EXTREME THEORY OF FUNCTIONAL CONNECTIONS FOR PROBLEMS INVOLVING DIFFERENTIAL EQUATIONS WITH APPLICATIONS TO OPTIMAL CONTROL PROBLEMS

by

Enrico Schiassi

A Dissertation Submitted to the Faculty of the

DEPARTMENT OF SYSTEMS AND INDUSTRIAL ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY

In the Graduate College

THE UNIVERSITY OF ARIZONA

2022

THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

As members of the Dissertation Committee, we certify that we have read the dissertation prepared by Enrico Schiassi, titled: “Extreme Theory of Functional Connections for Problems involving Differential Equations with Applications to Optimal Control Problems”

and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

_____ Date: _____
Roberto Furfaro

_____ Date: _____
Clayton T. Morrison

_____ Date: _____
Jianqiang Cheng

_____ Date: _____
Larry Head

Final approval and acceptance of this dissertation is contingent upon the candidate’s submission of the final copies of the dissertation to the Graduate College.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

_____ Date: _____
Dr. Roberto Furfaro
Dissertation Committee Chair
Professor, Systems and Industrial Engineering
Director, Space Situational Awareness

Contents

List of Figures

Abstract

In recent years, Neural Networks (NNs) have become widespread across many scientific fields. This requires designing them to satisfy application-specific constraints, such as conservation laws, symmetries, or other domain-specific knowledge. These constraints are usually imposed as soft penalties during network training and act as regularizers within the loss function. *Physics-Informed Neural Networks (PINNs)* are examples of this philosophy, where the outputs of the network are constrained to approximately satisfy specific physics laws, modeled as a set of Differential Equations (DEs).

This dissertation presents a novel, accurate, fast, flexible, reliable, and robust PINN framework for forward and inverse problems governed by DEs. This framework is called *Extreme Theory of Functional Connections (X-TFC)*. The proposed method, for the first time, merges the *Theory of Functional Connections (TFC)* and the *Extreme Learning Machines (ELM)*. TFC enables functional interpolation for a large class of mathematical objects. According to TFC, any mathematical problem solution can be represented via the Constrained Expressions (CEs). The CEs are functional. These functionals are the sum of a free function and a functional that analytically satisfies the problem constraints. When applied to problems involving DEs, the mathematical problem is represented by the DE itself, where the constraints are the DE initial and/or boundary conditions. The DE solution is approximated with the CE, where the free function, according to X-TFC, is chosen to be a shallow NN trained via the Extreme Learning Machine (ELM) algorithm. ELM algorithm randomly selects input weights and biases, which are not tuned during the training, leaving the output weights as the only trainable parameters. Thus, the training is performed with a fast and robust least-squares.

This work shows the X-TFC advantage over the classic PINN framework and some of its variants. In particular, it will show many X-TFC applications in solving forward and inverse problems involving DEs. X-TFC has been applied to physics-driven solutions of nuclear reactor dynamics (e.g., point kinetic equations with temperature feedback). Another application is for physics-driven solutions of Optimal Control Problems (OCPs), both via the application of the indirect method (e.g., Pontryagin Minimum Principle) and the Bellman Principle of Optimality (BPO), both for general and aerospace OCPs. X-TFC has also been applied for data-physics-driven parameter discovery of epidemiological models that regulate virus spread.

Chapter 1

Introduction

1.1 Overview and Opportunities

In mathematics, a Differential Equation (DE) is an equation that links one or several unknown functions, and their derivatives [1]. In applications, the functions usually are physical quantities. The derivatives are their rates of change. The DE defines a relationship among them. For instance, in classical mechanics, the motion of a body is described by its position (unknown function), velocity (first derivative), and acceleration (second derivative), as the time varies. That is,

$$a = \frac{d\,v(t)}{dt} = \frac{d^2\,r(t)}{dt^2} = \frac{F}{m} \tag{1.1}$$

where t is the time, a is the acceleration, v is the velocity, r is the position, F is the force acting on the body, and m is the body's mass. t is the independent variable, a, v, and r are the dependent variables, $\frac{F}{m}$ is the forcing term. DEs with only one independent variable are called Ordinary Differential Equations (ODEs). DEs with more than one independent variable are called Partial Differential Equations (PDEs).
In many real-world applications, closed-form expressions for the DE solutions are not available. Thus, the solutions are approximated via numerical methods.
Many real-world problems are modeled via DEs in fields such as physics, engineering, finance, biology, chemistry, oceanography, and epidemiology, to name a few. Hence, there is a need to have accurate, efficient, robust, flexible, and reliable tools to solve problems involving DEs. These problems are divided into two macro-categories:

- Forward problems, where the goal is to find the unknown functions that solve the DEs (e.g., the body position in Eq. (1.1))
- Inverse problems, where the goal is to estimate the parameters governing the DEs along with the unknown functions, given some data (e.g., to estimate the body mass in Eq. (1.1) given some measurements about the body position over time)

There exist many DEs solvers. For ODEs, the most common and widely used solvers are the Runge-Kutta family methods and their variants, such as Runge-Kutta second-order

method, Runge-Kutta fourth-order method, and Milne's predictor-corrector method [2, 3, 4]. For PDEs the most common and widely used solvers are Finite Difference Method (FDM) [5], Finite Element Method (FEM) [6, 7, 8, 9, 10, 11], Finite Volume Method (FVM) [12, 13, 14, 15] and all their variants. All these methods represent the state-of-the-art of DEs solvers. They have been used for years for academic research and by industries and companies in many disciplines, as they are accurate, robust, and reliable. Nonetheless, they present some non-negligible limitations. These methods employ orthogonal basis functions, such as orthogonal polynomials, to approximate the DE solutions, which are computed on collocation points arranged on mesh-grid schemes. Employing orthogonal basis functions and having the collocation points arranged on mesh-grid schemes make these methods suffer the *curse of dimensionality* [16]. The equation's space complexity is exponential to the number of dimensions (e.g., independent variables), d. That is, the volume of the computational domain grows too rapidly, causing the available collocation points and basis functions to become sparse. Hence, to obtain reliable approximated solutions, the number of collocations points and basis function required increases exponentially with the dimension of the problems. The latter makes these methods too computationally expensive and prohibitive to use for large-scale problems (e.g., PDEs with $d > 3$) such as, for instance, the Hamilton–Jacobi–Bellman (HJB) equation for optimal control problems [17, 18]. Moreover, all these methods provide a purely numerical DE solution approximation. Thus, to evaluate the DE solution on query points different than the collocation points, we either need to perform interpolation or re-compute the solutions using the query points as collocation points. The former will lead to a loss of accuracy in the approximated solution, which is already affected by an approximation error (e.g., the error to the DE real solution, which is unknown). The latter may be cost-effective in terms of computational efforts. Additionally, if we need to manipulate the approximated solution (e.g., evaluating its integral and/or derivatives), numerical methods based on discretization techniques must be employed, which will lead to an increase in the approximation error (to the real integral and derivatives). Finally, these methods are designed to tackle only forward problems. Hence, for inverse problems, they need to be modified and/or coupled with other methods. This can be a non-trivial task in terms of framework design as well as in terms of coding.
Therefore, the research goal is to develop and test an efficient, robust, reliable, flexible, and unified framework to tackle forward and inverse problems governed by DEs that overcomes the main limitations of the state-of-the-art DE solvers.

Neural Networks (NNs) represent an attractive solution to the limitations discussed above [19]. Being universal approximators [20, 21], NNs are functions with enough parameters that can approximate any continuous function within a specific given domain. Thus, with a proper choice of these parameters, NNs can approximate the DE solution in the domain of interest.

Consider a generic single PDE,

$$\mathcal{D}\left(u(\boldsymbol{x});\boldsymbol{\Gamma}\right) = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega \tag{1.2}$$

$$\mathcal{B}\left(u(\boldsymbol{x})\right) = g(\boldsymbol{x}), \quad \boldsymbol{x} \in \delta\Omega \tag{1.3}$$

where $\mathcal{D}(\cdot)$ and $\mathcal{B}(\cdot)$ are some linear or nonlinear differential operators, $u : \mathbb{R}^n \to \mathbb{R}$ is the unknown exact DE solution, $\boldsymbol{x} = \{x_1, \ldots, x_n\}^T$ are the independent variables (with $x_i \in \mathbb{R} \, \forall \, i = 1, \ldots, n$), $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}$ are some known functions, Ω is a bounded open domain with boundary $\delta\Omega$, and $\boldsymbol{\Gamma} \in \mathbb{R}^m$ are some parameters governing the DE. When dealing with forward problems, $\boldsymbol{\Gamma}$ is known.
On top of the author's knowledge, Dissanayake and Phan-Thien were among the first researchers to propose NN for approximating DE solutions to tackle forward problems [22], followed by Lagaris et al. [23]. In [22] u is represented through a NN. That is,

$$u(\boldsymbol{x}) \simeq u(\boldsymbol{x};\theta) = u_\theta(\boldsymbol{x}) \tag{1.4}$$

where θ are the NN parameters (e.g., weights and bias). Considering, as an example, a Multiple-Input Single-Output (MISO) single layer (e.g., shallow) NN as represented in figure 1.1, $u_\theta(\boldsymbol{x})$ is expressed as following,

$$u_\theta(\boldsymbol{x}) = \sum_{j=1}^{L} \beta_j \sigma\left(\boldsymbol{w}_j^T \, \boldsymbol{x} + b_j\right) \tag{1.5}$$

where σ's are some nonlinear activation functions (such as sigmoid, hyperbolic tangent, gaussian, to name a few), $W = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_L\} \in \mathbb{R}^{L \times n}$ (with $\boldsymbol{w}_j = \{w_{j,1}, \ldots, w_{j,n}\}^T \in \mathbb{R}^L$, $\forall \, j = 1, \ldots L$) is the input weights matrix, $\boldsymbol{b} = \{b_1, \ldots, b_L\}^T \in \mathbb{R}^L$ is the bias vector, $\boldsymbol{\beta} = \{\beta_1, \ldots, \beta_L\}^T \in \mathbb{R}^L$ is the output weights vector, and $\theta = \{W | \, \boldsymbol{b} | \, \boldsymbol{\beta}\} \in \mathbb{R}^{L \times (n+2)}$ is the NN parameters matrix. $u_\theta(\boldsymbol{x})$ is plugged into the DE residual and into the boundary conditions (BC), which are then collocated on the collocation (or training) points $\{\boldsymbol{x}_{\Omega,i}\}_{i=1}^{N_\Omega}$ and $\{\boldsymbol{x}_{\delta\Omega,i}\}_{i=1}^{N_{\delta\Omega}}$, that can be chosen from any collocation scheme. Finally, the following optimization problem is solved via gradient-based techniques,

$$\min_{\theta} \quad \sum_{i=1}^{N_\Omega} \left(||\mathcal{D}(u_\theta(\boldsymbol{x}_{\Omega,i};\boldsymbol{\Gamma})) - f(\boldsymbol{x}_{\Omega,i})||^2 \right) + \sum_{i=1}^{N_{\delta\Omega}} \left(||\mathcal{B}\left(u_\theta(\boldsymbol{x}_{\delta\Omega,i}) - g(\boldsymbol{x}_{\delta\Omega,i})||^2 \right) \right. \tag{1.6}$$

In Machine Learning (ML), optimization problems like those represented by Eq. (1.6) are called "training," especially when dealing with NNs. The specific type of training in Eq. (1.6) is called *physics-driven training* since the information to train the NN is brought by the physics of the problem, modeled via a DE. Once the training is completed, the optimal

NN parameters, θ^*, are learned and plugged back into Eq. (1.5). That is,

$$u_{\theta^*}(\boldsymbol{x}) = \sum_{j=1}^{L} \beta_j^* \sigma \left(\boldsymbol{w}_j^{*,T} \boldsymbol{x} + b_j^* \right) \tag{1.7}$$

which is a closed-form analytical approximation of the DE true unknown solution u. From either Eq. (1.5) or Eq. (1.7) can be seen how using NN to tackle DEs solves the state-of-the-art solver issue of providing purely numerical solutions. Being a closed-form analytical approximation, $u_{\theta^*}(\boldsymbol{x})$ can be evaluated of any desired query (or test) points within the domain where the DE was solved that are different from the training points. Moreover, integrals and derivatives of $u_{\theta^*}(\boldsymbol{x})$ can be evaluated analytically. Furthermore, since the training points can be selected by any collocation scheme (even Monte Carlo, e.g., random training points) and the activation functions are non (necessarily) orthogonal, the curse of dimensionality is heavily mitigated when using NN to tackle DEs. Indeed, although the number of training points and activation functions (e.g., the number of neurons) required to have accurate solutions may be high for high dimensional PDEs, it does not increase exponentially with the dimension of the PDE. Although physic-driven training is an attractive paradigm to tackle DEs, until recently ($\approx$ 2015-2019) it has not been often used. The main drawback is the computation and tracking of the (partial) derivatives of u during the network training via gradient-based techniques.
An alternative to the *physics-driven training* is the *data-driven training*. Within this scenario, NNs learn the DE solution u by fitting a labeled data set,

$$\{\boldsymbol{x}_i, \hat{u}_i\}_{i=1}^{N}$$

The label data can either be experimental (e.g., measurements of a phenomenon described by the DE of interest), synthetic (e.g., generated by solving the DE of interest with some state-of-the-art solver), or both. Here, the optimization problem to solve is the following,

$$\min_{\theta} \quad \sum_{i=1}^{N} ||\hat{u}_i - u_\theta(\boldsymbol{x}_i)||^2 \tag{1.8}$$

Although widely used because of its simplicity and relatively reasonable computational cost, data-driven training has a main limitation: the only information to train the NN comes from the data. Therefore, within the domain region where the data are sparse (or missing) u_θ may fail to represent u properly.
Leveraging on the increase in the computational capability brought with the ML explosion, especially Deep Learning (DL), which allows to easily compute and track the (partial) derivatives of u, Raissi et al. proposed a solution to the data-driven training issue [24]. In the pioneering work [24], the authors combine data-driven and physics-driven training in the so-called *data-physics-driven* training. In this scenario, the optimization problem to solve is

the following,

$$\min_{\theta} \quad \sum_{i=1}^{N} ||\hat{u}_i - u_\theta(\boldsymbol{x}_i)||^2 + \sum_{i=1}^{N_\Omega} \left(||\mathcal{D}(u_\theta(\boldsymbol{x}_{\Omega,i}; \boldsymbol{\Gamma})) - f(\boldsymbol{x}_{\Omega,i})||^2 \right) + \sum_{i=1}^{N_{\delta\Omega}} \left(||\mathcal{B}\left(u_\theta(\boldsymbol{x}_{\delta\Omega,i}) - g(\boldsymbol{x}_{\delta\Omega,i})||^2 \right) \right. \tag{1.9}$$

In [24] the authors also coin the term *Physics-Informed Neural Network (PINN)* referring to u_θ. Indeed, u_θ is physic-informed as the information for the training is brought from both the data and the physics of the problem under study, modeled via DE. It is trivial to see that even in the solely physic-driven scenario, u_θ is a PINN.
Another advantage of the data-physics-driven training is that it can also be used to solve inverse problems by simply adding the unknown $\boldsymbol{\Gamma}$ among the optimization problem decision variables. That is,

$$\min_{\theta,\boldsymbol{\Gamma}} \quad \sum_{i=1}^{N} ||\hat{u}_i - u_\theta(\boldsymbol{x}_i)||^2 + \sum_{i=1}^{N_\Omega} \left(||\mathcal{D}(u_\theta(\boldsymbol{x}_{\Omega,i}; \boldsymbol{\Gamma})) - f(\boldsymbol{x}_{\Omega,i})||^2 \right) + \sum_{i=1}^{N_{\delta\Omega}} \left(||\mathcal{B}\left(u_\theta(\boldsymbol{x}_{\delta\Omega,i}) - g(\boldsymbol{x}_{\delta\Omega,i})||^2 \right) \right. \tag{1.10}$$

Therefore, NNs, particularly PINNs, can overcome all the main limitations of state-of-the-art DE solvers. Nevertheless, PINNs have limitations too. For example, classic PINNs (e.g., as introduced by Raissi et al. in [24]) have two main limitations. The first is the use of gradient-based methods to train the network. The second one is that DE constraints (e.g., initial and/or boundary conditions) must be included in the optimization problem's objective function (or loss). The details regarding these limitations will be explained in the following chapter.

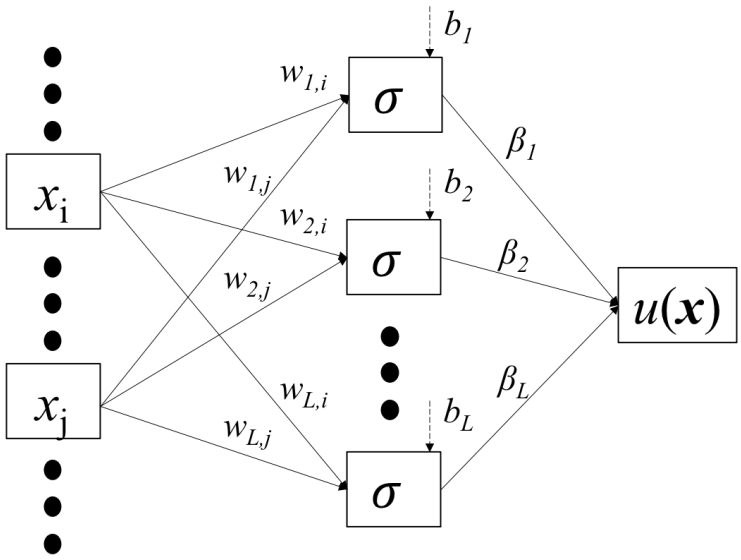


Figure 1.1: MISO NN schematic.

1.2 Aims, Novelty, and Contribution

This dissertation aims to introduce a novel and improved PINN framework called the *Extreme Theory of Functional Connections (X-TFC)* [25] for solving problems involving differential equations with applications to optimal control problems. The novelty of X-TFC is that for the first time, shallow NN with random features and the *Theory of Functional Connections (TFC)* [26, 27] are brought together. This synergy forms an efficient, robust, reliable, flexible, and unified framework to tackle forward and inverse problems governed by DE. TFC enables functional interpolation for a large class of mathematical objects. According to TFC, any mathematical problem solution can be represented via the so-called Constrained Expressions (CEs). The CEs are a sum of a free function and a functional that analytically satisfies the problem constraints. When applied to problems involving DEs, the mathematical problem is represented by the DE itself. The DE solution is approximated with the CE, where the free function, according to the X-TFC, is a shallow NN trained via the Extreme Learning Machine (ELM) algorithm [28]. Within the ELM algorithm, input weights and bias are randomly selected and not tuned during the training, leaving the output weights as the only trainable parameters. Thus, the training is reduced to a fast and robust least-squares.
Thanks to these features, X-TFC overcomes the main limitations of the classic PINN framework. This dissertation will show the X-TFC advantage over the classic and other PINN frameworks. In particular, it will show X-TFC flexibility with many applications in solving real-world problems involving differential equations. X-TFC has been applied to physics-driven solutions of several benchmark ODEs and PDEs, where it has been compared with some of the most common state-of-the-art DE solvers (Appendix A). X-TFC has been employed for physics-driven solutions of nuclear reactor dynamics (e.g., point kinetic equations with temperature feedback, Appendix B). This application was chosen because of the author's background. The author of this work earned his bachelor's and master's degrees in Energy and Nuclear Engineering. Another application is for physics-driven solutions of Optimal Control Problems (OCPs), both via the application of the indirect method (e.g., Pontryagin Minimum Principle) and the Bellman Principle of Optimality, both for general and aerospace optimal control problems (Appendices C, and D). The applications to OCPs were chosen as OCPs represent one of the leading research interests in the author's lab (Space Systems Engineering Laboratory, SSEL). Finally, X-TFC has also been applied for data-physics-driven parameter discovery of epidemiological models that regulate virus spread (Appendix E). This application was chosen in light of the COVID-19 pandemic.

The author of this dissertation conceptualized and was the lead developer of the X-TFC framework. In addition, the author had the lead in several research (peer-reviewed) articles where X-TFC has been applied to many real-world problems.

Chapter 2

Physics-Informed Neural Networks Frameworks and Extreme Theory of Functional Connections

In recent years, Neural Networks (NNs) have become widespread across many scientific fields. This requires designing them to satisfy application-specific constraints, such as conservation laws, symmetries, or other domain-specific knowledge. These constraints are usually imposed as soft penalties during network training and act as regularizers within the loss function. PINNs are examples of this philosophy, where the network outputs are constrained to approximately satisfy specific physics laws, modeled as a set of Differential Equations (DEs). More precisely, the term PINN refers to NNs that use physics as a regularizer in the loss function. For instance, assume that one wants to perform a regression of a set of experimental data using a NN and that such data represent physical phenomena modeled via a set of DEs. In classic regression, one could approximate the data using a NN trained to minimize a Mean Squared Error (MSE) as a loss function. However, there is no guarantee that the physics phenomena governing the data would not be violated. PINNs are introduced to add the DEs, modeling the physics of the experimental data set, as a penalty to the loss function. This additional term acts as a regularizer that penalizes the training when the DE and its constraints (e.g., initial conditions ICs and/or boundary conditions BCs) are violated. Overall, one aims to ensure that the process's physics is not violated. This approach is called the data-physics-driven solution of DEs. Conversely, when the goal is to estimate parameters governing some physical phenomena modeled through a DE (e.g., the thermal conductivity in the heat equation), one usually talks about data-physics-driven parameter discovery (or estimation) of DEs (e.g., inverse problems). When data is unavailable, and therefore the loss function contains only the residual of the DEs and the corresponding DE's constraints, PINNs are used for approximating the solutions of problems involving DEs solely in a physics-driven fashion.

In this chapter, we will show, with an example, how the classic PINN framework is applied to solve a DE in a physics-driven fashion. We will discuss its main limitations. Then, with the same example, we will show how a couple of variants of the classic PINN framework work

and how they mitigate and remove some of the classic PINN framework's main limitations. Finally, we will introduce X-TFC and explain how it mitigates and removes all the classic PINN framework's main limitations.

2.1 Classic PINN

Consider, as an example, the following bivariate PDE,

$$-u_t - u_x x + \frac{1}{4}u_x^2 = 0, \quad \forall t \in [0,1], \forall x \in [-1,1] \tag{2.1}$$

subject to the following constraints (e.g., final time condition),

$$u(1,x) = x^2, \quad \forall x \in [-1,1] \tag{2.2}$$

where u depends on time t and space x, u_t is the partial derivative with respect t, and u_x is the partial derivative with respect x (e.g., the gradient of u). According to the classic PINN framework, u (in strong form [29]) is approximated with a Deep NN (DNN). That is,

$$u(t,x) \simeq u(t,x;\theta) = u^\theta(t,x) \tag{2.3}$$

where θ are the DNN parameters to be learned. u^θ is a PINN as its training will be driven by the problem physics, modeled via DE. u^θ is then plugged into (2.1) and (2.2) to form the interior residual $\mathcal{R}_{\text{int},\theta}$ and temporal boundary residual $\mathcal{R}_{tb,\theta}$, The interior residual is,

$$\mathcal{R}_{\text{int},\theta}(t,x) = -u_t^\theta - u_x^\theta\, x + \frac{1}{4}(u_x^\theta)^2, \quad \forall t \in [0,1], \forall x \in [-1,1] \tag{2.4}$$

The temporal boundary residual is,

$$\mathcal{R}_{tb,\theta}(x) = u^\theta(1,x) - x^2, \quad \forall x \in [-1,1] \tag{2.5}$$

The following scalar physics-driven loss function, $\mathcal{L}(\theta)$, is formed by collocating the residuals on the training points,

$$\mathcal{L}(\theta) = \lambda_{\text{int}} \sum_{n=1}^{N_{\text{int}}} ||\mathcal{R}_{\text{int},\theta}(t_n,x_n)||^2 + \lambda_{\text{tb}} \sum_{n=1}^{N_{\text{tb}}} ||\mathcal{R}_{tb,\theta}(x_n)||^2 \tag{2.6}$$

where, N_{int} are the interior training points (e.g., the points sampled from $t \in [0,1]$ and $x \in [-1,1]$), N_{tb} are the temporal boundary training points, (e.g., the points sampled from $x \in [-1,1]$), and λ_{int} and λ_{tb} are hyperparameters for balancing the interior residual over the temporal boundary residual. Now, to find PINN, $u^* = u^{\theta^*}$, to approximate the true solution

u of the problem (2.1)-(2.2) as best as possible, the following optimization problem is solved

$$\theta^* = \min_{\theta \in \Theta} \mathcal{L}(\theta) \tag{2.7}$$

where Θ is the search space of the NN parameters, whose formal definition is given in Ref. [30].
The classic PINN framework (e.g., as introduced by Raissi et al. in [24]) presents two main drawbacks: using standard DNN to approximate the DE solutions and the fact that the equation's constraints must be included in the loss function. The use of standard DNNs introduces four main issues,

- Gradient-based methods are required to minimize the scalar loss function. These methods are computationally expensive, especially computational time.
- The complexity of the DNN may cause the divergence of the training. Indeed a non-convex optimization problem is solved.
- Linear DEs are treated as nonlinear ones, as the network parameters appear non-linearly to approximate the DE solutions.
- For nonlinear DEs, it is prohibitive to provide an informative initial guess on the solution if needed

Having the equation's constraints included in the loss function introduces the issues of having competing objectives during the PINN training: learning the DE solution within the domain and learning its constraints. This leads to unbalanced gradients when training the networks via gradient-based techniques that cause PINNs to have difficulties approximating the DE solution accurately [31]. Indeed, gradient-based optimization techniques may get stuck in limit cycles or diverge if multiple competing objectives are present [32, 33].
In [31], to overcome the issue caused by having competing objectives during the PINN training, the authors proposed a learning rate annealing algorithm. It uses gradient statistics to adaptively assign proper weights to the different terms forming the PINN loss function (e.g., DE residuals within the domain and DE residuals on the boundaries) during the network training. With the algorithm proposed in [31], the issue of having competing objectives during the PINN training is mitigated, but the issue of using standard DNN remains.
Thus, a significant advance in the PINN research area would be to introduce a PINN framework able to remove both the issues discussed above.

