mathematics

MDPI

Article

Pontryagin Neural Networks with Functional Interpolation for Optimal Intercept Problems

Andrea D'Ambrosio [1,2], Enrico Schiassi [2], Fabio Curti [1] and Roberto Furfaro [2,3,*]

[1] School of Aerospace Engineering, Sapienza University of Rome, 00138 Rome, Italy; andrea.dambrosio@uniroma1.it (A.D.); fabio.curti@uniroma1.it (F.C.)
[2] System & Industrial Engineering, University of Arizona, Tucson, AZ 85721, USA; eschiassi@email.arizona.edu
[3] Aerospace & Mechanical Engineering, University of Arizona, Tucson, AZ 85721, USA
* Correspondence: robertof@arizona.edu; Tel.: +1-520-621-2525

Abstract: In this work, we introduce Pontryagin Neural Networks (PoNNs) and employ them to learn the optimal control actions for unconstrained and constrained optimal intercept problems. PoNNs represent a particular family of Physics-Informed Neural Networks (PINNs) specifically designed for tackling optimal control problems via the Pontryagin Minimum Principle (PMP) application (e.g., indirect method). The PMP provides first-order necessary optimality conditions, which result in a Two-Point Boundary Value Problem (TPBVP). More precisely, PoNNs learn the optimal control actions from the unknown solutions of the arising TPBVP, modeling them with Neural Networks (NNs). The characteristic feature of PoNNs is the use of PINNs combined with a functional interpolation technique, named the Theory of Functional Connections (TFC), which forms the so-called PINN-TFC based frameworks. According to these frameworks, the unknown solutions are modeled via the TFC's constrained expressions using NNs as free functions. The results show that PoNNs can be successfully applied to learn optimal controls for the class of optimal intercept problems considered in this paper.

Keywords: Pontryagin Neural Networks; Physics-Informed Neural Networks; functional interpolation; Theory of Functional Connections; Extreme Learning Machine; optimal control

Citation: D'Ambrosio, A.; Schiassi, E.; Curti, F.; Furfaro, R. Pontryagin Neural Networks with Functional Interpolation for Optimal Intercept Problems. *Mathematics* **2021**, *9*, 996. https://doi.org/10.3390/math9090996

Academic Editor: Georgios Tsekouras

Received: 29 March 2021
Accepted: 26 April 2021
Published: 28 April 2021

1. Introduction

Over the last century, optimal control theory has garnered interest in many fields of study. Understanding the dynamics of systems and optimization methods is a difficult challenge for researchers in various scientific fields, from space guidance to epidemiology. Many numerical methods have been developed for optimizing the cost function of the optimal control problems, which usually depends on the state and control variables governing the problem. In general, solution methods for optimal control problems fall into two categories: direct methods and indirect methods [1]. In the direct method, the system's state and control are discretized and, thus, the problem is transcribed into a Non-Linear Optimization Problem or Non-Linear Programming Problem (NLP). These NLPs can be solved using well-known optimization techniques, such as Trust Region Method, Nelder–Mead Method, or Interior Point Methods [2]. The optimal solution is found by transcribing an optimization problem from infinite-dimensional to finite-dimensional, according to these techniques. Conversely, indirect methods are based on the first-order necessary conditions retrieved by direct application of *Pontryagin Minimum Principle* (PMP). The necessary conditions result in a Two-Point Boundary Value Problem (TPBVP) in the state-costate pair that, if an analytical solution does not exist, must be solved via numerical techniques (e.g., single and multiple shooting methods [3,4], orthogonal collocation [5], or pseudo-spectral methods [6]). Generally, the indirect method provides a solution that is guaranteed to be optimal by definition. This is the reason why it should be preferred with respect to the direct method. However, the numerical solution of the TPBVP is often

complicated to obtain because of how sensitive it is to initial guesses, which is required for a numerical solution. Moreover, it is also challenging to provide good initial guesses for the costates, since they do not represent any physical quantity. On the contrary, although the direct method usually calculates solutions that are not exactly optimal, they are close enough and more straightforward to obtain. Thus, the direct method is more prevalently applied in the aerospace community. Furthermore, path and control constraints can be more easily included within the direct method formulation. Luckily, the two methods can also be combined. Indeed, the solution obtained with the direct method can be exploited as an initial guess for the indirect method, which can better refine the solution and, at the same time, guarantee optimality.

In this work, we will tackle optimal intercept problems applying the PMP and modeling a Neural Network (NN) representation of the state-costate pair, for which the residuals of the TPBVP are as close to zero as possible. This will be done by employing *Physics-Informed Neural Networks (PINN)*. We will use a particular PINN framework, as developed by the authors, called *Extreme Theory of Functional Connections (X-TFC)*. Here, PINNs are specifically trained to learn optimal control actions that satisfy the PMP and, therefore, the optimality is guaranteed by learning the solutions of the associated TPBVP. For this reason, we name these PINNs *Pontryagin Neural Networks (PoNNs)*. More specifically, this paper aims to show the effectiveness of PoNNs in learning optimal control actions for intercept problems, achieving results that were comparable with the other state-of-the-art methods. Additionally, we will show how PoNNs can represent an alternative to these methods and be used in synergy with them.

PINNs are machine learning methods that insert physics into data-driven functional representations of input–output paring collections. The term PINN has been coined to indicate those NNs for which the loss function includes the physics as a regularization term, as defined by Raissi et al. [7]. For instance, suppose facing a regression problem using a NN and having some experimental data representing a physical phenomenon modeled via DEs. The common machine learning approach would be to approximate the data with a NN trained via minimizing the Mean Squared Error (MSE) as the loss function. However, in this case, there are no guarantees that the physics of the problem is not violated. Therefore, PINNs are introduced to take the DEs describing the physics behind the collected data into account. Indeed, the implicit forms of the DEs considered are embedded within the loss function as additional terms, which penalize the training when the DEs and its constraints (e.g., the Initial Conditions or ICs and/or Boundary Conditions or BCs) are violated. This approach is also referred to as data-driven solution of DEs. In the limit when only the residual of the DEs is considered (e.g., data are not available), PINNs learn the solutions of problems involving Ordinary (ODEs) and Partial (PDEs) DEs in a solely physics-driven fashion. In this regard, PoNNs represent a particular case of PINNs, where the DEs modeling the physics constraints are obtained via the application of the PMP. On the other hand, the data-driven parameters discovery of DEs (e.g., inverse problems) [7,8] refers to the estimation of parameters appearing within DEs (e.g., in the orbit determination framework). The major drawback of the standard PINN framework, as introduced by Raissi et al. [7], is that the DE constraints are not analytically satisfied and, therefore, they need to be simultaneously learned with the DE solution within the domain. Thus, we deal with competing objectives during the network's training: learning the DE latent solution within the domain and learning the DE latent solution on the boundaries. This leads to unbalanced gradients during the network training via gradient-based techniques that causes PINNs to often struggle to accurately learn the underlying DE solution [9]. It is well known that gradient-based methods may diverge when multiple competing objectives are present or may eventually get stuck in limit cycles [10,11]. In [9], in order to overcome this issue, the authors developed a learning rate annealing algorithm that employs gradient statistics to adaptive assign appropriate weights to different terms (e.g., DE residuals within the domain and DE residuals on the boundaries) in the PINNs loss function during the network training. Here, we propose to employ a different and more

robust PINN model, as developed by the authors, which merges NN and TFC [12,13]. That is, we will use a PINN-TFC based model called X-TFC. This PINN-TFC based framework approximates the latent solution of the DEs via the so-called Constrained Expressions (CEs). The CEs are the critical ingredient of the *Theory of Functional Connections or TFC*, developed by Mortari [14], which is a method for functional interpolation where functions are approximated using these CEs. A constrained expression is a functional that is made up of the sum of two terms. The first term is a free-function, while the second term is a functional that analytically fulfils the constraints independently from the choice of the free-function [14,15]. Thanks to the CEs analytically embedding the constraints, TFC has many applications. In particular, TFC is widely used for the solution of DEs, because the CEs remove the "curse of the equation constraints" [16–18]. Additionally, TFC has already been used to solve several classes of optimal control space guidance problems such as energy optimal landing on large and small planetary bodies [19,20], fuel optimal landing on large planetary bodies [21], and energy optimal relative motion problems subject to Clohessy–Wiltshire dynamics [22]. For solving DEs, the standard TFC method uses a linear combination of orthogonal polynomials, such as Legendre or Chebyshev polynomials [16,17], as a free function. Using orthogonal polynomials as a free-function makes the standard TFC framework suffer from the curse of dimensionality, especially when solving large-scale PDEs. In order to overcome this limitation, X-TFC uses a single layer NN trained via the Extreme Learning Machine (ELM) algorithm [23] to represent the free-function. This is the reason why X-TFC can be labeled as a PINN framework. Although, in this work, X-TFC is used to learn the state-costate solution that solves the TPBVP arising from the application of the PMP solely in a physics-driven fashion, being a PINN framework, it can also be used for the data-driven solution of DEs, data-driven DE parameter discovery, and data-driven discovery of DEs [8]. The manuscript is organized, as follows. First, we explain how PoNNs are modeled and trained to learn control actions for optimal control problems tackled via the indirect method. PoNNs are then employed to learn optimal control actions for the minimum time-energy optimal intercept problem, both with unbounded and bounded control. Finally, the results will be discussed in the remaining sections.

2. Pontryagin Neural Networks for Optimal Control Problems

PoNNs represent a particular family of PINNs specifically designed to tackle optimal control problems via the Pontryagin Minimum Principle (PMP) application (e.g., indirect method). The term "Pontryagin Neural Networks" (PoNNs) is coined here by the authors to refer in a more compact form to "PINNs designed to learn optimal control via indirect method". The PMP provides first-order necessary optimality conditions, which result in a Two-Point Boundary Value Problem (TPBVP). More precisely, PoNNs learn the optimal control actions from the unknown solutions of the arising TPBVP, modeling them with Neural Networks (NNs). That is, PoNNs model a NN representation of functions for which the residuals of the TPBVP are as close to zero as possible, i.e., the solutions of the TPBVP. This NN representation of the TPBVP unknown solutions is done combining PINNs with TFC. Together, PINNs and TFC form the so-called PINN-TFC based framework. A generic differential equation's unknown solutions are modeled via the TFC's constrained expressions using NNs as free functions, according to this framework. When Deep NNs (DNNs) are used as free function, the PINN-TFC framework is known as *Deep-TFC* [12]. When shallow NNs, trained via Extreme Learning Machine (ELM) algorithm, are used as free function, the PINN-TFC framework is known as *Extreme TFC (X-TFC)* [13]. In particular, PoNNs uses the X-TFC approach.

