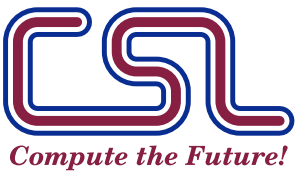
Computational Science Laboratory Report CSL-TR-22-2 May 5, 2022

Jostein Barry-Straume, Arash Sarshar,
Andrey A. Popov, and Adrian Sandu

"Physics-informed neural networks for PDE-constrained optimization and control"

Computational Science Laboratory
"Compute the Future!"

Department of Computer Science
Virginia Tech
Blacksburg, VA 24060
Phone: (540) 231-2193
Fax: (540) 231-6075
Email: sandu@cs.vt.edu
Web: https://csl.cs.vt.edu



Abstract

A fundamental problem of science is designing optimal control policies that manipulate a given environment into producing a desired outcome. Control Physics-Informed Neural Networks simultaneously solve a given system state, and its respective optimal control, in a one-stage framework that conforms to physical laws of the system. Prior approaches use a two-stage framework that models and controls a system sequentially, whereas Control PINNs incorporates the required optimality conditions in its architecture and loss function. The success of Control PINNs is demonstrated by solving the following open-loop optimal control problems: (i) an analytical problem (ii) a one-dimensional heat equation, and (iii) a two-dimensional predator-prey problem.

1 Introduction

Scientific Machine Learning (SciML) has arisen as a replacement to traditional numerical discretization methods. The main driving force behind this replacement is neural networks (NN), largely due to their success in natural language processing and computer vision problems [2]. As a vehicle to approximate the solution to a given partial differential equation (PDE) or ordinary differential equation (ODE), NNs offer a mesh-free approach via auto differentiation, and break the curse of dimensionality [12]. Combining scientific computing and ML, SciML offers the potential to improve "predictions beyond state-of-the-art physical models with smaller number of samples and generalizability in out-of-sample scenarios" [22]. Willard *et* al. provide a structured overview of physics-based modeling approaches with ML techniques, and summarize current areas of application with regard to science-guided ML in [22].

Physics-informed Neural Networks (PINNs) [18] solve semi-supervised learning tasks while respecting the properties of physical laws. This is achieved by informing the loss function about the mathematical equations that govern the physical system. Raissi *et* al. utilize PINNs for solving physical equations, and for data-driven discovery of partial differential equations [18]. The general procedure for solving a differential equation with a PINN involves finding the parameters of a network that minimize a loss function involving the mismatch of output and data, as well as residuals of the boundary and initial conditions, PDE equations, and any other physical constraints required [12]. The recent survey paper by Cuomo *et* al. [6] provide a comprehensive overview of PINNS and discusses a variety of customizations "through different activation functions, gradient optimization techniques, neural network structures, and loss functions". Since their introduction, PINNs have been leveraged to solve a wide range of problems including, but not limited to, inverse problems [4, 9, 12, 16, 17, 19], solution of fractional differential equations [15], and stochastic differential equations [14, 23–25].

Some recent work has began exploring the application of PINNs to solve optimal control problems [1, 3, 10, 13, 20]. However, existing approaches to use PINNs for optimal control problems either feed control data to a PINN, thereby training on precomputed control signals, or use an external control mechanism in conjunction with the PINN model of the physical system.

This paper presents a new PINN framework, named Control PINNs, for solving open loop optimal control problems. Control PINNs simultaneously solve the learning tasks of the system state, the adjoint system state, and the optimal control signal, without the need for either a priori controller data or an external controller. Moreover, Control PINNs can find optimal control solutions to complex computational scientific problems more efficiently using a comprehensive one-stage framework.

The remainder of the paper is organized as follows. In 2 we review prior work in the literature that applies PINNs to control problems, and substantiate the novelty of the Control PINNs approach developed herein. Section 3 covers the methodology of the novel approach that this paper offers. Section 4 validates the methodology via implementation of an analytical problem. Section 5 presents and discusses experimental results of a one-dimensional heat equation. Section 6 offers a more challenging optimal control problem for a two-dimensional predator-prey reaction diffusion problem. Section 7 summarizes the results and details the groundwork for future directions.

2 Current state-of-the-art in optimal control with PINNs

Chen *et* al. train an input convex recurrent neural network and subsequently solve a convex model predictive control (MPC) problem on the learned model [3]. The main strength of this approach is the guarantee of an optimal solution, due to the convex nature of the model. The main limitation is similar to [10], in that they employ a two-stage framework of system identification and controller design. Success is evaluated on four different experiments conducted in the paper where the input convex recurrent neural network results are compared to that of a standard multilayer perceptron (MLP). The control action is also compared to that of the baselines of conventional optimizations.

Antonelo *et* al. introduce a new framework called Physics-Informed Neural Nets for Control (PINC) [1]. PINC uses data from the control action and initial state to solve an optimal control problem. One strength of this approach is the ability make predictions beyond the training time horizon for an indefinite period of time without a significant reduction in prediction capability. A limitation of this approach is offline learning the control separately from the solution operator. In other words, PINC is essentially a PINN that is amenable to being trained on the actions of an external controller, instead of learning the optimal control unsupervised. Success is evaluated through Mean Squared Error (MSE) validation error of the solution on the Van der Pol Oscillator problem.

Wang *et* al. leverage physics-informed DeepONets "as a fast and differentiable surrogate for tackling high-dimensional PDE-constrained optimization problems via gradient-based optimization in near real-time" [20]. The foundational idea behind their approach is to optimize a network that associates an outcome with a set of controllable variables. The strength of the approach comes from leveraging DeepONets in a physics-informed fashion. This allows smaller training datasets, as their framework makes for an effective emulator. The limitation of their approach involves learning the control separately from the system state. The paper proposes sequentially training a neural network to learn the solution operator of a given PDE system, and then passing that information to another neural network to learn to associate the input system state with a certain control action. This approach, like others mentioned previously, is a two-stage framework. Moreover, the challenge of incorporating adjoint equations into their framework is circumvented, where indeed there is empirical evidence in favor of using adjoint information [5]. Here, one measure of performance considered is training time of PINNs compared to that of traditional numerical solvers. This benchmarking is conducted for both optimal control of heat transfer, and drag minimization of obstacles in Stokes flow. Moreover, a numerical solver is utilized to validate and test the inferred control solution versus the found solution.

Hwang *et* al. propose a two-stage framework for solving PDE-constrained control problems using operator learning [10]. They first train an autoencoder model, and then infer the optimal control by fixing the learnable parameter and minimizing their objective function. One strength of their approach is the ability to apply their framework to both data-driven and data-free cases. The main downside to their approach is the two-stage nature of the framework, as the control is found only after a surrogate model has been trained. Success is measured through tracking the relative error against numerical simulation in the case of data-driven experiments, and the relative error against an analytical solution in the case of data-free experiments. Additionally, visual inspection of the trained solution operators is conducted.

Mowlavi and Nabi conduct an evaluation of the comparative performance between traditional PINNs and classic direct-adjoint-looping (DAL) to solve optimal control problems [13]. Their optimal control problem is separated into two subproblems. At each state of the system, the PDE is solved with one neural network. That information is then used by another neural network to solve for the optimal control at that given state of the PDE. Afterwards, the adjoint PDE is solved in backwards time. The strength of Mowlavi and Nabi's approach is providing an evaluative comparison between PINNs and DAL frameworks for solving PDE-constrained optimal control problems. Their comparison provides a frame of reference for PINNs. Success is measured via validation and evaluation steps. Validation is done by monitoring residual, boundary, and initial loss components with a known solution. Evaluation is done by comparing the control cost objective with a solution found by a high-fidelity numerical solver. One limitation of this approach is using the more easily solvable steady state Navier-Stokes, instead of unsteady state Navier-Stokes. Moreover, manual derivation is used in their DAL approach, which is unnecessary because DAL can use automatic differentiation (AD). Consequently, to a certain extent, the optimal control problem is being solved manually. Furthermore, the control of the system is being dampened over time. This is suspicious, as it might mean this dampening approach was added post hoc because of struggling results. It should be noted that the adjoint PDE is not being

solved in their cost function, and thus the respective adjoint formulas are not present in said cost function. This brings us to the the main contributions of this paper.

