T' H_T^{xx} \partial_u^0 \right).
\]

Here, $X_0 = \frac{\partial x_0}{\partial \theta} = 0$ because $x_0$ in (1) is given; $\bar{J}$ is the defined control objective function which needs to be optimized in the auxiliary control system; and $\text{Tr}$ denotes matrix trace. Before presenting the key results, we make some comments on the above auxiliary control system $\Sigma(\xi_\theta)$. First, its state and control variables are both matrix variables defined in (15). Second, its dynamics is linear and control objective function $\bar{J}$ is quadratic, for which the coefficient matrices are given in (14). Third, its dynamics and objective function are determined by the trajectory $\xi_\theta$ of the system $\Sigma(\theta)$ in forward pass, and this is why we denote it as $\Sigma(\xi_\theta)$. Finally, we have the following important result.

**Lemma 5.1.** Let $\{X^0_0, U^0_0\}$ be a stationary solution to the auxiliary control system $\Sigma(\xi_\theta)$ in (16). Then, $\{X^\theta_0, U^\theta_0\}$ satisfies Pontryagin’s Maximum Principle of $\Sigma(\xi_\theta)$, which is (13), and

\[
\{X^\theta_0, U^\theta_0\} = \left\{\frac{\partial x^0_0}{\partial \theta}, \frac{\partial u^0_0}{\partial \theta}\right\} = \frac{\partial \xi_\theta}{\partial \theta}.
\]

A proof of Lemma 5.1 is in Appendix A. Lemma 5.1 states two assertions. First, the PMP condition for the auxiliary control system $\Sigma(\xi_\theta)$ is exactly the differential PMP in (13) for the original system $\Sigma(\theta)$; and second, importantly, the trajectory $\{X^\theta_0, U^\theta_0\}$ produced by the auxiliary control system $\Sigma(\xi_\theta)$ is exactly the derivative of trajectory of the original system $\Sigma(\theta)$ with respect to the parameter $\theta$. Based on Lemma 5.1, we can obtain $\frac{\partial \xi_\theta}{\partial \theta}$ from $\Sigma(\xi_\theta)$ efficiently by the lemma below.

**Lemma 5.2.** If $H_t^{uu}$ in (16) is invertible for all $t = 0, 1, \ldots, T-1$, define the following recursions

\[
P_t = Q_t + A_t' (I + P_{t+1} R_t)^{-1} P_{t+1} A_t,
\]

\[
W_t = A_t' (I + P_{t+1} R_t)^{-1} (W_{t+1} + P_{t+1} M_t) + N_t,
\]

with $P_T = H_T^{xx}$ and $W_T = H_T^{xx}$. Here, $I$ is identity matrix, $A_t = F_t - G_t (H_t^{uu})^{-1} H_t^{xx}, R_t = G_t (H_t^{uu})^{-1} G_t', M_t = E_t - G_t (H_t^{uu})^{-1} H_t^{ue}, Q_t = H_t^{ux} - H_t^{xx} (H_t^{uu})^{-1} H_t^{ux}, N_t = H_t^{xe} - H_t^{xx} (H_t^{uu})^{-1} H_t^{ue}$ are all known given (14). Then, the stationary solution $\{X^\theta_0, U^\theta_0\}$ in (17) can be obtained by iteratively solving the following equations from $t = 0$ to $T - 1$ with $X^\theta_0 = X_0 = 0$:

\[
U^\theta_t = -(H_t^{uu})^{-1} \left( H_t^{xx} X^\theta_t + H_t^{xe} + G_t' (I + P_{t+1} R_t)^{-1} \left( P_{t+1} A_t X^\theta_{t+1} + P_{t+1} M_t + W_{t+1} \right) \right),
\]

\[
X^\theta_{t+1} = F_t X^\theta_t + G_t U^\theta_t + E_t.
\]

A proof of Lemma 5.2 is in Appendix B. Lemma 5.2 states that the trajectory of the above auxiliary control system $\Sigma(\xi_\theta)$ can be obtained by two steps: first, iteratively solve (18) backward in time to
We investigate three learning modes of PDP, as described in Section 3. For each mode, we demonstrate the capability in four environments listed in Table 2, and a baseline and a state-of-the-art method are compared. Both PDP and environment codes are available at https://github.com/wanxinjin.