2.2 PIELM

In [34], the authors overcome the issues introduced by using standard DNN by employing shallow NN trained via Extreme Learning Machine (ELM) algorithm [28]. The advantage of the ELM algorithm is that input weights and biases are randomly selected. The training

is solely necessary for the output weights, with a significant reduction in the computational effort. This PINN framework is called *Physics-Informed Extreme Learning Machines* (PIELM).

Consider the problem (2.1)-(2.2). According to PIELM, u is approximated with a shallow NN trained via ELM algorithm or simply an ELM. That is,

$$u(t,x) \simeq u(t,x;\boldsymbol{\beta}) = u^{\boldsymbol{\beta}}(t,x) \tag{2.8}$$

where $\boldsymbol{\beta} \in \mathbb{R}^L$ are the ELM output weights. $u^{\boldsymbol{\beta}}$ is then plugged into (2.1) and (2.2) to form the interior residual $\mathcal{R}_{\text{int},\boldsymbol{\beta}}$ and temporal boundary residual $\mathcal{R}_{tb,\boldsymbol{\beta}}$, The interior residual is,

$$\mathcal{R}_{\text{int},\boldsymbol{\beta}}(t,x) = -u_t^{\boldsymbol{\beta}} - u_x^{\boldsymbol{\beta}}\, x + \frac{1}{4}(u_x^{\boldsymbol{\beta}})^2, \quad \forall t \in [0,1],\, \forall x \in [-1,1] \tag{2.9}$$

The temporal boundary residual is,

$$\mathcal{R}_{tb,\boldsymbol{\beta}}(x) = u^{\boldsymbol{\beta}}(1,x) - x^2, \quad \forall x \in [-1,1] \tag{2.10}$$

The following vectorial physics-driven loss function, $\boldsymbol{\mathcal{L}}(\boldsymbol{\beta})$, is formed by collocating the residuals on the training points,

$$\boldsymbol{\mathcal{L}}(\boldsymbol{\beta}) = \begin{Bmatrix} \mathcal{R}_{\text{int},\boldsymbol{\beta}}(t_1,x_1) \\ \vdots \\ \mathcal{R}_{\text{int},\boldsymbol{\beta}}(t_{N_{\text{int}}},x_{N_{\text{int}}}) \\ \mathcal{R}_{tb,\boldsymbol{\beta}}(x_1) \\ \vdots \\ \mathcal{R}_{tb,\boldsymbol{\beta}}(x_{N_{\text{tb}}}) \end{Bmatrix} \tag{2.11}$$

Now, to find PINN, $u^* = u^{\boldsymbol{\beta}^*}$, to approximate the true solution u of the problem (2.1)-(2.2) as best as possible the following optimization problem is solved,

$$\boldsymbol{\beta}^* = \min_{\boldsymbol{\beta} \in \mathbb{R}^L} \boldsymbol{\mathcal{L}}(\boldsymbol{\beta}) \tag{2.12}$$

If the DE is linear, the optimization problem (2.12) reduces to a linear system of algebraic equations, which is solved via a single pass linear least-squares. If the DE is nonlinear, as in our example, the optimization problem (2.12) is solved via any iterative least-squares procedure. That is,

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k + \Delta\boldsymbol{\beta}_k$$

where the k subscript is the iteration counter. The $\Delta\boldsymbol{\beta}_k$ term can be computed via linear least-squares at each iterative least-square procedure iteration. For instance, if a classic linear least-square is used, then,

$$\Delta\boldsymbol{\beta}_k = -\left(\mathbb{J}^T(\boldsymbol{\beta}_k)\mathbb{J}(\boldsymbol{\beta}_k)\right)^{-1}\mathbb{J}^T(\boldsymbol{\beta}_k)\boldsymbol{\mathcal{L}}(\boldsymbol{\beta}_k)$$

where $\mathbb{J}$ is the Jacobian matrix containing the derivatives of the physics-driven loss function $\boldsymbol{\mathcal{L}}(\boldsymbol{\beta})$ with respect to the output weights $\boldsymbol{\beta}$. One may obtain the Jacobian either by hand or by employing numerical toolboxes, such as automatic differentiation and symbolic computation. The iterative process is repeated until the following condition is met,

$$|\,||\boldsymbol{\mathcal{L}}(\boldsymbol{\beta}_{k+1})|| - ||\boldsymbol{\mathcal{L}}(\boldsymbol{\beta}_k)||\,| < \epsilon$$

where ϵ is a user-prescribed tolerance.
Nevertheless, PIELM does not remove the issues of having the DE constraints included in the loss function, although it mitigates it, especially when the DE is sufficiently complex (e.g., presenting sharp gradients, stiffness, etc.).

2.3 Deep-TFC

In [35], the authors remove the issue of having the DE constraints included in the loss function by merging, for the first time, DNNs and TFC. This PINN framework is called *Deep Theory of Functional Connections* (Deep-TFC).
Consider the problem (2.1)-(2.2). According to Deep-TFC, u is approximated with a TFC CE, where the free function $g(t,x)$ is a DNN. That is,

$$u(t,x) \simeq u(t,x;\theta) = u^\theta(t,x) = g^\theta(t,x) + \Big(x^2 - g^\theta(1,x)\Big) \tag{2.13}$$

where θ are the DNN parameters to be learned. u^θ is a PINN as its training will be driven by the problem physics, modeled via DE. u^θ is then plugged into (2.1) and (2.2) to form the interior residual $\mathcal{R}_{\text{int},\theta}$ and temporal boundary residual $\mathcal{R}_{tb,\theta}$, The interior residual is,

$$\mathcal{R}_{\text{int},\theta}(t,x) = -u_t^\theta - u_x^\theta\, x + \frac{1}{4}(u_x^\theta)^2, \quad \forall t \in [0,1],\, \forall x \in [-1,1] \tag{2.14}$$

The temporal boundary residual is,

$$\mathcal{R}_{tb,\theta}(x) = u^\theta(1,x) - x^2 = 0, \quad \forall x \in [-1,1] \tag{2.15}$$

The following scalar physics-driven loss function, $\mathcal{L}(\theta)$, is formed by collocating the residuals on the training points,

$$\mathcal{L}(\theta) = \sum_{n=1}^{N_{\text{int}}} ||\mathcal{R}_{\text{int},\theta}(t_n,x_n)||^2 \tag{2.16}$$

Now, to find PINN, $u^* = u^{\theta^*}$, to approximate the true solution u of the problem (2.1)-(2.2) as best as possible the following optimization problem is solved

$$\theta^* = \min_{\theta\in\Theta} \mathcal{L}(\theta) \tag{2.17}$$

where Θ is the search space of the NN parameters, whose formal definition is given in Ref. [30].
Nevertheless, Deep-TFC does not remove the issues caused by having u approximated via a DNN.

2.4 X-TFC

In [25], for the first time, the authors merge ELM and TFC, generating a PINN framework that overcomes all the main issues of the classic one. Indeed, using TFC removes the issue of having the DE constraints included in the loss function. Hence, this PINN framework is called *Extreme Theory of Functional Connections* (X-TFC). Using ELM removes the issues caused by approximating u via a DNN. Of course, the convergence issue for highly nonlinear and non-convex problems remains. Although it is heavily mitigated, as shown in the results, it is not solved. However, in most applications for highly nonlinear and non-convex problems, X-TFC seems not to be highly sensitive to the choice of the initial guess for the solutions. Whereas an informative initial guess was needed, thanks to the ELM properties, it is possible to provide one.
Consider the problem (2.1)-(2.2). According to X-TFC, u is approximated with a TFC CE, where the free function $g(t,x)$ is an ELM. That is,

$$u(t,x) \simeq u(t,x;\boldsymbol{\beta}) = u^{\boldsymbol{\beta}}(t,x) = g^{\boldsymbol{\beta}}(t,x) + \Big(x^2 - g^{\boldsymbol{\beta}}(1,x)\Big) \tag{2.18}$$

where $\boldsymbol{\beta} \in \mathbb{R}^L$ are the ELM output weights. $u^{\boldsymbol{\beta}}$ is then plugged into (2.1) and (2.2) to form the interior residual $\mathcal{R}_{\text{int},\boldsymbol{\beta}}$ and temporal boundary residual $\mathcal{R}_{tb,\boldsymbol{\beta}}$, The interior residual is,

$$\mathcal{R}_{\text{int},\boldsymbol{\beta}}(t,x) = -u_t^{\boldsymbol{\beta}} - u_x^{\boldsymbol{\beta}}\, x + \frac{1}{4}(u_x^{\boldsymbol{\beta}})^2, \quad \forall t \in [0,1],\, \forall x \in [-1,1] \tag{2.19}$$

The temporal boundary residual is,

$$\mathcal{R}_{tb,\boldsymbol{\beta}}(x) = u^{\boldsymbol{\beta}}(1,x) - x^2 = 0, \quad \forall x \in [-1,1] \tag{2.20}$$

The following vectorial physics-driven loss function, $\boldsymbol{\mathcal{L}}(\boldsymbol{\beta})$, is formed by collocating the residuals on the training points,

$$\boldsymbol{\mathcal{L}}(\boldsymbol{\beta}) = \begin{Bmatrix} \mathcal{R}_{\text{int},\boldsymbol{\beta}}(t_1,x_1) \\ \vdots \\ \mathcal{R}_{\text{int},\boldsymbol{\beta}}(t_{N_{\text{int}}},x_{N_{\text{int}}}) \end{Bmatrix} \tag{2.21}$$

Now, to find PINN, $u^* = u^{\boldsymbol{\beta}^*}$, to approximate the true solution u of the problem (2.1)-(2.2) as best as possible the following optimization problem is solved,

$$\boldsymbol{\beta}^* = \min_{\boldsymbol{\beta} \in \mathbb{R}^L} \boldsymbol{\mathcal{L}}(\boldsymbol{\beta}) \tag{2.22}$$

If the DE is linear, the optimization problem (2.22) reduces to a linear system of algebraic equations, which is solved via a single pass linear least-squares. If the DE is nonlinear, as in our example, the optimization problem (2.22) is solved via the same iterative least-squares procedure used for PIELM.

The peer-reviewed paper [25] where X-TFC is introduced is attached in Appendix A. More details on X-TFC can be found there. The dissertation author's contribution to the work mentioned above is the following,

- Conceptualization, methodology, investigation
- Writing original and revised manuscripts
- Manuscript structure organization
- Problems selection
- Software

Chapter 3

X-TFC for Learning Forward Problem Solutions

This chapter shows the X-TFC employment to learn the solutions for several forward problems selected from different engineering and science fields. These solutions are learned solely in a physics-driven fashion. The main aims of this chapter are the following:

- To test and validate X-TFC performances over a variety of DEs with comparisons against the state-of-the-art methods
- To show the X-TFC flexibility and adaptability. Indeed, the reader will appreciate how the X-TFC application, in terms of problem formulation and coding, remains substantially the same from problems with completely different characteristics and properties belonging to different fields
- To present how X-TFC is adapted to tackle initial value ODEs (of any order) with long integration time and/or presenting stiffness in the dynamics

All the selected problems are coded in MATLAB (2019a and higher), Python (3.7 and higher), or both, and run with an Intel Core i7 - 9700 CPU PC with 64 GB of RAM.

3.1 X-TFC for General Benchmarks ODEs and PDEs

X-TFC has been tested on several benchmarks, ODEs, and PDEs with known real solutions. This allows us to evaluate the generalization error that X-TFC generates in approximating those solutions. Thus, it will provide an overall fair glance at how X-TFC will perform in real-world problems, where the real exact solutions are not available. These problems are found in Appendix A [25].
The maximum absolute (training and test) error has been used to evaluate the X-TFC performances in terms of accuracy. That is,

$$\max_{i=1,\dots,N}(\boldsymbol{\varepsilon}) = \max_{i=1,\dots,N}(|\boldsymbol{u} - \boldsymbol{u}^*|) \tag{3.1}$$

where $\boldsymbol{u} = \{u_1, \ldots, u_N\}^T$ is the real DE solution, $\boldsymbol{u}^* = \{u_1^*, \ldots, u_N^*\}^T$ is the X-TFC approximation of the DE solution, and N is the number of training/test points.
As previously mentioned, the work in Appendix A introduces X-TFC. The introduction provides an extensive literature review on numerical methods for solving DEs, and the motivations for X-TFC are provided in detail. In the methodology section, how X-TFC is applied to learn the solution of generic DEs, both in a physics-driven and data-physics-driven fashion, is presented in detail. A couple of simple procedural examples are also provided. Moreover, some preliminary theoretical considerations on the X-TFC error in approximating generic DE solutions are discussed. In the results section, X-TFC is applied to tackle several benchmarks ODEs and PDEs, and the performances are compared against some of the most common and used state-of-the-art methods. A nonlinear ODE taken from [36] and a system of nonlinear ODEs taken from [23] are solved. X-TFC solutions are compared with the methods presented in [36] and [23], respectively, and references within. The results show that X-TFC outperforms all the methods it has been compared with in terms of accuracy and computational effort. However, regarding these two examples, the most interesting comparison is with classic (or Vanilla, as defined in [37] and [38]) TFC. The performances are comparable in terms of accuracy and computational efforts. As previously explained, TFC is a general mathematical framework for functional interpolation, invented and introduced in [26] by Mortari. TFC can potentially be applied to any mathematical problem. For instance, TFC has been applied to quadratic and nonlinear programming under equality constraints [39]. Another TFC application is for homotopy continuation algorithm with application to dynamics and control problems [40]. As of today, most of the TFC applications are related to DEs [41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55]. For solving DE, the classic TFC employs a linear combination of orthogonal polynomials as free-function [56, 57]. Although employing orthogonal polynomials allows us to achieve highly accurate results, it affects the original TFC framework with the curse of dimensionality. In particular, when solving large-scale PDEs (e.g., PDEs with more than three independent variables), the computational cost grows exponentially with the number of bases required to compute accurate solutions [58]. This is the primary motivation that leads the authors of this dissertation to develop X-TFC. That is, to have a TFC-based framework able to tackle large-scale PDEs. However, when working on the X-TFC development, the author, thanks to the two examples reported in [25] and many other test cases, showed that X-TFC and classic TFC performances are comparable even for ODEs and small-scale PDEs. Thus, the author presented X-TFC as a general method for all types of DEs, not only large-scale PDEs. In the results section of the work in Appendix A, several PDEs are also solved. Among those, the first three PDEs considered are the most interesting, where X-TFC solutions are compared with the companion Deep-TFC and FEM. The results show that X-TFC outperforms both Deep-TFC and FEM in the accuracy and computational efforts.

The dissertation author's contributions to the peer-reviewed article [25] attached in Appendix A are listed at the end of Chapter 2.

3.2 X-TFC for Nuclear Reactor Dynamics

In Appendix C, X-TFC has been employed to learn the solutions of the Point Kinetics Equations (PKEs) with temperature feedback for nuclear reactor dynamics. This real-world application has been selected because it is attractive to the nuclear power industry (where the dissertation author has his background) and is computationally challenging.

An accurate and fast solution of the reactor kinetic equations is essential for the reactor transient simulation during safety analyses. The general and complete solution of the time-dependent problem demands solving the linear Boltzmann equation coupled with the precursors' equations, typically characterized by very different timescales, generating a system of stiff DEs. Moreover, the neutron-precursors equations must then be coupled with the temperature distribution through the thermal feedback, introducing non-linearity into the dynamical systems. To make the problem computationally manageable, approximations to simplify the reactor kinetic equations have been developed and introduced. The most common approximation is the PKEs, firstly derived by Henry [59] and still used in many applications of existing codes. The basic idea behind PKEs is to factorize the solution of the time-dependent neutron-precursor balance equations in amplitude (function of the time only) and shape (function of space, energy, and angle) and project the equations onto a weight. This reduces the mathematical problem from a system of integro-differential equations to a system of ODEs. More accurate approximations of the reactor kinetics equations, such as the quasi-static approach [60] consider the evolution of the shape on a longer timescale. However, the resulting set of differential equations modeling the kinetics of the amplitude in the quasi-static approximation is still in the form of the PKEs. Similarly, other classical methods like multi-point kinetics or modal methods can be reduced to a mathematical problem equivalent to the PKEs (Picca et al. [61]).

As the analytical solution of PKEs is only possible in idealized cases, numerical methods have been developed to simulate transients of interest in applications. Although there is extensive literature on solution methods for PKEs, the topic is still attracting the interest of various research groups, both for improving the efficiency of solution techniques and deriving highly accurate benchmark solutions to test commercial codes.

The most interesting computational challenge of the PKEs is that they are systems of highly nonlinear stiff ODEs, especially when the temperature feedback is considered, with a long integration time. Thus, X-TFC needs to be adapted to tackle these problems to control and minimize the carryover of the generalization error. The time interval is divided into smaller sub-intervals. As showed in Fig. 4 of Appendix A, the domain is divided into n time steps of equal size $h = t_i - t_{i-1}$, with $i = 1, ..., n$. Starting from the first time step, we employ the initial conditions given by the problem at time t_0. The single-time step is discretized into n_x training points. By solving the DEs, we find the solutions at time t_1. These solutions become the initial values for the time step 2, and so on. Thus, for each time step, X-TFC is applied recursively up to the last time step of the total time interval, which, for the PKEs, represents the total time for the reactor operation.

In Appendix C, the accuracy of X-TFC is tested against several published benchmarks for

different test cases of PKEs, showing its performance in terms of accuracy and computational time.
All the problems considered in [62] have been coded in MATLAB R2020a and run with an Intel Core i7 - 9700 CPU PC with 64 GB of RAM.

The dissertation author's contributions to the peer-reviewed article [62] attached in Appendix C are listed below,

- Conceptualization, methodology, investigation
- Adaptation of X-TFC to stiff systems of nonlinear ODE with a long integration time
- Writing the original manuscript
- Manuscript structure organization
- Problems selection
- Software

3.3 X-TFC for Optimal Control Problems

In Appendices C and D, X-TFC has been employed to learn OCP solutions via the application of the indirect method and the application of the BPO. More specifically,

- Appendix C [63] shows how X-TFC is employed to learn the optimal control actions for planar orbit transfer.
- Appendix D [64] introduces *Bellman Neural Networks* (BeNN) and how they are used to learn closed-loop optimal control actions for OCPs that present continuous solutions. Moreover, the estimate of the generalization error of BeNNs in learning OCP solutions is derived.

The continued and exponential increase in computational capabilities has the potential to enable a new level of autonomy and decision-making, especially for complex systems, such as aerospace systems. In particular, the ability to generate real-time trajectories may be considered a technology of paramount importance for managing autonomous systems within complex environments. Similarly, new analytical and data-driven methods for real-time synthesis of closed-loop optimal controllers may integrate autonomy with robustness and stability. Optimal trajectory generation and closed-loop control rely on optimal control theory, a well-established discipline. Indeed, given a prescribed model of the dynamical system under consideration, it aims to find optimal policies and trajectories by minimizing a cost function that accounts for all constraints and design objectives. Usually, two methods are used to solve OCPs, i.e., direct and indirect.

Direct methods discretize the continuous states and controls by transforming the original continuous problem into a Non Linear Programming (NLP) problem [65, 66, 67]. The latter is formulated as a finite constrained optimization problem solved via numerical algorithms that find a local minimum (e.g., thrust region method [68]). Direct methods have been employed to tackle a large variety of OCPs, for example, in the aerospace field (e.g., [69, 70, 71, 72]). The general NLP problem is considered NP-hard, i.e., non-deterministic polynomial-time hard. NP-hard problems imply that the amount of computational time needed to find the optimal solution does not have a predetermined bound (i.e., a bound cannot be a priori determined). NP-hard problems are such that the computational time necessary to converge to the solution is unknown. Consequently, the lack of assured convergence may question the proposed approach's reliability, especially for real-time applications. More recently, researchers have been experimenting with transforming OCPs from a general non-convex formulation into a convex optimization problem [73, 74]. Here, the goal is to leverage the assured convergence properties of convex problems. Indeed, convex optimization problems are computationally tractable as their related numerical algorithms guarantee convergence to a globally optimal solution in a polynomial time. The general convex methodology needs the optimal guidance problem formulated as convex optimization whenever appropriate convexification techniques are applied to transform the problem from a non-convex into a convex one. For instance, these methodologies have been used to solve optimal guidance and control via direct method in a large variety of aerospace problems including, planetary landing [73, 74], entry atmospheric guidance [75, 76], rocket ascent guidance [77], and low thrust [78].

OCP can also be solved by *Indirect methods*. The latter derive the first-order necessary conditions by direct application of *Pontryagin Maximum Principle* (PMP) or by the calculus of variations. The necessary conditions result in a Two-Point Boundary Value Problem (TPBVP) in the state-costate pair that must be generally solved by the application of numerical techniques such as single and multiple shooting methods [79, 80], orthogonal collocation [81], or pseudospectral methods [82]. The indirect method generates a solution that is guaranteed to be optimal. This is why it should be preferred to the direct method. However, the solution of the TPBVPs is very cumbersome because of the strong sensitivity of the solution to the initial guesses. Moreover, it is also challenging to provide good initial guesses for the costates as they do not represent any physical quantity. Conversely, although the direct method can provide not necessarily optimal solutions, the solution is more straightforward to compute. Thus, the direct method is much more spread within the scientific community. In principle, the two approaches can also be merged. That is, the solution obtained with the direct method can be used as the initial guess for the indirect method, which will refine the solution and, simultaneously, guarantees optimality.