In this section, all of the key elements of the PoNNs will be explored in detail. First, how OCPs are faced using the indirect method is presented. Moreover, bounds on the control will also be taken into consideration. We will then explain how standard PINNs and X-TFC are employed to learn the unknown solutions of the arising TPBVP, which consequently lead to retrieve the optimal control actions.

2.1. Optimal Control Problems via Indirect Method

In this section, Unconstrained Optimal Control Problems are first introduced and then the extension to Constrained Optimal Control Problems is presented following the procedure that was proposed in [24]. UOCPs are generally based on a cost function, whose expression is given by:

$$\mathcal{J} = \mathbf{\Phi}(\boldsymbol{x}(t_0), t_0, \boldsymbol{x}(t_f), t_f) + \int_{t_0}^{t_f} \mathscr{L}(\boldsymbol{x}(t), \boldsymbol{u}(t), t)\, dt \tag{1}$$

subject to the dynamic constraints, expressed as,

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \tag{2}$$

and the boundary conditions,

$$\mathbf{\Phi}(\boldsymbol{x}(t_0), t_0) = \mathbf{\Phi}_0 \tag{3}$$
$$\mathbf{\Phi}(\boldsymbol{x}(t_f), t_f) = \mathbf{\Phi}_f \tag{4}$$

where $\boldsymbol{x}(t)$ is the state vector, $\boldsymbol{u}(t)$ is the control vector, t is the independent variable, t_0 is the initial time, and t_f is the final time. The terms $\mathbf{\Phi}$ and $\mathscr{L}$, appearing in Equation (1) are the end-point cost and Lagrangian, respectively [2]. If the Pontryagin Maximum (or Minimum) Principle (PMP) [25] is employed within the indirect method, then the Hamiltonian of the problem needs to be defined as

$$H = \mathscr{L} + \boldsymbol{\lambda}^{\mathrm{T}} \boldsymbol{f} \tag{5}$$

where $\boldsymbol{\lambda}$ represent the costate (or adjoint variables). According to the first-order optimality conditions of the PMP, the optimal control is derived, as follows,

$$\frac{\partial H}{\partial \boldsymbol{u}} = 0 \tag{6}$$

Once the Equation (6) is solved for $\boldsymbol{u}$, the result is plugged into (5), and the remaining first-order necessary conditions for the state and costate variables can be obtained are derived,

$$\dot{\boldsymbol{x}} = \frac{\partial H}{\partial \boldsymbol{\lambda}} \tag{7}$$
$$\dot{\boldsymbol{\lambda}} = -\frac{\partial H}{\partial \boldsymbol{x}} \tag{8}$$

Finally, the transversality conditions have to be applied, if any (e.g., when the corresponding state variable is free). The possible transversality conditions are the following,

$$\boldsymbol{\lambda}(t_0) = -\frac{\partial \mathcal{J}}{\partial \boldsymbol{x}_0} \tag{9}$$
$$H(t_0) = \frac{\partial \mathcal{J}}{\partial t_0} \tag{10}$$
$$\boldsymbol{\lambda}(t_f) = \frac{\partial \mathcal{J}}{\partial \boldsymbol{x}_f} \tag{11}$$
$$H(t_f) = -\frac{\partial \mathcal{J}}{\partial t_f} \tag{12}$$

Equations (7) and (8), along with the transversality conditions on the Hamiltonian, represent a BVP, which, in the proposed method, is solved using PINNs, in particular X-TFC. One should note that the transversality conditions on the costate will be a priori satisfied with the X-TFC approach, as will be explained below. Thus, they do not explicitly appear in the BVP.

To consider control constraints for COCPs, some modifications should be introduced to the previous formulation (see [24] for more detail). First, we need to define a new unconstrained control variable w_i for each control constraint that is present in the problem $d_i(u)$ belonging to the interval $[d_i^-, d_i^+]$, with $i = 1, \ldots, q$. The idea is to replace the constraint d_i with a saturation function, as follows

$$d_i = \phi_i(w_i) \tag{13}$$

$$\phi_i(w_i) = d_i^+ - \frac{d_i^+ - d_i^-}{1 + \exp(s w_i)} \quad \text{with} \quad s = \frac{c}{d_i^+ - d_i^-} \tag{14}$$

where ϕ_i is a smooth and monotonically increasing function and c is a constant value, which useful for modifying the slope of ϕ_i at $w_i = 0$. The advantages of using a saturation function is that it is defined within the range $u_i \in [d_i^-, d_i^+]$ and it approaches the saturation limits asymptotically for $w_i \to \pm\infty$. Other saturation function, such as tanh, can also be employed. One can note that control constraints represent input constraints in the sense that the order of derivative in which the saturation function appears in the Hamiltonian is zero, as shown by Equation (16). Because a new control variable is introduced, a new term in the cost function, called the regularization term, is added, as follows,

$$\bar{\mathcal{J}} = \mathcal{J} + \epsilon \int_{t_0}^{t_f} ||w_i||^2 \, dt \tag{15}$$

where ϵ is the regularization parameter. Having a low value of ϵ means that the new control problem with the new unconstrained control becomes similar to the original constrained control problem, as desired. However, this parameter can affect either the saturation function or its derivative, according to the order of the constraint. Indeed, decreasing ϵ results in higher values of the augmented control w, which would approach the asymptotic limits. Therefore, if ϵ approaches 0, w could tend to infinity, leading to numerical problems. To summarize, even if ϵ close to 0 leads to having the solution of the new UOCP closer to the one of the original COCP, one should be careful to set the value of ϵ. For more information about the influence of ϵ on the control and the other optimization parameters, the reader can refer to Ref. [26]. In order to obtain accurate solutions, a continuation procedure on ϵ is usually exploited to bring the new UOCP control problem to match the original COCP. According to Equation (15), the new Hamiltonian of the problem is,

$$H = \mathscr{L} + \boldsymbol{\lambda}^{\mathrm{T}} \boldsymbol{f} + \epsilon ||w_i||^2 + \sum_{i=1}^{q} \nu_i (d_i(u) - \phi_i(w_i)) \tag{16}$$

where ν_i represents additional multipliers that take the new equality constraints into account. It is worth to note that it can happen that the optimality condition on the control $\boldsymbol{u}$ ($\frac{\partial H}{\partial \boldsymbol{u}} = 0$) is represented by a trascendental form and is no longer able to provide a closed form solution for $\boldsymbol{u}$. Because a new unconstrained control has been added, another equation should be added to the optimality condition on the control,

$$\frac{\partial H}{\partial w_i} = 2\epsilon w_i - \nu_i \phi_i'(w_i) = 0 \tag{17}$$

where ϕ_i' indicates the derivative of the saturation function with respect to w_i. Moreover, the following constraint equation has to be added to the BVP,

$$d_i(u) - \phi_i(w_i) = 0 \tag{18}$$

Finally, the new UOCP is totally defined and the lower the value of ϵ is, the more the solution of the new UOCP approaches the solution of the original COCP.

2.2. Physics-Informed Neural Networks and Functional Interpolation

Raissi et al. have defined PINNs [7], following the original idea that was introduced by Lagaris et al. [27], to tackle two main problems categories: data-driven discovery and data-driven solution of PDEs. Even though, in [7], PINNs were introduced to tackle problems governed by PDEs, the same methodology is also used to solve problems involving ODEs. PINN uses a single neural network (either shallow or deep) to approximate the solution of DEs. In particular, the training set is represented by points randomly sampled from the full high-resolution data-set and by collocation points that were randomly chosen to enforce the BCs, the DE, and its related constraints. Once the network's output is obtained, it is substituted in the residual of both the DE and BCs. The weights and bias parameters are learned through the training process that minimizes the physics-based loss function via gradient-based methods. Hence, the main characteristic of PINNs is to include the DEs and BCs describing the physics of the problem in their cost functions.The theoretical formulation to learn a generic TPBVP is reported below in order to better understand how a PINN works.

2.2.1. PINN for TPBVPs

The vector differential equation of a generic TPBVP in the time domain can be expressed, in its implicit form, as follows,

$$F_i\Big(t, y_j(t), \dot{y}_j(t), \ddot{y}_j(t)\Big) = 0 \qquad \text{subject to:} \begin{cases} y_j(t_0) = y_{0_j} \\ y_j(t_f) = y_{f_j} \\ \dot{y}_j(t_0) = \dot{y}_{j_0} \\ \dot{y}_j(t_f) = \dot{y}_{f_j} \end{cases} \tag{19}$$

where i is the number of differential equations forming the ODEs system, and j is the number of unknown functions ($y_j(t)$) that are the solutions of the system. The independent variable is the time $t \in [t_0, t_f]$. According to the standard PINN framework, the unknown solutions of the system (19) are approximated with a NN. Following [28], as we deal with a system of ODEs, we use a Single Input and Multiple Outputs (SIMO) NN. More precisely, the input is represented by the time t, and the outputs by the y_j's. That is,

$$y_j(t) = y_j^{\mathrm{NN}}(t, \boldsymbol{\theta}) \tag{20}$$

where $\boldsymbol{\theta}$ are the NN parameters that are trained via gradient based methods. Equation (20) represents the PoNN. That is, PoNN is a particular NN that is trained to satisfy the PMP by learning the solution of the arising TPBVP. To train the PoNN, the Mean Square Error (MSE) is to be minimized with respect to the NN parameters,

$$\min_{\boldsymbol{\theta}} MSE \tag{21}$$

where,

$$MSE = MSE_{\mathcal{F}} + MSE_{BC_0} + MSE_{BC_f} \tag{22}$$

with,

$$MSE_{\mathcal{F}} = \frac{1}{N_{\mathcal{F}}} \sum_{i=1}^{N_{\mathrm{eq}}} \sum_{k=1}^{N_{\mathcal{F}}} \Big| F_i\Big(t_{\mathcal{F}}^k, y_j^{\mathrm{NN}}(t_{\mathcal{F}}^k), \dot{y}_j^{\mathrm{NN}}(t_{\mathcal{F}}^k), \ddot{y}_j^{\mathrm{NN}}(t_{\mathcal{F}}^k)\Big)\Big|^2 \tag{23}$$

$$MSE_{BC_0} = \sum_{j=1}^{N_y} \Big| y_{0_j} - y_j^{\mathrm{NN}}(t_0) \Big|^2 + \sum_{j=1}^{N_y} \Big| \dot{y}_{0_j} - \dot{y}_j^{\mathrm{NN}}(t_0) \Big|^2 \tag{24}$$

$$MSE_{BC_f} = \sum_{j=1}^{N_y} \Big| y_{f_j} - y_j^{\mathrm{NN}}(t_f) \Big|^2 + \sum_{j=1}^{N_y} \Big| \dot{y}_{f_j} - \dot{y}_j^{\mathrm{NN}}(t_f) \Big|^2 \tag{25}$$

Here, $MSE_{\mathcal{F}}$, MSE_{BC_0}, and MSE_{BC_f} are the MSEs that take the dynamics (e.g., the ODEs forming the TPBVP, for our purposes), the initial conditions, and the final conditions, respectively, into account. N_{eq} is the number of ODEs in the system (19), N_y is the number of unknown solutions y_j's, and $\{t_{\mathcal{F}}^k\}_{k=1}^{N_{\mathcal{F}}}$ represent the training points for each $\{F_i\}_{i=1}^{N_{\text{eq}}}$. Finally, the derivatives of the $y_j^{\text{NN}}(t,\boldsymbol{\theta})$'s with respect to time are computed via automatic differentiation.