The proposed framework in this paper goes beyond the approaches in [1, 3, 10, 13, 20]. Control PINNs do not rely on data from an external controller. Instead, Control PINNs solve simultaneously for the optimal solution and the optimal control signal with respect to a given cost function, and constrained by the system governing equations. This approach is different from building accurate PINNs and using them as differentiable surrogate models in the optimization solution for control applications. Rather, we propose a framework that can be considered a new taxonomical entity within the genus of PINNs, wherein the PDE-constrained optimization is fused with the training loss function to create a one-stage approach to directly learning both the solution and the optimal control.

3 Methodology

Hairer and Wanner provide an overview of control problems, and more specifically, optimal control problems in [8, pp. 461-463]. Here, we present the problem as an ordinary differential equation of the form $\mathbf{y}' = f(\mathbf{y}, \mathbf{u})$ where $\mathbf{u} = \mathbf{u}(x, t)$ represents a time-varying distributed control applied to the system. We are interested in optimal control solutions $(\mathbf{y}^*, \mathbf{u}^*)$ that satisfy the ODE equation while minimizing a given cost function Ψ that may depend on the solution, control, or both.

Formally, we seek to solve the PDE-constrained control problem:

$$\begin{aligned} \mathbf{u}^* = \arg\min_{\mathbf{u}} \Psi(\mathbf{u}) &= \int_{t_0}^{t_f} g\left(\mathbf{y}, \mathbf{u}\right) \mathrm{d}t + w(\mathbf{y}|_{t_f}), \\ \mathbf{y}'(t,x) &= f\left(\mathbf{y}, \mathbf{u}\right), \quad \forall t \in [t_0, t_f], \quad \forall x \in \Omega, \\ \text{subject to} \quad \mathbf{y}(t_0, x) &= \mathbf{y}_0, \quad \forall x \in \Omega, \\ \mathbf{y}(t,x) &= \mathbf{b}(t,x), \quad \forall t \in [t_0, t_f], \quad \forall x \in \partial\Omega. \end{aligned} \tag{1}$$

We denote by $\mathbf{y} = \mathbf{y}(t, x)$ the solution of the semi-discretized PDE , and by $\mathbf{u} = \mathbf{u}(t, x)$ the control signal. $\mathbf{y}|_t$ is shorthand for $\mathbf{y}(t, x)$. The cost function in eq. (1) includes penalty terms for the solution trajectory as well as a desired final-time solution. The Lagrangian for this optimization is written as:

$$\mathfrak{L} = \int_{t_0}^{t_f} \left[g\left(\mathbf{y}, \mathbf{u}\right) - \boldsymbol{\lambda}(t)^T \left(\mathbf{y}' - f(\mathbf{y}, \mathbf{u})\right)\right] \mathrm{d}t + w(\mathbf{y}|_{t_f}), \tag{2a}$$

$$\frac{\partial \mathfrak{L}}{\partial \mathbf{u}} = \int_{t_0}^{t_f} \left[g_{\mathbf{y}}\left(\mathbf{y}, \mathbf{u}\right) \mathbf{y}_{\mathbf{u}} + g_{\mathbf{u}}\left(\mathbf{y}, \mathbf{u}\right) - \boldsymbol{\lambda}(t)^T \left(\mathbf{y}'_{\mathbf{u}} - f_{\mathbf{u}}\left(\mathbf{y}, \mathbf{u}\right) - f_{\mathbf{y}}\left(\mathbf{y}, \mathbf{u}\right) \mathbf{y}_{\mathbf{u}}\right)\right] \mathrm{d}t + w_{\mathbf{y}}(\mathbf{y}|_{t_f}\, \mathbf{y}_{\mathbf{u}}|_{t_f}). \tag{2b}$$

Using integration by parts to eliminate $\mathbf{y}'_{\mathbf{u}}$ in eq. (2b) we have

$$\int_{t_0}^{t_f} \boldsymbol{\lambda}(t)^T \mathbf{y}'_{\mathbf{u}} \, \mathrm{d}t = \boldsymbol{\lambda}(t)^T \mathbf{y}_{\mathbf{u}} \Big|_{t_0}^{t_f} - \int_{t_0}^{t_f} \boldsymbol{\lambda}'(t)^T \mathbf{y}_{\mathbf{u}} \, \mathrm{d}t \,. \tag{3}$$

Setting $\boldsymbol{\lambda}(t_f) = w_{\mathbf{y}}(\mathbf{y}|_{t_f})$ in eq. (3) and substituting in eq. (2b)

$$\begin{aligned} \frac{\partial \mathfrak{L}}{\partial \boldsymbol{\lambda}} &= \int_{t_0}^{t_f} \left[\left(\mathbf{y}' - f(\mathbf{y}, \mathbf{u})\right)\right] \mathrm{d}t \,, \\ \frac{\partial \mathfrak{L}}{\partial \mathbf{u}} &= \int_{t_0}^{t_f} \left[\left(\boldsymbol{\lambda}'(t)^T + g_{\mathbf{y}}\left(\mathbf{y}, \mathbf{u}\right) + \boldsymbol{\lambda}(t)^T f_{\mathbf{y}}\left(\mathbf{y}, \mathbf{u}\right)\right) \mathbf{y}_{\mathbf{u}}\right] \mathrm{d}t \\ &\quad + \int_{t_0}^{t_f} \left[g_{\mathbf{u}}\left(\mathbf{y}, \mathbf{u}\right) + \boldsymbol{\lambda}(t)^T f_{\mathbf{u}}\left(\mathbf{y}, \mathbf{u}\right)\right] \mathrm{d}t \,. \end{aligned} \tag{4}$$

The first order optimality condition requires $\frac{\partial \mathfrak{L}}{\partial \mathbf{u}} = \mathbf{0}$, and, $\frac{\partial \mathfrak{L}}{\partial \boldsymbol{\lambda}} = \mathbf{0}$. The integrals in eq. (4) can be numerically approximated using a quadrature scheme. Alternatively, we can enforce the stronger

point-wise equations:

$$\mathbf{y}' = f(\mathbf{y}, \mathbf{u}), \ \mathbf{y}(0) = \mathbf{y}_0, \qquad \forall t \in [t_0, t_f], \tag{5a}$$

$$\boldsymbol{\lambda}'(t) = -\boldsymbol{\lambda}(t)^T f_{\mathbf{y}}(\mathbf{y}, \mathbf{u}) - g_{\mathbf{y}}(\mathbf{y}, \mathbf{u}), \ \boldsymbol{\lambda}(t_f) = w_{\mathbf{y}}(\mathbf{y}|_{t_f}), \qquad \forall t \in [t_f, t_0], \tag{5b}$$

$$\mathbf{0} = \boldsymbol{\lambda}(t)^T f_{\mathbf{u}}(\mathbf{y}, \mathbf{u}) + g_{\mathbf{u}}(\mathbf{y}, \mathbf{u}), \qquad \forall t \in [t_0, t_f]. \tag{5c}$$

Note that eq. (5a) may also contain boundary conditions similar to eq. (1). Equations (5a) to (5c) can be thought of respectively as the equations for system state, the system controller , and the optimality equation. As an example, in the context of an autonomous vehicles, the system state (the velocity and the position of the vehicle) are determined by the equations of motion. By the same token, the system controller would be the software that governs the steering wheel, acceleration, and braking. Finally, the system push back would be the response of the vehicle to the software's choices of direction and speed.