A third and complementary approach to solving OCP comes from the application of the *Bellman Principle of Optimality* (BPO). The latter provides the necessary and sufficient conditions for optimality which yields in writing the so-called *Hamilton-Jacobi-Bellman Equation* (HJB) equation. The HJB is a nonlinear PDE, which plays a crucial role in the optimal control theory. The solution of the HJB equation is the *value function* in all the time-

state space. The value function represents the optimal *cost-to-go* for a dynamical system with an associated cost function. Once the value function is known, the optimal control is also known as it is the gradient of the value function to the state variables. That is, if we learned the solution of the HJB equation in all the time-state space, we would also consequentially learn the optimal control actions in all the time-state space in a closed-loop fashion. However, since it is not trivial and, in many cases, it is most likely impossible to find an analytical solution to the HJB equation, one has to resort to numerical methods. Many numerical methods have been proposed in the literature to generate closed-loop solutions to the HJB equation. One possible approach is through generalized characteristics [83], which can quickly obtain the solution at a single point when the solution is smooth. Nevertheless, since the HJB is a nonlinear PDE, the solutions are generally non-smooth (except for the linear/quadratic case), which causes difficulties in finding the solution via the characteristics method. The theory of viscosity solutions has given a significant improvement in solving the HJB equation [84, 85]. This allows defining a continuous function as the unique solution to the HJB equations that do not admit continuous solutions in the classical sense (i.e., the solution must be differentiable over the domain of interest). As a result, the viscosity solution of the HJB equation is identical to the value function of the associated (stochastic or deterministic) OCP. Moreover, if the solution of the HJB equation is differentiable, then the viscosity solution and the classical solution are identical. To evaluate the viscosity solution of the HJB, various approaches have been used, such as the semi-Lagrangian schemes [86, 87] and grid-based methods, involving both the finite-difference and finite-element methods. In particular, a first-order semi-lagrangian scheme is used in [88] to roughly approximate the value function appearing in the HJB PDE, which is then used to provide a suitable initial guess for the PMP method. As can be easily understood, the advantage of using the HJB PDE-PMP coupling over other initialization techniques (such as a continuation or direct methods) resides in providing an initial guess close to the global minimum. This approach results in a very efficient algorithm. For what concern the finite-difference methods, they aim at approximating the dynamics of the system through Euler formula [89] or Runge-Kutta scheme [90]. Consequently, the OCP can then be approximated by solving the discrete dynamic programming problem, and the computed discrete solution is shown to (locally uniformly) converge to the viscosity solution of the HJB equation as the step size vanishes. On the other hand, FEMs [91, 29] have also been employed to study HJB equation solutions, such as in [92, 93], where finite-time problems with continuous and impulse controls for stationary and non-stationary systems have been considered. However, even for these cases, the method converges to the viscosity solution, and error estimates for the algorithm convergence are also provided. The main disadvantage of FEMs and FDMs is that they require a discretization of the state space, which causes different problems. First, an enormous amount of data must be stored and recalled in real-time to produce feedback control. Second, they are subject to the so-called *curse of dimensionality*, which is also typical of many other numerical techniques. The term *curse of dimensionality* was coined by Bellman, referring to the issue caused by the exponential increase in volume related to the addition of extra dimensions to a mathematical space. These problems make the applica-

tion of many numerical methods infeasible as vastly computational times are required when there are many independent variables (e.g., time and states) in the value function. This is why other methods have been investigated to try to limit as much as possible the drawback of the curse of dimensionality. For example, Galerkin's spectral method has been used in [94, 95] to solve the generalized HJB equation, thus improving the closed-loop performance of stabilizing feedback control laws for nonlinear systems. Galerkin's spectral method has been shown to approximate the HJB equation such that the resulting control is in a feedback form and stabilizes the closed-loop system. Thanks to this method, the discretization of time and space variables is avoided. The main advantages of this method can be listed as follows: it produces control laws in a feedback form, which are determined by a series of basis functions and a small number of coefficients; the controls are robust in the same sense as the optimal control; all computations can be performed off-line, and once a solution is found, the control can be implemented in hardware and run in real-time. Concerning the convergence of this method, it was guaranteed for a sufficiently large number of basis functions (N); however, an estimation of this number was not given. The method's main drawback is that $O(N^2)$ n-dimensional integrals need to be computed. Moreover, since the control laws are given as a series of basis functions, they are inherently complex. Max-plus basis methods have contributed significantly to the solution of HJB PDE [96] that exploits the max-plus algebra. They use a max-plus basis function expansion of the solution, and then numerical methods obtain the coefficients in the basis expansion. With the max-plus basis methods, the "time-step" is larger than what can be used in grid-based methods (since it includes the action of the semi-group propagation on each basis function), and so these methods can be quite fast on small-scale problems. However, the number of basis functions required grows exponentially with space dimension. Consequently, one still has the curse of dimensionality, and its growth is too fast that one cannot tackle general problems of more than dimension $d = 5$ on current machinery. Many researchers have noticed that introducing even a single, simple non-linearity into an otherwise linear control problem of high dimensionality has disastrous computational repercussions. Specifically, one goes from the solution of a d-dimensional Riccati equation to the solution of a grid-based or max-plus basis method n dimensions space. While the Riccati equation may be "relatively" easy to tackle for large d, the max-plus, and grid-based methods have no hope of obtaining solutions to general problems of dimension $d > 6$. Over the years, researchers have been trying to come up with methods and models to solve the HJB equation that are not affected by the curse of dimensionality, an inherent problem of the HJB equation. Recently, a curse-of-dimensionality-free method was developed to solve HJB PDE for nonlinear control problems, using semi-convex duality and max-plus analysis [97]. This method may be applied to HJB PDEs where the Hamiltonian is given as (or well-approximated by) a pointwise maximum of quadratic forms. The method builds the correct solution of an HJB PDE from a max-plus-linear combination of quadratics. The method completely avoids the curse of dimensionality and is subject to cubic computational growth as a function of spatial dimension. However, it is subject to a *curse-of-complexity*. The number of quadratics in the approximation increases exponentially with the iteration number. The efficacy of such a method depends on the pruning of quadrat-

ics to keep the complexity growth at a reasonable level. In [98], a pruning algorithm based on semi-definite programming is employed to speed up the computation. Other methods to solve HJB PDE rely on the Level Set Method, a general framework for the theoretical and numerical study of an evolving domain according to a velocity field. It was first introduced in [99] and applied, for example, in [100] to compute an approximate solution to the HJB equation. In recent years, the machine learning community has also faced the problem of solving the HJB equation. Indeed, an empirical study of iterative least squares minimization of the Hamilton–Jacobi–Bellman (HJB) residual with a neural network (NN) approximation of the value function is proposed in [101]. Moreover, HJB PDE can be solved via Dynamic Programming (DP), which can be handled via Reinforcement Learning (RL) algorithms. RL algorithms aim to learn the optimal policy and value function, and unlike traditional optimal control, RL finds the solution to the HJB PDE online in real-time. Moreover, thanks to RL, the requirement of complete knowledge about the system dynamics can be relaxed, thus considering uncertain dynamical systems. Two main approaches can be pursued via RL: the Policy Iteration (PI) [102], and the online actor-critic [103], which has led to the definition of Adaptive Dynamic Programming (ADP) algorithms. PI is based on a two-step iteration: policy evaluation and policy improvement. The PI algorithm starts by evaluating the cost of a given initial admissible control policy. This is done by solving a nonlinear Lyapunov equation. This new cost is then used to obtain a new, improved (i.e., which will have a lower associated cost) control policy. These two steps of policy evaluation and improvement are repeated until the policy improvement step no longer changes the actual policy. Thus convergence to the optimal controller is achieved. On the other hand, in the actor-critic approach, the critic approximator estimates the value function and is updated to minimize the Bellman error, whereas the actor approximator approximates the control policy and is updated to minimize the value function [104]. The actor/critic approach has been widely employed in literature for both discrete [105] and continuous-time systems [106, 107, 108, 109]. Subsequently, the actor-critic approach has been extended with the actor-critic-identifier (ACI) structure [110], in which three NNs are used to approximate the optimal control, the optimal value function, and the uncertain dynamical system, respectively. One of the most successful applications of RL algorithms is aerospace applications. Indeed, in the last few years, a multitude of applications to spacecraft guidance have been developed, ranging from low thrust trajectory design [111, 112, 113, 114], formation flying [115], and rendezvous [116, 117, 118]. RL, and Reinforcement Meta-Learning (RML), have been explored for space engineering, regarding planetary landing [119, 120, 121], asteroids close-proximity operations [122], and relative motion [123, 124] frameworks. All these works have proven the ability of RL to generate guidance laws in complex environments even with non-convex constraints scenarios, with the added advantage of having a low computational cost at the test phase, which could enable real-time use. Aerospace real-time applications have also been studied in [125], where Bellman's principle of optimality has been used to develop a fast algorithm to obtain accurate low-thrust optimal trajectories. Moreover, Bellman's principle of optimality has been exploited together with pseudospectral methods in [126] to solve satellite attitude control problems and generic OCPs.

In this dissertation will be shown how X-TFC is employed to learn OCPs solutions both via the indirect method and via the BPO application.
Consider a general OCP,

$$\min_{\boldsymbol{u}\in\mathcal{U}} J(t,\boldsymbol{x},\boldsymbol{u}) = \boldsymbol{\phi}(\boldsymbol{x}(t_f)) + \int_{t_0}^{t_f} \mathcal{L}(t,\boldsymbol{x},\boldsymbol{u}) \tag{3.2}$$

subject to

$$\begin{aligned} &\dot{\boldsymbol{x}} = \boldsymbol{f}(t,\boldsymbol{x},\boldsymbol{u}) \\ &\boldsymbol{x}(t_0) = \boldsymbol{x_0} \\ &\boldsymbol{x}(t_f) \in \mathcal{C} \\ &t \in [t_0, t_f] \end{aligned} \tag{3.3}$$

where $\boldsymbol{x} \in \Omega \subseteq \mathbb{R}^n$, $\boldsymbol{u} \in \mathcal{U} \subseteq \mathbb{R}^m$, and $\mathcal{C} \subset \Omega$ are the states (that eventually can be subject also to path constraints), control (that eventually can be subject also to inequality constraints), and terminal conditions, respectively. In general, the final time t_f may be fixed, or it might be a function of the initial states $\boldsymbol{x}_0$ (e.g., for the time-free problems).
As previously explained, according to the indirect methods, the OC is retrieved by direct application of the PMP or by the calculus of variations, which provide the first-order necessary optimality conditions. The necessary conditions result in a TPBVP in the state-costate pair. Another approach to tackling OCPs comes from the application of the BPO. The latter provides the necessary and sufficient conditions for optimality, which yields a nonlinear PDE known as the HJB [85] equation. Within both approaches, the control, to be optimal, must satisfy these optimality principles, which lead to physical constraints represented by the TPVBP and the HJB equation, respectively.
Here, the question that automatically arises is the following "*which approach would be better to be used and why?*". One, in principle, would prefer to use the approach that relays on the application of the BPO, as it provides necessary and sufficient conditions for optimality. That is, if one can solve the HJB equation will have a closed-loop form of the optimal control within the time-state space where the HJB is solved. Here, the bottleneck is the solution of the HJB equation, which can be infeasible for many practical applications due to the curse of dimensionality. For instance, for a spacecraft's optimal 3D landing, the resulting HJB has at least seven independent variables: time, position (three components), and velocity (three components). Conversely, the TPBVP in the state-costate pair arising from the application of the PMP, being a system of ODEs, is easier to tackle, but the optimality conditions are only necessary. The resulting optimal control is open-loop. Thus, each time initial and/or final conditions change, the optimal control actions must be retrieved again. Moreover, the solution of the TPBVP is very cumbersome because of the strong sensitivity of the solution to the initial guesses. In particular, it is difficult to provide good initial guesses for the costates since they do not represent any physical quantity. This represents the bottleneck on the indirect methods and why they are unreliable for real-time applications. Hence, although

the direct method provides control actions that are not necessarily optimal, the solution is more straightforward and stable. For this reason, direct methods are more spread within the scientific community. However, one in principle could combine the BPO and the PMP using coarse HJB solutions to initialize the TPVBP solutions [88]. Doing so could guarantee that the optimal control actions would be retrieved from the TPVBP. Then, for real-time applications, one would need to compute open-loop solutions fast enough to generate a closed-loop solution with a series of open-loop ones. Conversely, open-loop optimal control actions can be generated within a specific time-state space and then used to learn the closed-loop solution in a data-physics-driven fashion within the same time-state space. In this case, the challenge will then become to guarantee that the BPO, transformed into physics constraints in the form of the HJB equation, is not violated.
X-TFC represents a unique PINN framework, which enables to learn the control actions directly from the TPVBP and the HJB equation in a (data-) physics-driven fashion. Hence, this guarantees that the learned control is optimal while respecting the physics constraints arising from the optimality principles. More precisely, if the PMP approach is used, the goal is to model a PINN approximation of the state-costate pair for which the residuals of the TPVBP are as close to zero as possible. We will refer to this PINN model as *Portraying Neural Networks* (PoNN) [127]. Likewise, if the BPO approach is used, the goal is to model a PINN representation of the value function for which the residual of the HJB equation is close to zero. We will refer to this PINN model as *Bellman Neural Networks* (BeNN).

The dissertation author's contributions to the peer-reviewed article [63] attached in Appendix C are listed below,

- Methodology, investigation
- Writing original manuscript and revised manuscript
- Manuscript structure organization
- Problems selection
- Software

The dissertation author's contributions to the peer-reviewed article [64] attached in Appendix D are listed below,

- Methodology, investigation
- Conceptualization of the term *Bellman Neural Networks*
- Generalization error upper bounds derivation and corresponding theoretical proofs
- Writing the original manuscript and revised manuscript
- Manuscript structure organization

- Problems selection
- Software

Chapter 4

X-TFC for DE Parameters Discovery

Being a PINN framework, X-TFC can also be used for DE parameter discovery (e.g., inverse problems). When solving inverse problems, the PINN training is data-physics-driven.
In this chapter, we will see, with an example, how X-TFC is employed for inverse problems and the X-TFC application for Epidemiological Compartmental Models (ECMs) parameter estimation (Appendix G [128]).
For the sake of consistency, consider again the problem (2.1)-(2.2), where now we add three unknown parameters, α, γ, and ω. That is,

$$-\alpha\, u_t - \gamma\, u_x x + \frac{\omega}{4} u_x^2 = 0, \quad \forall t \in [0,1],\, \forall x \in [-1,1] \tag{4.1}$$

subject to the following constraints (e.g., final time condition),

$$u(1,x) = x^2, \quad \forall x \in [-1,1] \tag{4.2}$$

To perform the data-physics-driven training, we need to collect some data on either u or its partial derivatives. For this example, consider that the data are collected on u,

$$\{t_i, x_i, \hat{u}_i\}_{i=1}^{N} \tag{4.3}$$

where N is the number of labeled training points. Following X-TFC, u is approximated with the CE (as for the forward problems),

$$u(t,x) \simeq u(t,x;\boldsymbol{\beta}) = u^{\boldsymbol{\beta}}(t,x) = g^{\boldsymbol{\beta}}(t,x) + \left(x^2 - g^{\boldsymbol{\beta}}(1,x)\right) \tag{4.4}$$

where $\boldsymbol{\beta} \in \mathbb{R}^L$ are the ELM output weights. $u^{\boldsymbol{\beta}}$ is then plugged into (4.1) and (4.2) to form the interior residual $\mathcal{R}_{\text{int},\boldsymbol{\Xi}}$ and temporal boundary residual $\mathcal{R}_{tb,\boldsymbol{\Xi}}$, The interior residual is,

$$\mathcal{R}_{\text{int},\boldsymbol{\Xi}}(t,x) = -\alpha\, u_t^{\boldsymbol{\beta}} - \gamma\, u_x^{\boldsymbol{\beta}}\, x + \frac{\omega}{4}(u_x^{\boldsymbol{\beta}})^2, \quad \forall t \in [0,1],\, \forall x \in [-1,1] \tag{4.5}$$

where $\boldsymbol{\Xi} = \{\boldsymbol{\beta}, \alpha, \gamma, \omega\}^T$. The temporal boundary residual is,

$$\mathcal{R}_{tb,\boldsymbol{\Xi}}(x) = u^{\boldsymbol{\beta}}(1, x) - x^2 = 0, \quad \forall x \in [-1, 1] \tag{4.6}$$

Here, we also need the residual on the data-set, which we will call data residual,

$$\mathcal{R}_{\text{data},\boldsymbol{\Xi}}(t, x) = \hat{u} - u^{\boldsymbol{\beta}} \tag{4.7}$$

The following vectorial data-physics-driven loss function, $\boldsymbol{\mathcal{L}}(\boldsymbol{\Xi})$, is formed by collocating the residuals on the corresponding training points,

$$\boldsymbol{\mathcal{L}}(\boldsymbol{\Xi}) = \begin{Bmatrix} \mathcal{R}_{\text{data},\boldsymbol{\Xi}}(t_1, x_1) \\ \vdots \\ \mathcal{R}_{\text{data},\boldsymbol{\Xi}}(t_N, x_N) \\ \mathcal{R}_{\text{int},\boldsymbol{\Xi}}(t_1, x_1) \\ \vdots \\ \mathcal{R}_{\text{int},\boldsymbol{\Xi}}(t_{N_{\text{int}}}, x_{N_{\text{int}}}) \end{Bmatrix} \tag{4.8}$$

Now, to find PINN, $u^* = u^{\boldsymbol{\beta}^*}$, to approximate the true solution u of the problem (4.1)-(4.2) as best as possible and the unknown parameters α, γ, and ω the following optimization problem is solved,

$$\boldsymbol{\Xi}^* = \min_{\boldsymbol{\Xi} \in \mathbb{R}^{L+3}} \boldsymbol{\mathcal{L}}(\boldsymbol{\Xi}) \tag{4.9}$$

The presence of the unknown parameters makes (4.9) always nonlinear. Thus, the optimization problem (4.9) is solved via the same iterative least-squares procedure used for PIELM.

4.1 X-TFC for Epidemiological Compartmental Models Discovery

In Appendix E, X-TFC has been employed to learn the unknown parameters for ECMs that regulate virus spread. This real-world application has been selected because COVID-19 pandemic.

The concern for viruses spread has been in the researchers' interest for years [129, 130, 131, 132]. Particularly, in the last two years and a half, due to the COVID-19 pandemic, this concern has become a hot topic for many research groups, in many fields [133, 134, 135, 136, 137, 138, 139, 140, 141]. There exist, several models, to study the virus spread. A first categorization of these models can be made for deterministic and stochastic models [142, 143, 144, 145].

Deterministic models have fixed input variables. They are compartmental models because the individuals in the population are assigned to different subgroups or compartments, each representing a specific individual condition in the epidemic situation [146]. Derivatives in

time model the transition rates of individuals from one compartment to another. Thus, these models are built as ODE systems.
Stochastic models take into account variations in input variables and provide results in terms of probability. Unlike deterministic models, stochastic ones consider random variations in one or more inputs over time. Thus, an estimation of the probability distributions of the outcomes can be evaluated. The variables that change in time can be the exposure risk, recovery rate, and other disease dynamics. As it is possible to insert the variability of the input data, the stochastic models are more complex than the deterministic ones. However, they manage to be more adherent to reality [147].
A second categorization between the models can be made by taking or not into account the vital dynamics. The vital dynamics represent the demography dynamics, in which the naturally occurring births and deaths are included [148]. In work in Appendix G [128], deterministic models with vital dynamics are studied. Precisely, the Susceptible-Infectious-Recovered (SIR), Susceptible-Exposed-Infectious-Recovered (SEIR), and Susceptible-Exposed-Infectious-Recovered-Susceptible (SEIR) models are considered, with the vaccination factor for one of those [149, 150]. The results show the high accuracy and effectiveness of X-TFC in performing data-physics-driven parameters' estimation of ECMs with unperturbed and perturbed data.

The dissertation author's contributions to the peer-reviewed article [128] attached in Appendix E are listed below,

- Conceptualization, methodology, investigation
- Writing original manuscript and revised manuscript
- Manuscript structure organization
- Problems selection
- Software

Chapter 5

Conclusions and Outlooks

This dissertation introduces a novel and improved Physics-Informed Neural Network (PINN) framework called the Extreme Theory of Functional Connections (X-TFC) for solving problems involving DEs of interest in engineering and science.
X-TFC, for the first time, merges PINNs with a functional interpolation technique called Theory of Functional Connections (TFC) to overcome the main drawbacks of the classic PINN framework and the classic TFC.
This work aims to show the X-TFC advantage (in terms of accuracy, solution generalization, and computational efforts) over the existing PINN frameworks and the state-of-the-art methods to solve DEs, such as the Finite Difference Method (FDM) or Finite Element Method (FEM). Moreover, it shows several X-TFC applications in solving problems of interest for different types of real-world applications involving DEs, proving the flexibility of the proposed method. This work shows how, X-TFC has been applied to physics-driven solutions of nuclear reactor dynamics (e.g., point kinetic equations with temperature feedback). Another application is for physics-driven solutions of OCPs, both via the application of the indirect method (e.g., the Pontryagin Minimum Principle) and the Bellman Principle of Optimality (either for general or aerospace optimal control problems). X-TFC has also been applied for data-physics-driven parameter discovery of epidemiological models that regulate virus spread.
As the author of this dissertation devised and developed the X-TFC framework during his Ph.D. journey, he has been involved in many projects where X-TFC has been employed. This has allowed him to produce many journals and conference articles, some as the leading author and some as a co-author [25, 62, 63, 128, 151, 127, 152, 153, 154, 155, 156, 157, 158, 64]. The dissertation expected outcomes, in terms of journal publications with the author as the leading author, are listed below.

- Ref. [25] (Appendix A) is a published article where X-TFC has been introduced and tested against other state-of-the-art methods in solving several benchmark problems of interest in physics and engineering. As expected, the results of this work show that X-TFC's performances (in terms of accuracy and computational time) are comparable and, in many cases, better than the state-of-the-art classical methods and, especially,

the existing PINN frameworks.

- Ref. [62] (Appendix B) is a published article where X-TFC is applied to learn the solutions of the Point Kinetic Equations (PKEs) for nuclear reactor dynamics, which is of extreme importance for the simulation of reactor transient in safety analyses. Here, X-TFC can match all the state-of-the-art methods in terms of accuracy and overcome them in terms of computational time and generalization of the solutions.

- Ref. [63] (Appendix C) is a published article where PoNNs are employed to learn optimal control actions for a series of optimal planar orbit transfer problems, which are essential to design space exploration missions accurately. In this work, X-TFC's performances are compared to a commercial optimization software called GPOPS-II [159]. In all the problems, the accuracy of the two methods is comparable. At the same time, in many of them, X-TFC outperforms GPOPS-II in terms of computational time, making the X-TFC-based algorithm potentially suitable for real-time applications.

- Ref. [64] (Appendix D) is a published article which introduces the *Bellman Neural Networks (BeNNs)*. BeNNs represent a particular family of PINNs specifically designed to tackle optimal control problems via the Bellman Principle of Optimality application. The authors coined the term "Bellman Neural Networks" to refer in a more compact form to "PINNs designed to learn optimal control via the BPO." The characteristic feature of BeNNs is to use X-TFC as the PINN framework. In particular, this paper aims to provide upper bounds on the generalization error of X-TFC in learning optimal control actions for the class of optimal control with integral quadratic cost tackled via the BPO. The findings are supported and validated by numerical experiments on a couple of benchmark problems with linear and nonlinear dynamics.

- Ref. [128] (Appendix E) is a published article where X-TFC is applied to perform data-physics-driven parameters discovery of epidemiological compartmental models, which are typical dynamical systems to model the spread of viruses. In this work, X-TFC is applied to estimate, from the data, the unknown parameters governing the epidemiological compartmental models, such as mortality rate, infectious rate, etc. Furthermore, it is shown how X-TFC can estimate these parameters with good accuracy even when the data are noisy.

All these works proved the X-TFC efficiency, robustness, reliability, and flexibility. So far, in all the applications where it has been tested, its performances have been competitive with state-of-the-art methods. In many applications, it outperformed them in accuracy and computational efforts.
Of course, these works and this dissertation are only the beginning. Being a newly developed method, X-TFC opens the doors for many future works. Moreover, as it is a method for solving forward and inverse problems governed by DEs, it can be applied to any real-world problem governed by DEs. For instance, this dissertation's author is currently working on applying X-TFC to OCPs tackled via BPO for space applications such as optimal landing or

orbit transfer. As shown in [62] and [156] when the problems present stiffness in the dynamics and/or solutions with sharp gradients, X-TFC may need to be adapted to learn and generalize the solutions accurately. In [62] and [156] a domain decomposition technique was devised and employed. However, this strategy was not optimized as the domain decomposition was fixed and done in a "trail and fail" fashion. Thus, although great accuracy was achieved, the computational effort, although comparable with optimized methods, was not optimized. To this end, works are in progress to devise adaptive domain decomposition techniques to optimize the computational effort. Moreover, the domain decomposition technique used in [62] and [156] was devised solely for IVP ODE. Efforts are being made to make a domain decomposition also for TPBVP ODEs and PDEs.

The ultimate goal would be to optimize and improve X-TFC to the point where it could be systematically and widely used (either replacing or in synergy with the existing methods) for any large-scale real-world problems to help researchers worldwide, both in industry and academia.

Finally, theoretical works are in progress to derive the estimates on the generalization error of X-TFC in learning the solutions for several classes of DEs.

Bibliography

[1] D. G. Zill, *A first course in differential equations with modeling applications*. Cengage Learning, 2012.

[2] J. C. Butcher, "On the implementation of implicit runge-kutta methods," *BIT Numerical Mathematics*, vol. 16, no. 3, pp. 237–240, 1976.

[3] J. C. Butcher, "A history of runge-kutta methods," *Applied numerical mathematics*, vol. 20, no. 3, pp. 247–260, 1996.

[4] R. W. Hamming, "Stable predictor-corrector methods for ordinary differential equations," *Journal of the ACM (JACM)*, vol. 6, no. 1, pp. 37–47, 1959.

[5] N. Perrone and R. Kao, "A general finite difference method for arbitrary meshes," *Computers & Structures*, vol. 5, no. 1, pp. 45–57, 1975.

[6] G. Dhatt, E. Lefrançois, and G. Touzot, *Finite element method*. John Wiley & Sons, 2012.

[7] T. J. Hughes, *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2012.

[8] S. S. Rao, *The finite element method in engineering*. Butterworth-heinemann, 2017.