The attentive reader can clearly see how the MSE is made by multiple objects that, competing during the training, would likely lead to not properly learn the underlying DE solution. Furthermore, when dealing with a system of ODEs, the number of competing objectives increases: learning the ODE latent solution within the domain for all of the equations of the systems, and learning the ODE latent solution on the boundaries for all the unknowns. As previously mentioned, in this research, all of these issues are removed thanks to the combination of the classic PINN methods with the TFC.

2.2.2. X-TFC for TPBVPs

As previously stated, in this work we will use the PINN-TFC based methods, called X-TFC. Here, how to apply X-TFC to solve a generic TPBVP in the time domain is presented in details. When considering the system that is expressed by (19), the first step of the X-TFC method follows the derivation of the constrained expressions, and their derivatives, as developed by TFC,

$$y_j^{(\ell)}(t) = g_j^{(\ell)}(t) + \sum_{k=1}^{n_j} \eta_{k_j} s_k^{(\ell)}(t) \tag{26}$$

where the ℓ superscripts refers to the ℓth derivative with respect to the independent variable, n_j is the number of constraints for the jth unknown function and/or its derivatives, η_{k_j} are some coefficients, and $g_j(t)$ is the free function. According to [14], the functions $s_k(t)$, called support functions, can be selected, as following,

$$s_k(t) = t^{k-1} \tag{27}$$

Once the constrained expression is defined, substituting the constraints on $y_j(t)$ and/or its derivatives at the time boundary (e.g., t_0 and t_f) into the constrained expression creates a set of linear algebraic equations, which is then solved for the η_{k_j} coefficients. Once the η_{k_j} coefficients are calculated, the boundary constraints of Equation (19) are analytically embedded within then constrained expression. Now, substituting the constrained expressions into the F_i differential equations transforms them into new set of equations that we define with $\tilde{F}_i$. This new set of equations is now just a function of the free functions $g_j(t)$, and their derivatives, and the independent variable t. That is,

$$\tilde{F}_i\Big(t, g_j(t), \dot{g}_j(t), \ddot{g}_j(t)\Big) = 0 \tag{28}$$

This vector differential equation is unconstrained, because the boundary conditions are embedded within it through the derived η_{k_j} values. To solve Equation (28), X-TFC uses as free functions, $g_j(t)$, a shallow Single Input Single Output (SISO) NN trained via ELM algorithm [23]. That is,

$$g_j(z) = \sum_{q=1}^{L} \beta_{j,q}\sigma\left(w_q z + b_q\right) = \begin{bmatrix}\sigma_1 \\ \vdots \\ \sigma_L\end{bmatrix}^T \boldsymbol{\beta}_j = \boldsymbol{\sigma}^{\intercal}(z)\boldsymbol{\beta}_j \tag{29}$$

where b_q is the bias of the qth hidden neuron, $w_q \in \mathbb{R}$ is the input weights vector connecting the qth hidden neuron and the input nodes, $\beta_q \in \mathbb{R}$ with $q = 1, \ldots, L$ is the qth output weight connecting the qth hidden neuron and the output node, L is the number of hidden

neurons, and $\sigma_j(\cdot)$ are activation functions for the $g_j(z)$ free chosen function. Equation (29) represents the X-TFC key difference with respect to the standard TFC, and the reason why the X-TFC framework belongs to the family of the PINN frameworks. Using a NN as free function instead of orthogonal polynomials allows the X-TFC approach to (1) highly reduce the curse of dimensionality in ODE/PDE problems with respect the standard TFC and (2) be framed as PINN method. Because we are using the ELM algorithm to train the NN, the only unknowns to compute are the output weights $\boldsymbol{\beta_j} = \left[\beta_{j,1}, \ldots, \beta_{j,L}\right]^{\mathrm{T}}$. The attentive reader will notice the use of a different independent variable, z. This happens because the domain of the activation functions z and the problem domain t are usually not coincident. Therefore, we must map the domain t into the domain z and vice-versa,

$$z = z_0 + c(t - t_0) \quad \longleftrightarrow \quad t = t_0 + \frac{1}{c}(z - z_0) \tag{30}$$

where c is a mapping coefficient that is,

$$c = b^2 = \frac{z_f - z_0}{t_f - t_0} \tag{31}$$

Because c is always a positive number, it is convenient to rewrite it as $c = b^2$. Because to the mapping, all subsequent derivatives of $g_j(t)$ are defined, as follows,

$$\frac{\mathrm{d}^n g_j}{\mathrm{d}t^n} = \boldsymbol{\beta}_j^{\mathrm{T}} \frac{\mathrm{d}^n \boldsymbol{\sigma}_j(z)}{\mathrm{d}z^n} \left(\frac{\mathrm{d}z}{\mathrm{d}t}\right)^n = \boldsymbol{\beta}^{\mathrm{T}} \frac{\mathrm{d}^n \boldsymbol{\sigma}_j(z)}{\mathrm{d}z^n} (b^2)^n \tag{32}$$

It is to be noticed that, for the optimal control problems where the final time is free, the mapping coefficient becomes an unknown that must be computed along with all of the $\boldsymbol{\beta}_j$'s. Hence, the transformation of the free function and its derivatives between the t domain to the z domain can be summarized, as follows,

$$\begin{cases} g_j(t) = \boldsymbol{\sigma}_j^{\mathrm{T}}(z)\boldsymbol{\beta}_j \\ \dot{g}_j(t) = b^2\, \boldsymbol{\sigma'}_j^{\mathrm{T}}(z)\boldsymbol{\beta}_j \\ \ddot{g}_j(t) = b^4\, \boldsymbol{\sigma''}_j^{\mathrm{T}}(z)\boldsymbol{\beta}_j \end{cases} \tag{33}$$

where $\boldsymbol{\sigma}'_j(z)$ is the abbreviation for $\frac{\mathrm{d}\boldsymbol{\sigma}_j(z)}{\mathrm{d}z}$. Equation (28) in the z domain then becomes,

$$\tilde{F}_i(z, \boldsymbol{\beta}_j) = 0 \tag{34}$$

The z domain must be discretized into n points in order to numerically solve this TPBVP. One can note that z is usually discretized with equally spaced points or using the Chebyshev-Gauss-Lobatto points, but one can use any desired quadrature scheme. The unconstrained set of differential equations in Equation (34) can then be expressed as loss functions at each discretization point,

$$\mathcal{L}_i(\boldsymbol{\beta}_j) = \begin{Bmatrix} \tilde{F}_i(z_0, \boldsymbol{\beta}_j) \\ \vdots \\ \tilde{F}_i(z_d, \boldsymbol{\beta}_j) \\ \vdots \\ \tilde{F}_i(z_n, \boldsymbol{\beta}_j) \end{Bmatrix} \tag{35}$$

Now, by combining the differential equation for each dimension, an augmented loss function can be written, as follows,

$$\mathbb{L} = \begin{bmatrix} \mathcal{L}_1, & \ldots, & \mathcal{L}_i, & \ldots, & \mathcal{L}_{N_{eq}} \end{bmatrix}^{\mathrm{T}} \tag{36}$$

and imposing that to be a true solution, this vector should be equal to **0**. This allows for the $\boldsymbol{\beta}_j$ coefficients to be learnt via different optimization schemes, e.g., least-square for linear problems [16] and iterative least-squares for non-linear problems [17]. If the iterative least-square method is needed, then the estimation of the unknowns is updated at each iteration, as follows,

$$\Xi_{k+1} = \Xi_k + \Delta\Xi_k \tag{37}$$

where Ξ is the augmented vector containing all the vector $\boldsymbol{\beta}_j$ (and eventually the square root of mapping coefficient, b) and the k subscript indicates the current iteration. In general, the $\Delta\Xi_k$ term can be obtained by performing classic linear least-square at each step of the iterative least-square procedure,

$$\Delta\Xi_k = -\Big(\mathbb{J}(\Xi_k)^{\mathrm{T}}\mathbb{J}(\Xi_k)\Big)^{-1}\mathbb{J}(\Xi_k)^{\mathrm{T}}\mathbb{L}(\Xi_k) \tag{38}$$

where $\mathbb{J}$ is the Jacobian matrix containing the derivatives of the losses with respect to all of the unknowns. The Jacobian matrix can be computed either by hand or by means of computing tools, such as the Symbolic or the Automatic Differentiation Toolbox. The iterative process is repeated until the following condition is met,

$$L_2[\mathcal{L}(\Xi_k)] < \varepsilon \tag{39}$$

where ϵ defines some user prescribed tolerance. Once we are sure that the convergence is achieved by meeting the criteria $L_2[\mathcal{L}(\Xi_k)] < \epsilon$, the nonlinear DE can be solved using the following criteria $L_2[\mathcal{L}(\Xi_{k+1})] > L_2[\mathcal{L}(\Xi_k)]$. Doing so, round-off error causes the convergence and the use of the user prescribed tolerance ϵ is completely avoided. Thus, the solution accuracy is pushed to the limit (e.g., it is possible to reach the best solution accuracy possible for the specific DE).

Using the X-TFC method for learning the solution of the TPBVP, arising from the PMP, Equation (26) represents the PoNN. For the convenience of the reader, Figure 1 presents a schematic summarizing how the PoNNs are modeled and trained for learning optimal control actions for a generic unconstrained OCP.