Table 2: Experimental environments (results for 6-DoF rocket landing is in Appendix I)

<table>
<thead>
<tr>
<th>Systems</th>
<th>Dynamics parameter $\theta_{dyn}$</th>
<th>Control objective parameter $\theta_{obj}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cartpole</td>
<td>cart mass, pole mass and length</td>
<td>$c(x, u) =</td>
</tr>
<tr>
<td>Two-link robot arm</td>
<td>length and mass for each link</td>
<td>$h(x, u) =</td>
</tr>
<tr>
<td>6-DoF quadrotor maneuvering</td>
<td>mass, wing length, inertia matrix</td>
<td></td>
</tr>
<tr>
<td>6-DoF rocket powered landing</td>
<td>mass, rocket length, inertia matrix</td>
<td></td>
</tr>
</tbody>
</table>

We fix the unit weight to $||u||^2$, because estimating all weights will incur ambiguity [48]; $x_g$ is the goal state.

IRL/IoC Mode. The parameterized $\Sigma(\theta)$ is in (1) and the loss in (4). In the forward pass of PDP, $\xi_0$ is solved from $\Sigma(\theta)$ by any OC solver. In the backward pass, $\frac{\partial \xi}{\partial \theta}$ is computed from the auxiliary control system $\Sigma(\xi_0)$ in (16) using Algorithm 1. The full algorithm is in Appendix D.

Experiment: imitation learning. We use IRL/IoC Mode to solve imitation learning in environments in Table 2. The true dynamics is parameterized, and control objective is parameterized as a weighted distance to the goal, $\theta = (\theta_{dyn}, \theta_{obj})$. Set imitation loss $L(\xi_0, \theta) = ||\xi_0 - \xi_0||^2$. Two other methods are compared: (i) neural policy cloning, and (ii) inverse KKT [52]. We set learning rate $\eta = 10^{-4}$ and run five trials given random initial $\theta_0$. The results in Fig. 3a-3c show that PDP significantly outperforms the policy cloning and inverse-KKT for a much lower training loss and faster convergence. In Fig. 3d, we apply the PDP to learn a neural control objective function for the robot arm using the same demonstration data in Fig. 3b, and we also compare with the GAIL [56]. Results in Fig. 3d show that the PDP successfully learns a neural objective function and the imitation loss of PDP is much lower than that of GAIL. It should note that because the demonstrations are not strictly realizable (optimal) under the parameterized neural objective function, the final loss for the PDP is strictly small but not zero. This indicates that given sub-optimal demonstrations, PDP can still find the ‘best’ control objective function within the function set $J(\theta)$ such that its reproduced $\xi_0$ has the minimal distance to the demonstrations. Please refer to Appendix E.2 for more experiment details and additional validations.

Figure 3: (a-c) imitation loss v.s. iteration, (d) PDP learns a neural objective function and comparison.

SysID Mode. In this mode, $\Sigma(\theta)$ is (5) and loss is (6). PDP is greatly simplified: in forward pass, $\xi_0$ is solved by integrating the difference equation (5). In the backward pass, $\Sigma(\xi_0)$ is reduced to

$$\Sigma(\xi_0) : \text{dynamics: } X_{t+1}\theta = F_tX_t\theta + E_t \text{ with } X_0 = 0.$$ (20)
This is because $\Sigma(\theta)$ in (5) results from letting $J(\theta) = 0$, (13b-13d) and $\mathcal{J}$ in (16) are then trivialized, and due to $u_{0:T-1}$ given, $U^T_0 = 0$ in (13a). The algorithm is in Appendix D.

**Experiment: system identification.** We use the SysID Mode to identify the dynamics parameter $\theta_{dyn}$ for the systems in Table 2. Set the SysID loss $L(\xi_{\theta}, \theta) = \| \xi_{\theta} - \xi_{\theta} \|^2$. Two other methods are compared: (i) learning a neural network (NN) dynamics model, and (ii) DMDc [57]. For all methods, we set learning rate $\eta = 10^{-4}$, and run five trials with random $\theta_0$. The results are in Fig. 4. Fig. 4a-4c show an obvious advantage of PDP over the NN baseline and DMDc in terms of lower training loss and faster convergence speed. In Fig. 4d, we compare PDP and Adam [58] (here both with $\eta = 10^{-4}$) for training the same neural dynamics model for the robot arm. The results again show that PDP outperforms Adam for faster learning speed and lower training loss. Such advantages are due to that PDP has injected an inductive bias of optimal control into learning, making it more efficient for handling dynamical systems. More experiments and validations are in Appendix E.3.