[9] J. N. Reddy, *Introduction to the finite element method*. McGraw-Hill Education, 2019.

[10] T. G. Zieli, "Introduction to the finite element method," 1992.

[11] O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu, *The finite element method: its basis and fundamentals*. Elsevier, 2005.

[12] R. Eymard, T. Gallouët, and R. Herbin, "Finite volume methods," *Handbook of numerical analysis*, vol. 7, pp. 713–1018, 2000.

[13] H. Jasak, "Error analysis and estimation for the finite volume method with applications to fluid flows.," 1996.

[14] F. Moukalled, L. Mangani, and M. Darwish, "The finite volume method," in *The finite volume method in computational fluid dynamics*, pp. 103–135, Springer, 2016.

[15] H. K. Versteeg and W. Malalasekera, *An introduction to computational fluid dynamics: the finite volume method*. Pearson education, 2007.

[16] R. Bellman, "Dynamic programming princeton university press princeton," *New Jersey Google Scholar*, 1957.

[17] D. E. Kirk, *Optimal control theory: an introduction*. Courier Corporation, 2004.

[18] M. Bardi, I. C. Dolcetta, *et al.*, *Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations*, vol. 12. Springer, 1997.

[19] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[20] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[21] K. Hornik, M. Stinchcombe, and H. White, "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks," *Neural networks*, vol. 3, no. 5, pp. 551–560, 1990.

[22] M. Dissanayake and N. Phan-Thien, "Neural-network-based approximations for solving partial differential equations," *communications in Numerical Methods in Engineering*, vol. 10, no. 3, pp. 195–201, 1994.

[23] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE transactions on neural networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[24] 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*, vol. 378, pp. 686–707, 2019.

[25] E. Schiassi, R. Furfaro, C. Leake, M. De Florio, H. Johnston, and D. Mortari, "Extreme theory of functional connections: A fast physics-informed neural network method for solving ordinary and partial differential equations," *Neurocomputing*, vol. 457, pp. 334–356, 2021.

[26] D. Mortari, "The theory of connections: Connecting points," *Mathematics*, vol. 5, no. 4, p. 57, 2017.

[27] C. Leake, H. Johnston, and D. Mortari, *The Theory of Functional Connections: A Functional Interpolation. Framework with Applications*. Morrisville NC: Lulu, 2022. This if the first (and only) book on functional interpolation. The book is available from: http://www.lulu.com.

[28] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1-3, pp. 489–501, 2006.

[29] J. Fish and T. Belytschko, *A first course in finite elements*, vol. 1. John Wiley & Sons New York, 2007.

[30] S. Mishra and R. Molinaro, "Estimates on the generalization error of physics-informed neural networks for approximating pdes," *IMA Journal of Numerical Analysis*, 2022.

[31] S. Wang, Y. Teng, and P. Perdikaris, "Understanding and mitigating gradient flow pathologies in physics-informed neural networks," *SIAM Journal on Scientific Computing*, vol. 43, no. 5, pp. A3055–A3081, 2021.

[32] P. Mertikopoulos, C. Papadimitriou, and G. Piliouras, "Cycles in adversarial regularized learning," in *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 2703–2717, SIAM, 2018.

[33] D. Balduzzi, S. Racaniere, J. Martens, J. Foerster, K. Tuyls, and T. Graepel, "The mechanics of n-player differentiable games," in *International Conference on Machine Learning*, pp. 354–363, PMLR, 2018.

[34] V. Dwivedi and B. Srinivasan, "Physics informed extreme learning machine (pielm)–a rapid method for the numerical solution of partial differential equations," *Neurocomputing*, vol. 391, pp. 96–118, 2020.

[35] C. Leake and D. Mortari, "Deep theory of functional connections: A new method for estimating the solutions of partial differential equations," *Machine learning and knowledge extraction*, vol. 2, no. 1, pp. 37–55, 2020.

[36] Y. Yang, M. Hou, and J. Luo, "A novel improved extreme learning machine algorithm in solving ordinary differential equations by legendre neural network methods," *Advances in Difference Equations*, vol. 2018, no. 1, pp. 1–24, 2018.

[37] H. Johnston, "The theory of functional connections: A journey from theory to application," *arXiv preprint arXiv:2105.08034*, 2021.

[38] C. Leake, "The multivariate theory of functional connections: An n-dimensional constraint embedding technique applied to partial differential equations," *arXiv preprint arXiv:2105.07070*, 2021.

[39] T. Mai and D. Mortari, "Theory of functional connections applied to quadratic and nonlinear programming under equality constraints," *Journal of Computational and Applied Mathematics*, vol. 406, p. 113912, 2022.

[40] Y. Wang and F. Topputo, "A tfc-based homotopy continuation algorithm with application to dynamics and control problems," *Journal of Computational and Applied Mathematics*, vol. 401, p. 113777, 2022.

[41] H. Johnston, E. Schiassi, R. Furfaro, and D. Mortari, "Fuel-efficient powered descent guidance on large planetary bodies via theory of functional connections," *The journal of the astronautical sciences*, vol. 67, no. 4, pp. 1521–1552, 2020.

[42] K. Drozd, R. Furfaro, E. Schiassi, H. Johnston, and D. Mortari, "Energy-optimal trajectory problems in relative motion solved via theory of functional connections," *Acta Astronautica*, vol. 182, pp. 361–382, 2021.

[43] M. De Florio, E. Schiassi, R. Furfaro, B. D. Ganapol, and D. Mostacci, "Solutions of chandrasekhar's basic problem in radiative transfer via theory of functional connections," *Journal of Quantitative Spectroscopy and Radiative Transfer*, vol. 259, p. 107384, 2021.

[44] C. Leake, H. Johnston, L. Smith, and D. Mortari, "Analytically embedding differential equation constraints into least squares support vector machines using the theory of functional connections," *Machine learning and knowledge extraction*, vol. 1, no. 4, p. 60, 2019.

[45] M. De Florio, E. Schiassi, A. D'Ambrosio, D. Mortari, and R. Furfaro, "Theory of functional connections applied to linear odes subject to integral constraints and linear ordinary integro-differential equations," *Mathematical and Computational Applications*, vol. 26, no. 3, p. 65, 2021.

[46] C. Yassopoulos, C. Leake, J. Reddy, and D. Mortari, "Analysis of timoshenko–ehrenfest beam problems using the theory of functional connections," *Engineering Analysis with Boundary Elements*, vol. 132, pp. 271–280, 2021.

[47] D. Mortari and R. Furfaro, "Univariate theory of functional connections applied to component constraints," *Mathematical and Computational Applications*, vol. 26, no. 1, p. 9, 2021.

[48] A. K. de Almeida Junior, H. Johnston, C. Leake, and D. Mortari, "Fast 2-impulse non-keplerian orbit transfer using the theory of functional connections," *The European Physical Journal Plus*, vol. 136, no. 2, pp. 1–21, 2021.

[49] H. Johnston, C. Leake, and D. Mortari, "Least-squares solutions of eighth-order boundary value problems using the theory of functional connections," *Mathematics*, vol. 8, no. 3, p. 397, 2020.

[50] A. D'Ambrosio, E. Schiassi, H. Johnston, F. Curti, D. Mortari, and R. Furfaro, "Time-energy optimal landing on planetary bodies via theory of functional connections," *Advances in Space Research*, vol. 69, no. 12, pp. 4198–4220, 2022.

[51] D. Mortari, "Using the theory of functional connections to solve boundary value geodesic problems," *Mathematical and Computational Applications*, vol. 27, no. 4, p. 64, 2022.

[52] S. Li, Y. Yan, K. Zhang, and X. Li, "Fuel-optimal ascent trajectory problem for launch vehicle via theory of functional connections," *International Journal of Aerospace Engineering*, vol. 2021, 2021.

[53] A. De Almeida Junior, H. Johnston, C. Leake, and D. Mortari, "Evaluation of transfer costs in the earth-moon system using the theory of functional connections," in *Proceedings of the AAS/AIAA Astrodynamics Specialist Conference, Lake Tahoe, CA, USA*, pp. 9–13, 2020.

[54] H. Johnston and D. Mortari, "Orbit propagation via the theory of functional connections," in *Proceedings of the 2019 AAS/AIAA Astrodynamics Specialist Conference, Portland, ME, USA*, pp. 11–15, 2019.

[55] R. Furfaro and D. Mortari, "Least-squares solution of a class of optimal space guidance problems via theory of connections," *Acta Astronautica*, vol. 168, pp. 92–103, 2020.

[56] D. Mortari, "Least-squares Solution of Linear Differential Equations," *MDPI Mathematics*, vol. 5, no. 48, pp. 1–18, 2017.

[57] D. Mortari, H. Johnston, and L. Smith, "High accuracy least-squares solutions of nonlinear differential equations," *Journal of Computational and Applied Mathematics*, vol. 352, pp. 293 – 307, 2019.

[58] C. Leake, H. Johnston, and D. Mortari, "The multivariate theory of functional connections: Theory, proofs, and application in partial differential equations," *Mathematics*, vol. 8, no. 8, p. 1303, 2020.

[59] A. F. Henry, "The application of reactor kinetics to the analysis of experiments," *Nuclear Science and Engineering*, vol. 3, no. 1, pp. 52–70, 1958.

[60] K. Ott and D. Meneley, "Accuracy of the quasistatic treatment of spatial reactor kinetics," *Nuclear Science and Engineering*, vol. 36, no. 3, pp. 402–411, 1969.

[61] P. Picca, R. Furfaro, and B. D. Ganapol, "A highly accurate technique for the solution of the non-linear point kinetics equations," *Annals of Nuclear Energy*, vol. 58, pp. 43–53, 2013.

[62] E. Schiassi, M. De Florio, B. D. Ganapol, P. Picca, and R. Furfaro, "Physics-informed neural networks for the point kinetics equations for nuclear reactor dynamics," *Annals of Nuclear Energy*, vol. 167, p. 108833, 2022.

[63] E. Schiassi, A. D'Ambrosio, K. Drozd, F. Curti, and R. Furfaro, "Physics-informed neural networks for optimal planar orbit transfers," *Journal of Spacecraft and Rockets*, vol. 59, no. 3, pp. 834–849, 2022.

[64] E. Schiassi, A. D'Ambrosio, and R. Furfaro, "Bellman neural networks for the class of optimal control problems with integral quadratic cost," *IEEE Transactions on Artificial Intelligence*, 2022.

[65] C. L. Darby, W. W. Hager, and A. V. Rao, "An hp-adaptive pseudospectral method for solving optimal control problems," *Optimal Control Applications and Methods*, vol. 32, no. 4, pp. 476–502, 2011.

[66] F. Fahroo and I. M. Ross, "Direct trajectory optimization by a chebyshev pseudospectral method," *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 1, pp. 160–166, 2002.

[67] I. M. Ross and F. Fahroo, "Pseudospectral knotting methods for solving nonsmooth optimal control problems," *Journal of Guidance, Control, and Dynamics*, vol. 27, no. 3, pp. 397–405, 2004.

[68] R. H. Byrd, J. C. Gilbert, and J. Nocedal, "A Trust Region Method Based on Interior Point Techniques for Nonlinear Programming," *Mathematical programming*, vol. 89, no. 1, pp. 149–185, 2000.

[69] S. Josselyn and I. M. Ross, "Rapid verification method for the trajectory optimization of reentry vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 26, no. 3, pp. 505–508, 2003.

[70] K. F. Graham and A. V. Rao, "Minimum-time trajectory optimization of multiple revolution low-thrust earth-orbit transfers," *Journal of Spacecraft and Rockets*, vol. 52, no. 3, pp. 711–727, 2015.

[71] A. T. Miller and A. V. Rao, "Rapid ascent-entry vehicle mission optimization using hp-adaptive gaussian quadrature collocation," in *AIAA Atmospheric Flight Mechanics Conference*, p. 0249, 2017.

[72] X. Jiang, S. Li, and R. Furfaro, "Integrated guidance for mars entry and powered descent using reinforcement learning and pseudospectral method," *Acta Astronautica*, vol. 163, pp. 114–129, 2019.

[73] B. Acikmese and S. R. Ploen, "Convex Programming Approach to Powered Descent Guidance for Mars Landing," *Journal of Guidance, Control, and Dynamics*, vol. 30, no. 5, pp. 1353–1366, 2007.

[74] L. Blackmore, B. Acikmese, and D. P. Scharf, "Minimum-Landing-Error Powered-Descent Guidance for Mars Landing using Convex Optimization," *Journal of guidance, control, and dynamics*, vol. 33, no. 4, pp. 1161–1171, 2010.

[75] Z. Wang and M. J. Grant, "Constrained trajectory optimization for planetary entry via sequential convex programming," in *AIAA Atmospheric Flight Mechanics Conference*, p. 3241, 2016.

[76] Z. Wang and M. J. Grant, "Autonomous entry guidance for hypersonic vehicles by convex optimization," *Journal of Spacecraft and Rockets*, vol. 55, no. 4, pp. 993–1006, 2018.

[77] K. Zhang, S. Yang, and F. Xiong, "Rapid ascent trajectory optimization for guided rockets via sequential convex programming," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, p. 0954410019830268, 2019.

[78] Z. Wang and M. J. Grant, "Minimum-fuel Low-thrust Transfers for Spacecraft: A Convex Approach," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 54, no. 5, pp. 2274–2290, 2018.

[79] H. B. Keller, *Numerical solution of two point boundary value problems*, vol. 24. SIaM, 1976.

[80] J. Stoer and R. Bulirsch, *Introduction to numerical analysis*, vol. 12. Springer Science & Business Media, 2013.

[81] S. Oh and R. Luus, "Use of orthogonal collocation method in optimal control problems," *International Journal of Control*, vol. 26, no. 5, pp. 657–673, 1977.

[82] F. Fahroo and I. Ross, "Trajectory optimization by indirect spectral collocation methods," in *Astrodynamics specialist conference*, p. 4028, 2000.

[83] A. Melikyan, *Generalized characteristics of first order PDEs: applications in optimal control and differential games*. Springer Science & Business Media, 2012.

[84] M. G. Crandall, H. Ishii, and P.-L. Lions, "User's guide to viscosity solutions of second order partial differential equations," *Bulletin of the American mathematical society*, vol. 27, no. 1, pp. 1–67, 1992.

[85] M. Bardi and I. Capuzzo-Dolcetta, *Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations*. Springer Science & Business Media, 2008.

[86] Z. Chen and P. A. Forsyth, "A semi-lagrangian approach for natural gas storage valuation and optimal operation," *SIAM Journal on Scientific Computing*, vol. 30, no. 1, pp. 339–368, 2008.

[87] M. Falcone and R. Ferretti, "Semi-lagrangian schemes for hamilton–jacobi equations, discrete representation formulae and godunov methods," *Journal of computational physics*, vol. 175, no. 2, pp. 559–575, 2002.

[88] E. Cristiani and P. Martinon, "Initialization of the shooting method via the hamilton-jacobi-bellman approach," *Journal of Optimization Theory and Applications*, vol. 146, no. 2, pp. 321–346, 2010.

[89] I. C. Dolcetta, "On a discrete approximation of the hamilton-jacobi equation of dynamic programming," *Applied Mathematics and Optimization*, vol. 10, no. 1, pp. 367–377, 1983.

[90] M. Falcone and R. Ferretti, "Discrete time high-order schemes for viscosity solutions of hamilton-jacobi-bellman equations," *Numerische Mathematik*, vol. 67, no. 3, pp. 315–344, 1994.

[91] J. N. Reddy, "An introduction to the finite element method," *Journal of Pressure Vessel Technology*, vol. 111, pp. 348–349, 08 1989.

[92] R. Gonzalez and E. Rofman, "On deterministic control problems: An approximation procedure for the optimal cost i. the stationary problem," *SIAM Journal on Control and Optimization*, vol. 23, no. 2, pp. 242–266, 1985.

[93] R. Gonzalez and E. Rofman, "On deterministic control problems: An approximation procedure for the optimal cost ii. the nonstationary case," *SIAM journal on control and optimization*, vol. 23, no. 2, pp. 267–285, 1985.

[94] R. W. Beard, G. N. Saridis, and J. T. Wen, "Galerkin approximations of the generalized hamilton-jacobi-bellman equation," *Automatica*, vol. 33, no. 12, pp. 2159–2177, 1997.

[95] R. W. Beard, G. N. Saridis, and J. T. Wen, "Approximate solutions to the time-invariant hamilton–jacobi–bellman equation," *Journal of Optimization theory and Applications*, vol. 96, no. 3, pp. 589–626, 1998.

[96] W. H. Fleming and W. M. McEneaney, "A max-plus-based algorithm for a hamilton–jacobi–bellman equation of nonlinear filtering," *SIAM Journal on Control and Optimization*, vol. 38, no. 3, pp. 683–710, 2000.

[97] W. M. McEneaney, "A curse-of-dimensionality-free numerical method for solution of certain hjb pdes," *SIAM journal on Control and Optimization*, vol. 46, no. 4, pp. 1239–1276, 2007.

[98] W. M. McEneaney, A. Deshpande, and S. Gaubert, "Curse-of-complexity attenuation in the curse-of-dimensionality-free method for hjb pdes," in *2008 American Control Conference*, pp. 4684–4690, IEEE, 2008.

[99] S. Osher and J. A. Sethian, "Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations," *Journal of computational physics*, vol. 79, no. 1, pp. 12–49, 1988.

[100] C. O. Aguilar and A. J. Krener, "Numerical solutions to the bellman equation of optimal control," *Journal of optimization theory and applications*, vol. 160, no. 2, pp. 527–552, 2014.

[101] Y. Tassa and T. Erez, "Least squares solutions of the hjb equation with neural network value-function approximators," *IEEE transactions on neural networks*, vol. 18, no. 4, pp. 1031–1041, 2007.

[102] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[103] P. Werbos, "Beyond regression: new tools for prediction and analysis in the behavioral sciences," *Ph. D. dissertation, Harvard University*, 1974.

[104] B. Kiumarsi, K. G. Vamvoudakis, H. Modares, and F. L. Lewis, "Optimal and autonomous control using reinforcement learning: A survey," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 6, pp. 2042–2062, 2017.

[105] D. V. Prokhorov and D. C. Wunsch, "Adaptive critic designs," *IEEE transactions on Neural Networks*, vol. 8, no. 5, pp. 997–1007, 1997.

[106] L. C. Baird, "Reinforcement learning in continuous time: Advantage updating," in *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, vol. 4, pp. 2448–2453, IEEE, 1994.

[107] T. Hanselmann, L. Noakes, and A. Zaknich, "Continuous-time adaptive critics," *IEEE Transactions on Neural Networks*, vol. 18, no. 3, pp. 631–647, 2007.

[108] D. Vrabie and F. L. Lewis, "Adaptive optimal control algorithm for continuous-time nonlinear systems based on policy iteration," in *2008 47th IEEE Conference on Decision and Control*, pp. 73–79, IEEE, 2008.

[109] D. Vrabie, K. Vamvoudakis, and F. Lewis, "Adaptive optimal controllers based on generalized policy iteration in a continuous-time framework," in *2009 17th Mediterranean Conference on Control and Automation*, pp. 1402–1409, IEEE, 2009.

[110] S. Bhasin, R. Kamalapurkar, M. Johnson, K. G. Vamvoudakis, F. L. Lewis, and W. E. Dixon, "A novel actor–critic–identifier architecture for approximate optimal control of uncertain nonlinear systems," *Automatica*, vol. 49, no. 1, pp. 82–92, 2013.

[111] H. Holt, R. Armellin, A. Scorsoglio, and R. Furfaro, "Low-thrust trajectory design using closed-loop feedback-driven control laws and state-dependent parameters," in *AIAA Scitech 2020 Forum*, p. 1694, 2020.

[112] A. Zavoli and L. Federici, "Reinforcement learning for low-thrust trajectory design of interplanetary missions," *arXiv preprint arXiv:2008.08501*, 2020.

[113] L. Arora and A. Dutta, "Reinforcement learning for sequential low-thrust orbit raising problem," in *AIAA Scitech 2020 Forum*, p. 2186, 2020.

[114] N. B. LaFarge, D. Miller, K. C. Howell, and R. Linares, "Guidance for closed-loop transfers using reinforcement learning with application to libration point orbits," in *AIAA Scitech 2020 Forum*, p. 0458, 2020.

[115] S. Silvestrini and M. R. Lavagna, "Spacecraft formation relative trajectories identification for collision-free maneuvers using neural-reconstructed dynamics," in *AIAA Scitech 2020 Forum*, p. 1918, 2020.

[116] K. Hovell and S. Ulrich, "On deep reinforcement learning for spacecraft guidance," in *AIAA Scitech 2020 Forum*, p. 1600, 2020.

[117] C. E. Oestreich, R. Linares, and R. Gondhalekar, "Autonomous six-degree-of-freedom spacecraft docking maneuvers via reinforcement learning," *arXiv preprint arXiv:2008.03215*, 2020.

[118] X. Wang, G. Wang, Y. Chen, and Y. Xie, "Autonomous rendezvous guidance via deep reinforcement learning," in *2020 Chinese Control And Decision Conference (CCDC)*, pp. 1848–1853, IEEE, 2020.

[119] R. Furfaro and R. Linares, "Waypoint-based generalized zem/zev feedback guidance for planetary landing via a reinforcement learning approach," *3rd IAA Conference on Dynamics and Control of Space Systems, Moscow, Russia*, 2017.

[120] R. Furfaro, A. Scorsoglio, R. Linares, and M. Massari, "Adaptive generalized zem-zev feedback guidance for planetary landing via a deep reinforcement learning approach," *Acta Astronautica*, 2020.

[121] B. Gaudet, R. Linares, and R. Furfaro, "Deep reinforcement learning for six degree-of-freedom planetary landing," *Advances in Space Research*, 2020.

[122] B. Gaudet, R. Linares, and R. Furfaro, "Terminal adaptive guidance via reinforcement meta-learning: Applications to autonomous asteroid close-proximity operations," *Acta Astronautica*, vol. 171, pp. 1–13, 2020.

[123] A. Scorsoglio, R. Furfaro, R. Linares, M. Massari, *et al.*, "Actor-critic reinforcement learning approach to relative motion guidance in near-rectilinear orbit," *Advances in the Astronautical Sciences*, pp. 1–20, 2019.

[124] A. Scorsoglio and R. Furfaro, "Elm-based actor-critic approach to lyapunov vector fields relative motion guidance in near-rectilinear orbit," in *2019 AAS/AIAA Astrodynamics Specialists Conference*, pp. 1–20, 2019.

[125] I. M. Ross, Q. Gong, and P. Sekhavat, "Low-thrust, high-accuracy trajectory optimization," *Journal of Guidance, Control, and Dynamics*, vol. 30, no. 4, pp. 921–933, 2007.

[126] I. M. Ross, Q. Gong, and P. Sekhavat, "Bellman pseudospectral method," in *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, p. 6448, 2008.

[127] A. D'ambrosio, E. Schiassi, F. Curti, and R. Furfaro, "Pontryagin neural networks with functional interpolation for optimal intercept problems," *Mathematics*, vol. 9, no. 9, p. 996, 2021.

[128] E. Schiassi, M. De Florio, A. D'ambrosio, D. Mortari, and R. Furfaro, "Physics-informed neural networks and functional interpolation for data-driven parameters discovery of epidemiological compartmental models," *Mathematics*, vol. 9, no. 17, p. 2069, 2021.

[129] P. M. Sharp, E. Bailes, D. L. Robertson, F. Gao, and B. H. Hahn, "Origins and evolution of aids viruses," *The Biological Bulletin*, vol. 196, no. 3, pp. 338–342, 1999.

[130] A. C. Lowen, S. Mubareka, T. M. Tumpey, A. García-Sastre, and P. Palese, "The guinea pig as a transmission model for human influenza viruses," *Proceedings of the National Academy of Sciences*, vol. 103, no. 26, pp. 9988–9992, 2006.

[131] J. L. Geoghegan, A. M. Senior, F. Di Giallonardo, and E. C. Holmes, "Virological factors that increase the transmissibility of emerging human viruses," *Proceedings of the National Academy of Sciences*, vol. 113, no. 15, pp. 4170–4175, 2016.

[132] M. I. Nelson, M. R. Gramer, A. L. Vincent, and E. C. Holmes, "Global transmission of influenza viruses from humans to swine," *The Journal of general virology*, vol. 93, no. Pt 10, p. 2195, 2012.

[133] E. Kıymet, E. Böncüoğlu, Ş. Şahinkaya, E. Cem, M. Y. Çelebi, M. Düzgöl, A. A. Kara, K. Ö. Arıkan, T. Aydın, R. İşgüder, *et al.*, "Distribution of spreading viruses during covid-19 pandemic: Effect of mitigation strategies," *American Journal of Infection Control*, 2021.