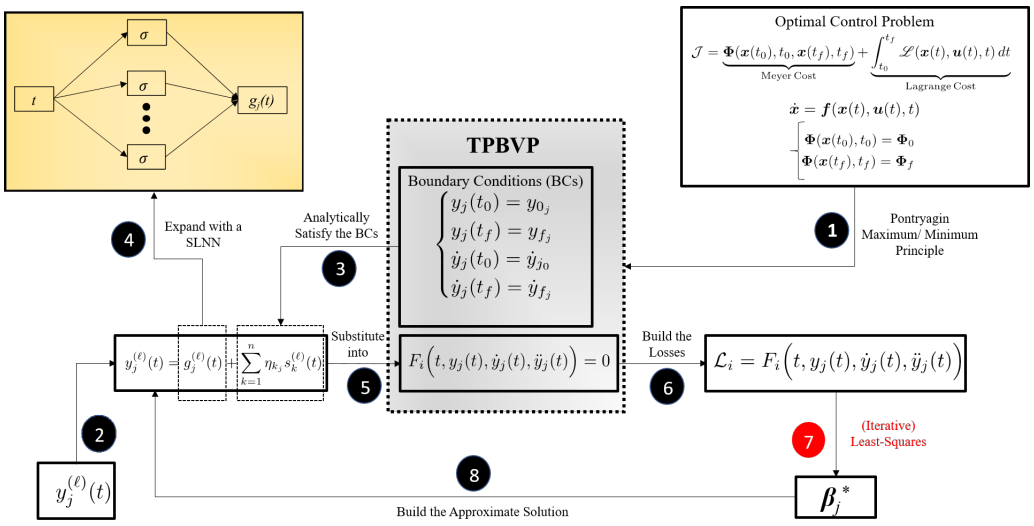


Figure 1. Schematic summarizing how the PoNNs are modeled and trained for learning optimal control actions for a generic unconstrained OCP.

3. Optimal Intercept Problem: Formulations and Results

In this section, we show how PoNNs are modeled and trained to learn optimal control actions for the minimum-time energy optimal intercept problem [19]. Unbounded and bounded control actions will both be considered.

All of the problems tackled in this manuscript have been coded in Matlab R2020a, and ran with an Intel Core i7 - 9700 CPU PC with 64 GB of RAM.

3.1. Unconstrained Problem

The optimal control problem that is considered here is the minimum time-energy optimal intercept problem. This problem describes the motion of an interceptor (M) chasing a target (T) both subjected to the same gravitational acceleration. The equations of motion in an inertial reference frame can be written, as follows,

$$\dot{\boldsymbol{r}} = \boldsymbol{v} \tag{40}$$

$$\dot{\boldsymbol{v}} = \boldsymbol{a}_T - \boldsymbol{a}_M \tag{41}$$

where $\boldsymbol{r} = \boldsymbol{r}_T - \boldsymbol{r}_M$ and $\boldsymbol{v} = \boldsymbol{v}_T - \boldsymbol{v}_M$ are, respectively, the relative distance and velocity between the target and the interceptor, while $\boldsymbol{a}_T$ and $\boldsymbol{a}_M$ are the thrust accelerations. The optimization problem to be addressed is a minimum time-energy optimal problem,

$$\min \quad \mathcal{J} = \Gamma t_f + \frac{1}{2}\int_{t_0}^{t_f} (\boldsymbol{a}_M^T \boldsymbol{a}_M)\, dt \tag{42}$$

subject to

$$\begin{aligned} &\dot{\boldsymbol{r}} = \boldsymbol{v} \\ &\dot{\boldsymbol{v}} = \boldsymbol{a}_T - \boldsymbol{a}_M \\ &t_0 \leq t \leq t_f \\ &\boldsymbol{r}(t_0) = \boldsymbol{r}_0 \\ &\boldsymbol{v}(t_0) = \boldsymbol{v}_0 \\ &\boldsymbol{r}(t_f) = \boldsymbol{0} \end{aligned} \tag{43}$$

where Γ is a non-negative coefficient, t_f is the free final time to be optimized, and the final relative velocity is chosen to be free. The Hamiltonian for this problem is,

$$H(t) = \frac{1}{2}\boldsymbol{a}_M^T \boldsymbol{a}_M + \boldsymbol{\lambda}_r^T \boldsymbol{v} + \boldsymbol{\lambda}_v^T (\boldsymbol{a}_T - \boldsymbol{a}_M) \tag{44}$$

The control optimality comes from,

$$\frac{\partial H}{\partial \boldsymbol{a}_M} = \boldsymbol{0} \quad \rightarrow \quad \boldsymbol{a}_M = \boldsymbol{\lambda}_v \tag{45}$$

By plugging Equation (45) into Equation (44), the Hamiltonian can be rewritten as,

$$H(t) = -\frac{1}{2}\boldsymbol{\lambda}_v^T \boldsymbol{\lambda}_v + \boldsymbol{\lambda}_r^T \boldsymbol{v} + \boldsymbol{\lambda}_v^T \boldsymbol{a}_T \tag{46}$$

The first-order necessary conditions for the state and costate are,

$$\dot{\boldsymbol{r}} = \frac{\partial H}{\partial \boldsymbol{\lambda}_r} = \boldsymbol{v} \tag{47}$$

$$\dot{\boldsymbol{v}} = \frac{\partial H}{\partial \boldsymbol{\lambda}_v} = \boldsymbol{a}_T - \boldsymbol{\lambda}_v \tag{48}$$

$$\dot{\boldsymbol{\lambda}}_r = -\frac{\partial H}{\partial \boldsymbol{r}} = \boldsymbol{0} \tag{49}$$

$$\dot{\boldsymbol{\lambda}}_v = -\frac{\partial H}{\partial \boldsymbol{v}} = -\boldsymbol{\lambda}_r \tag{50}$$

In addition, two transversality conditions need to be imposed,

$$H(t_f) + \Gamma = 0 \tag{51}$$

$$\boldsymbol{\lambda}_v(t_f) = \boldsymbol{0} \tag{52}$$

The system that is expressed by Equations (47)–(52) is a TPBVP, representing the physics constraints that drive the PoNN training.

The transversality condition (51) is necessary because the problem is final time free (e.g., the final time is in the cost function). This means that the Hamiltonian at the converged final time must be equal to Γ. As the system is autonomous (e.g., there is no explicit dependency on time in the dynamics), the Hamiltonian must be constant over time (e.g., $H(t) = -\Gamma$). Because $H(t_f) = -\Gamma$ (from the transversality condition), then $H(t) + \Gamma = 0$.

By writing the equations in components ($j = 1, 2, 3$), one can obtain

$$\begin{aligned} \dot{v}_j &= a_{T,j} - \lambda_{v,j} && (53)\\ \dot{\lambda}_{r,j} &= 0 && (54)\\ \dot{\lambda}_{v,j} &= -\lambda_{r,j} && (55)\\ H(t_f) + \Gamma &= 0 && (56) \end{aligned}$$

where Equation (47) is removed since it is redundant when employing the CEs approximation and Equation (52) is also removed as the boundary conditions are analytically satisfied by the CEs. Thus, the states and costates are approximated using the TFC's CEs, as follows

$$\begin{aligned} r_j &= \left(\sigma - \Omega_1\sigma_0 - \Omega_2\sigma_f - \Omega_3\sigma_0'\right)^{\mathrm{T}}\boldsymbol{\beta}_j + \Omega_1 r_{0_j} + \Omega_2 r_{f_j} + \frac{\Omega_3 v_{0_j}}{b^2}\\ v_j &= b^2\left[\left(\sigma' - \Omega_1'\sigma_0 - \Omega_2'\sigma_f - \Omega_3'\sigma_0'\right)^{\mathrm{T}}\boldsymbol{\beta}_j + \Omega_1' r_{0_j} + \Omega_2' r_{f_j} + \frac{\Omega_3' v_{0_j}}{b^2}\right]\\ a_j &= b^4\left[\left(\sigma'' - \Omega_1''\sigma_0 - \Omega_2''\sigma_f - \Omega_3''\sigma_0'\right)^{\mathrm{T}}\boldsymbol{\beta}_j + \Omega_1'' r_{0_j} + \Omega_2'' r_{f_j} + \frac{\Omega_3'' v_{0_j}}{b^2}\right] \end{aligned}$$

$$\begin{aligned} \lambda_{r,j} &= \sigma^{\mathrm{T}}\boldsymbol{\beta}_{r,j}\\ \dot{\lambda}_{r,j} &= b^2\sigma'^{T}\boldsymbol{\beta}_{r,j}\\ \lambda_{v,j} &= \left(\sigma - \sigma_f\right)^{\mathrm{T}}\boldsymbol{\beta}_{v,j} + \lambda_{v_{f,j}}\\ \dot{\lambda}_{v,j} &= b^2\sigma'^{T}\boldsymbol{\beta}_{v,j} \end{aligned}$$

where $\Omega_k(z)$ are called switching functions and they are reported in the Appendix A. Hence, the PoNN's parameter to be learned is,

$$\Xi = \begin{bmatrix}\boldsymbol{\beta}_{r,1} & \boldsymbol{\beta}_{r,2} & \boldsymbol{\beta}_{r,3} & \boldsymbol{\beta}_{\lambda_{r,1}} & \boldsymbol{\beta}_{\lambda_{r,2}} & \boldsymbol{\beta}_{\lambda_{r,3}} & \boldsymbol{\beta}_{\lambda_{v,1}} & \boldsymbol{\beta}_{\lambda_{v,2}} & \boldsymbol{\beta}_{\lambda_{v,3}} & b\end{bmatrix}^{\mathrm{T}}$$

An iterative least-square approach can be used to learn the PoNN's parameter that minimizes the following losses,

$$\begin{aligned} \mathcal{L}_{a,j} &= a_j - a_{T,j} + \lambda_{v,j} && (57)\\ \mathcal{L}_{\lambda_{r,j}} &= \dot{\lambda}_{r,j} && (58)\\ \mathcal{L}_{\lambda_{v,j}} &= \dot{\lambda}_{v,j} + \lambda_{r,j} && (59)\\ \mathcal{L}_{\lambda_H} &= \sum_{j=1}^{3}(\lambda_{r,j} v_j) + \Gamma && (60) \end{aligned}$$

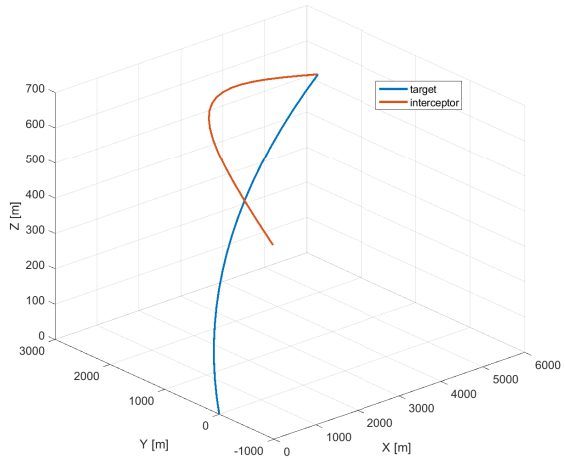
where $j = 1, 2, 3$, $a_j = \dot{v}_j$, and the last equation is meant to be computed at the final time. For what concerns the derivatives of the losses with respect to the unknowns (e.g., the Jacobian matrix), it can be computed either analytically, thanks to symbolic computation, or numerically, by automatic differentiation.