We design a neural network that generates triples $(\mathbf{y}, \mathbf{u}, \boldsymbol{\lambda})$ from input data (t, x), equipped with PINN loss functions that capture the first order optimality conditions according to eq. (5). The process of solving the control problem eq. (1) is outlined in algorithm 1. The main technical challenge involves learning the state of a dynamical system while at the same time finding its optimal control. We create a path in the computational graph for the back propagation of derivatives by placing the the deep layers that generate $\mathbf{u}$ after the layers that generate $\mathbf{y}$ and, similarly for $\boldsymbol{\lambda}$. This ensures that all necessary derivatives for eq. (5) can be computed using automatic differentiation.

Moreover, there is a tension between different terms in the loss function defined in algorithm 1. For example, satisfying the boundary conditions may oppose adhering to the constraints imposed by the physical laws. This is addressed by scaling factors in the loss function and treating them as hyper-parameters. We further note that as Control PINN is applied to increasingly more complex problems, a unique solution to the problem may not be available. We discuss the validation of the solution found by the PINN model as well as its optimality further in section 5.

Algorithm 1: The procedure to train a Control PINN model

Result: Training of a Control PINN that learns the optimal solution and the optimal control function for the given problem in (1)

1. Construct a network with inputs t, x, and outputs $\mathbf{y}$, $\mathbf{u}$, and $\boldsymbol{\lambda}$ based on the architecture in fig. 1
2. Using back-propagation, compute the necessary derivatives of the outputs w.r.t to the inputs and other outputs. These derivatives will be used in the loss function to compute the residuals in eq. (5)
3. With $\mathbf{y}^i_*$ denoting the training data for the input (t^i, x^i), and $\mathbf{y}^i, \mathbf{u}^i, \boldsymbol{\lambda}^i$ corresponding outputs of the network, $\boldsymbol{\theta}_k$ representing the weights of the network at iteration k, and $\|\cdot\|$ as L_2 norm, we seek to minimize the loss function:

$$\begin{aligned} L(\boldsymbol{\theta}_k) = &\sum_i \|\mathbf{y}^i_* - \mathbf{y}^i\|^2 \\ &+ \sum_i \|\frac{\partial \mathbf{y}^i}{\partial t} - f(\mathbf{y}^i, \mathbf{u}^i)\|^2 \\ &+ \sum_i \|\frac{\partial \boldsymbol{\lambda}^i}{\partial t} + (\boldsymbol{\lambda}^i)^T \frac{\partial}{\partial \mathbf{y}^i} f(\mathbf{y}^i, \mathbf{u}^i) + \frac{\partial}{\partial \mathbf{y}^i} g(\mathbf{y}^i, \mathbf{u}^i)\|^2 \\ &+ \sum_i \|(\boldsymbol{\lambda}^i)^T \frac{\partial}{\partial \mathbf{u}^i} f(\mathbf{y}^i, \mathbf{u}^i) + \frac{\partial}{\partial \mathbf{u}^i} g(\mathbf{y}^i, \mathbf{u}^i)\|^2. \end{aligned} \tag{6}$$

4. Update the weights of the network using the optimizer according to the loss function eq. (6).

$$\boldsymbol{\theta}_{k+1} = \texttt{Adam}(\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k)).$$

5. Repeat until convergence.

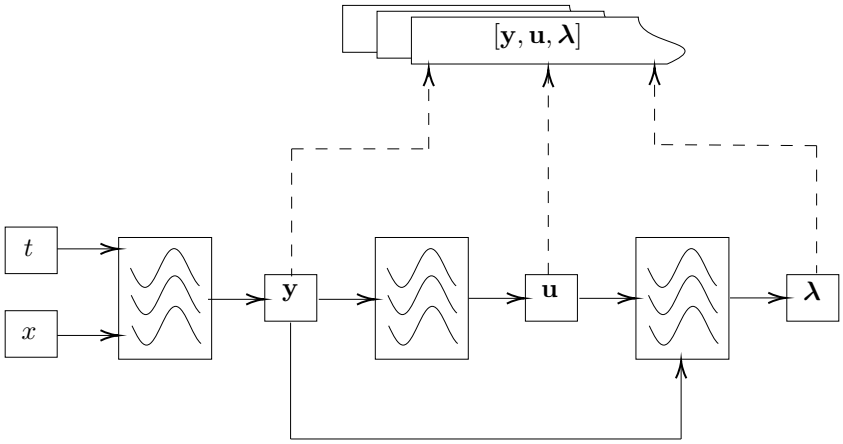


Figure 1: *Control PINN architecture:* Time t and space x are inputs to the model, whereas the system state y, control u, and system push back λ are the outputs to the model. The boxes with wave lines represent hidden layers. The dashed lines indicate the outputs. Note that the system state y is passed into both the control u and the system push back λ. This enables for the automatic differentiation of second order and mixed derivatives.

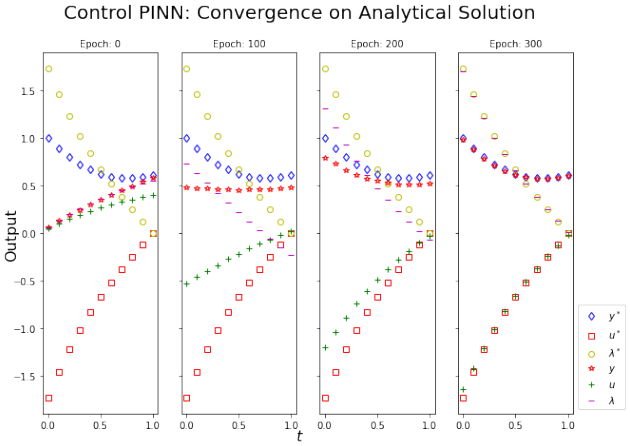


Figure 2: *Results of the analytical problem:* The optimal solutions of the system state, control, and system push back on the control are respectively denoted by y^*, u^*, and λ^*. Their corresponding learned solutions found by the Control PINN model are denoted by y, u, and λ. After 300 epochs the Control PINN has reached convergence on the analytical solution, and the solutions remain unchanged upon further training.

Control PINN architecture. A visual representation of the Control PINN architecture is found in fig. 1. Adaptive moment estimation (Adam) is used as the optimizer. The activation function of exponential linear unit (ELU) is used. The neural density is 100 neurons per layer. There are five hidden layers that proceed the input layer that takes in time (t) and space (x). The information of the system state ($\mathbf{y}$) is passed to the controller ($\mathbf{u}$) both directly and indirectly by a skip connection and three hidden layers, respectively. The aggregate information of the system state ($\mathbf{y}$) and controller ($\mathbf{u}$) is handled similarly in the context of the system's push back on the controller ($\boldsymbol{\lambda}$), except with only two hidden layers. This architecture enables for the automatic differentiation of second order and mixed derivatives. This is necessary to impose the custom loss function detailed in algorithm 1. All three experiments presented in this paper use the same architecture, further demonstrating the robustness of the Control PINN framework as increasingly more challenging problems are tackled.