[134] T. Galbadage, B. M. Peterson, and R. S. Gunasekera, "Does covid-19 spread through droplets alone?," *Frontiers in public health*, vol. 8, p. 163, 2020.

[135] M. Arti and K. Bhatnagar, "Modeling and predictions for covid 19 spread in india," *ResearchGate, DOI: DOI*, vol. 10, 2020.

[136] M. C. Castro, S. Kim, L. Barberia, A. F. Ribeiro, S. Gurzenda, K. B. Ribeiro, E. Abbott, J. Blossom, B. Rache, and B. H. Singer, "Spatiotemporal pattern of covid-19 spread in brazil," *Science*, vol. 372, no. 6544, pp. 821–826, 2021.

[137] C. A. Varotsos and V. F. Krapivin, "A new model for the spread of covid-19 and the improvement of safety," *Safety science*, vol. 132, p. 104962, 2020.

[138] G. Caspi, U. Shalit, S. L. Kristensen, D. Aronson, L. Caspi, O. Rossenberg, A. Shina, and O. Caspi, "Climate effect on covid-19 spread rate: an online surveillance tool," *MedRxiv*, 2020.

[139] K. Aabed and M. M. Lashin, "An analytical study of the factors that influence covid-19 spread," *Saudi Journal of Biological Sciences*, vol. 28, no. 2, pp. 1177–1195, 2021.

[140] E. L. Piccolomiini and F. Zama, "Monitoring italian covid-19 spread by an adaptive seird model," *MedRxiv*, 2020.

[141] K. M. Al-Kindi, A. Alkharusi, D. Alshukaili, N. Al Nasiri, T. Al-Awadhi, Y. Charabi, and A. M. El Kenawy, "Spatiotemporal assessment of covid-19 spread over oman using gis techniques," *Earth Systems and Environment*, vol. 4, no. 4, pp. 797–811, 2020.

[142] D. Adak, A. Majumder, and N. Bairagi, "Mathematical perspective of covid-19 pandemic: disease extinction criteria in deterministic and stochastic models," *Chaos, Solitons & Fractals*, vol. 142, p. 110381, 2021.

[143] S. V. Petrovskii, H. Malchow, F. M. Hilker, and E. Venturino, "Patterns of patchy spread in deterministic and stochastic models of biological invasion and biological control," *Biological Invasions*, vol. 7, no. 5, pp. 771–793, 2005.

[144] N. C. Perera, *Deterministic and stochastic models of virus dynamics*. PhD thesis, Texas Tech University, 2003.

[145] I. Sazonov, D. Grebennikov, M. Kelbert, and G. Bocharov, "Modelling stochastic and deterministic behaviours in virus infection dynamics," *Mathematical Modelling of Natural Phenomena*, vol. 12, no. 5, pp. 63–77, 2017.

[146] D. Breda, O. Diekmann, W. De Graaf, A. Pugliese, and R. Vermiglio, "On the formulation of epidemic models (an appraisal of kermack and mckendrick)," *Journal of biological dynamics*, vol. 6, no. sup2, pp. 103–117, 2012.

[147] T. Britton, "Stochastic epidemic models: a survey," *Mathematical biosciences*, vol. 225, no. 1, pp. 24–35, 2010.

[148] H. W. Hethcote, "Three basic epidemiological models," in *Applied mathematical ecology*, pp. 119–144, Springer, 1989.

[149] G. Huang, Y. Takeuchi, W. Ma, and D. Wei, "Global stability for delay sir and seir epidemic models with nonlinear incidence rate," *Bulletin of mathematical biology*, vol. 72, no. 5, pp. 1192–1207, 2010.

[150] M. B. Trawicki, "Deterministic seirs epidemic model for modeling vital dynamics, vaccinations, and temporary immunity," *Mathematics*, vol. 5, no. 1, p. 7, 2017.

[151] M. De Florio, E. Schiassi, B. D. Ganapol, and R. Furfaro, "Physics-informed neural networks for rarefied-gas dynamics: Thermal creep flow in the bhatnagar–gross–krook approximation," *Physics of Fluids*, vol. 33, no. 4, p. 047110, 2021.

[152] A. D'Ambrosio, E. Schiassi, F. Curti, and R. Furfaro, "Physics-informed neural networks applied to a series of constrained space guidance problems," in *31st AAS/AIAA Space Flight Mechanics Meeting*, 2021.

[153] E. Schiassi, A. D'Ambrosio, A. Scorsoglio, R. Furfaro, and F. Curti, "Class of optimal space guidance problems solved via indirect methods and physics-informed neural networks," 2021.

[154] A. D'Ambrosio, E. Schiassi, F. Curti, and R. Furfaro, "Physics-informed neural networks for optimal proximity maneuvers with collision avoidance around asteroids,"

[155] E. Schiassi, A. D'Ambrosio, R. Furfaro, and F. Curti, "Physics-informed neural networks for optimal intercept problem,"

[156] M. De Florio, E. Schiassi, and R. Furfaro, "Physics-informed neural networks and functional interpolation for stiff chemical kinetics," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 32, no. 6, p. 063107, 2022.

[157] R. Furfaro, A. D'Ambrosio, E. Schiassi, and A. Scorsoglio, "Physics-informed neural networks for closed-loop guidance and control in aerospace systems," in *AIAA SCITECH 2022 Forum*, p. 0361, 2022.

[158] M. De Florio, E. Schiassi, B. D. Ganapol, and R. Furfaro, "Physics-informed neural networks for rarefied-gas dynamics: Poiseuille flow in the bgk approximation," *Zeitschrift für angewandte Mathematik und Physik*, vol. 73, no. 3, pp. 1–18, 2022.

[159] M. A. Patterson and A. V. Rao, "Gpops-ii: A matlab software for solving multiple-phase optimal control problems using hp-adaptive gaussian quadrature collocation methods and sparse nonlinear programming," *ACM Transactions on Mathematical Software (TOMS)*, vol. 41, no. 1, pp. 1–37, 2014.

Appendix A

Schiassi et al. (2021): Extreme Theory of Functional Connections

Reproduced from Schiassi, E., Furfaro, R., Leake, C., De Florio, M., Johnston, H. and Mortari, D., 2021. Extreme theory of functional connections: A fast physics-informed neural network method for solving ordinary and partial differential equations. *Neurocomputing*, 457, pp.334-356.https://doi.org/10.1016/j.neucom.2021.06.015 [25], with the permission of Elsevier.

Neurocomputing 457 (2021) 334–356

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Extreme theory of functional connections: A fast physics-informed neural network method for solving ordinary and partial differential equations

Enrico Schiassi [a], Roberto Furfaro [a,b,*], Carl Leake [c], Mario De Florio [a], Hunter Johnston [c], Daniele Mortari [c]

[a] Department of Systems and Industrial Engineering, The University of Arizona, AZ, Tucson, USA
[b] Department of Aerospace and Mechanical Engineering, The University of Arizona, AZ, Tucson, USA
[c] Department of Aerospace Engineering, Texas A and M University, TX, USA

ARTICLE INFO

Article history:
Received 12 July 2020
Revised 9 March 2021
Accepted 6 June 2021
Available online 8 June 2021

Keywords:
Physics-informed neural networks
Extreme learning machine
Functional interpolation
Numerical methods
Universal approximator
Least-squares

ABSTRACT

We present a novel, accurate, fast, and robust physics-informed neural network method for solving problems involving differential equations (DEs), called *Extreme Theory of Functional Connections*, or X-TFC. The proposed method is a synergy of two recently developed frameworks for solving problems involving DEs: the *Theory of Functional Connections* TFC, and the *Physics-Informed Neural Networks* PINN. Here, the latent solution of the DEs is approximated by a TFC constrained expression that employs a Neural Network (NN) as the free-function. The TFC approximated solution form always analytically satisfies the constraints of the DE, while maintaining a NN with unconstrained parameters. X-TFC uses a single-layer NN trained via the *Extreme Learning Machine* (ELM) algorithm. This choice is based on the approximating properties of the ELM algorithm that reduces the training of the network to a simple least-squares, because the only trainable parameters are the output weights. The proposed methodology was tested over a wide range of problems including the approximation of solutions to linear and nonlinear ordinary DEs (ODEs), systems of ODEs, and partial DEs (PDEs). The results show that, for most of the problems considered, X-TFC achieves high accuracy with low computational time, even for large scale PDEs, without suffering the curse of dimensionality.

1. Introduction

Differential Equations (DEs) are a powerful tool used for the mathematical modelling of various problems arising in scientific fields such as physics, engineering, finance, biology, chemistry, and oceanography. There exist two types of DEs: (1) ordinary DEs (ODEs) which are univariate independent variable equations, and (2) partial DEs (PDEs) which are multivariate independent variable equations. Such equations can be employed to simulate, identify, characterize, design, and verify the design of a variety of systems. In many practical problems, it is not trivial and most likely impossible to find an analytical solution to these DEs. Thus, one has to resort to numerical methods.

To numerically solve ODEs, a variety of methods exist, the most popular belonging to the Runge–Kutta family [1]. Other methods include finite difference, Chebyshev-Picard iteration [2], and pseudo-spectral methods [3]. However, a recently developed method called the *Theory of Functional Connections* (TFC) [4–6], has significantly improved the state-of-the-art of computing numerical solutions of ODEs. According to the TFC methodology, the unknown (or latent [7]) solution of the equation is approximated via a *constrained expression*. The constrained expression comprises (1) a function that analytically satisfies the constraints, and (2) a functional comprising a freely-chosen function that projects it onto the space of functions that vanish at the constraints. In the classic TFC method, the free-function is approximated by a linear combination of orthogonal polynomials, such as Chebyshev polynomials [5,6]. Whereas the free-function could easily be defined by an explicit n^{th} degree polynomial. Orthogonal polynomials representations are generally preferred due to their approximating and convergence properties. For example, Chebyshev polynomials generate a function that minimizes the maximum error in its application, and therefore, they are well suited for approximating other functions [8,9]. Studies have shown that the TFC method can be used to numerically approximate the solution of linear and nonlinear ODEs with machine-level error in

* Corresponding author at: Department of Systems and Industrial Engineering, The University of Arizona, AZ, Tucson, USA.
E-mail address: robertof@email.arizona.edu (R. Furfaro).

https://doi.org/10.1016/j.neucom.2021.06.015
0925-2312/

milliseconds [5,6,10,11]. For this reason, TFC represents an appealing choice for many different applications. For example, TFC has already been used to solve ODEs with initial value constraints, boundary value constraints [5,6], relative constraints, integral constraints [12], and linear combinations of constraints. Additionally, TFC has been employed to solve Two-Point Boundary Value Problems (TPBVP) arising in aerospace optimal control problems such as computation of energy optimal [10] and fuel optimal landing trajectories on large planetary bodies [11]. Additionally, in De Florio et. al. [13], the TFC was leveraged to solve a class of Radiative Transfer Problems, matching the benchmarks published by Garcia and Siewert [14] and Ganapol [15]. Although at first glance TFC might seem similar to the spectral method, their difference lies in how the differential constraints are enforced. While TFC analytically enforces these constraints, spectral method requires the constraints to be numerically imposed by adding them to the minimization process to compute the solution [3]. Therefore, the minimization process involves a higher number of equations, and the overall accuracy of the solution is generally worse than the accuracy achieved with TFC [11].

For PDEs, a multitude of numerical methods exist as well, the most widely used being the Finite Element Method (FEM) [16–19]. In fact, FEM has been successfully applied to solve PDEs in a variety of fields such as structures, fluids, and acoustics. In this method, the domain is discretized into smaller parts called finite elements and simple polynomials are used to interpolate the solution within these elements. Ultimately, these elements are then assembled into a larger system of equations that model the entire problem. However, the major drawback to FEM is the number of subdivisions needed to capture large variations in the solution. For example, this discretization works well for low-dimensional PDEs, but suffers in higher dimensions; the number of elements grows exponentially with the number of independent variables. Thus, the discretization becomes prohibitive as the number of variables increases. Moreover, in the FEM framework, the PDE is solved at discrete nodes and if the solution is needed at different points, an interpolation scheme is required. As mentioned in Ref. [20], this affects the accuracy of the solution at the interpolated points. Furthermore, extra numerical techniques are needed to perform further manipulation of the FEM solution such as taking the gradients, or computing the integrals.

The solution of ODEs and PDEs has also become an area of interest in the machine learning community. Indeed, various authors have explored the feasibility of using Neural Networks (NNs) to solve ODEs and PDEs. There are two main motivations behind this choice, (1) according to the *Universal Approximation Theorem* [21,22], NNs are universal approximators, and thus they can potentially be used to approximate the solution of any DE [21,23], and (2) using NNs to approximate DE solutions avoids the discretization step, as well as discretization based physics errors [24]. Chen and Chen [25] have shown that NNs can approximate nonlinear operators, virtually opening the way to learn operators from data. Furthermore, Pinkus [26] and Lu et al. [27] have proved that a single layer NN with a sufficiently large number of hidden neurons can simultaneously and uniformly approximate a function and its partial derivatives. Importantly, it has been demonstrated that NNs are able to approximate the solutions for some classes of PDEs such as quasilinear parabolic PDEs [28], the Black–Scholes PDEs [29], and the Hamilton–Jacobi PDEs [30].

For ODEs, NN-based solutions have been recently proposed. Yang et al. [31] proposed a method based on NNs, in particular, Legendre Neural Networks (LeNNs). LeNNs are single layer NNs where the activation functions are Legendre polynomials and the network is trained via the Extreme Learning Machine (ELM) algorithm, proposed by Huang et al. [32]. This algorithm is used for Single-hidden Layer Feed-forward Networks (SLFNs). It randomly selects hidden input weights and biases, and computes the output weights via least-squares [32]. Although the results obtained from the LeNN method are fast and accurate, the accuracy is affected by the fact that the equation constraints (e.g. initial and/or boundary conditions) are not analytically satisfied. Indeed, the method adds the constraints as a penalty to the loss function which is minimized when training the neural networks.

Recently, Raissi et al. [7] defined a framework that use NNs and Deep-NNs (DNNs) to solve PDEs as *Physics-Informed Neural Networks* (PINNs), wherein the latent solution of a PDE is approximated via a NN or DNN. The PINN methods are a novel approach for the direct solution of DEs, data-driven solution of DEs, and data-driven parameter discovery of DEs [7]. The term PINN, as described by Raissi et al. in [7], defines NNs that use the physics as regulators during the training. More precisely, assume that one wants to perform a regression of a set of data using a NN, and that these data represent a physics phenomena modeled via DEs. In a normal regression, one would approximate the data using a NN, and train the NN to minimize the Mean Squared Error (MSE) between the NN approximation and the data set. However, there is no guarantee that the physics phenomena governing the data would not be violated. Within a PINN framework, the DEs that model the physics of the data set are added in their implicit form along with the initial and/or boundary conditions (ICs and/or BCs) to the MSE. Theses additional terms act as a regulator that penalizes the training when the physics and the ICs and/or BCs are violated. A regression that involves data and the physics of the problem modeled via DEs is called a data-driven solution of a DE. If instead the goal is to estimate parameters governing some physical phenomena modeled through a DE, e.g., the thermal conductivity in the heat equation, then the problem is called data-driven parameter discovery of DEs. When data is not available, and therefore the MSE contains only the DEs in their implicit form and the ICs and/or BCs, PINNs are used for approximating the solutions of ODEs and PDEs. To this end, PINNs are frameworks that approximate solutions of DEs using NNs.

Physics-informed NN algorithms have already been applied to a variety of different problems involving PDEs [33], such as fractional PDEs [34,35], stochastic differential equations [28,36], biomedical problems [37], and fluid mechanics [38], to name a few.

The physics-informed NN methods explored in Refs. [7,39] show that the use of NNs overcomes some of FEM's limitations. One major benefit of physics-informed methods is that the points can be randomly sampled from the domain. Therefore, the discretization of the entire domain into a number of elements that grows exponentially with the scale of the problems is avoided. Thus, PINN frameworks are not affected by the curse of dimensionality, and thus are suitable to solve large-scale PDEs. Moreover, the NN is an analytical approximation of the latent solution. This leads to two major advantages: (1) once the network is trained, no interpolation schemes are needed when estimating the solution at points that did not appear during training, and (2) further solution manipulation, such as computing gradients or integrals, can be done analytically. Although this method was created to numerically estimate the solutions of PDEs, it can also be used to approximate the solutions of ODEs. According to this, both the TFC and the LeNN methods can be seen as physics-informed methods. Demonstrated in Raissi et al.[7], such method can also be used to solve inverse problems for parameter estimation, where the physics phenomena are modeled via DEs. However, what was presented in Refs. [7,39] can still be improved both in terms of solution accuracy and computational speed.

The above mentioned technique relies on DNNs to approximate the latent solution. The latter affects the computational cost because gradient descent-based methods, used to train the networks, are generally computationally expensive. Such limitations

may be in principle overcome by approximating the latent solution with a SLFN trained via the ELM algorithm [24] [1]. Another potential drawback of the DNN-based PINN approach is that the equation constraints, i.e. initial conditions (IC), and boundary conditions (BC), are managed by adding extra terms to the loss function which may negatively affect the solution accuracy and increase the computational cost.

One way to overcome such limitation is to approximate the solution in such a way that the constraints are automatically satisfied, independently of the values of the training parameters in the network. The latter implies some decoupling of initial and boundary conditions from the domain solution. Lagaris et al. [20] handled such constraints by imposing *ad hoc* functions that satisfy ICs/BCs when solving both ODEs and PDEs via a physics-informed NN method. The approach leveraged a technique similar to the Coons' patch [40] to analytically satisfy the constraints. Generally, analytical satisfaction of the constraints is of significant interest for a variety of problems. This is particularly true when the confidence in the constraint information is high. Moreover, embedding the constraints in such a way allows the NN to sample points only from interior of the domain [41]. The latter reduces the number of training points needed, and thereby decreases the computational cost of training. Whereas the method proposed by Lagaris et al. [20] works well for ODEs and low-dimensional PDEs with simple boundary constraints, its application is relatively limited because it does not provide a generalized framework to derive higher-dimensional or more complex constraints. Fortunately, the TFC framework may overcome such limitations.

In fact, an extension of the univariate TFC to n-dimensions was recently formalized [42,43]. This extension highlighted a succinct method to derive constrained expressions for value constraints and arbitrary order derivative constraints of $(n-1)$-dimensional manifolds in n-dimensional space. Consequently, the TFC framework can generate constrained expressions that satisfy the boundary constraints of multidimensional, large-scale, PDEs [44,42]. In fact, this framework has already been utilized to solve PDEs in combination with machine learning algorithms, such as support vector machines [45] and DNNs [41].

As previously mentioned, numerical techniques based on the TFC framework to solve ODEs have typically used a linear combination of orthogonal polynomials with unknown coefficients as the free-function. This approach leads to a solution via least-squares for linear problems [5], and iterative least-squares for nonlinear problems [6]. In later studies, this method was extended to the solution of bivariate PDEs [44], where the free function was expressed as the product of two linear combinations of orthogonal polynomials with unknown coefficients. Since the free-function remained linear in these coefficients, a linear least-squares or iterative least-squares method could still be used to estimate the PDEs solution. However, as the dimension of the problem increases or the scale of the problem becomes larger, this approach becomes computationally prohibitive, because it suffers the curse of dimensionality. One way to overcome this limitation is to model the free-function using NNs[41]. Here, the authors developed a framework called *Deep-TFC* where DNNs where employed to approximate the free-function, and TFC is employed to analytically satisfy the ICs/BCs via the appropriate constrained expressions. The results showed that the method was an improvement of the NN approach proposed by Lagaris et al. in terms of both accuracy and computational cost. However, both methods rely on gradient based optimization to train their networks which are known to be computationally expensive.

[1] In this manuscript SLFNs trained via the ELM algorithm will be refereed to simply as ELMs

This paper introduces the *Extreme Theory of Functional Connections* (X-TFC), a method that for the first time brings together TFC and single layer NNs trained via ELM algorithm as a PINN framework, which can be used to solve problems involving linear and nonlinear ODEs and PDEs.

We aim at defining a framework that is simultaneously accurate, robust and, computationally efficient by leveraging (1) the ability of TFC to analytically satisfy ICs/BCs via constrained expressions, and (2) the ability of ELM algorithms to learn the boundary-free function without iterative tuning for fast convergence. As such, the solution of any DE is modeled as a combination of two components. The first component defines the constrained expressions necessary to analytically satisfy the space–time boundary constraints exhibited by the problem. The second component is modeled as a SLFN that learns to approximate the problem solution on the domain according to the ELM theories, i.e. in a least-square or iterative least-square fashion.

X-TFC is intended to be a novel physics-informed NN method devised to solve forward and inverse problems involving DEs. The proposed framework is able to overcome some of the limitations of the TFC approaches and the other state-of-the-art physics-informed NN methods. Indeed, X-TFC can handle large-scale problems (e.g. PDEs with several independent variables), while preserving high accuracy and low computational time compared to competing methods, without being affected by the curse of dimensionality. Importantly, this approach can be legitimately classified as a physics-informed NN method for two reasons. First, the X-TFC framework uses a NN to both directly solve DEs with known parameters, and to provide data-driven solutions of DEs with known parameters, where the approximated solution is posed in such a way that the physics of the problems are not violated. Second, although in this paper we focus only on the solution of DEs with known parameters (i.e. forward modeling), X-TFC can also be used for data-driven parameters discovery of DEs (i.e. solutions of inverse problems for parameter estimation) both in deterministic and probabilistic (e.g. Bayesian Inversion) fashions.

The remainder of this article is organized as follows. First, the theory for the X-TFC framework is explained in detail (Section 2). In this section we also provide some general theoretical discussion on X-TFC and the relationship with the theory of PINN frameworks for solving DEs. Next, the results are presented and discussed (Section 3). Results are compared with the other state-of-the-art methods when possible. Particular attention is put on the comparison of X-TFC with the standard TFC, for ODEs, and with Deep-TFC, for PDEs. The results in the ODEs section show that X-TFC and the standard TFC are comparable both in terms of accuracy and computational efficiency for the solution of linear and nonlinear ODEs, and systems of ODEs (SODEs). As for the PDEs, the results show that X-TFC outperforms Deep-TFC (both in terms of accuracy and computational speed) for smooth PDEs such as the heat equation, but Deep-TFC preforms better on sufficiently complex problems such as the Navier–Stokes equations. To this end, we show how X-TFC and Deep-TFC complement each other. That is, X-TFC and Deep-TFC form a synergy that defines a PINN-TFC based method for the solution of linear and nonlinear ODEs and PDEs. Conclusions are reported in Section 4.

2. Extreme theory of functional connections

As previously mentioned, the X-TFC is a novel PINN framework developed to solve direct and inverse problems involving DEs with high accuracy and low computational time.

From the computer/data science prospective solving direct problems means performing a regression (or a classification) over a given data set. When NNs are used to perform a regression, the

goal is to train the network to learn the functional relation among the data. Most of the times the data are affected by noise and this can lead to the issue of the over-fitting. To avoid this problem, regularization techniques are needed [46,47]. Usually the regulators are mathematical artifacts that regulate the training of the network by penalizing it when over-fitting happens. With PINN, the training is regularized by the physics governing the data that is modeled via DEs. To accomplish this, the PINN frameworks add the DE in its implicit form and constraint terms to the MSE to help keep the NN from violating the physics. This particular regression is also called *data-driven solution* of DEs [7]. PINN can also be used to approximate the solution of DEs, where the MSE accounts only contains the residual of the DEs [7].

From the physics prospective, when solving direct problems involving DEs governed by certain parameters (e.g., thermal conductivity in the heat equation), the parameters governing the equations are assumed to be known within a certain accuracy. Therefore, two possible scenarios exist: (1) the DE exactly approximates the physical phenomena that it describes, i.e. there are no modeling errors; hence, these problems are called *exact problems*, and (2) the DE approximates the physical phenomena that it describes with non-negligible modeling errors. In the first scenario no data is needed to solve the equation with good accuracy. Thus, when applying the X-TFC method, the solution of the DE reduces to an unconstrained optimization problem. In the second scenario, data is needed to solve the equation with good accuracy. In this case, the solution of the DE can also be called a *data-driven solution* [7].

When solving inverse problems involving DEs, the parameters governing the DE are unknown and need to be estimated. These kind of problems are called *data-driven parameters discovery* of DEs [7], as the goal is to discover (i.e. estimate) the parameters that govern the equation by comparing the equation solution with data. For example, a typical field where solving inverse problems is of extreme interest is remote sensing [48–51]. For instance, in Ref. [52], the authors couple radiative and heat transfer equations to form a set of DEs. The solutions of this set of DEs is compared with real data to discover the thermal inertia and the grain size of planetary regoliths, which are the parameters governing the DEs.