Results

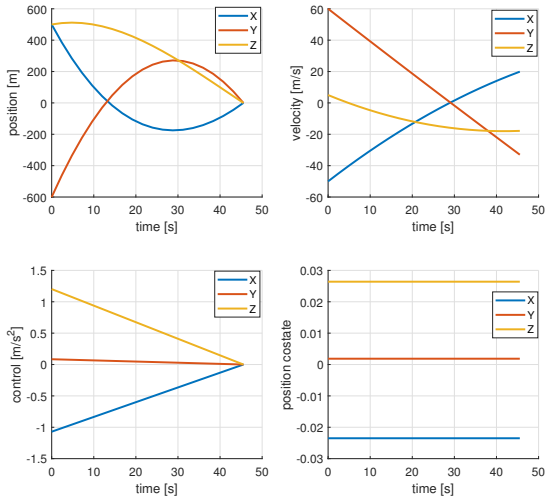
The initial conditions for this problem are taken from Furfaro and Mortari [19]. The initial relative position and velocity vectors are chosen to be respectively $\boldsymbol{r}_0 = [500, -600, 500]$m and $\boldsymbol{v}_0 = [-50, 60, 5]$ m/s, whereas the initial absolute velocity of the target is equal to $\boldsymbol{v}_{T,0} = [100, 100, 10]$ m/s. The target is supposed to be located at the origin of the absolute frame at the initial time instant and is subjected to a constant acceleration command $\boldsymbol{a}_T = [1, -2, 0.1]$ m/s^2. The activation function that is used in this problem is the hyperbolic tangent. The weights and bias are sampled from $\mathcal{U}(-1; 1)$. Because the problem is nonlinear, an initial guess on the unknowns is required to initialize the iterative least-square. In this work, as a first guess, the PoNN's parameters $\boldsymbol{\Xi}$ have been randomly sampled from $\mathcal{U}(0; 1)$. Figure 2 shows the trajectories of target and intercept learnt by the PoNN, and the time history of state and costates, with $\Gamma = 1$. For these results, we set $N = 20$ and $L = 16$. The training's convergence is achieved in nine iterations, with a training time of 0.01 s. The average loss and Hamiltonian are 1.3×10^{-9} and 2.5×10^{-8}, respectively. The Hamiltonian at the final time is 7.3×10^{-13}. Moreover, Figure 3 shows that $H(t) + \Gamma$ is approximately equal to zero, as expected. The learnt time of flight is 45.54 s, and the associated cost function is $\mathcal{J} = 65.30$. In Table 1, it can be seen that the average error on the dynamics and the optimality of the learnt solutions improve via increasing N and L, at the expense of the training time, always for $\Gamma = 1$. It is worth noting that the learned trajectories are still accurate and optimal when low N and L hyperparameters are chosen. Furthermore, the training time is such that the algorithm could be used for real-time application (e.g., a closed control loop made by a series of open control loops that were computed instant by instant). A major difference in the formulation of the minimum time-energy intercept problem that we are reporting here and what was solved by Furfaro and Mortari [19] is that they fixed the final time to 50 s, while, in this work, we are also optimizing the final time, which is another parameter to be learnt by the PoNN.

Table 1. Summary of the performances of PoNNs in learning the solutions for the minimum time-energy optimal intercept, with $\Gamma = 1$.

N	L	# of Iterations	Training Time [s]	Mean($\lvert\mathbb{L}\rvert$)	Mean($\lvert H + \Gamma \rvert$)	$\lvert H(t_f) + \Gamma \rvert$	t_f [s]	$\mathcal{J}$
20	8	8	0.005	2.2×10^{-6}	2.5×10^{-6}	1.3×10^{-9}	45.54	65.30
20	12	5	0.006	1.7×10^{-7}	3.9×10^{-5}	6.6×10^{-12}	45.54	65.30
20	16	9	0.01	1.3×10^{-9}	2.5×10^{-8}	7.3×10^{-12}	45.54	65.30
30	16	9	0.02	2.6×10^{-9}	2.1×10^{-7}	6.2×10^{-13}	45.54	65.30
30	30	7	0.03	2.0×10^{-10}	7.4×10^{-8}	2.8×10^{-10}	45.54	65.30
30	30	7	0.03	2.0×10^{-10}	7.4×10^{-8}	2.8×10^{-10}	45.54	65.30



(**a**) Trajectories of Target and Intercept



(**b**) Time history of state and costates

Figure 2. Unconstrained intercept problem with penalty $\Gamma = 1$ ($N = 20$ and $L = 16$).

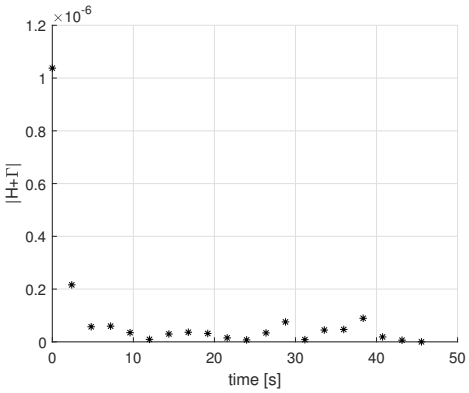
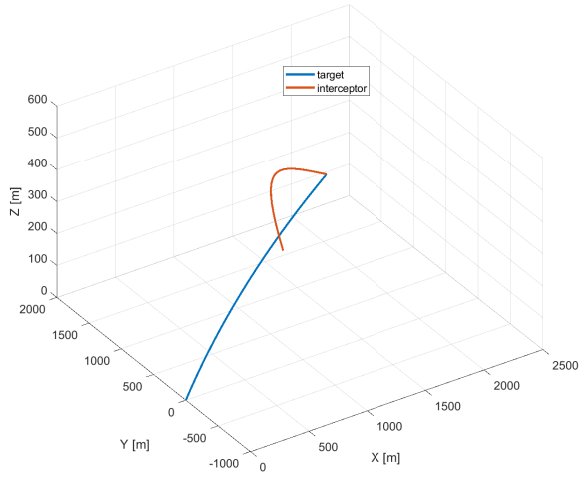


Figure 3. Hamiltonian loss for the unconstrained intercept problem with penalty $\Gamma = 1$ ($N = 20$ and $L = 16$).

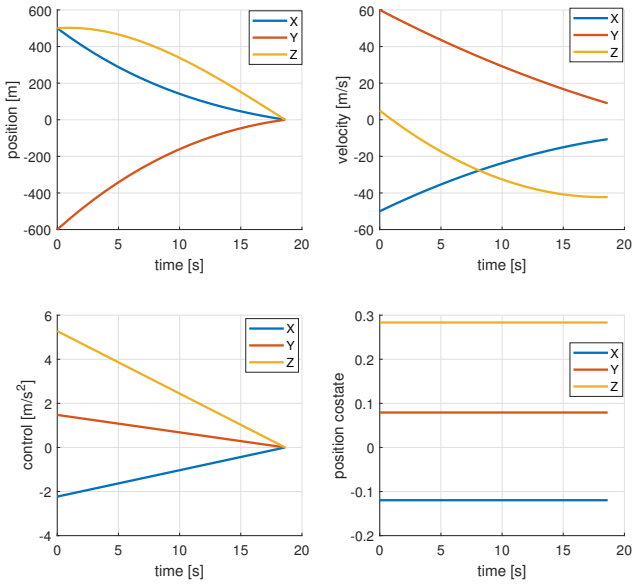
Table 2 also reports the solutions with different values of the penalty Γ. As expected, increasing the penalty, the time of flight decreases at the expense of a higher commanded accelerations (see Figure 4b). Even for the case of $\Gamma = 10$, Figure 5 shows that $H(t) + \Gamma$ is

approximately equal to zero, as expected, proving the optimality of the results. In order to test the reliability of our results, the software GPOPS-II [29] has been run, just for this example, with $\Gamma = 1$ and $\Gamma = 10$. The results obtained with GPOPS-II are the following,

- $\Gamma = 1$: CPU time = 1.48 s, mean$(|H + \Gamma|) = 2.35 \times 10^{-6}$, $|H(t_f) + \Gamma| = 2.35 \times 10^{-6}$, $t_f = 45.54$, $\mathcal{J} = 65.27$
- $\Gamma = 10$: CPU time = 15.41 s, mean$(|H + \Gamma|) = 1.27 \times 10^{-6}$, $|H(t_f) + \Gamma| = 1.27 \times 10^{-6}$, $t_f = 18.63$, $\mathcal{J} = 294.97$



(a) Trajectories of Target and Interceptor



(b) Time history of state and costates

Figure 4. Unconstrained intercept problem with penalty $\Gamma = 10$ ($N = 20$ and $L = 16$).

Table 2. A summary of the performances of PoNNs in learning the solutions for the minimum time-energy optimal intercept, with $N = 20$ and $L = 16$.

Γ	# of Iterations	CPU Time [s]	Mean($\|\mathbb{L}\|$)	Mean($\|H + \Gamma\|$)	$\|H(t_f) + \Gamma\|$	t_f [s]	$\mathcal{J}$
0	9	0.01	1.0×10^{-8}	2.8×10^{-7}	3.6×10^{-8}	61.63	12.53
1	9	0.01	1.3×10^{-9}	2.5×10^{-8}	7.3×10^{-12}	45.54	65.30
5	7	0.01	1.5×10^{-9}	1.1×10^{-7}	1.5×10^{-9}	23.8	191.75
10	6	0.009	3.7×10^{-9}	5.7×10^{-8}	9.6×10^{-8}	18.62	295.11

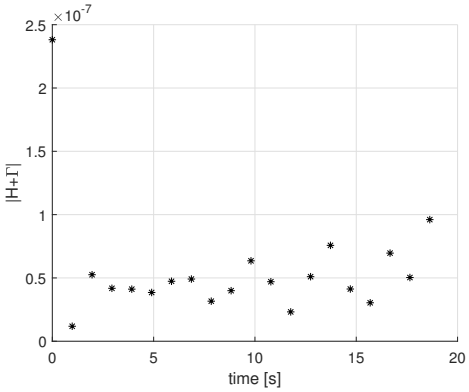


Figure 5. Hamiltonian loss for the unconstrained intercept problem with penalty $\Gamma = 10$ ($N = 20$ and $L = 16$).