4 Analytical problem

As a proof of concept, and to provide a foundation for methodology validation, consider the following ODE control problem [7, Ch. 6, pp. 272-273, eqn. 65],

$$
\begin{aligned}
u^* = \arg\min_u \Psi(u) = \int_{t_0}^{t_f} \left(y^2(t) + \frac{1}{2} u^2(t) \right) \mathrm{d}t \,, \\
y'(t) = \frac{1}{2} y + u, \qquad \forall t \in [t_0, t_f], \\
\text{subject to} \quad y(t_0) = y_0 = 1, \\
\lambda'(t) = -\frac{1}{2} \lambda(t) - 2y, \quad \forall t \in [t_f, t_0], \quad \lambda(t_f) = 0, \\
0 = -\lambda(t) - u, \qquad \forall t \in [t_0, t_f],
\end{aligned}
\tag{7}
$$

where $t_0 = 0$, $t_f = 1$. An analytical form of the optimal solution is available:

$$
y^*(t) = \frac{2e^{3t} + e^3}{e^{3t/2}(2 + e^3)}, \quad u^*(t) = \frac{2(e^{3t} - e^3)}{e^{3t/2}(2 + e^3)}, \quad \lambda^*(t) = -u^*(t). \tag{8}
$$

After 300 epochs of training, the Control PINN converges on the optimal solution. This is shown in fig. 2, wherein y^*, u^*, and λ^* respectively represent the reference solution for the system state y, the reference solution for the system controller u, and the reference solution of the system push back on the controller λ. An animation of the convergence of the outputs to the reference solution as training progresses can found at: https://github.com/ComputationalScienceLaboratory/control-pinns/blob/main/animations/AnalyticalProblem_Convergence.gif.

5 One-dimensional heat equation

Fourier's famous heat equation $\frac{\partial u}{\partial t} = \alpha^2 \frac{\partial^2 u}{\partial x^2}$ is the origin of the Fourier series theory [8, p. 30], and is thus a well known and established problem with which to illustrate the robustness of Control PINN. Imagine an infinitesimally thin steel beam being heated by a heat pad. This heat pad will be controlled in heating the steel beam such that a given uniform temperature will be reached at the final time. The problem set up is as follows:

$$
\begin{aligned}
u^* = \arg\min_u \Psi(u) = \int_\Omega \left(\left(y(u(x, t_0), t_0) - y^*(x, t_0) \right) \right)^2 \mathrm{d}x \\
+ \int_\Omega \left(y(u(x, t_f), t_f) - y^*(x, t_f) \right)^2 \mathrm{d}x \\
+ \int_\Omega \int_{t_0}^{t_f} \left(u(x, t) \right)^2 \mathrm{d}t\mathrm{d}x, \\
\text{subject to} \quad \frac{\partial y}{\partial t} = 0.1 \frac{\partial^2 y}{\partial x^2} + u(x, t), \quad \forall t \in [t_0, t_f], \ \Omega = [0, 1], \\
y(x, t_0) = \sin(\pi x) \sin(2\pi x), \quad \forall x \in \Omega, \\
y(x, t) = 0, \qquad \forall t \in [t_0, t_f], \ x \in \{0, 1\}, \\
y^*(x, t) = k \left(\exp\{-\pi^2 t\} - \cos\left(\frac{\pi t}{2}\right) + 2\pi \sin\left(\frac{\pi t}{2}\right) \right) \sin(\pi x), \quad k = 2/(\pi + 4\pi^3).
\end{aligned}
\tag{9}
$$

The cost function in eq. (9) consists of desired initial and final time solutions and an l_2 regularization term that favors minimal smooth control functions $\mathbf{u}$ over space and time. Figure 4 illustrates the results of the model finding the solutions of the system state and the control over the time span $[0, 1]$. A reference solution for the system push back (λ) is not known, and is therefore not included in the plot for comparative purposes. There is an exact match between the reference solution and learned solution of the system state. Table 1 lists the relative error between the learned solution of Control PINN to that of the direct numerical simulation.

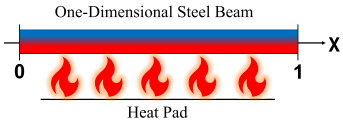


Figure 3: *Explanatory diagram:* A one-dimensional steel beam is heated by a controllable heat pad over the space x from x_0 to x_f such that at the final time a uniform temperature of the beam will be realized.

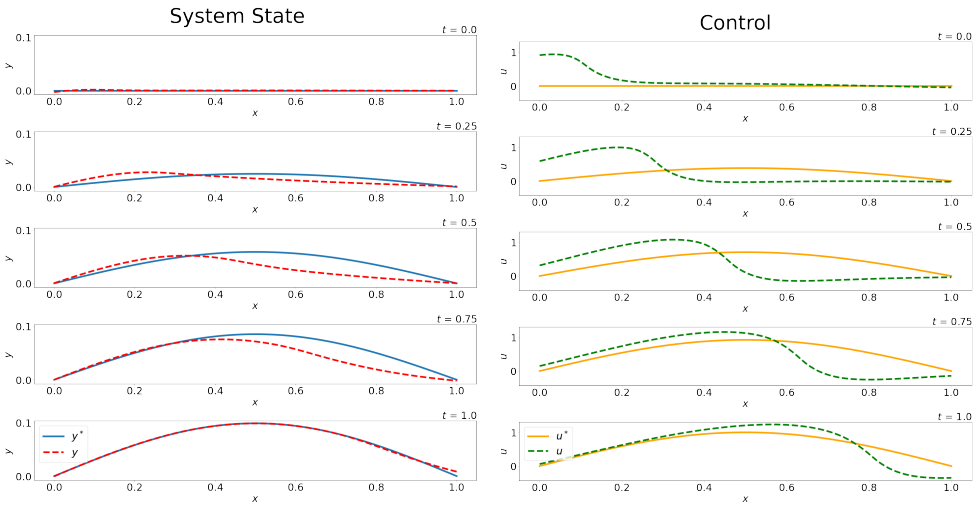


Figure 4: *One-dimensional heat equation results: Left:* Comparison of the learned system state y and the reference system state y^*. *Right:* Comparison of the learned control u and the reference control u^* . Note that this problem has a non-unique solution. Control PINN found a different control that minimizes the control action on the system state.

Of note is the shape of the optimal control found by the model. The optimal control of the reference solution reaches a parabolic curve over the domain as time approaches $t = 1$. The trajectory of the model's solution dips more sharply downward towards the end of the feature space. This begs the question of if the model's solution of the control is valid.

To validate the solution found by the Control PINN, we generated high resolution offline data of the control and the solution outputs at different values of time and space. A direct numerical solver (DNS) using finite differences was then used along with the control data to generate DNS solutions for this problem . Figure 7 shows the validation of the solution for this problem. We see good agreement between the two approaches. Table 1 shows the error computed as the L_2 norm of the difference between the PINN model and DNS on the spatial domain (discretized uniformly with 1000 points) for a number of different timepoints.

If this framework was implemented in the field there would often be cases of non-unique solutions. With this in mind, it is not a problem that the model found a different non-unique solution. Figure 7 shows the analytical solution with respect to the numerical solution with the learned control. In other words, Control PINN found the control u, whose data in turn was used to solve the PDE in eq. (9). Figure 7 validates that the control learned by Control PINN is less than that of the analytical solution. This means for a problem with a non-unique solution, Control PINN was able to find a solution with the smallest amount of control.