Although this paper focuses on the solution of exact problems, in this section we present the general X-TFC method that can be also used for data-driven solutions of DEs, and for data-driven parameter discovery of DEs. Additionally, for the convenience of the reader, two simple examples are provided where the X-TFC method is applied to the solution of a linear and a nonlinear bivariate PDE. Finally, preliminary theoretical evaluations regarding the proposed physics-informed X-TFC framework are provided.

For the interested reader, we also present in detail how to build the constrained expressions [41], and give a brief description of the ELM algorithm and its properties [32] in the Appendix.

2.1. Method

As previously stated, differential Equations (DEs) are a powerful tool used for the mathematical modelling of various problems arising in scientific fields such as physics, engineering, finance, biology, chemistry, and oceanography, to name few. We can express DEs, in their most general implicit form, as,

$$\gamma f_t + \mathscr{N}[f;\boldsymbol{\lambda}] + \varepsilon - \mathscr{U} = 0 \tag{1}$$

subject to constraints given by the initial conditions (IC) and boundary conditions (BC). In Eq. (1), t is a non-negative real number, $\boldsymbol{x} \in \mathbb{D} \subseteq \mathbb{R}^n$, $f = f(t,\boldsymbol{x};\gamma(t,\boldsymbol{x}),\boldsymbol{\lambda}(t,\boldsymbol{x}))$ is the unknown solution of Eq. (1) (in strong form [53]) and is a real valued function, $\gamma = \gamma(t,\boldsymbol{x}) \in \mathbb{G} \subseteq \mathbb{R}$ and $\boldsymbol{\lambda} = \boldsymbol{\lambda}(t,\boldsymbol{x}) \in \mathbb{L} \subseteq \mathbb{R}^m$are the parameters governing the DE (which are known when dealing with direct problems, and latent when dealing with inverse problems) [2], $\mathscr{N}[\cdot;\boldsymbol{\lambda}]$ is a linear or nonlinear operator acting on f and parameterized by $\boldsymbol{\lambda}$, the subscript t refers to the partial derivative of f with respect to t, i.e. $f_t = \frac{\partial f}{\partial t}$ [3] [4], ε is the modeling error that is negligible when solving *exact problems*, and $\mathscr{U}$ is a known function that in general can depend on t and $\boldsymbol{x}$, and can be parametrized by γ and $\boldsymbol{\lambda}$. Note that in accordance with the notation, the operator $\mathscr{N}[\cdot;\boldsymbol{\lambda}]$ contains derivatives with respect to $\boldsymbol{x}$ only. Higher order derivatives with respect to t, which can be eventually considered, are not part of the $\mathscr{N}[\cdot;\boldsymbol{\lambda}]$ operator.

The first step in our general physics-informed framework is to approximate the latent solution f with a constrained expression that analytically satisfies the constraints as follows,

$$f(\mathbf{x};\boldsymbol{\Theta}) = f_{CE}(\mathbf{x},g(\mathbf{x});\boldsymbol{\Theta}) = A(\mathbf{x};\boldsymbol{\Theta}) + B(\mathbf{x},g(\mathbf{x});\boldsymbol{\Theta}),$$

where $\mathbf{x} = [t,\boldsymbol{x}]^{\mathrm{T}} \in \boldsymbol{\Omega} \subseteq \mathbb{R}^{n+1}$ with $t \geqslant 0$, $\boldsymbol{\Theta} = [\gamma,\boldsymbol{\lambda}]^{\mathrm{T}} \in \mathbb{P} \subseteq \mathbb{R}^{m+1}$, $A(\mathbf{x};\boldsymbol{\Theta})$ analytically satisfies the constraints, and $B(\mathbf{x},g(\mathbf{x});\boldsymbol{\Theta})$ projects the free-function $g(\mathbf{x})$, which is a real valued function, onto the space of functions that vanish at the constraints [41]. According to the X-TFC method, the free-function, $g(\mathbf{x})$, is chosen to be a single layer feed forward NN, in particular, an ELM [32]. That is,

$$g(\mathbf{x}) = \sum_{j=1}^{L} \beta_j \sigma\left(\mathbf{w}_j^{\mathrm{T}}\mathbf{x} + b_j\right) = [\sigma_1,\ldots,\sigma_L]\,\boldsymbol{\beta} = \boldsymbol{\sigma}^{\mathrm{T}}\boldsymbol{\beta}, \tag{2}$$

where L is the number of hidden neurons, $\mathbf{w}_j = \left[w_{j,1},\ldots,w_{j,n+1}\right]^{\mathrm{T}} \in \mathbb{R}^{n+1}$ is the input weights vector connecting the j^{th} hidden neuron and the input nodes, $\beta_j \in \mathbb{R}$ with $j = 1,\ldots,L$ is the j^{th} output weight connecting the j^{th} hidden neuron and the output node, and b_j is the threshold (aka bias) of the j^{th} hidden neuron, and $\sigma(\cdot)$ are activation functions. According to the ELM algorithm [32], input weights and biases are randomly selected and not tuned during the training, thus they are known parameters. The activation functions, $\sigma(\cdot)$, are chosen by the user, so they are also known. Therefore, the only unknowns to compute are the output weights $\boldsymbol{\beta} = [\beta_1,\ldots,\beta_L]^{\mathrm{T}}$. Fig. 1 shows an example of a general Multiple Input Single Output (MISO) single layer NN trained via the ELM method that is used within the physics-informed X-TFC framework [5]. Moreover, the partial derivatives of Eq. (2), with respect to the independent variables, are,

$$\frac{\partial^k g(\mathbf{x})}{\partial t^k} = \sum_{j=1}^{L} \beta_j w_{j,1}^k \frac{\partial^k \sigma\left(\mathbf{w}_j^{\mathrm{T}}\mathbf{x} + b_j\right)}{\partial t^k}$$

$$\frac{\partial^k g(\mathbf{x})}{\partial x_i^k} = \sum_{j=1}^{L} \beta_j w_{j,i}^k \frac{\partial^k \sigma\left(\mathbf{w}_j^{\mathrm{T}}\mathbf{x} + b_j\right)}{\partial x_i^k}$$

where k refers to the k^{th} order derivative, the index $i = 2,\ldots,(n+1)$ refers to the x_i^{th} independent variable, $w_{j,1}$ where $j = 1,\ldots,L$ are the input weights associated with the independent variable t, and $w_{j,i}$ where $j = 1,\ldots,L$ are the input weights associated with the x_i^{th} inde-

[2] In general, even if it is not explicitly reported in the notation, f is a function of t and $\boldsymbol{x}$, and it is parameterized by γ and $\boldsymbol{\lambda}$, that in general can be t and $\boldsymbol{x}$ dependent as well.

[3] When dealing with ODEs, the partials derivatives become ordinary derivatives, thus, $f_t = \frac{\partial f}{\partial t} = \frac{df}{dt}$.

[4] The same notation referring to the derivatives is used to express the partial derivatives with respect any independent variable, for example $f_{xx} = \frac{\partial^2 f}{\partial x^2}$, $f_{xy} = \frac{\partial^2 f}{\partial x \partial y}$, etc.

[5] To clarify, for the free-function, X-TFC uses a MISO single layer NN (Fig. 1) when solving a PDE, whereas it uses a Single Input Single Output (SISO) single layer NN when solving an ODE.

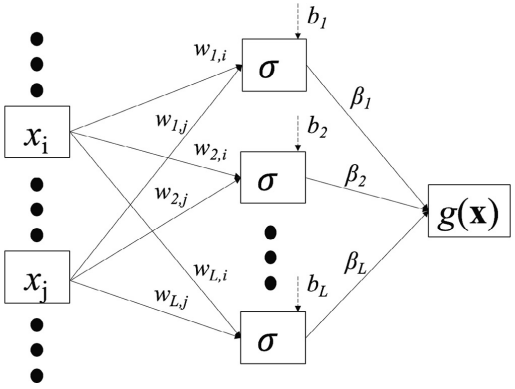


Fig. 1. Example of a MISO single layer NN, trained via ELM, used as the free-function within the X-TFC method to solve problems involving PDEs.

pendent variable.

Therefore,

$$f(\mathbf{x};\boldsymbol{\Theta}) = f_{CE}(\mathbf{x}, g(\mathbf{x});\boldsymbol{\Theta}) = \widehat{f}_{CE}(\mathbf{x},\boldsymbol{\beta};\boldsymbol{\Theta}).$$

The step-by-step process to derive the constrained expression is provided in A.

Since the constrained expressions are analytical, all the partial derivatives of $\widehat{f}_{CE}(\mathbf{x},\boldsymbol{\beta};\boldsymbol{\Theta})$ could be computed analytically. However, one can compute these partials by means of numerical tools such as automatic differentiation [54] at the cost of a higher computational time. These tools are particularly useful for the solutions of PDEs.

Now that f is approximated with a NN, the second step of the X-TFC physics-informed method is to define a loss function for the differential equation,

$$\mathscr{L}(\mathbf{x},\boldsymbol{\beta};\boldsymbol{\Theta}) = \gamma f_{CE,t} + \mathscr{N}[f_{CE};\boldsymbol{\lambda}] + \varepsilon - \mathscr{U} \tag{3}$$

In the X-TFC methodology, a constrained expression is used as the latent solution, where an ELM is used as the free function, and the parameters of the ELM are tuned to minimize $\mathscr{L}(\mathbf{x},\boldsymbol{\beta};\boldsymbol{\Theta})$, which is considered to be, according to Raissi et al. [7], a *physics-informed neural network*.

Now, the Mean Square Errors (*MSEs*) for the latent solution, MSE_f, and for the loss, $MSE_{\mathscr{L}}$, are defined,

$$MSE_f = \frac{1}{2N}\sum_{p=1}^{N}\left(\widetilde{f}_p - \widehat{f}_{p,CE}(\mathbf{x}_p,\boldsymbol{\beta},\boldsymbol{\Theta})\right)^2$$

$$MSE_{\mathscr{L}} = \frac{1}{2N}\sum_{p=1}^{N}\left(\mathscr{L}_p(\mathbf{x}_p,\boldsymbol{\beta};\boldsymbol{\Theta})\right)^2$$

Here, $\left[\mathbf{x}_p,\widetilde{f}_p\right]_{p=1}^{N}$ are N distinct sample data points with $\mathbf{x}_p = [t_p, x_{p,1},\ldots,x_{p,n}]^{\mathrm{T}} \in \mathbb{R}^{n+1}$, and $\widetilde{f}_p \in \mathbb{R}$ for each $p = 1,\ldots,N$. $\widehat{f}_{p,CE}$ and $\mathscr{L}_p$ are the CE and the loss function, respectively, evaluated at $\mathbf{x}_p$.

The final step is to compute the latent solution f, and in the case of the inverse problem the parameters $\boldsymbol{\Theta}$, that minimize the cost function $\mathscr{J}$,

$$\min_{\boldsymbol{\beta},\boldsymbol{\Theta}} \mathscr{J} = \Gamma_f MSE_f + \Gamma_{\mathscr{L}} MSE_{\mathscr{L}}, \tag{4}$$

where Γ_f and $\Gamma_{\mathscr{L}}$ are positive parameters whose values are chosen based on the relative importance of the MSEs in the computation of the unknowns. In other words, these values are problem-dependent and user specified. Using the X-TFC framework, the problem posed in Eq. (4) is an unconstrained regularized regression problem that can be solved via any optimization technique such as least-squares.

As mentioned previously, this article only focuses on the solutions of exact problems; problems in which the parameters $\boldsymbol{\Theta}$ are known within a certain accuracy, and the model error, ε, is negligible (i.e. $\varepsilon = 0$). Therefore the only quantity to compute is the latent solution, f, which is approximated via the constrained expression, f_{CE}, as explained above. For exact problems, the loss is the following,

$$\mathscr{L}(\mathbf{x},\boldsymbol{\beta};\boldsymbol{\Theta}) = \gamma f_{CE,t} + \mathscr{N}[f_{CE};\boldsymbol{\lambda}] - \mathscr{U}. \tag{5}$$

For linear DEs by imposing $\mathscr{L}(\mathbf{x},\boldsymbol{\beta};\boldsymbol{\Theta})$ to be equal to $\mathbf{0}$, the terms in Eq. (5) can be rearranged to obtain a system of linear algebraic equations whose solutions are the unknown output weights $\boldsymbol{\beta}$. This happens since the ELM algorithm does not train the input weights and biased of the NN, leaving the output weights $\boldsymbol{\beta}$ as only unknowns to be computed. That is,

$$A\boldsymbol{\beta} = \boldsymbol{b}, \tag{6}$$

which is solved via a least-squares technique [5]. For instance, using a classic least-squares, we have,

$$\boldsymbol{\beta} = \left(A^{\mathrm{T}}A\right)^{-1}A^{\mathrm{T}}\boldsymbol{b}$$

For nonlinear DEs, an iterative least-squares approach is required [6]. According to the iterative procedure, the estimation of the output weights are updated at each iteration as follows,

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k + \Delta\boldsymbol{\beta}_k$$

where the k subscript refers to the current iteration. In general, the $\Delta\boldsymbol{\beta}_k$ term can be defined by performing linear least-squares at each iteration of the iterative least-square procedure. For instance, if a classic linear least-square is useds, then,

$$\Delta\boldsymbol{\beta}_k = -\left(\mathbb{J}^{\mathrm{T}}(\boldsymbol{\beta}_k)\mathbb{J}(\boldsymbol{\beta}_k)\right)^{-1}\mathbb{J}^{\mathrm{T}}(\boldsymbol{\beta}_k)\mathscr{L}(\boldsymbol{\beta}_k)$$

where $\mathbb{J}$ is the Jacobian matrix containing the derivatives of the loss with respect to the output weights $\boldsymbol{\beta}$. Again, one may compute the Jacobian by hand or by means of automatic differentiation as mentioned previously. The iterative process is repeated until the following conditions is met,

$$L_2[\mathscr{L}(\boldsymbol{\beta}_k)] < \epsilon$$

where ϵ is some user prescribed tolerance [6].

Fig. 2 is a schematic summary that shows how the physic-informed X-TFC framework works for solving exact forward problems involving DEs. This schematic mirrors the schematic reported in [27] that summarizes the standard PINN framework as introduced by Raissi et al. [7]. This is done purposefully to highlight the major differences between the standard PINN and the physics-informed X-TFC: these differences stem mainly from how the constraints of the problem are managed.

2.2. Procedural examples

This subsection provides two simple examples that demon-

[6] Once we are sure that the converged is achieved by meeting the criteria $L_2[\mathscr{L}(\boldsymbol{\beta}_k)] < \epsilon$, the nonlinear DE can be solved using the following criteria $L_2[\mathscr{L}(\boldsymbol{\beta}_{k+1})] > L_2[\mathscr{L}(\boldsymbol{\beta}_k)]$. Doing so, the converged is caused by round-off error 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.)

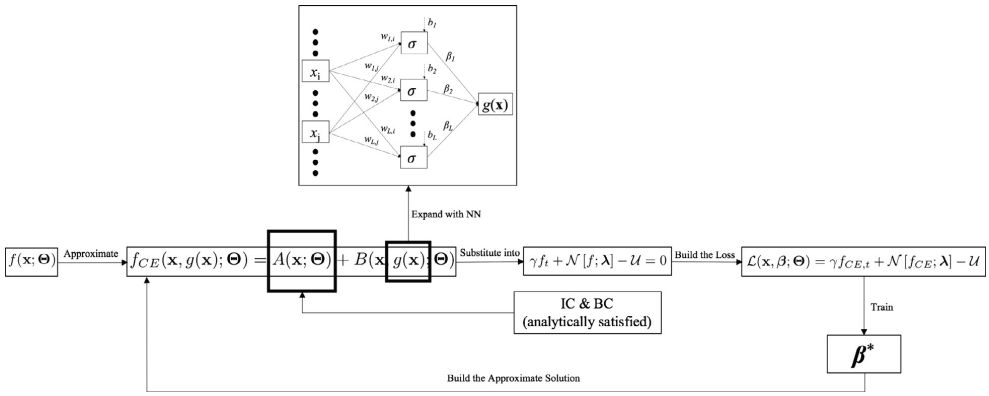


Fig. 2. Schematic of the physics-informed X-TFC framework to solve exact forward problems involving DEs.

strate how the X-TFC framework is applied to solve a linear and a nonlinear bivariate PDE.

2.2.1. Linear Bivariate PDE

The example problem proposed here is a mono-dimensional time-dependent heat diffusion equation.

$$f_{xx}(t,x) = \kappa f_t(t,x)$$

where $t, x \in [0,1] \times [0,1]$ and $\kappa = 1$, subject to

$$f(t,0) = 0$$
$$f(t,1) = 0$$
$$f(0,x) = \sin(\pi x).$$

The constrained expression in its compact form is (see A for the detailed derivation),

$$f(t,x) = g(t,x) + \mathbf{v}(x)^{\mathrm{T}} \mathscr{M}(c(t,x) - g(t,x))\mathbf{v}(t),$$

where

$$\mathscr{M}(c(t,x)) = \begin{bmatrix} 0 & c(0,x) \\ c(t,0) & -c(0,0) \\ c(t,L) & -c(0,L) \end{bmatrix},$$

and

$$v(x) = \{1, \quad 1-x, \quad x\}^{\mathrm{T}}, \quad v(t) = \{1, \quad 1\}^{\mathrm{T}}.$$

In its expanded form the constrained expression is,

$$\begin{aligned} f(t,x) = \; & g(t,x) + (x-1)g(t,0) - xg(t,1) - xg(0,0) + xg(0,1) \\ & - g(0,x) + g(0,0) + \sin(\pi x). \end{aligned}$$

By using the expression $g(\mathbf{x}) = g(t,x)$ given by Eq. (2), one can rewrite the constrained expression as,

$$\begin{aligned} f(t,x) = \; & \boldsymbol{\sigma}^{\mathrm{T}}\boldsymbol{\beta} + (x-1)\boldsymbol{\sigma}_{t,0}^{\mathrm{T}}\boldsymbol{\beta} - x\boldsymbol{\sigma}_{t,1}^{\mathrm{T}}\boldsymbol{\beta} - x\boldsymbol{\sigma}_{0,0}^{\mathrm{T}}\boldsymbol{\beta} + x\boldsymbol{\sigma}_{0,1}^{\mathrm{T}}\boldsymbol{\beta} - \boldsymbol{\sigma}_{0,x}^{\mathrm{T}}\boldsymbol{\beta} \\ & + \boldsymbol{\sigma}_{0,0}^{\mathrm{T}}\boldsymbol{\beta} + \sin(\pi x). \end{aligned}$$

The loss function is given by

$$\mathscr{L} = f_{xx} - f_t = \frac{\partial^2 f}{\partial x^2} - \frac{\partial f}{\partial t}. \tag{7}$$

For an exact forward problem, it is desired that Eq. (7) is equal to $\mathbf{0}$. The next step is to compute f_{xx} and f_t,

$$\frac{\partial^2 f}{\partial x^2} = \left[\frac{\partial^2 \boldsymbol{\sigma}}{\partial x^2} - \frac{\partial^2 \boldsymbol{\sigma}}{\partial x^2}\Big|_{0,x}\right]^{\mathrm{T}} \boldsymbol{\beta} - \pi^2 \sin(\pi x) \tag{8}$$

$$\frac{\partial f}{\partial t} = \left[\frac{\partial \boldsymbol{\sigma}}{\partial t} + (x-1)\frac{\partial \boldsymbol{\sigma}}{\partial t}\Big|_{t,0} - x\frac{\partial \boldsymbol{\sigma}}{\partial t}\Big|_{t,1}\right]^{\mathrm{T}} \boldsymbol{\beta}. \tag{9}$$

As previously stated, the partial derivatives can also be computed by means of automatic differentiation tools, at the cost of a higher computational time. Substituting Eqs. (8) and (9) into Eq. (7) and re-arranging yields,

$$\underbrace{\left[\frac{\partial^2 \boldsymbol{\sigma}}{\partial x^2} - \frac{\partial \boldsymbol{\sigma}}{\partial t} - \frac{\partial^2 \boldsymbol{\sigma}}{\partial x^2}\Big|_{x,0} - (x-1)\frac{\partial \boldsymbol{\sigma}}{\partial t}\Big|_{t,0} + x\frac{\partial \boldsymbol{\sigma}}{\partial t}\Big|_{t,1}\right]^{\mathrm{T}}}_{a(x,t)} \boldsymbol{\beta} = \pi^2 \sin(\pi x).$$

Finally, once x and t are discretized, this simplifies to,

$$A\boldsymbol{\beta} = \boldsymbol{b}.$$

2.2.2. Nonlinear Bivariate PDE

The second example problem proposed here is a mono-dimensional time-dependent Bateman-Burgers equation.

$$f_t(t,x) + f(t,x)f_x(t,x) = \nu f_{xx}(t,x)$$

where $x, t \in [0,1] \times [0,1]$ and $\nu = 1$, subject to

$$f(t,0) = 0$$
$$f(t,1) = 0$$
$$f(0,x) = \sin(\pi x).$$

For this problem, the constrained expression in its compact form is,

$$f(t,x) = g(t,x) + \mathbf{v}(x)^{\mathrm{T}} \mathscr{M}(c(t,x) - g(t,x))\mathbf{v}(t),$$

where

$$\mathscr{M}(c(x,t)) = \begin{bmatrix} 0 & c(0,x) \\ c(t,0) & -c(0,0) \\ c(t,L) & -c(0,L) \end{bmatrix},$$

and

$$v(x) = \{1, \quad 1-x, \quad x\}^{\mathrm{T}}, \quad v(t) = \{1, \quad 1\}^{\mathrm{T}}.$$

In its expanded form, the constrained expression becomes,

$$f(t,x) = g(t,x) + (x-1)g(t,0) - xg(t,1) - xg(0,0) + xg(0,1) - g(0,x) + g(0,0) + \sin(\pi x).$$

Using Eq. (2) this can be rewritten as,

$$f(t,x) = \boldsymbol{\sigma}^{\mathrm{T}}\boldsymbol{\beta} + (x-1)\boldsymbol{\sigma}_{t,0}^{\mathrm{T}}\boldsymbol{\beta} - x\boldsymbol{\sigma}_{t,1}^{\mathrm{T}}\boldsymbol{\beta} - x\boldsymbol{\sigma}_{0,0}^{\mathrm{T}}\boldsymbol{\beta} + x\boldsymbol{\sigma}_{0,1}^{\mathrm{T}}\boldsymbol{\beta} - \boldsymbol{\sigma}_{0,x}^{\mathrm{T}}\boldsymbol{\beta} + \boldsymbol{\sigma}_{0,0}^{\mathrm{T}}\boldsymbol{\beta} + \sin(\pi x).$$

The function for this problem is given by,

$$\mathscr{L} = f_t + ff_x - f_{xx} = \frac{\partial f}{\partial t} + f\frac{\partial f}{\partial x} - \frac{\partial^2 f}{\partial x^2}.$$

Next, the derivatives f_t, f_x, and f_{xx} are computed,

$$\frac{\partial f}{\partial t} = \left[\frac{\partial \boldsymbol{\sigma}}{\partial t} + (x-1)\frac{\partial \boldsymbol{\sigma}}{\partial t}|_{t,0} - x\frac{\partial \boldsymbol{\sigma}}{\partial t}|_{t,1}\right]^{\mathrm{T}}\boldsymbol{\beta}$$

$$\frac{\partial f}{\partial x} = \left[\frac{\partial \boldsymbol{\sigma}}{\partial x} - \frac{\partial \boldsymbol{\sigma}}{\partial x}|_{0,x} + \boldsymbol{\sigma}_{t,0} - \boldsymbol{\sigma}_{t,1} - \boldsymbol{\sigma}_{0,0} + \boldsymbol{\sigma}_{0,1}\right]^{\mathrm{T}}\boldsymbol{\beta} + \pi\cos(\pi x)$$

$$\frac{\partial^2 f}{\partial x^2} = \left[\frac{\partial^2 \boldsymbol{\sigma}}{\partial x^2} - \frac{\partial^2 \boldsymbol{\sigma}}{\partial x^2}|_{0,x}\right]^{\mathrm{T}}\boldsymbol{\beta} - \pi^2\sin(\pi x).$$

Since the problem is nonlinear, the Jacobian matrix is computed,

$$\mathbb{J}(\boldsymbol{\beta}) = \frac{\partial \mathscr{L}}{\partial \boldsymbol{\beta}}$$

and then the iterative least-squares approach as described in Section 2.1 can be used.