As can be seen, the cost function and time of flight are comparable, while the accuracy on the quantity $|H(t) + \Gamma|$ is higher for PoNNs.

3.2. Constrained Problem

Here, the optimization problem to be addressed is a minimum time-energy optimal problem with bounded control. The OCS is cast as follows,

$$\min \quad \mathcal{J} = \Gamma t_f + \frac{1}{2}\int_{t_0}^{t_f} (\boldsymbol{a}_M^T \boldsymbol{a}_M)\, dt + \epsilon \int_{t_0}^{t_f} \mathbf{w}^T \mathbf{w}\, dt \tag{61}$$

subject to

$$\begin{aligned} &\dot{\boldsymbol{r}} = \boldsymbol{v} \\ &\dot{\boldsymbol{v}} = \boldsymbol{a}_T - \boldsymbol{a}_M \\ &t_0 \leq t \leq t_f \\ &\boldsymbol{r}(t_0) = \boldsymbol{r}_0 \\ &\boldsymbol{v}(t_0) = \boldsymbol{v}_0 \\ &\boldsymbol{r}(t_f) = \mathbf{0} \\ &u_{min,i} \leq a_{M,i} \leq u_{max,i} \end{aligned} \tag{62}$$

where Γ is a non-negative coefficient, $u_{min,i}$ and $u_{max,i}$ are the bounds of the control acceleration vector components, t_f is the free final time to be optimized, and the final relative velocity is chosen to be free. In particular, $\mathbf{w} = [w_x, w_y, w_z]$ is the new fictitious control acceleration vector that is required to transform the original constrained OCP into its unconstrained version. The Hamiltonian for this problem is,

$$H(t) = \frac{1}{2}\boldsymbol{a}_M^T \boldsymbol{a}_M + \epsilon \mathbf{w}^T \mathbf{w} + \boldsymbol{\lambda}_r^T \boldsymbol{v} + \boldsymbol{\lambda}_v^T (\boldsymbol{a}_T - \boldsymbol{a}_M) + \boldsymbol{\nu}^T (\boldsymbol{a}_M - \boldsymbol{\phi}(\mathbf{w})) \tag{63}$$

with $\boldsymbol{\nu} = [\nu_x, \nu_y, \nu_z]$ and $\boldsymbol{\phi}(\mathbf{w}) = [\phi(w_x), \phi(w_y), \phi(w_z)]$ being the additional multipliers vector and the saturation functions vector, respectively. One can note that, in this case, one equality constraint is added for each component of $\boldsymbol{a}_M$. The first-order necessary conditions for the state and costate equations and the control optimality are given by,

$$\frac{\partial H}{\partial \boldsymbol{a}_M} = \boldsymbol{a}_M - \boldsymbol{\lambda}_v + \boldsymbol{\nu} = \mathbf{0} \tag{64}$$

$$\frac{\partial H}{\partial \boldsymbol{w}} = 2\epsilon \mathbf{w} - \boldsymbol{\nu} \odot \boldsymbol{\phi}^{'}(\mathbf{w}) = \mathbf{0} \tag{65}$$

$$\dot{\boldsymbol{r}} = \frac{\partial H}{\partial \boldsymbol{\lambda}_r} = \boldsymbol{v} \tag{66}$$

$$\dot{\boldsymbol{v}} = \frac{\partial H}{\partial \boldsymbol{\lambda}_v} = \boldsymbol{a}_T - \boldsymbol{a}_M \tag{67}$$

$$\dot{\boldsymbol{\lambda}}_r = -\frac{\partial H}{\partial \boldsymbol{r}} = \mathbf{0} \tag{68}$$

$$\dot{\boldsymbol{\lambda}}_v = -\frac{\partial H}{\partial \boldsymbol{v}} = -\boldsymbol{\lambda}_r \tag{69}$$

$$\boldsymbol{a}_M - \boldsymbol{\phi}(\mathbf{w}) = \mathbf{0} \tag{70}$$

where the symbol $'$ indicates the derivatives with respect to the new fictitious control and the symbol $\odot$ identifies the Hadamard product (e.g., element-wise multiplication). In addition, two transversality conditions need to be imposed,

$$H(t_f) + \Gamma = 0 \tag{71}$$

$$\boldsymbol{\lambda}_v(t_f) = \mathbf{0} \tag{72}$$

The system of equations, along with the transversality condition on the Hamiltonian at the final time, forms the TPBVP, which represent the physics constraints that drive the PoNN's training.

The transversality condition (71) is necessary, because the problem is final time free (e.q., the final time is in the cost function). This means that the Hamiltonian at the converged final time must be equal to Γ. Because the system is autonomous (e.g., there is no explicit dependency on time in the dynamics), the Hamiltonian must be constant over time (e.g., $H(t) = -\Gamma$). As $H(t_f) = -\Gamma$ (from the transversality condition), then $H(t) + \Gamma = 0$.

The equations then are,

$$\begin{aligned}
\boldsymbol{\mathcal{L}}_a &= \dot{\boldsymbol{v}} - \boldsymbol{a}_T + \boldsymbol{\phi}(\mathbf{w}) && (73)\\
\boldsymbol{\mathcal{L}}_{\lambda_r} &= \dot{\boldsymbol{\lambda}}_r && (74)\\
\boldsymbol{\mathcal{L}}_{\lambda_v} &= \dot{\boldsymbol{\lambda}}_v + \boldsymbol{\lambda}_r && (75)\\
\boldsymbol{\mathcal{L}}_u &= \boldsymbol{\phi}(\mathbf{w}) - \boldsymbol{\lambda}_v + \boldsymbol{\nu} && (76)\\
\boldsymbol{\mathcal{L}}_w &= 2\epsilon\mathbf{w} - \boldsymbol{\nu} \odot \boldsymbol{\phi}^{'}(\mathbf{w}) && (77)\\
H(t_f) + \Gamma &= 0 && (78)
\end{aligned}$$

where Equation (66) is removed, since it is redundant and the control $\boldsymbol{a}_M$ is substituted by $\boldsymbol{\phi}(\mathbf{w})$. The unknown solutions are approximated with the TFC's CEs,

$$\begin{aligned}
r_j &= \left(\sigma - \Omega_1\sigma_0 - \Omega_2\sigma_f - \Omega_3\sigma_0'\right)^{\mathrm{T}}\boldsymbol{\beta}_j + \Omega_1 r_{0_j} + \Omega_2 r_{f_j} + \frac{\Omega_3 v_{0_j}}{b^2}\\
v_j &= b^2\left[\left(\sigma' - \Omega_1'\sigma_0 - \Omega_2'\sigma_f - \Omega_3'\sigma_0'\right)^{\mathrm{T}}\boldsymbol{\beta}_j + \Omega_1' r_{0_j} + \Omega_2' r_{f_j} + \frac{\Omega_3' v_{0_j}}{b^2}\right]\\
a_j &= b^4\left[\left(\sigma'' - \Omega_1''\sigma_0 - \Omega_2''\sigma_f - \Omega_3''\sigma_0'\right)^{\mathrm{T}}\boldsymbol{\beta}_j + \Omega_1'' r_{0_j} + \Omega_2'' r_{f_j} + \frac{\Omega_3'' v_{0_j}}{b^2}\right]\\
\lambda_{r,j} &= \sigma^{\mathrm{T}}\boldsymbol{\beta}_{r,j}\\
\dot{\lambda}_{r,j} &= b^2\sigma^{'T}\boldsymbol{\beta}_{r,j}\\
\lambda_{v,j} &= \left(\sigma - \sigma_f\right)^{\mathrm{T}}\boldsymbol{\beta}_{v,j} + \lambda_{v_{f,j}}\\
\dot{\lambda}_{v,j} &= b^2\sigma^{'T}\boldsymbol{\beta}_{v,j}\\
w_j &= \sigma^{\mathrm{T}}\boldsymbol{\beta}_{w,j}\\
\nu_j &= \sigma^{\mathrm{T}}\boldsymbol{\beta}_{\nu,j}
\end{aligned}$$

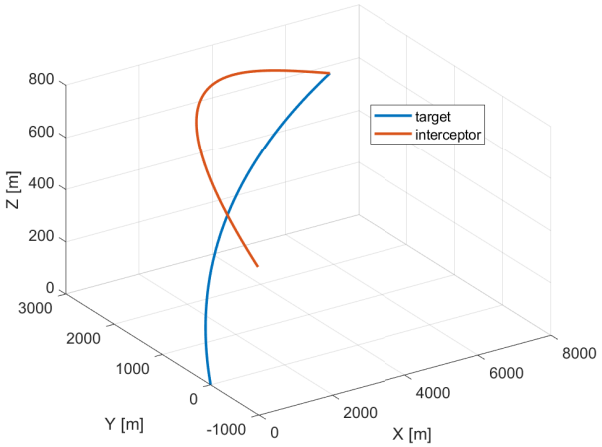
Even for this case, the switching functions $\Omega_k(z)$ are reported in the Appendix A. Hence, the parameters of PoNN to be learnt are,

$$\Xi = \left\{ \begin{array}{ccccccccccc} \beta_{r,1} & \beta_{r,2} & \beta_{r,3} & \beta_{\lambda_{r,1}} & \beta_{\lambda_{r,2}} & \beta_{\lambda_{r,3}} & \beta_{\lambda_{v,1}} & \beta_{\lambda_{v,2}} & \beta_{\lambda_{v,3}} & \beta_{w,1} \\ \beta_{w,2} & \beta_{w,3} & \beta_{\nu,1} & \beta_{\nu,2} & \beta_{\nu,3} & b & & & & \end{array} \right\}^{\mathrm{T}}$$

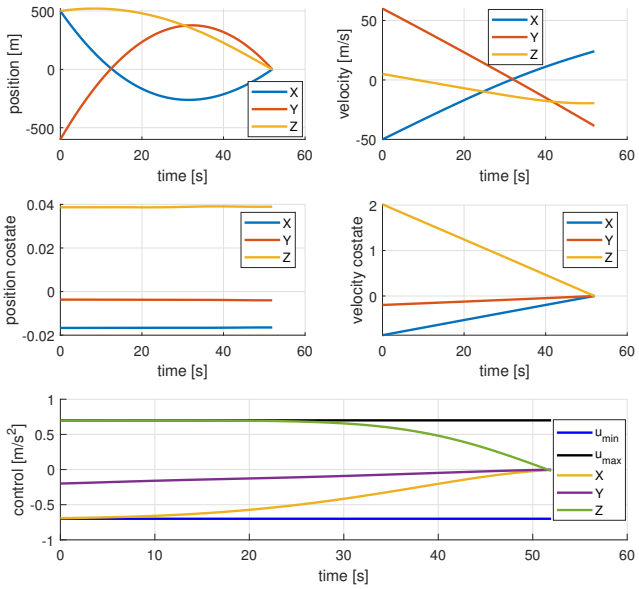
An iterative least-square approach can be used for minimizing the losses and train the PoNN.