**Control/Planning Mode.** The parameterized system $\Sigma(\theta)$ is (7) and loss is (8). PDP for this mode is also simplified. In forward pass, $\xi_{\theta}$ is solved by integrating a (controlled) difference equation (7). In backward pass, $\mathcal{J}$ in the auxiliary control system (16) is trivialized because we have considered $J(\theta) = 0$ in (7). Since the control is now given by $u_t = u(t, x_t, \theta)$, $U^\theta_t$ is obtained by differentiating the policy on both side with respect to $\theta$, that is, $U^\theta_t = U^\theta_t X^\theta_t + U^\theta_t$ with $U^\theta_t = \frac{\partial u_t}{\partial x_t}$ and $U^\theta_t = \frac{\partial u_t}{\partial \theta}$. Thus, $\Sigma(\xi_{\theta})$: dynamics: $X_{t+1} = F_t X_t + G_t U^\theta_t$ with $X_0 = 0$,
control policy: $U^\theta_t = U^\theta_t X^\theta_t + U^\theta_t$.

Integrating (21) from $t = 0$ to $T$ leads to $\{X_{0:T}, U_{0:T-1}\} = \frac{\partial x}{\partial T}$. The algorithm is in Appendix D.

**Experiment: control and planning.** Based on identified dynamics, we learn policies of each system to optimize a control objective with given $\theta_{obj}$. We set loss (8) as the control objective (below called control loss). To parameterize policy (7), we use a Lagrange polynomial of degree $N$ (for planning) or neural network (for feedback control). iLQR [38] and guided policy search (GPS) [59] are compared. We set learning rate $\eta = 10^{-4}$ or $10^{-6}$ and run five trials for each system. Fig. 5a-5b are learning neural network feedback policies for the cart-pole and robot arm, respectively. The results show that PDP outperforms GPS for having lower control loss. Fig. 5c is motion planning for quadrotor using a neural network (for feedback control). iLQR [38] and guided policy search (GPS) [59] are compared. We set loss (8) as the control objective (below called control loss). To parameterize policy (7), we use a Lagrange polynomial of degree $N$ (for planning) or neural network (for feedback control). iLQR [38] and guided policy search (GPS) [59] are compared. We set learning rate $\eta = 10^{-4}$ or $10^{-6}$ and run five trials for each system. Fig. 5a-5b are learning neural network feedback policies for the cart-pole and robot arm, respectively. The results show that PDP outperforms GPS for having lower control loss. Fig. 5c is motion planning for quadrotor using a neural network (for feedback control). iLQR [38] and guided policy search (GPS) [59] are compared. We set learning rate $\eta = 10^{-4}$ or $10^{-6}$ and run five trials for each system. Fig. 5a-5b are learning neural network feedback policies for the cart-pole and robot arm, respectively. The results show that PDP outperforms GPS for having lower control loss. Fig. 5c is motion planning for quadrotor using a neural network (for feedback control). iLQR [38] and guided policy search (GPS) [59] are compared. We set learning rate $\eta = 10^{-4}$ or $10^{-6}$ and run five trials for each system. Fig. 5a-5b are learning neural network feedback policies for the cart-pole and robot arm, respectively. The results show that PDP outperforms GPS for having lower control loss. Fig. 5c is motion planning for quadrotor using a neural network (for feedback control). iLQR [38] and guided policy search (GPS) [59] are compared. We set learning rate $\eta = 10^{-4}$ or $10^{-6}$ and run five trials for each system. Fig. 5a-5b are learning neural network feedback policies for the cart-pole and robot arm, respectively. The results show that PDP outperforms GPS for having lower control loss. Fig. 5c is motion planning for quadrotor using a neural network (for feedback control). iLQR [38] and guided policy search (GPS) [59] are compared. We set learning rate $\eta = 10^{-4}$ or $10^{-6}$ and run five trials for each system. Fig. 5a-5b are learning neural network feedback policies for the cart-pole and robot arm, respectively. The results show that PDP outperforms GPS for having lower control loss.
7 Discussion

The related end-to-end learning frameworks. Two lines of recent work are related to PDP. One is the recent work [60–64] that seeks to replace a layer within a deep neural network by an argmin layer, in order to capture the information flow characterized by a solution of an optimization. Similar to PDP, these methods differentiate the argmin layer through KKT conditions. They mainly focus on static optimization problems, which can not directly be applied to dynamical systems. The second line is the recent RL development [65–68] that embeds an implicit planner within a policy. The idea is analogous to MPC, because using a predictive OC system (i.e., embedded planner) to generate controls leads to better adaption to unseen situations. The key problem in these methods is to learn a planner (i.e., OC system), which is similar to our formulation. [65, 66] learn a path-integral OC system [69], which is a special class of OC systems. [68] learns an OC system in a latent space. However, all these methods adopt the ‘unrolling’ strategy to facilitate differentiation. Specifically, they treat the forward pass of solving an OC problem as an ‘unrolled’ computational graph of multiple steps of applying gradient descent, because by this computational graph, automatic differentiation tool [70] can be immediately applied. The drawbacks of this ‘unrolling’ strategy are apparent: (i) they need to store all intermediate results over the entire computational graph, thus are memory-expensive; and (ii) the accuracy of gradient depends on the length of the ‘unrolled’ graph, thus facing trade-off between complexity and accuracy. To address these, [67] develops a differentiable MPC framework, where in forward pass, a LQR approximation of the OC system is obtained, and in backward pass, the gradient is solved by differentiating such LQR approximation. Although promising, this framework has one main weakness: differentiating LQR requires to solve a large linear equation, which involves the inverse of a matrix of size $(2n+m)T \times (2n+m)T$, thus can incur huge cost when handling systems of longer horizons $T$. Detailed descriptions for all these methods is in Appendix F.