2.3. Preliminary theoretical considerations on physics-informed neural network frameworks

As stated in [33], despite their empirical success, there is still little theoretical justification for the physics-informed NN methodology. More specifically, there is lack of an established analysis that may guarantee the convergence of the learning process to the true solutions of the DEs. To this end, the authors are currently working to establish a mathematical foundation that guarantees the convergence for the proposed physics-informed X-TFC framework. Here, a few preliminary theoretical considerations are reported.

As stated in the introduction, the interest of the machine learning community in using NNs (shallow and/or deep) is mainly driven by the fact that NNs are universal approximators [22,26]. Indeed, the *Universal Approximation Theorem* states that any NN with a single hidden layer and a finite number of hidden neurons can approximate arbitrarily well any continuous function on any compact subset of $I \subset \mathbb{R}^n$, i.e. for any continuous sigmoidal function, the family of functions parametrized by one hidden layer NN is dense, with respect to the supremum norm in $\mathscr{C}^0(I)$ [21]. Various methods have been employed in the 80s and 90s to develop the mathematical proof including inverse Radon transform [55], Hahn–Banach and Riesz representation theorem [21], Stone-Weierstrass theorem and trigonometric functions [22], just to mention a few. More recently, ELM theories have demonstrated that single-layer forward networks with random hidden nodes act as universal approximators as well. Indeed, Huang et al. [23,32] have shown that any continuous function can be approximated arbitrarily well in the L^2 sense by single-layer NNs for both data regression and classification (see Theorems 2.1 and 2.2).

Whereas NNs can approximate any continuous function on a compact set, many applied engineering problems that involve systems identification and/or with computing the output of dynamical system in space and time, must deal with approximating nonlinear continuous functionals and/or operators on some infinite dimensional function space. Two key results have been developed by Chen and Chen [56,25] which can be summarized in the following theorems:

Theorem 2.1. *Suppose that $\sigma \in (TW)$(Tauber-Wiener function), X is a Banach space, $K \subset X$is a compact set, V is a compact set in $C(K)$,f is a continuous functional defined on V. Then for any $\epsilon > 0$, there exist positive integers n, m points $x_1, x_2, \ldots, x_m \in K$, and real constants $c_i, \theta_i, \xi_{ij}, i = 1, \ldots n, j = 2, \ldots m$, such that $|f(u) - \sum_{i=1}^{n} c_i \sigma\left(\sum_{j=1}^{m} \xi_{ij} u(x_j) + \theta_i|\right) < \epsilon$, holds for all $u \in V$.*

Theorem 2.2. *Suppose that $\sigma \in (TW)$(Tauber-Wiener function), X is a Banach space, $K_1 \subset X, K_2 \subset \mathbb{R}^d$ are two compact sets in X and $\mathbb{R}^d$, respectively, V is a compact set in $C(K_1)$, G is a nonlinear continuous operator which maps V into $C(K_2)$. Then for any $\epsilon > 0$, there exist positive integers n, p, m, constants $c_i^k, \xi_{ij}^k, \theta_i^k, \zeta_k \in \mathbb{R}, w_k \in \mathbb{R}^d, x_i \in K_1$, $i = 1, \ldots, n, k = 1, \ldots, p, j = 1, \ldots m$ such that $|G(u)(y) - \sum_{k=1}^{p}\sum_{i=1}^{n} c_i^k \sigma\left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k|\right)\sigma(w_k \cdot y + \zeta_k)| < \epsilon$, holds for all $u \in V$ and $y \in K_2$.*

Here, $C(K)$ represents the Banach space of all continuous functions defined on a compact set $K \subset X$ equipped with the L^∞ norm. *Theorem 4.1* states that we can define a NN model to approximate nonlinear functionals on some compact set $C(K)$. Subsequently, *Theorem 4.2* states that we can define a NN model to approximate nonlinear operators, i.e. the output of some dynamical system as a whole, thus effectively providing system identification. The Tauber-Wiener (TW) functions [25] are a critical element of the approximation. Importantly, virtually any known activation function (e.g. ReLu, sigmoid, *tanh*) are TW functions. Theorem 4.2 is the most relevant to our X-TFC framework as it indicates that NNs can effectively learn nonlinear operators from data, although it does not tell us how to practically learn such operators [57]. Generally, as stated in [33,27], the challenge is to prove that there exists a NN capable of simultaneously satisfying both the DE and its initial and/or boundary conditions. Pinkus [26] and Lu et al. [27] have shown that feed-forward NNs with enough hidden neurons are able to uniformly and simultaneously approximate any function and its partial derivatives (see Theorem 2.1 in [27], where the interested reader can also find the formal proof). Thus, based on these theoretical results, NNs can be effectively employed to approximate the solution of DEs. Nevertheless, NNs in practice have limited size. Let $\mathscr{F}$ be the family of all the functions that can be approximated with a particular NN (either shallow or deep), and let f^* be the true solution of the DE. With high probability f^* does not belong to $\mathscr{F}$. Thus, we define $f_{\mathscr{F}} = \arg\min_{f_{NN} \in \mathscr{F}} ||f_{NN} - f^*||$ as the closest function in $\mathscr{F}$ to f^*, where f_{NN} is the NN approximation of f^*. This introduces an error, called the approximation error $\epsilon_{\text{app}} = ||f_{\mathscr{F}} - f^*||$, which is the theoretical minimum loss the neural network could achieve if it were trained on all data points in the domain and achieved the global minimum during training. Of course, the NN cannot be trained on all points in the domain, which leads to the generalization error, $\epsilon_{\text{gen}} = ||f_{\mathscr{F}} - f_{\mathscr{F}}||$, where $f_{\mathscr{F}}$ is the NN with tuned parameters such that the loss for some training set training set $\mathscr{T}$ is at the global minimum. In addition, the NN may not be trained to the global minimum, which introduces the optimization error, $\epsilon_{\text{opt}} = ||\hat{f}_{\mathscr{F}} - f_{\mathscr{F}}||$, which comes about as a result of the optimization technique used to compute $f_{\mathscr{F}}$. Therefore, the total error is [57],

$$\epsilon_{\text{tot}} = ||\hat{f}_{\mathscr{T}} - f^*|| \leqslant \epsilon_{\text{opt}} + \epsilon_{\text{gen}} + \epsilon_{\text{app}}$$

Additionally, Shin et al. [33] proved that for elliptic and parabolic DEs, adapting the Schuader approach, the sequence of minimizers $\hat{f}_{\mathscr{T}}$ strongly converges to the DE solution in L^2. Moreover, they have proved that if each minimizer satisfies the initial/boundary conditions of the DE, the convergence can be improved to H^1.

In [24,31,58], the authors have leveraged theoretical properties and merged them with the ELM algorithm to successfully develop physics-informed ELM based frameworks to solve DEs.

This work also leverages the approximation properties of the ELM algorithm, and combines it with the properties of the TFC framework to build a new physics-informed NN framework for solving DEs. Although a mathematical proof for the convergence of ELMs in linear regression problems has been established, a similar proof does not yet exist for X-TFC. When using an ELM in a TFC constrained expression and then substituting that constrained expression into the residual of a DE, the proof regarding convergence of ELMs in linear regression no longer applies. Future work will focus on establishing a convergence proof for the X-TFC case. Convergence that, at least for elliptic and parabolic DEs, according to [33], is expected to be in H^1 as in the X-TFC framework the IC and/or BC are analytically satisfied.

3. Results

In this section, the X-TFC method is tested on a series of problems of interest in physics and engineering. The problems considered are linear and nonlinear ODEs, System of ODEs (SODEs), and PDEs. Moreover, when they are available, the X-TFC method is compared with other state-of-the-art methods and with analytical solutions. The results show that the X-TFC method is as accurate or more accurate than all other methods except the classic TFC; although it should be noted here that in all cases the classic TFC and X-TFC have solutions errors that are on the same order of magnitude. For the sake of clarity, the solution accuracy is measured using the absolute error, i.e., the absolute value of the difference between the analytical and approximated solutions (computed at each training/test point).

Moreover, since in this paper we do not consider experimental data for data-driven solution of DEs, the parameters Γ_f and Γ_l in Eq. (4) are set equal to zero and one respectively.

3.1. ODEs

In this section, X-TFC is applied to linear and nonlinear ODEs and SODEs. Each problem was solved in MATLAB on an Intel Core i7 - 9700 CPU PC with 64 GB of RAM.

As explained previously, in the X-TFC method there are several hyperparameters that can be modified to obtain accurate solutions. These hyperparameters are the number of training points, N, the number of neurons, L, the type of activation function, and the probability distribution used to initialize the weights and biases of the ELM. An analysis was performed to study the sensitivity of the X-TFC framework to these hyperparameters. This analysis showed that, for the problems considered, the solution accuracy is not as sensitive to the type of activation function used or to the probability distribution used to initialize the weights and biases as it is to the number of training and the number of neurons. The solution accuracy trends for the number of training points and number of neurons for Problem 1 is shown in Fig. 3. Fig. 3(a) shows the solution accuracy as a function of the number of training points while holding the number of neurons constant, and Fig. 3(b) shows the solution accuracy as a function of the number neurons for a fixed number of training points. Since the X-TFC methodology uses random numbers to initialize the weights and biases that are untrained, the method is inherently stochastic. Thus, each point in the plots of Fig. 3 is the maximum absolute error of 10^3 Monte Carlo simulations. In this article the sensitivity analysis is only shown for problem 1, because the same behaviour was observed for all the problems analyzed.

For all the problems reported in this section we use $N = 50$ uniformly spaced training points, 100 uniformly spaced test points, $L = 50$, the activation function used in the ELM was a *logistic* activation function, and the weights and biases of the ELM were randomly sampled from a uniform distribution, $(w_j, b_j) \sim \mathscr{U}(-10, 10)$ where $j = 1, \ldots, L$. As previously mentioned, due to the inherently stochastic nature of the X-TFC method, for each problem solved in this section, 10^3 Monte Carlo simulations were performed to show the variability and test the robustness of the method.

3.1.1. Problem 1

The following equation is a nonlinear ODE taken from Yang et al. [31],

$$y_{tt} = \frac{1}{2t^2}\left(y^3 - 2y^2\right)$$

subject to $y(1) = 1$ and $y(2) = 4/3$ for $t \in [1, 2]$. The exact solution is $y(t) = \frac{2t}{t+1}$. The constrained expression for this problem is,

$$y(t) = \left[\boldsymbol{\sigma} + (t-2)\boldsymbol{\sigma}_0 + (1-t)\boldsymbol{\sigma}_f\right]^{\mathrm{T}}\boldsymbol{\beta} + \frac{t+2}{3},$$

where $\boldsymbol{\sigma}_0$ and $\boldsymbol{\sigma}_f$ are the free-functions computed at $t = 1$ and $t = 2$, respectively.

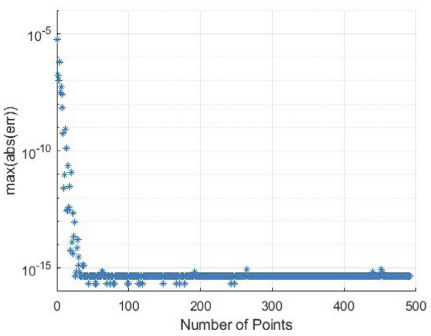
The results and X-TFC performances for Problem 1 are presented in Fig. 4 and 5, and Table 1. The results show that the average absolute errors for both training and testing are on the order of $\mathscr{O}\left(10^{-16}\right)$, with a standard deviation on the order of $\mathscr{O}\left(10^{-16}\right)$. The computational time is on the order of $\mathscr{O}\left(10^{-4}\right)$ seconds. The results reported in Table 1 are achieved by using $N = 50$ training points and $L = 50$ neurons computed at the test points. Note that the order of the error is the same for all test points, and the order of the test set error closely matches that of the training set error. In fact, this phenomenon has been noted when applying TFC-based methods. At this time there is no formal mathematical theorem/proof that explains this behavior. However, the current conjecture by the authors is that this behavior is due to the fact that after convergence TFC-based methods represent an analytical approximation of the solution of the differential equation, i.e., no interpolation is required to obtain the solution at points not seen during training. In contrast, other methods such as FEM require interpolation to calculate the solution at points not seen during training; depending on the interpolation scheme used and relative distance between training points, this can result in test points with errors of different orders of magnitude and a test set error that is orders of magnitude different than the training set errors. This behaviour is observed for all the problems solved in this work.

3.1.2. Problem 2

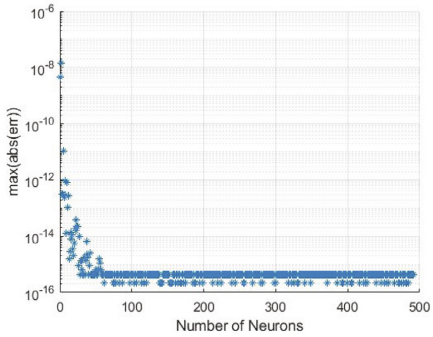
Problem 2 is a system of nonlinear ODEs taken from Lagaris et al. [20],

$$y_{1_t} = \cos t + y_1^2 + y_2 - \left(1 + t^2 + \sin^2 t\right)$$
$$y_{2_t} = 2t - \left(1 + t^2\right)\sin t + y_1 y_2$$

subject to $y_1(0) = 0$ and $y_2(0) = 1$ where $t \in [0, 3]$. The exact solutions are $y_1(t) = \sin(t)$and $y_2(t) = 1 + t^2$. The constrained expressions for this problem are,



(a) Maximum absolute error as a function of number of points n, for a fixed number of neurons $L = 100$



(b) Maximum absolute error as a function of number of neurons L, for a fixed number of points $N = 100$

Fig. 3. Solution accuracy sensitivity analysis for problem #1.

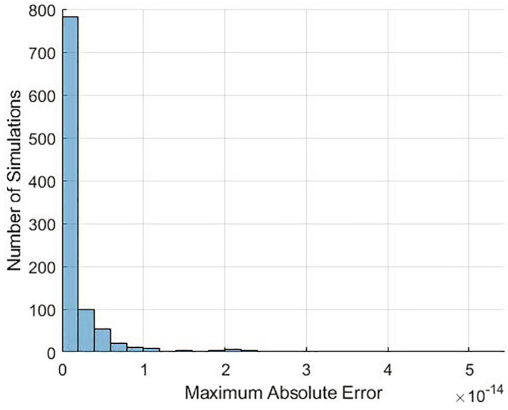


Fig. 4. Histogram of 10^3 Monte Carlo simulations for problem #1.

$$y_1(t) = [\boldsymbol{\sigma} - \boldsymbol{\sigma}_0]^{\mathrm{T}}\boldsymbol{\beta}_1$$
$$y_2(t) = [\boldsymbol{\sigma} - \boldsymbol{\sigma}_0]^{\mathrm{T}}\boldsymbol{\beta}_2 + 1$$

where $\boldsymbol{\sigma}_0$ is the free-function computed at $t = 0$.

The results and X-TFC performances for Problem 2 are reported in Figs. 6 and 7, and Tables 2 and 3. For y_1, the average absolute errors for both training and testing are on the order of $\mathcal{O}\left(10^{-11}\right)$, with a standard deviation on the order of $\mathcal{O}\left(10^{-11}\right)$. For y_2, the average absolute errors for both training and testing are on the order of $\mathcal{O}\left(10^{-11}\right)$, with a standard deviation on the order of $\mathcal{O}\left(10^{-10}\right)$. The computational time for both y_1 and y_2 is on the order of $\mathcal{O}\left(10^{-2}\right)$ seconds. The increase in the computational time due to the nonlinear nature of the problem. Thus, to compute the solution, an iterative-least square is required. The results reported in Tables 2 and 3 are achieved by using $N = 50$ training points and $L = 50$ neurons computed at the test points.

3.2. PDEs

This section presents the results of the proposed method when applied a variety of PDEs. For each problem, the PDE and its constraints are summarized along with the relevant equations needed to construct the constrained expression. Each problem follows the same format. First, the $\mathscr{M}$ tensor and $\mathbf{v}$ vectors are presented followed by an expanded form of the constrained expression. Problem 1 provides a reminder of the notation used in the constrained expressions, while the latter examples simply state the terms used. For a more detailed explanation on how to build constrained expression, consult Refs. [4,41,42].

All PDE problems shown in this article were implemented in Python and utilized the autograd package [59], which uses automatic differentiation [54] to compute the derivatives. Consequently, two specific computation times are provided, (1) the full run-time of the problem and (2) the computation time associated with the least-squares. As observed in the results, the full run-time is drastically affected by the computation overhead from autograd with full run times on the order of 1–100 s. However, the computation time for the least squares and nonlinear least squares is on the order of 1–100 ms. Therefore, the second time reported, the one associated with the least squares, is the expected computation time if the partial derivative of the problem were computed analytically and explicitly programmed and/or if the problems were solved in a compiled language. All input weights and biases for all problems were randomly sampled from a uniform distribution, $(\mathbf{w}_j, b_j) \sim \mathscr{U}(-1, 1)$ where $j = 1, \ldots, L$.

For the PDEs, the same sensitivity analysis was performed as for the ODEs. The results of one of these studies is shown in Figs. 8 and 9 for the PDE of problem number 1. Fig. 8 shows the solution error for the PDE of problem number 1 as a function of the number of training points in the grid. The x-axis in this figure shows the number of training points used per independent variable: the total number of training points used was the square of the values on the x-axis. Each data point in Fig. 8 used 170 basis functions. Fig. 9 shows the solution error for the PDE of problem number 1 as a function of the number of basis functions. The number of training points used for each point in Fig. 9 was 900, a 30×30 grid. The results are consistent with the results obtained for the ODEs: the solution error asymptotically decreases as the number of basis functions increases, and the solution error asymptotically decreases as the number of training points increases.

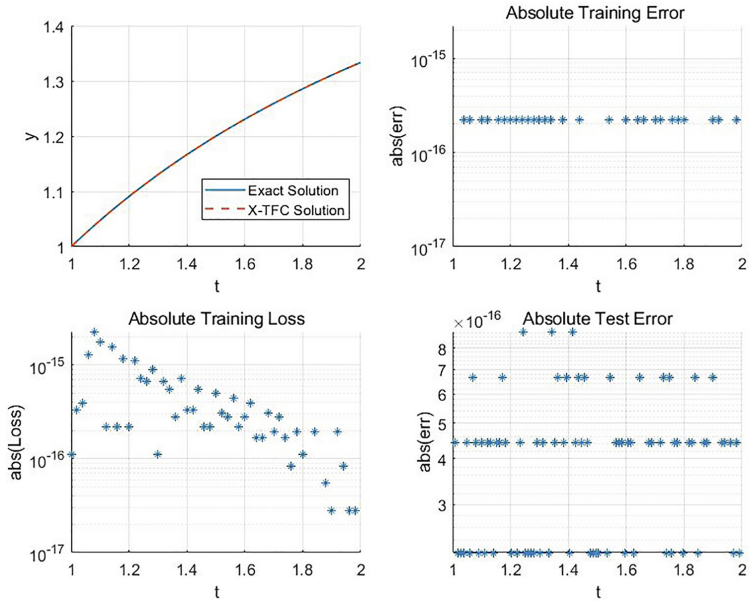


Fig. 5. Exact solution versus X-TFC solution, absolute training error, absolute loss, and absolute test error for problem #1. The convergence is achieved in 4 iterations with a tolerance set to 4.4×10^{-16}. The computational time is 0.8433milliseconds, the maximum absolute training error is 2.2×10^{-16}, the maximum absolute training loss is 2.3×10^{-15}, and the maximum absolute test error is 8.9×10^{-16}.

Table 1
Problem #1: X-TFC and TFC absolute errors with respect the exact solution. As reported in [31], the average absolute error of LeNN method is of $\mathcal{O}\left(10^{-3}\right)$.

Test Points	X-TFC	TFC
1	0	0
1.1	0	2.2×10^{-16}
1.2	2.2×10^{-16}	0
1.3	2.2×10^{-16}	2.2×10^{-16}
1.4	2.2×10^{-16}	0
1.5	2.2×10^{-16}	2.2×10^{-16}
1.6	0	2.2×10^{-16}
1.7	2.2×10^{-16}	2.2×10^{-16}
1.8	2.2×10^{-16}	2.2×10^{-16}
1.9	0	0
2	0	0

The tests in this section were performed on a MacBook Pro (2016) macOS Version 10.15.3 with a 3.3 GHz Dual-Core Intel® Core™ i7 and with 16 GB of RAM. All run times were calculated using the default_timer function in the Python `timeit` package.

3.2.1. Problem 1

Consider the PDE solved in Largaris et al. [20], Mall & Chakraverty [60], and Sun et al. [58],

$$f_{xx}(x,y) + f_{yy}(x,y) = e^{-x}(x - 2 + y^3 + 6y)$$

where $x, y \in [0, 1]$ and subject to,

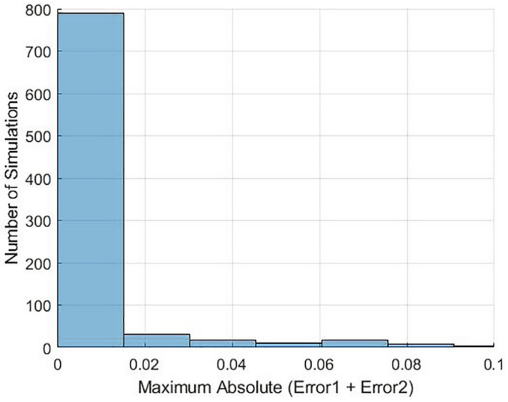
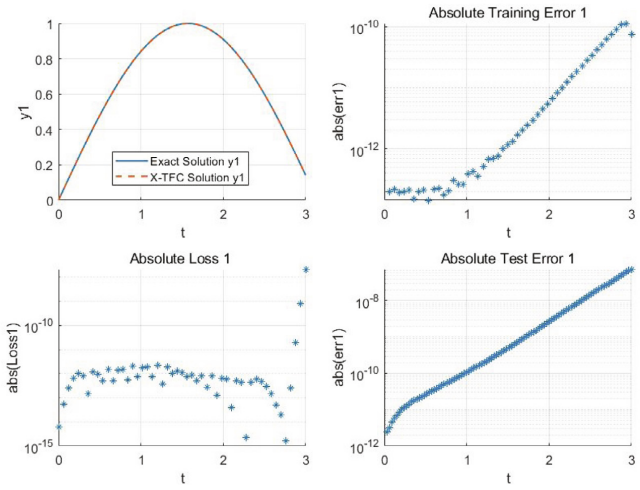


Fig. 6. Histogram of 10^3 Monte Carlo simulations for problem #2.