Results

The initial conditions and target constant acceleration command for this problem are the same of the previous unconstrained version. For what concerns the boundaries for the components of the control acceleration vector, they have all been set equal to $u_{min,i} = d_i^- = -0.7\ \mathrm{m/s^2}$ and $u_{max,i} = d_i^+ = 0.7\ \mathrm{m/s^2}$. The coefficient c for the saturation function appearing in Equation (14) has been set to 0.005 after a trial and error procedure in order to achieve a good accuracy. Moreover, a continuation procedure on the parameter ϵ has been carried out to increase the accuracy of the solution as the unconstrained OCP approaches its constrained version. In particular, the value of ϵ is decreased by a factor of 0.1 at each iteration starting from 0.5 and arriving at 5×10^{-11}. The activation function that is used in this problem is the logistic function. The weights and bias are sampled from $\mathcal{U}(-1;1)$. Even for this case, all of the PoNN's parameters have been initialized by randomly sample them from $\mathcal{U}(0;1)$. For this problem, we set $L = 16$ and $N = 100$, where the discretization points employed are the Chebyshev–Gauss–Lobatto points, represented by the equation $z_i = cos(i\pi)$, where i is equally spaced from -1 to 0. The results are reported here just for $\Gamma = 1$ and they are shown in Figures 6–8. It is actually bounded within the defined boundaries, as can be seen from the control in Figure 6b. As expected from Equations (67) and (68), the position and velocity costates have a constant and linear trend, respectively, proving the feasibility of the proposed solution. In addition, the precision on the dynamics is shown in the time history of all the losses that are reported in Figure 7, for which the mean value corresponds to 4.89×10^{-6}. Finally, the cost function that is learnt by the PoNN is 67.807 with a time of flight of 51.946 s.

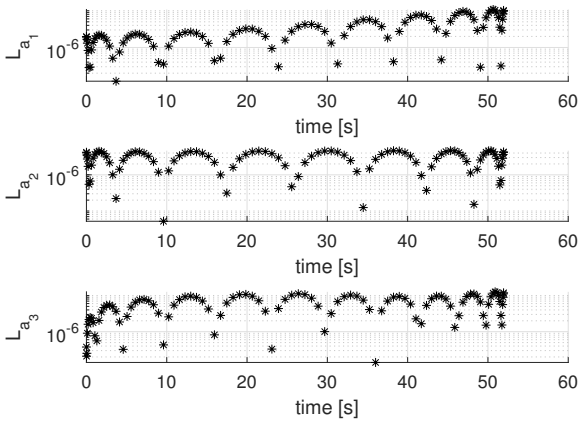


(**a**) Trajectories of Target and Interceptor

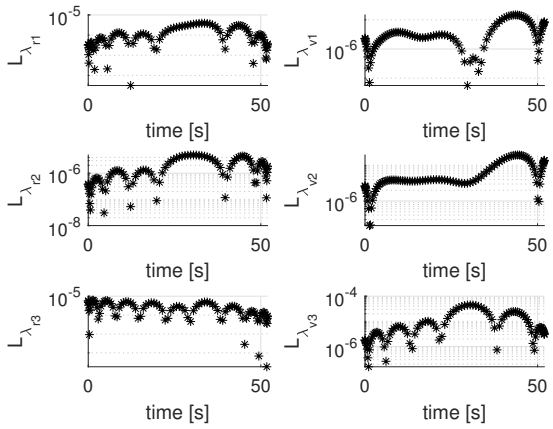


(**b**) Time history of state and costates

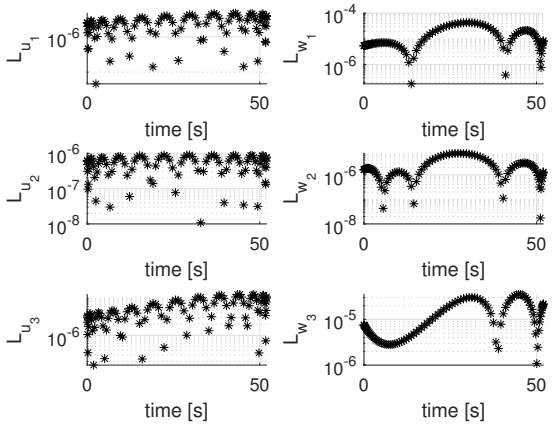
Figure 6. Constrained intercept problem with penalty $\Gamma = 1$ ($N = 100$ and $L = 16$).



(**a**) Acceleration losses



(**b**) Costate losses



(**c**) Control losses

Figure 7. Losses for the constrained intercept problem with penalty $\Gamma = 1$ ($N = 100$ and $L = 16$).

The optimality of the results is proved by the Hamiltonian loss (namely $|H(t) + \Gamma|$), whose mean and final value are 4.95×10^{-7} and 1.76×10^{-6}, respectively.

As can be observed, the PoNN is able to learn the optimal control action, even for this case of bounded control, with a good accuracy.

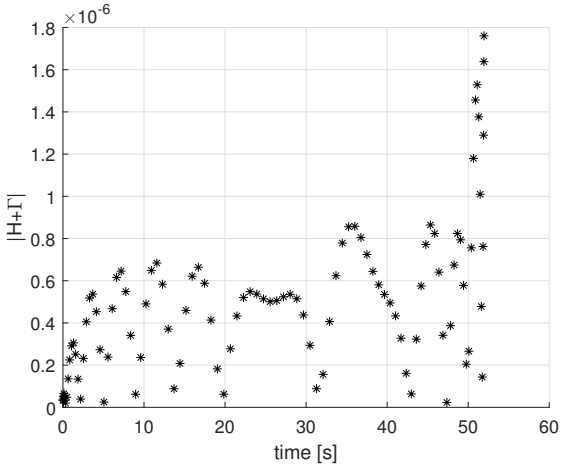


Figure 8. Hamiltonian loss for the constrained intercept problem with penalty $\Gamma = 1$ ($N = 100$ and $L = 16$).

4. Discussions

The results that are presented in this paper show that PoNNs can effectively learn the optimal control for the class of optimal intercept problems with the integral quadratic cost. In particular, the framework converges to the solution, even for a random initialization of the PoNN's parameters, which is an important advantage, since we know how the training can be significantly affected by the initial guesses when more traditional deterministic algorithms are used.

Importantly, once the optimal control is learned on the training points, its analytical representation is automatically obtained with the NNs. Therefore, it can be evaluated in points unseen during the training (e.g., test points) without recurring to interpolation techniques and/or no additional computational efforts.

Although the PoNN was solely trained in a physics-driven fashion to generate the results, the training could also be carried out in a data-physics-driven fashion. This becomes of extreme importance if the DEs do not precisely model the physics of the problem. For instance, when perturbations are present and/or when uncertain dynamical systems are considered, which is common when dealing with OCPs for space applications. Thanks to this unique feature, the proposed PoNNs could be used solo or in synergy with other states of the art methods for OCPs. For instance, one, in principle, could use PoNNs with GPOPS-II, where the problem to tackle is sufficiently complex and both of the methods alone would struggle. Within this scenario, optimal trajectories could be generated with GPOPS-II and then sampled to generate data, which can be added as additional terms in the PoNNs' loss function to better drive the training in a data-physics-driven manner. The same approach could be used with real data that intrinsically own information about perturbations and/or unmodeled terms of the real dynamics that were not accounted for in the modeled one.

5. Conclusions

Pontryagin Neural Networks (PoNNs) are introduced and tested to learn optimal control actions for instances of an optimal intercept problem. PoNNs represent a particular family of PINNs, specifically designed to learn control actions while satisfying the physics constraints coming from the PMP application. PoNNs are based on a newly developed PINN framework, named Extreme Theory of Functional Connections (X-TFC). PoNNs are SISO NNs, which are modeled and trained to satisfy the PMP via learning the solution of the arising TPBVP. In particular, the use of the Constrained Expressions (CEs), defined within the TFC, enables the boundary conditions' analytical satisfaction. This removes the issues that are related to having competing objects in the loss function, representing the major drawback of the original PINNs methods. The Extreme Learning Machine (ELM)

algorithm is employed to train the NN within the CEs. Thanks to the ELM, the network's training is reduced to a simple least-squares, reducing the training time significantly when compared to gradient-based methods. The results show how PoNNs successfully learn optimal control actions for the optimal intercept problem, with unbounded and bounded control. Although the training is always initialized with random initial guesses for both the unknown coefficients of the CEs and final time, the algorithm can still converge to a feasible solution in all the cases, proving the proposed method's reliability. Furthermore, the results have shown good accuracy for the precision of the dynamics and the results' optimality.

Future works will focus on applying PoNNs to other classes of OCPs for space applications involving discontinuities in control, such as for the bang-bang type of solutions.

Author Contributions: Conceptualization: A.D., E.S. and R.F.; methodology: A.D., E.S. and R.F.; software: A.D. and E.S.; validation: A.D. and E.S.; formal analysis: A.D. and E.S.; investigation: A.D. and E.S.; resources: A.D. and E.S.; writing—original draft preparation: A.D. and E.S.; writing—review and editing: R.F. and F.C.; visualization: A.D. and E.S.; supervision: R.F. and F.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors would like to acknowledge Kristopher Drozd for providing computational resources, and precious insights that have helped for the realization of this work.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Switching Functions

The detailed derivation of the switching functions is reported in the work of Johnston et al. [21]. Here, for the convenience of the reader we just report them. We define $\Delta z = z_f - z_0$ and $z_* = z - z_0$. The switching functions for a constrained expression with one constraint on f are given in Table A1. The switching functions for a constrained expression with one constraint on f' are given in Table A2. The switching functions for a constrained expression with two constraints on f are given in Table A3. The switching functions for a constrained expression with two constraints, one on f and one on f', are given in Table A4. The switching functions for a three constraints constrained expression, with two constraints on f and one on f', are given in Table A5. The switching functions for a four constraints constrained expression, with two constraints on both f and f', are given in Table A6.