6 Two-dimensional predator-prey problem

We next look at a two-dimensional predator-prey (Lotka-Volterra) problem formulated as a reaction-diffusion problem[11]. On a two-dimensional Cartesian grid, the system characterized by eq. (10) defines the interaction of two populations: the predator and the prey. The rate of birth for each

population follows an exponential law based on the current population and is also affected by the competing population. We are interested in controlling the prey population (over time and space) by inserting predators at different times and locations. For this example we have considered a starting profile of preys which is then "herded" into a desired profile over a specified timespan. The predator population in eq. (10) is denoted by y_1, whereas y_2 represents the prey. Similarly, u_1 and u_2 are predator and prey control functions, respectively. We have used $u_1(t, \mathbf{x}) = 0$ for this problem.

$$
\begin{aligned}
\mathbf{u}^* = \arg\min_{\mathbf{u}} \Psi(\mathbf{u}) = &\int_\Omega \int_{t_0}^{t_f} \|\mathbf{y}(\mathbf{x},t) - \mathbf{y}^*(\mathbf{x},t)\|_2^2 \,\mathrm{d}t\,\mathrm{d}\mathbf{x} + \int_\Omega \int_{t_0}^{t_f} \|\mathbf{u}(\mathbf{x},t)\|_2^2 \,\mathrm{d}t\mathrm{d}\mathbf{x}, \\
0 = &\frac{\partial y_1}{\partial t} - \frac{\partial^2 y_1}{\partial \mathbf{x}^2} - u_1(\mathbf{x},t) + y_1(\mathbf{x},t), \quad \forall t \in [t_0, t_f], \forall \mathbf{x} \in \Omega, \\
0 = &\frac{\partial y_2}{\partial t} - \frac{\partial^2 y_2}{\partial \mathbf{x}^2} - u_2(\mathbf{x},t) - y_2(\mathbf{x},t), \quad \forall t \in [t_0, t_f], \forall \mathbf{x} \in \Omega, \\
\text{subject to} \quad y_1(\mathbf{x},0) = &\sin(\pi x_1)\sin(\pi x_2), \quad \forall \mathbf{x} \in \Omega, \\
y_2(\mathbf{x},t) = &\, t\left(\sin(2\pi x_1)\sin(2\pi x_2)\right)^2 \quad \forall t \in [t_0, t_f], \forall \mathbf{x} \in \Omega, \\
&+ (1-t)\sin(\pi x_1)\sin(\pi x_2), \quad \text{where } \Omega = [0,1]^2.
\end{aligned}
\tag{10}
$$

Figure 10 plots the reaction and diffusion of predator and prey populations at three different time points $t = [0.0, 0.5, 1.0]$. The first column illustrates the reference solution of the predator population. The second column shows the model's learned solution of the predator population. Likewise, the next two columns respectively show the same for that of the prey population. The last column is the optimal control found by the model. The main takeaway from Figure 10 is that Control PINN is able to minimize the control needed on the prey population y_2 such that at the final time the desired population density of both the predator and prey populations are realized. Figure 5 is a comparison between the analytical and learned solutions of both the predator and prey populations in the form of a two-dimensional contour plot. The absolute error of the two selected timepoints shows how Control PINN reaches its objective at the end of the timespan. An animation of the Control PINN model converging on the optimal solution can be found at https://github.com/ComputationalScienceLaboratory/control-pinns/blob/main/animations/ControlPINN_Predator_Prey_Absolute_Error.gif. Figure 6 puts forth a granular comparison of the analytical solution and the learned solution of the prey population at the final time, along with the corresponding control. The four cones are nearly identical between the analytical and learned solutions. Of note is the challenge faced of Control PINN to learn the boundary solution exactly, shown in the y_2 subplot as the wavy boundary at the base of the four cones. [20] also makes note of their struggles to wrangle their framework's ability to balance solving the overall problem while maintaining satisfactory respect of the boundary conditions.

Figure 9 shows Control PINN learning the optimal control of the prey population from a top down two-dimensional grid perspective. Figure 11 illustrates the same solution with more timepoints, and additionally shows the prey push back λ in response to the control u_2. Figure 12 offers a zoomed in view of the tail-end of the timespan, which illustrates how Control PINN minimizes the control needed to reach the desired population states by waiting until near the end to enact significant control on the prey population.

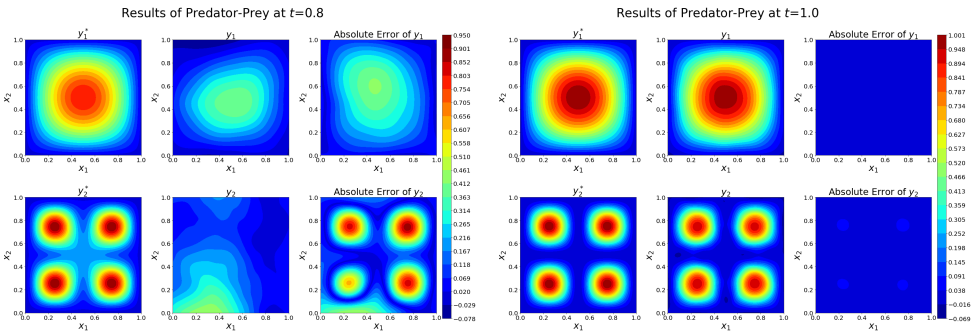


Figure 5: *Two-dimensional predator-prey results: Left:* Comparison of both the numerical simulation and the learned system states of the predator y_1 and prey y_2 populations at the time $t = 0.8$. The absolute error of the two comparisons is displayed in the third column. *Right:* Comparison of both the numerical simulation and the learned system states of the predator y_1 and prey y_2 populations at the time $t = 1.0$. The absolute error of the two comparisons is displayed in the third column. Note the decrease in absolute error at the final time $t = 1.0$. An expanded looked into the absolute errors throughout the timespan $t = [0.0, 1.0]$ is found in appendix B.

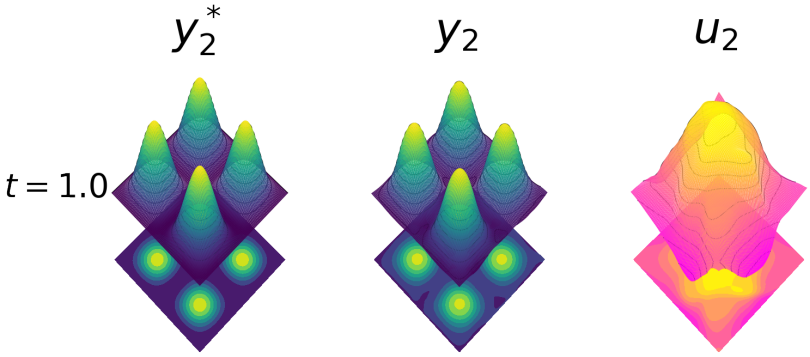


Figure 6: *Two-dimensional predator-prey problem: Left:* Prescribed prey population at the final time $t_f = 1.0$. *Middle:* Learned solution y_2 prey population at the final time $t_f = 1.0$. *Right:* Control signal u_2 on the predator population y_2 determined by the Control PINN at the final time $t_f = 1.0$. The three-dimensional surface plot is projected downward to a two-dimensional contour plot for increased discrepancy of comparative purposes. Note the differences in the bases of the surface plots, which illustrates the challenges of boundary condition enforcement.

7 Conclusions

This work provides a novel approach for solving the optimal control problem for PDEs using PINNs. In contrast to previous approaches, we integrate the optimality conditions from the control problem directly in a theory-guided and physics-informed manner. We dub this approach Control PINNs.