Compared to [35, 65–68], the efficiency of PDP stems from the following novel aspects. First, in forward pass, without needing an unrolled computational graph, PDP only computes and stores the resulting trajectory of the OC system, $\xi_\theta$, (does not care about how $\xi_\theta$ is solved). Second, without obtaining intermediate (LQR) approximations, PDP differentiates through PMP of the OC system to directly obtain the exact analytical gradient. Third, in the backward pass, unlike differentiable MPC which costs at least a complexity of $O\left((m+2n)^2T^2\right)$ to differentiate a LQR approximation, PDP explicitly solves $\frac{\partial \xi_\theta}{\partial \theta}$ by an auxiliary control system, where thanks to the recursion structure, the memory and computation complexity of PDP is only $O\left((m+2n)T\right)$. In Fig. 6, we have compared the running time of PDP with that of differentiable MPC. The results show PDP is 1000x faster than differentiable MPC. Due to space constraint, we put the detailed complexity analysis of PDP in Appendix G.

Convergence and limitation of PDP. Since all gradient quantities in PDP are analytical and exact, and the development of PDP does not involves any second-order derivative of functions or models, PDP essentially is a first-order gradient-descent framework to solve non-convex bi-level optimization. Therefore, in general, PDP can only achieve local minima. As explored by [71], if we pose further assumptions such as convexity and smoothness on all functions (dynamics, policy, loss, and control objective function), the global convergence of the bi-level programming could be established. But we do think these conditions are too restrictive for dynamical control systems. As a direction of future work, we will investigate the mild conditions for good convergence by taking advantage of control theory, e.g., Lyapunov theory. Due to space constraint, limitation of PDP is detailed in Appendix H.

8 Conclusions

This paper proposes a Pontryagin differentiable programming (PDP) methodology to establish an end-to-end learning framework for solving a range of learning and control tasks. The key contribution in PDP is that we incorporate the knowledge of optimal control theory as an inductive bias into the learning framework. Such combination enables PDP to achieve higher efficiency and capability than existing learning and control methods in solving many tasks including inverse reinforcement learning, system identification, and control/planning. We envision the proposed PDP could benefit to both learning and control fields for solving many high-dimensional continuous-space problems.
**Broader Impact**

This work is expected to have the impacts on both learning and control fields.

- To the learning field, this work connects some fundamental topics in machine learning to their counterparts in the control field, and unifies some concepts from reinforcement learning, backpropagation/deep learning, and control theory in one generic learning framework. The contribution of this framework is a deep integration of optimal control theory into end-to-end learning process, leading to an optimal-control-informed end-to-end learning framework that is flexible enough to solve a broad range of learning and control tasks and efficient enough to handle high-dimensional and continuous-space problems. In a broad perspective, we hope that this paper could motivate more future work that integrates the benefits of both control and learning to promote efficiency and explainability of artificial intelligence.

- To the control field, this work proposes a generic paradigm, which shows how a challenging control task can be converted into a learning formulation and solved using readily-available learning techniques, such as (deep) neural networks and backpropagation. For example, the proposed framework, equipped with (deep) neural networks, shows significant advantage for handling non-linear system identification and optimal control over state-of-the-art control methods. Since classic control theory typically requires knowledge of models, we expect that this work could pave a new way to extend classic control with data-driven techniques.

Since the formulation of this paper does not consider the boundness or constraints of a decision-making system, the real-world use of this work on physical systems might possibly raise safety issues during the training process; e.g., the state or input of the physical system at some time instance might exceeds the safety bounds that are physically required. One option to address this is to include these safety boundness as soft constraints added to the control objective or loss that is optimized. In future work, we will formally discuss PDP within a safety framework.

**Acknowledgments and Disclosure of Funding**

We acknowledge support for this research from Northrop Grumman Mission Systems’ University Research Program.

**References**