Table A1. Switching functions for a one constraint constrained expression, with the constraint of f, defined on domain of $z \in [z_0, z_f]$.

	Initial/Final Value $\boldsymbol{\Omega_1(z)}$
$(\cdot)$	1
$\frac{d}{dz}(\cdot)$	0

Table A2. Switching functions for a one constraint constrained expression, with the constraint of f', defined on domain of $z \in [z_0, z_f]$.

	Initial/Final Value $\boldsymbol{\Omega_1(z)}$
$(\cdot)$	z
$\frac{d}{dz}(\cdot)$	1

Table A3. Switching functions for a two constraints constrained expression, with both constrains on f, defined on domain of $z \in [z_0, z_f]$.

	Initial Value $\boldsymbol{\Omega_1(z)}$	**Final Value** $\boldsymbol{\Omega_2(z)}$
$(\cdot)$	$\frac{z_f - z}{\Delta z}$	$\frac{z - z_0}{\Delta z}$
$\frac{d}{dz}(\cdot)$	$-\frac{1}{\Delta z}$	$\frac{1}{\Delta z}$
$\frac{d^2}{dz^2}(\cdot)$	0	0

Table A4. Switching functions for a two constraints constrained expression, with one constraint on f and one on f', defined on domain of $z \in [z_0, z_f]$.

	Initial Value $\boldsymbol{\Omega_1(z)}$	**Final Value** $\boldsymbol{\Omega_2(z)}$
$(\cdot)$	1	$z - z_0$
$\frac{d}{dz}(\cdot)$	0	1
$\frac{d^2}{dz^2}(\cdot)$	0	0

Table A5. Switching functions for a three constraints constrained expression, with two constraints on f and one on f', defined on domain of $z \in [z_0, z_f]$.

	Initial Value $\boldsymbol{\Omega_1(z)}$	**Final Value** $\boldsymbol{\Omega_2(z)}$	**Initial Derivative** $\boldsymbol{\Omega_3(z)}$
$(\cdot)$	$\frac{(z_f - z)(z - 2z_0 + z_f)}{\Delta z^2}$	$\frac{(z - z_0)^2}{\Delta z^2}$	$\frac{(z - z_0)(z_f - z)}{\Delta z}$
$\frac{d}{dz}(\cdot)$	$\frac{-2(z - z_0)}{\Delta z^2}$	$\frac{2(z - z_0)}{\Delta z^2}$	$\frac{-2z + z_0 + z_f}{\Delta z}$
$\frac{d^2}{dz^2}(\cdot)$	$\frac{-2}{\Delta z^2}$	$\frac{2}{\Delta z^2}$	$\frac{-2}{\Delta z}$

Table A6. Switching functions for a four constraints constrained expression, with two constraints on f and two constraints on f', defined on domain of $z \in [z_0, z_f]$.

	Initial Value $\boldsymbol{\Omega_1(z)}$	**Final Value** $\boldsymbol{\Omega_2(z)}$	**Initial Derivative** $\boldsymbol{\Omega_3(z)}$	**Final Derivative** $\boldsymbol{\Omega_4(z)}$
$(\cdot)$	$1 + \frac{2z_*^3}{\Delta z^3} - \frac{3z_*^2}{\Delta z^2}$	$-\frac{2z_*^3}{\Delta z^3} + \frac{3z_*^2}{\Delta z^2}$	$z_* + \frac{z_*^3}{\Delta z^2} - \frac{2z_*^2}{\Delta z}$	$\frac{z_*^3}{\Delta z^2} - \frac{z_*^2}{\Delta z}$
$\frac{d}{dz}(\cdot)$	$\frac{6z_*^2}{\Delta z^3} - \frac{6z_*}{\Delta z^2}$	$-\frac{6z_*^2}{\Delta z^3} + \frac{6z_*}{\Delta z^2}$	$1 + \frac{3z_*^2}{\Delta z^2} - \frac{4z_*}{\Delta z}$	$\frac{3z_*^2}{\Delta z^2} - \frac{2z_*}{\Delta z}$
$\frac{d^2}{dz^2}(\cdot)$	$\frac{12z_*}{\Delta z^3} - \frac{6}{\Delta z^2}$	$-\frac{12z_*}{\Delta z^3} + \frac{6}{\Delta z^2}$	$\frac{6z_*}{\Delta z^2} - \frac{4}{\Delta z}$	$\frac{6z_*}{\Delta z^2} - \frac{2}{\Delta z}$

References

1. Rao, A.V. A survey of numerical methods for optimal control. *Adv. Astronaut. Sci.* **2009**, *135*, 497–528.
2. Poe, W.A.; Mokhatab, S. *Modeling, Control, and Optimization of Natural Gas Processing Plants*; Gulf Professional Publishing: Houston, TX, USA, 2016.
3. Keller, H.B. *Numerical Solution of Two Point Boundary Value Problems*; SIaM: Philadelphia, PA, USA, 1976; Volume 24.
4. Stoer, J.; Bulirsch, R. *Introduction to Numerical Analysis*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2013; Volume 12.
5. Oh, S.; Luus, R. Use of orthogonal collocation method in optimal control problems. *Int. J. Control.* **1977**, *26*, 657–673. [CrossRef]
6. Fahroo, F.; Ross, I. Trajectory optimization by indirect spectral collocation methods. In Proceedings of the Astrodynamics Specialist Conference, Denver, CO, USA, 14–17 August 2000; p. 4028.

7. Raissi, M.; Perdikaris, P.; Karniadakis, G.E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* **2019**, *378*, 686–707. [CrossRef]
8. Schiassi, E.; D'Ambrosio, A.; De Florio, M.; Furfaro, R.; Curti, F. Physics-Informed Extreme Theory of Functional Connections Applied to Data-Driven Parameters Discovery of Epidemiological Compartmental Models. *arXiv* **2020**, arXiv:2008.05554.
9. Wang, S.; Teng, Y.; Perdikaris, P. Understanding and mitigating gradient pathologies in physics-informed neural networks. *arXiv* **2020**, arXiv:2001.04536.
10. Mertikopoulos, P.; Papadimitriou, C.; Piliouras, G. Cycles in adversarial regularized learning. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans LA, USA, 7–10 January 2018*; SIAM: Philadelphia, PA, USA, 2018; pp. 2703–2717.
11. Balduzzi, D.; Racaniere, S.; Martens, J.; Foerster, J.; Tuyls, K.; Graepel, T. The mechanics of n-player differentiable games. In *Proceedings of the International Conference on Machine Learning, Virtual, 22–24 June 2021*; ML Research Press: Stockholm, Sweden, 2018; pp. 354–363.
12. Leake, C.; Mortari, D. Deep theory of functional connections: A new method for estimating the solutions of partial differential equations. *Mach. Learn. Knowl. Extr.* **2020**, *2*, 37–55. [CrossRef] [PubMed]
13. Schiassi, E.; Leake, C.; De Florio, M.; Johnston, H.; Furfaro, R.; Mortari, D. Extreme Theory of Functional Connections: A Physics-Informed Neural Network Method for Solving Parametric Differential Equations. *arXiv* **2020**, arXiv:2005.10632.
14. Mortari, D. The Theory of Connections: Connecting Points. *Mathematics* **2017**, *5*, 57. [CrossRef]
15. Mortari, D.; Leake, C. The Multivariate Theory of Connections. *Mathematics* **2019**, *7*, 296. [CrossRef]
16. Mortari, D. Least-Squares Solution of Linear Differential Equations. *Mathematics* **2017**, *5*, 48. [CrossRef]
17. Mortari, D.; Johnston, H.; Smith, L. High accuracy least-squares solutions of nonlinear differential equations. *J. Comput. Appl. Math.* **2019**, *352*, 293–307. [CrossRef] [PubMed]
18. Leake, C.; Johnston, H.; Mortari, D. The Multivariate Theory of Functional Connections: Theory, Proofs, and Application in Partial Differential Equations. *Mathematics* **2020**, *8*, 1303. [CrossRef]
19. Furfaro, R.; Mortari, D. Least-squares solution of a class of optimal space guidance problems via Theory of Connections. *Acta Astronaut.* **2020**. [CrossRef]
20. Schiassi, E.; D'Ambrosio, A.; Johnston, H.; Furfaro, R.; Curti, F.; Mortari, D. Complete Energy Optimal Landing on Small and Large Planetary Bodies via Theory of Functional Connections. 2020. Available online: https://www.researchgate.net/publication/343628030_Complete_Energy_Optimal_Landing_on_Small_and_Large_Planetary_Bodies_via_Theory_of_Functional_Connections (accessed on 26 April 2021).
21. Johnston, H.; Schiassi, E.; Furfaro, R.; Mortari, D. Fuel-Efficient Powered Descent Guidance on Large Planetary Bodies via Theory of Functional Connections. *arXiv* **2020**, arXiv:2001.03572.
22. Drozd, K.; Furfaro, R.; Schiassi, E.; Johnston, H.; Mortari, D. Energy-optimal trajectory problems in relative motion solved via Theory of Functional Connections. *Acta Astronaut.* **2021**, *182*, 361–382. [CrossRef]
23. Huang, G.B.; Zhu, Q.Y.; Siew, C.K. Extreme learning machine: Theory and applications. *Neurocomputing* **2006**, *70*, 489–501. [CrossRef]
24. Graichen, K.; Kugi, A.; Petit, N.; Chaplais, F. Handling constraints in optimal control with saturation functions and system extension. *Syst. Control. Lett.* **2010**, *59*, 671–679. [CrossRef]
25. Ross, I.M. *A Primer on Pontryagin's Principle in Optimal Control*; Collegiate Publications: Wales, UK, 2009.
26. Antony, T. Large Scale Constrained Trajectory Optimization Using Indirect Methods. Ph.D. Thesis, Purdue University, West Lafayette, IN, USA, 2018.
27. Lagaris, I.E.; Likas, A.; Fotiadis, D.I. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans. Neural Netw.* **1998**, *9*, 987–1000. [CrossRef] [PubMed]
28. Lu, L.; Meng, X.; Mao, Z.; Karniadakis, G.E. DeepXDE: A deep learning library for solving differential equations. *arXiv* **2019**, arXiv:1907.04502.
29. Patterson, M.A.; Rao, A.V. GPOPS-II: A MATLAB software for solving multiple-phase optimal control problems using hp-adaptive Gaussian quadrature collocation methods and sparse nonlinear programming. *ACM Trans. Math. Softw. (TOMS)* **2014**, *41*, 1–37. [CrossRef]