The Control PINN methodology is able to simultaneously learn the system state solution and the optimal control for a general class of PDEs. We illustrate the validity of our approach on a diverse set of problems: a simple control problem for which an analytical solution is known, a one-dimensional heat equation for which a control that is not optimal is known, and a two-dimensional predator-prey problem which might not even have an optimal control. In higher dimensional problems, a tension exists in Control PINNs between respecting the boundary conditions of the given system state while learning the solution and corresponding optimal control. Adaptive methods for choosing the scaling of the terms in the loss similar to [21] are of future interest.

One potential application is leveraging Control PINNS as agents in deep reinforcement learning (DRL) [6]. In DRL, finding the state of a robot after a given action requires solving a number of physical equations (e.g. equation of motion and balance of force). This issue can be circumvented

by leveraging PINNs as an agent because PINNs penalize deviations from physical constraints by design. Furthermore, an agent that can simultaneously solve a system state and a corresponding optimal control, such as Control PINN, would be useful in Q-learning to efficiently optimize the value of action-state policies. Future work will also involve extending Control PINNs to operate on closed-loop control problems.

Acknowledgments and Disclosure of Funding

Special thanks to Austin Chennault for pointing out many relevant literature to this project. This work was supported by DOE through award ASCR DE-SC0021313, by NSF through award CDS&E–MSS 1953113, and by the Computational Science Laboratory at Virginia Tech. All data and codes used in this manuscript are publicly available on GitHub at: https://github.com/ComputationalScienceLaboratory/control-pinn.

References

[1] E. A. Antonelo, E. Camponogara, L. O. Seman, E. R. de Souza, J. P. Jordanou, and J. F. Hubner. Physics-Informed Neural Nets for Control of Dynamical Systems, 2021.

[2] P. J. Atzberger. Importance of the Mathematical Foundations of Machine Learning Methods for Scientific and Engineering Applications, 2018.

[3] Y. Chen, Y. Shi, and B. Zhang. Optimal Control Via Neural Networks: A Convex Approach, 2019.

[4] Y. Chen, L. Lu, G. E. Karniadakis, and L. Dal Negro. Physics-informed neural networks for inverse problems in nano-optics and metamaterials. *Optics Express*, 28(8):11618, Apr 2020. ISSN 1094-4087. doi: 10.1364/oe.384875. URL http://dx.doi.org/10.1364/OE.384875.

[5] A. Chennault, A. A. Popov, A. N. Subrahmanya, R. Cooper, A. Karpatne, and A. Sandu. Adjoint-Matching Neural Network Surrogates for Fast 4D-Var Data Assimilation, 2021. URL https://arxiv.org/abs/2111.08626.

[6] S. Cuomo, V. S. di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli. Scientific Machine Learning through Physics-Informed Neural Networks: Where we are and What's next, 2022.

[7] W. W. Hager. Runge-Kutta methods in optimal control and the transformed adjoint system. *Numerische Mathematik*, 87:247–282, 2000.

[8] E. Hairer and G. Wanner. Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems. 2002.

[9] Q. He, D. Barajas-Solano, G. Tartakovsky, and A. M. Tartakovsky. Physics-informed neural networks for multiphysics data assimilation with application to subsurface transport. *Advances in Water Resources*, 141:103610, Jul 2020. ISSN 0309-1708. doi: 10.1016/j.advwatres.2020.103610. URL http://dx.doi.org/10.1016/j.advwatres.2020.103610.

[10] R. Hwang, J. Y. Lee, J. Y. Shin, and H. J. Hwang. Solving PDE-constrained Control Problems using Operator Learning, 2021.

[11] A. J. Lotka. Contribution to the Theory of Periodic Reactions. *The Journal of Physical Chemistry*, 14(3):271–274, 1910. doi: 10.1021/j150111a004. URL https://doi.org/10.1021/j150111a004.

[12] L. Lu, X. Meng, Z. Mao, and G. Karniadakis. DeepXDE: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021. doi: 10.1137/19M1274067.

[13] S. Mowlavi and S. Nabi. Optimal control of PDEs using physics-informed neural networks, 2021.

[14] M. A. Nabian and H. Meidani. A deep learning solution approach for high-dimensional random differential equations. *Probabilistic Engineering Mechanics*, 57:14–25, Jul 2019. ISSN 0266-8920. doi: 10.1016/j.probengmech.2019.05.001. URL http://dx.doi.org/10.1016/j.probengmech.2019.05.001.

[15] G. Pang, L. Lu, and G. E. Karniadakis. fPINNs: Fractional Physics-Informed Neural Networks. *SIAM Journal on Scientific Computing*, 41(4):A2603–A2626, Jan 2019. ISSN 1095-7197. doi: 10.1137/18m1229845. URL http://dx.doi.org/10.1137/18M1229845.

[16] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations. *arXiv preprint arXiv:1711.10561*, 2017.

[17] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations. *arXiv preprint arXiv:1711.10566*, 2017.

[18] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[19] M. Raissi, A. Yazdani, and G. Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367:eaaw4741, 01 2020. doi: 10.1126/science.aaw4741.

[20] S. Wang, M. A. Bhouri, and P. Perdikaris. Fast PDE-constrained optimization via self-supervised operator learning. *CoRR*, abs/2110.13297, 2021. URL https://arxiv.org/abs/2110.13297.

[21] S. Wang, H. Wang, and P. Perdikaris. Improved architectures and training algorithms for deep operator networks, 2021. URL https://arxiv.org/abs/2110.01654.

[22] J. Willard, X. Jia, S. Xu, M. Steinbach, and V. Kumar. Integrating Scientific Knowledge with Machine Learning for Engineering and Environmental Systems, 2021.

[23] L. Yang, D. Zhang, and G. E. Karniadakis. Physics-Informed Generative Adversarial Networks for Stochastic Differential Equations, 2018.

[24] D. Zhang, L. Guo, and G. E. Karniadakis. Learning in Modal Space: Solving Time-Dependent Stochastic PDEs Using Physics-Informed Neural Networks, 2019.

[25] D. Zhang, L. Lu, L. Guo, and G. E. Karniadakis. Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. *Journal of Computational Physics*, 397:108850, Nov 2019. ISSN 0021-9991. doi: 10.1016/j.jcp.2019.07.048. URL http://dx.doi.org/10.1016/j.jcp.2019.07.048.

A One-dimensional heat equation

Table 1: *One-dimensional heat equation relative error:* Error of the trained Control PINN relative to direct numerical simulation.

Time	Relative Error
0.1	1.2309
0.2	0.2646
0.3	0.1805
0.4	0.2010
0.5	0.1843
0.6	0.1514
0.7	0.1168
0.8	0.0843
0.9	0.0535
1.0	0.0232

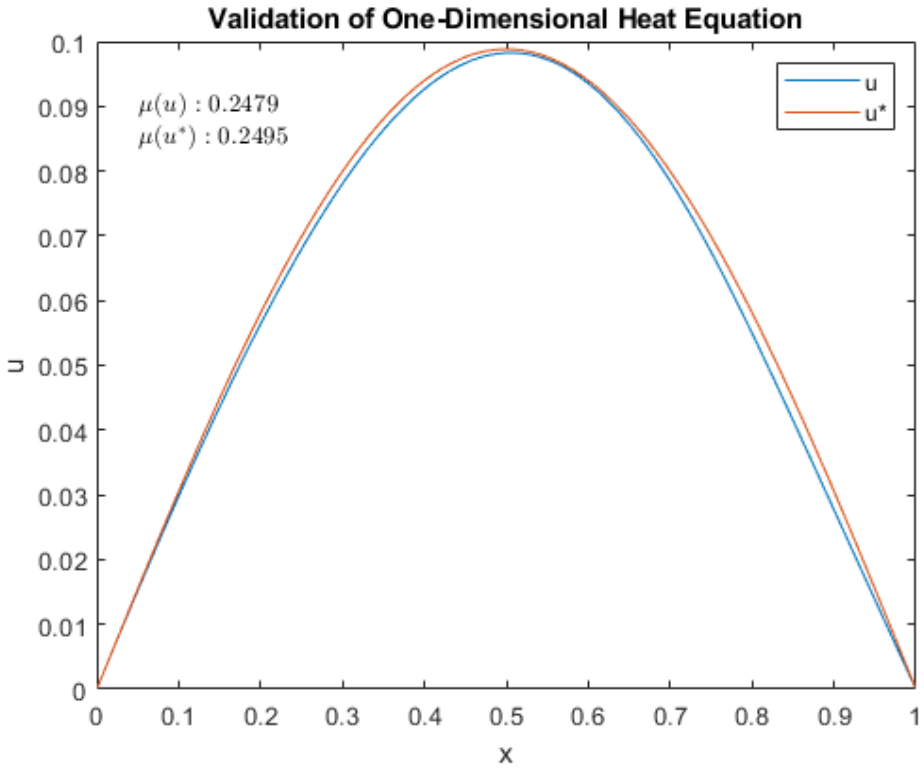


Figure 7: *Validation of 1-D Heat Equation:* Comparison of the control from the analytical solution to that of Control PINN. The mean of control action of Control PINN is less than that of the analytical solution.

B Two-dimensional predator-prey problem

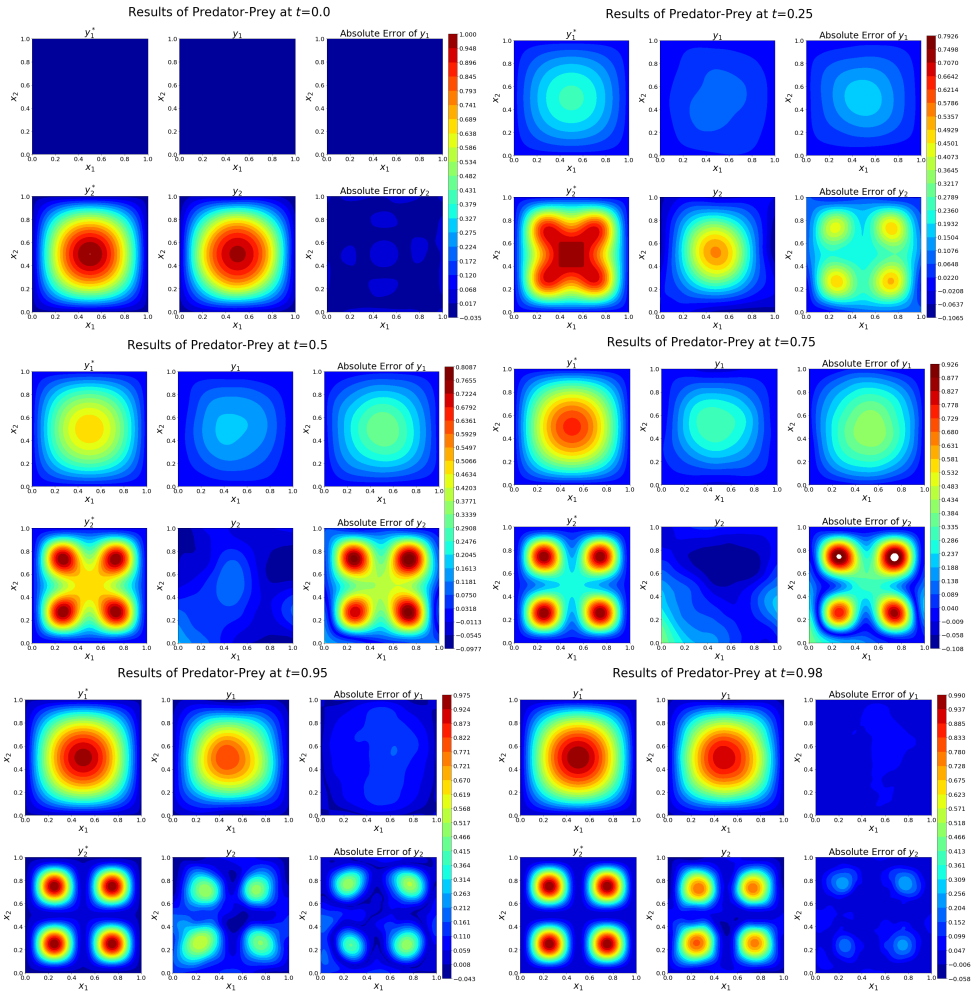


Figure 8: *Two-dimensional predator-prey results:* Expanded comparison of the absolute error between the learned and analytical solutions for the timepoints $t \in [0.0, 0.25, 0.50, 0.75, 0.95, 0.98]$. *First column of each subplot:* The analytical solutions for the predator y_1^* and y_2^* populations. *Second column of each subplot:* The learned solutions for the predator y_1^* and y_2^* populations. *Third column of each subplot:* The absolute error between the analytical and learned solutions for the predator y_1^* and y_2^* populations.

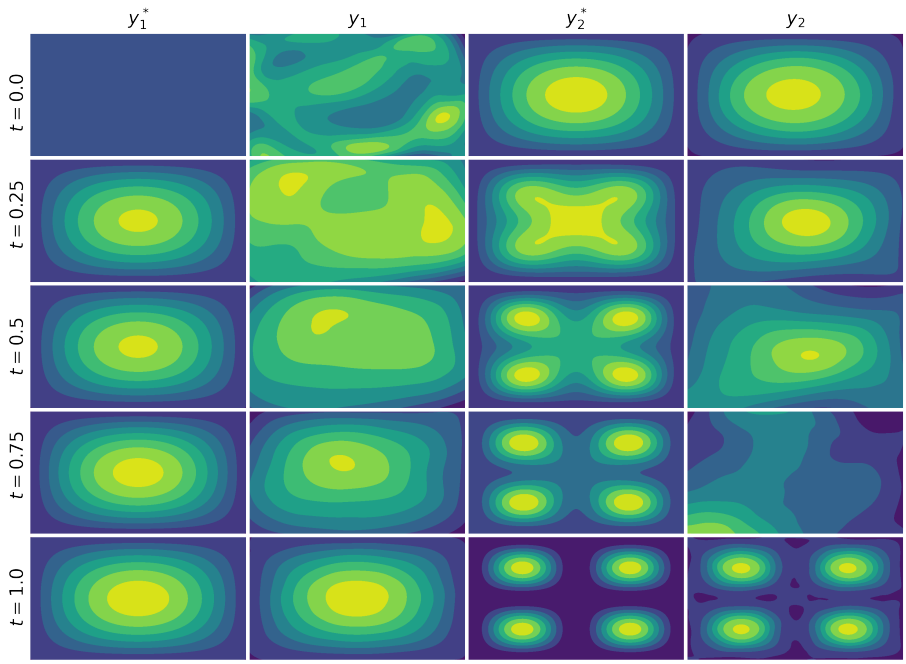


Figure 9: *Two-dimensional predator prey problem:* Contour plot comparison at varying time points between the analytical solutions and the learned solutions by Control PINN of the predator and prey populations. *First column:* The analytical solution of the predator y_1^* population at varying time snapshots. *Second column:* The learned solution by Control PINN of the predator y_1 population at varying time snapshots. *Third column:* The analytical solution of the prey y_2^* population at varying time snapshots. *Fourth column:* The analytical solution by Control PINN of the prey y_2 population at varying time snapshots.

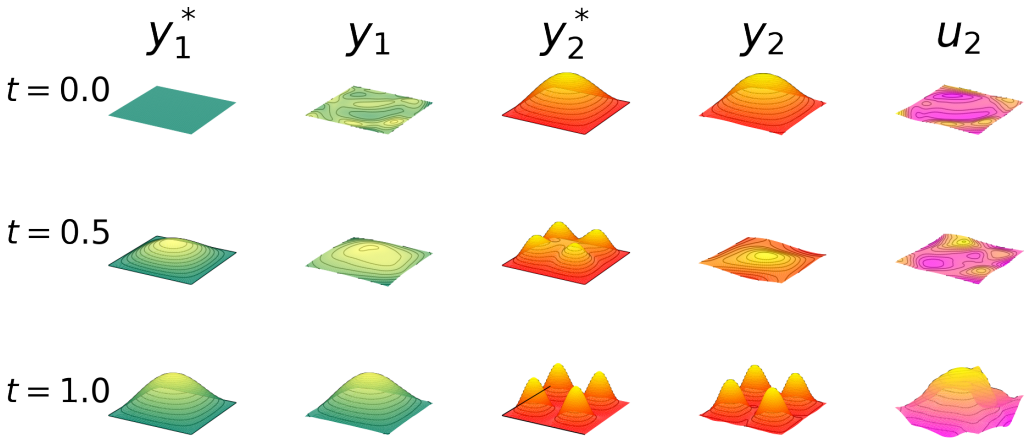


Figure 10: *Two-dimensional predator-prey problem:* Snapshots from the beginning, middle, and end of the timespan. *First column:* The reference solution of the predator population y_1^*. *Second column:* The learned solution of the predator population y_1 by Control PINN. *Third column:* The reference solution of the prey population y_2^*. *Fourth column:* The learned solution of the prey population y_2 by Control PINN. *Fifth column:* The learned control u_2 of the prey population y_2 by Control PINN.

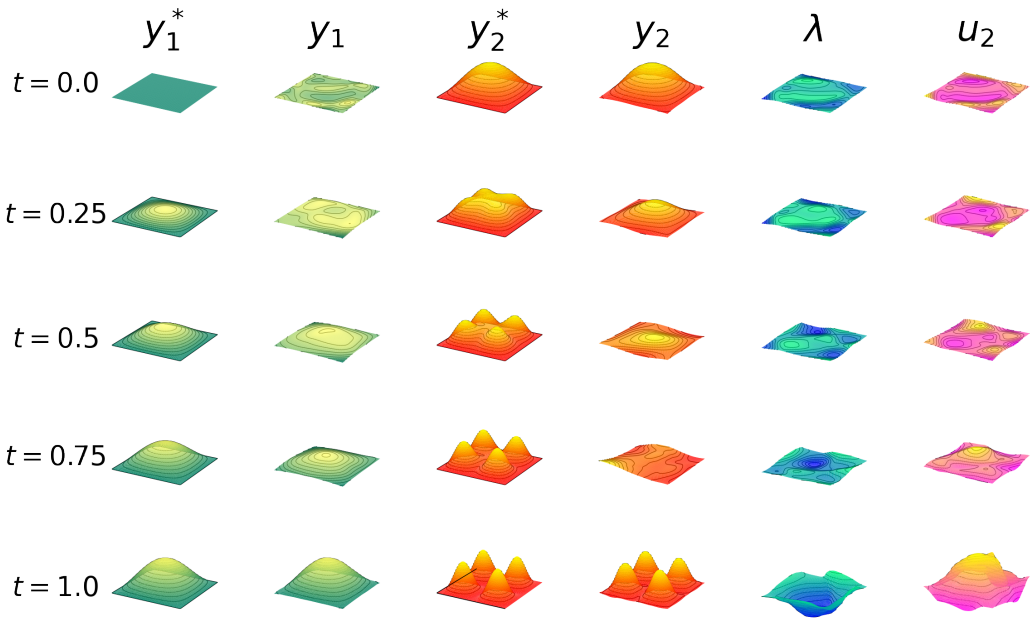


Figure 11: *Two-dimensional predator prey problem:* Surface plot comparison at varying time points with included system push back λ. *First column:* The analytical solution of the predator y_1^* population at varying time snapshots. *Second column:* The learned solution by Control PINN of the predator y_1 population at varying time snapshots. *Third column:* The analytical solution of the prey y_2^* population at varying time snapshots. *Fourth column:* The analytical solution by Control PINN of the prey y_2 population at varying time snapshots. *Fifth column:* The system push back λ from the control u_2 on the prey population y_2 by Control PINN. *Sixth column:* The control u_2 of the prey population y_2 by Control PINN.

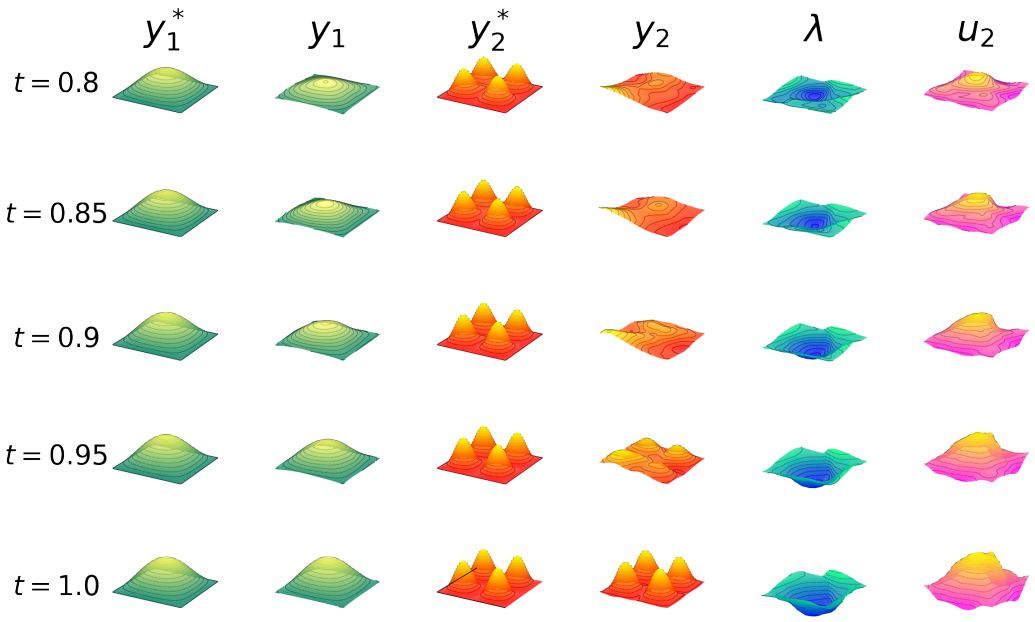


Figure 12: *Two-dimensional predator prey problem:* Surface plot comparison at varying time points near the end of the time span with included system push back λ. *First column:* The analytical solution of the predator y_1^* population at varying time snapshots. *Second column:* The learned solution by Control PINN of the predator y_1 population at varying time snapshots. *Third column:* The analytical solution of the prey y_2^* population at varying time snapshots. *Fourth column:* The analytical solution by Control PINN of the prey y_2 population at varying time snapshots. *Fifth column:* The system push back λ from the control u_2 on the prey population y_2 by Control PINN. *Sixth column:* The control u_2 of the prey population y_2 by Control PINN.