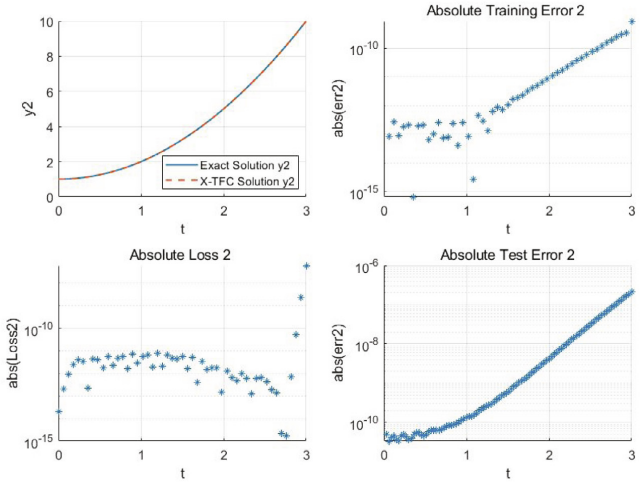
$$f(0,y) = y^3$$
$$f(1,y) = (1 + y^3)e^{-1}$$
$$f(x,0) = xe^{-x}$$
$$f(x,1) = e^{-x}(x + 1),$$

which has the true solution $f(x,y) = e^{-x}(x + y^3)$. Using the TFC [42,43], the constrained expression for the specified boundary conditions can be written in its compact form,

$$f(x,y) = g(x,y) + \mathbf{v}(x)^{\mathrm{T}}.\mathscr{M}(c(x,y) - g(x,y))\mathbf{v}(y),$$



(a) Exact solution versus X-TFC solution, absolute training error, absolute loss, and absolute test error for y_1. The convergence is achieved in 9 iterations with a tolerance set to 1.0×10^{-6}. The computational time is 11.09 milliseconds, the maximum absolute training error is 1.1×10^{-10}, the maximum absolute training loss is 2.1×10^{-8}, and the maximum absolute test error is 3.4×10^{-8}



(b) Exact solution versus X-TFC solution, absolute training error, absolute loss, and absolute test error for y_2. The convergence is achieved in 9 iterations with a tolerance set to 1.0×10^{-6}. The computational time is 11.09 milliseconds, the maximum absolute training error is 8.6×10^{-10}, the maximum absolute training loss is 6.1×10^{-8}, and the maximum absolute test error is 9.6×10^{-8}

Fig. 7. Solution and performances for problem #2.

where $g(x, y)$ will be defined as an ELM, the c terms are defined by the constraints (e.g. $c(0, y) := f(0, y)$), and the values $c(0,0), c(0,1), c(1,0)$, and $c(1,1)$ coincide with the intersection of the constraints and are therefore defined by the constraints. Furthermore, for this problem,

$$\mathscr{M}(c(x,y)) = \begin{bmatrix} 0 & c(x,0) & c(x,1) \\ c(0,y) & -c(0,0) & -c(0,1) \\ c(1,y) & -c(1,0) & -c(1,1) \end{bmatrix}.$$

and

Table 2
Problem #2: X-TFC and TFC absolute errors on y_1 with respect the analytical solution. The maximum absolute error obtained with the ANN method [20] is of $\mathcal{O}\left(10^{-4}\right)$

Test Points	X-TFC	TFC
0	0	0
0.3	1.1×10^{-13}	1.2×10^{-13}
0.6	1.7×10^{-13}	2.9×10^{-13}
0.9	3.0×10^{-13}	7.2×10^{-13}
1.2	6.7×10^{-13}	1.6×10^{-12}
1.5	1.7×10^{-12}	4.1×10^{-12}
1.8	4.4×10^{-12}	1.1×10^{-11}
2.1	1.2×10^{-11}	3.0×10^{-11}
2.4	3.4×10^{-11}	8.0×10^{-11}
2.7	9.4×10^{-11}	1.8×10^{-10}
3	1.8×10^{-10}	1.6×10^{-10}

Table 3
Problem #2: X-TFC and TFC absolute errors on scattered flux y_2 with respect the exact solution. The maximum absolute error obtained with the ANN method [20] is of $\mathcal{O}\left(10^{-5}\right)$

Test Points	X-TFC	TFC
0	0	0
0.3	6.5×10^{-14}	2.2×10^{-13}
0.6	1.3×10^{-13}	2.0×10^{-13}
0.9	2.5×10^{-13}	1.2×10^{-12}
1.2	6.6×10^{-13}	1.4×10^{-12}
1.5	2.0×10^{-12}	5.5×10^{-12}
1.8	6.3×10^{-12}	1.5×10^{-11}
2.1	2.1×10^{-11}	5.2×10^{-11}
2.4	7.0×10^{-11}	1.6×10^{-10}
2.7	2.3×10^{-10}	4.5×10^{-10}
3	5.2×10^{-10}	4.7×10^{-10}

$$\mathbf{v}(x) = \{\,1, \quad 1-x, \quad x\,\}^{\mathrm{T}}, \quad \mathbf{v}(y) = \{\,1, \quad 1-y, \quad y\,\}^{\mathrm{T}}.$$

It follows that the expanded constrained expression is,

$$\begin{aligned} f(x,y) = {} & g(x,y) - (x-1)\big(y(-g(0,0)+g(0,1)-1)+g(0,0)+y^3\big) \\ & + (x-1)g(0,y) + x(yg(1,1) - (y-1)g(1,0)) - xg(1,y) \\ & + (y-1)g(x,0) - yg(x,1) + \frac{xy(y^2-1)}{e} + e^{-x}(x+y) \end{aligned}$$

For this problem, the free-function, $g(x,y)$, was chosen to be an ELM with 170 neurons that used `tanh` as the nonlinear activation function. Then, the constrained expression and its derivatives were substituted into the differential equation, which converts the differential equation into an algebraic equation. To solve this algebraic equation, the problem was discretized over 30×30 training points that spanned the domain. This system of equations was solved using NumPy's `lstsq` function.

The total execution time was 3.48 s, and the least-squares took 8.07 ms. Additionally, the training set maximum error was 3.808×10^{-13}, and the training set average error was 6.475×10^{-14}. The test set maximum error was 5.054×10^{-13}, and the test set average error was 7.640×10^{-14}; the test set was a 100×100 grid of uniformly spaced points. Fig. 10 shows a plot of the error over the domain, and Table 4 compares the X-TFC solution with Deep-TFC, the FEM, and Refs. [20,60,58]. For the Deep-TFC results, a fully connected neural network with 4 hidden layers, 10 neurons per layer, and one linear output layer was used. The nonlinear activation function used in the hidden layers was the hyperbolic tangent. To train the network, a 30×30 grid of evenly distributed points was used. The training took about 301.4 s. The interested reader can find more details regarding the training of the DNN in [41]. Fig. 10 shows that the error is distributed approximately evenly throughout the domain. Table 4 shows that the X-TFC method outperforms the other methods in terms of solution error by 5 to 11 orders of magnitude.

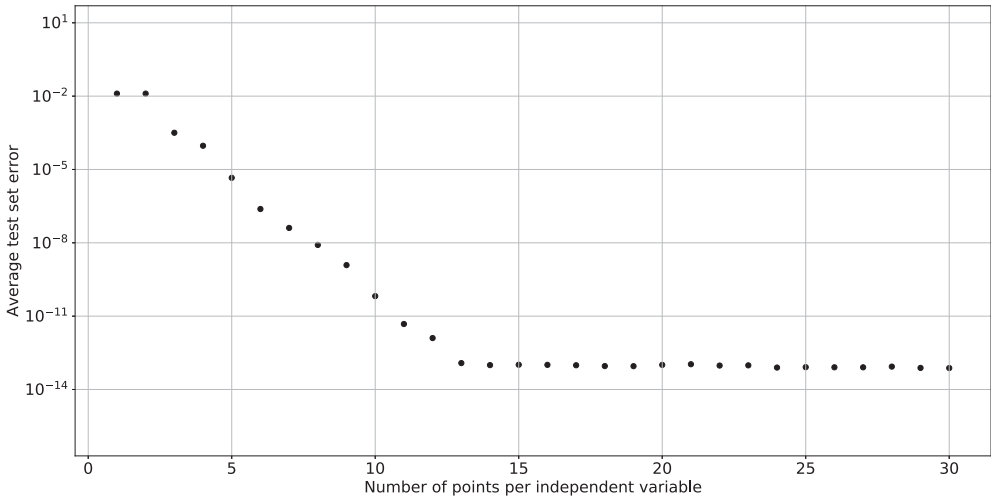


Fig. 8. Average test set error as a function of the number of points per side in the grid.

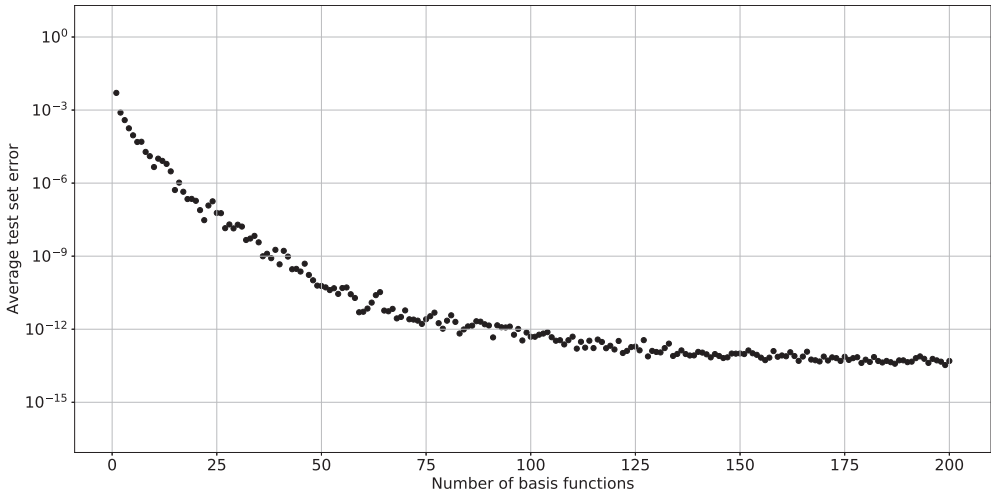


Fig. 9. Average test set error as a function of the number of basis functions.

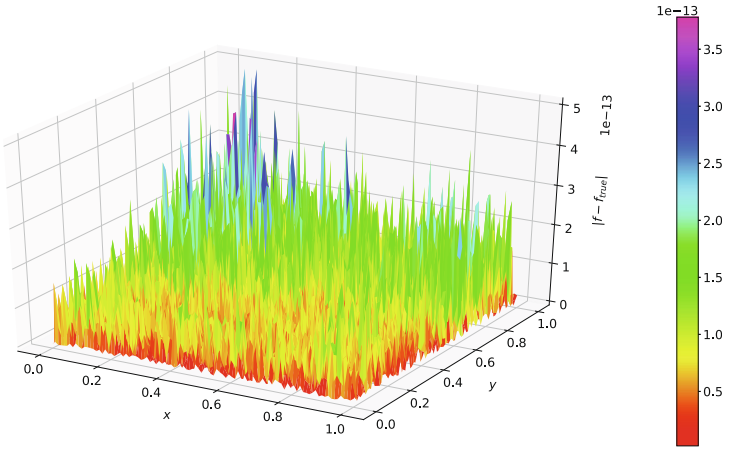


Fig. 10. Solution error using X-TFC in problem 1.

Table 4
Comparison of maximum training and test error of X-TFC with current state-of-the-art techniques for Problem 3.

Method	Training Set Maximum Error	Test Set Maximum Error
X-TFC	3.8×10^{-13}	5.1×10^{-13}
Deep-TFC	2×10^{-7}	2×10^{-7}
FEM	2×10^{-8}	1.5×10^{-5}
Ref. [20]	5×10^{-7}	5×10^{-7}
Ref. [60]	–	3.2×10^{-2}
Ref. [58]	–	2.4×10^{-4}

3.2.2. Problem 2

Consider the PDE solved in Largaris et al. [20] and Mall & Chakraverty [60],

$$f_{xx}(x,y)+f_{yy}(x,y)=\left(2-\pi^2y^2\right)\sin(\pi x)$$

where $x,y\in[0,1]$ and subject to,

$$f(0,y)=0$$
$$f(1,y)=0$$
$$f(x,0)=0$$
$$f_y(x,1)=2\sin(\pi x),$$

which has the true solution $f(x,y)=y^2\sin(\pi x)$.

. Constrained expression (compact):

$$f(x,y)=g(x,y)+\mathbf{v}(x)^{\mathrm{T}}\mathscr{M}(c(x,y)-g(x,y))\mathbf{v}(y)$$

where

$$\mathscr{M}(c(x,y))=\begin{bmatrix}0 & c(x,0) & c_y(x,1)\\ c(0,y) & -c(0,0) & -c_y(0,1)\\ c(1,y) & -c(1,0) & -c_y(1,1)\end{bmatrix}$$

and

$$\mathbf{v}(x)=\{1,\quad 1-x,\quad x\}^{\mathrm{T}},\quad \mathbf{v}(y)=\{1,\quad 1,\quad y\}^{\mathrm{T}}.$$

Constrained expression (expanded):

$$\begin{aligned}f(x,y)=y\big((1-x)g_y(0,1)+xg_y(1,1)-g_y(x,1)+2\sin(\pi x)\big)\\ -(1-x)g(0,y)-xg(1,y)+g(x,y)+(1-x)g(0,0)\\ +xg(1,0)-g(x,0)\end{aligned}$$

For this problem, the free-function was chosen to be an with 170 neurons that used `tanh` as the activation function. The problem was discretized over 30 × 30 training points that spanned the domain, and the least-squares problem was solved using NumPy's `lstsq` function.

The total execution time was 3.54 s, and the least-squares took 9.34 ms. Furthermore, the training set maximum error was 6.332×10^{-12}, and the training set average error was 1.187×10^{-12}. The test set maximum error was 7.581×10^{-12}, and the test set average error was 1.322×10^{-12} the test set was a 100×100 grid of uniformly spaced points. Fig. 11 shows a plot of the error over the domain, and Table 5 compares the X-TFC solution with, the FEM and, Refs. [20,60]. For the Deep-TFC results a fully connected neural network with 4 hidden layers, 10 neurons per layer, and one linear output layer was used. The nonlinear activation function used in the hidden layers was the hyperbolic tangent. To train the network, a 30×30 grid of evenly distributed points was used. The training time was 424.43 seconds. The interested reader can find more details regarding the training of the DNN in [41]. Fig. 11 shows that the solution error is lower near where constraints are defined on the boundary value than where they are defined on the derivative. Intuitively, this makes sense as the TFC constrained expressions guarantees there will be no error in the solution value for constraints defined on the boundary value, but not for constraints on the derivative (there it only garuantees that the deriative will have no error). Table 5 shows that the X-TFC method outperforms all other methods in terms of accuracy by 5 to 9 orders of magnitude.

3.2.3. Problem 3

Consider the PDE solved in Largaris et al. [20],

$$\begin{aligned}f_{xx}(x,y)+f_{yy}(x,y)+f(x,y)f_y(x,y)\\ =\sin(\pi x)\left(2-\pi^2y^2+2y^3\sin(\pi x)\right)\end{aligned}$$

where $x,y\in[0,1]$ and subject to,

$$f(0,y)=0$$
$$f(1,y)=0$$
$$f(x,0)=0$$
$$f_y(x,1)=2\sin(\pi x),$$

which has the true solution $f(x,y)=y^2\sin(\pi x)$.

. Constrained expression (compact):

$$f(x,y)=g(x,y)+\mathbf{v}(x)^{\mathrm{T}}\mathscr{M}(c(x,y)-g(x,y))\mathbf{v}(y)$$

where

$$\mathscr{M}(c(x,y))=\begin{bmatrix}0 & c(x,0) & c_y(x,1)\\ c(0,y) & -c(0,0) & -c_y(0,1)\\ c(1,y) & -c(1,0) & -c_y(1,1)\end{bmatrix}$$

and

$$\mathbf{v}(x)=\{1,\quad 1-x,\quad x\}^{\mathrm{T}},\quad \mathbf{v}(y)=\{1,\quad 1,\quad y\}^{\mathrm{T}}.$$

Constrained expression (expanded):

$$\begin{aligned}f(x,y)=y\big((1-x)g_y(0,1)+xg_y(1,1)-g_y(x,1)+2\sin(\pi x)\big)\\ -(1-x)g(0,y)-xg(1,y)+g(x,y)+(1-x)g(0,0)\\ +xg(1,0)-g(x,0)\end{aligned}$$

For this problem, the free-function was chosen to be an ELM with 150 neurons that used `tanh` as the activation function. The problem was discretized over 20×20 training points that spanned the domain, and each iteration of the nonlinear least-squares was solved using NumPy's `lstsq` function.

The total execution time was 22.48 s, and the nonlinear least-squares, which needed 10 iterations, took 52.6 ms. In addition, the training set maximum error was 7.634×10^{-11}, and the training set average error was 9.497×10^{-12}. The test set maximum error was 8.977×10^{-11}, and the test set average error was 1.068×10^{-11}; the test set was a 100×100 grid of uniformly spaced points. Fig. 12 shows a plot of the error over the domain, and Table 6 compares the X-TFC solution with, the FEM method, and Ref. [20]. For the Deep-TFC results a fully connected neural network with 4 hidden layers, 10 neurons per layer, and one linear output layer was used. The nonlinear activation function used in the hidden layers was the hyperbolic tangent. To train the network, a 30×30 grid of evenly distributed points was used. The train to about 334.26 s. The interested reader can find more details regarding the training of the DNN in [41]. As in problem 2, Fig. 12 shows illustrates that the solution error is lower near where constraints are defined on the boundary value than where they are defined on the derivative, because the TFC constrained expressions guarantees there will be no error in the solution value for constraints defined on the boundary. Table 6 shows that the X-TFC method outperforms all other methods in terms of accuracy by 4 to 6 orders of magnitude.

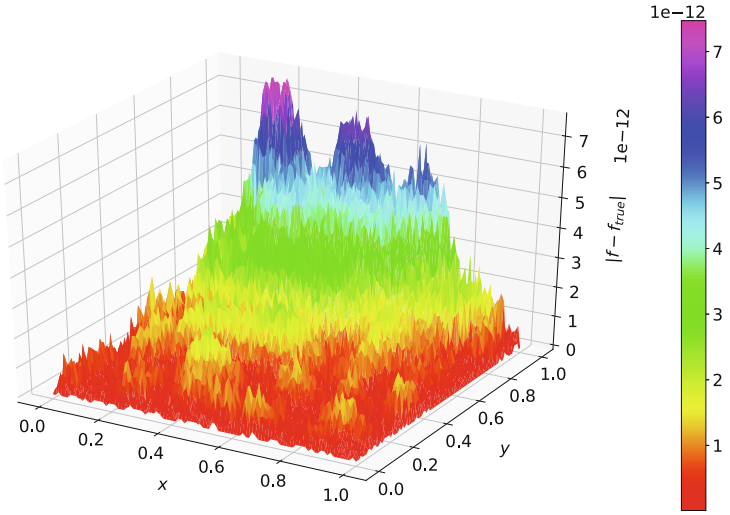


Fig. 11. Solution error using X-TFC in problem 2.

Table 5
Comparison of maximum training and test error of X-TFC with current state-of-the-art techniques for Problem 1.

Method	Training Set Maximum Error	Test Set Maximum Error
X-TFC	6.3×10^{-12}	7.6×10^{-12}
Deep-TFC	6×10^{-6}	6×10^{-6}
FEM	7×10^{-7}	4×10^{-5}
Ref. [20]	6×10^{-6}	6×10^{-6}
Ref. [60]	–	3×10^{-3}

Table 6
Comparison of maximum training and test error of X-TFC with current state-of-the-art techniques for Problem 2.

Method	Training Set Maximum Error	Test Set Maximum Error
X-TFC	8.8×10^{-11}	9.0×10^{-11}
Deep-TFC	3.8×10^{-6}	3.8×10^{-6}
FEM	6×10^{-7}	4×10^{-5}
Ref. [20]	1.5×10^{-5}	1.5×10^{-5}

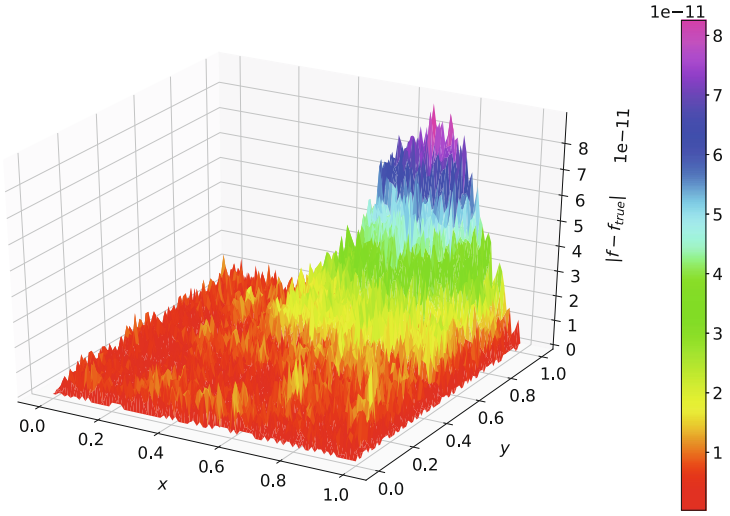


Fig. 12. Solution error using X-TFC in problem 3.

3.2.4. Problem 4 (2D Time-Dependent Heat Equation)

$$f_{xx}(x,y,t)+f_{yy}(x,y,t)=\kappa f_t(x,y,t)$$

where $x,y,t\in[0,L]\times[0,H]\times[0,1]$, subject to

$$\begin{aligned}&f(0,y,t)=0\\&f(L,y,t)=0\\&f(x,0,t)=0\\&f(x,H,t)=0\\&f(x,y,0)=\sin\left(\tfrac{\pi x}{L}\right)\sin\left(\tfrac{\pi y}{H}\right),\end{aligned}$$

which has the true solution $f(x,y,t)=\sin\left(\frac{\pi x}{L}\right)\sin\left(\frac{\pi y}{H}\right)e^{-\left(\frac{\pi^2}{L^2}+\frac{\pi^2}{H^2}\right)t}$. Here, the values $L=2, H=1$, and $\kappa=1$ are used.

. Constrained expression (compact):

$$f(x,y,t)=g(x,y,t)+\mathscr{M}_{ijk}(c(x,y,t)-g(x,y,t))v_i(x)v_j(y)v_k(t)$$

where

$$\mathscr{M}_{ij1}(c(x,y,t))=\begin{bmatrix}0 & c(x,0,t) & c(x,H,t)\\ c(0,y,t) & -c(0,0,t) & -c(0,H,t)\\ c(L,y,t) & -c(L,0,t) & -c(L,H,t)\end{bmatrix}$$

$$\mathscr{M}_{ij2}(c(x,y,t))=\begin{bmatrix}c(x,y,0) & -c(x,0,0) & -c(x,H,0)\\ -c(0,y,0) & c(0,0,0) & c(0,H,0)\\ -c(L,y,0) & c(L,0,0) & c(L,H,0)\end{bmatrix}$$

and

$$v_i(x)=\left\{1,\ \tfrac{L-x}{L},\ \tfrac{x}{L}\right\}^{\mathrm{T}},\quad v_j(y)=\left\{1,\ \tfrac{H-y}{H},\ \tfrac{y}{H}\right\}^{\mathrm{T}},\quad v_k(t)=\{1,\ 1\}^{\mathrm{T}}.$$

Constrained expression (expanded):

$$\begin{aligned}f(x,y,t)=&\frac{(H-y)\left(\frac{(L-x)g(0,0,t)}{L}+\frac{xg(L,0,t)}{L}-g(x,0,t)\right)}{H}\\&+\frac{y\left(\frac{(L-x)g(0,H,t)}{L}+\frac{xg(L,H,t)}{L}-g(x,H,t)\right)}{H}\\&+\frac{(H-y)\left(-\frac{g(0,0,0)(L-x)}{L}-\frac{xg(L,0,0)}{L}+g(x,0,0)\right)}{H}\\&+\frac{y\left(-\frac{(L-x)g(0,H,0)}{L}-\frac{xg(L,H,0)}{L}+g(x,H,0)\right)}{H}\\&-\frac{(L-x)g(0,y,t)}{L}-\frac{xg(L,y,t)}{L}+\frac{(L-x)g(0,y,0)}{L}\\&+\frac{xg(L,y,0)}{L}+g(x,y,t)-g(x,y,0)\\&+\sin\left(\frac{\pi y}{H}\right)\sin\left(\frac{\pi x}{L}\right)\end{aligned}$$

For this problem, the free-function was chosen to be an ELM with 400 neurons that used `tanh` as the activation function. The problem was discretized over 13×13×13 training points that spanned the domain, and the least-squares problem was solved using NumPy's `lstsq` function.

The total execution time was 159.3 s, and the nonlinear least-squares took 55.4 ms. Furthermore, the training set maximum error was 3.985×10^{-4}, and the training set average error was 1.705×10^{-5}. The test set maximum error was 4.025×10^{-4}, and the test set average error was 2.090×10^{-5}; the test set was a $25\times25\times25$ grid of uniformly spaced points.

3.2.5. Problem 5 (Nonlinear 3D time-dependent PDE)

$$\begin{aligned}&f_x(x,y,z,t)f_y(x,y,z,t)f_z(x,y,z,t)+f_{tt}(x,y,z,t)\\&=\ ((t-1)tx(z-1)+x^2\cos\left(x^2y\right)+\tfrac{3}{2}x\sqrt{yz})\cdot\\&\left((t-1)ty(z-1)+2xy\cos\left(x^2y\right)+y^{3/2}z\right)\cdot\\&\left(2\pi t^2\cos(2\pi z)+(t-1)txy+xy^{3/2}\right)+2xy(z-1)\\&+2\sin(2\pi z)\end{aligned}$$

where $x,y,t\in[0,1]$, subject to

$$\begin{aligned}&f(0,y,z,t)=t^2\sin(2\pi z)\\&f(x,0,z,t)=t^2\sin(2\pi z)\\&f(x,y,1,t)=\sin\left(x^2y\right)+x\sqrt{y}\\&f(x,y,z,0)=\sin\left(x^2y\right)+x\sqrt{yz}\\&f(x,y,z,1)=\sin\left(x^2y\right)+x\sqrt{yz}+\sin(2\pi z),\end{aligned}$$

which has the true solution $f(x,y,z,t)=t^2\sin(2\pi z)+\sin\left(x^2y\right)+xy^{3/2}z+xyt(z-1)(t-1)$.

. Constrained expression (compact):

$$\begin{aligned}f(x,y,z,t)=&\,g(x,y,z,t)\\&+\mathscr{M}_{ijkl}(c(x,y,z,t)-g(x,y,z,t))v_i(x)v_j(y)v_k(z)v_l(t)\end{aligned}$$

where

$$\mathscr{M}_{ij11}(c(x,y,z,t))=\begin{bmatrix}0 & c(x,0,z,t)\\ c(0,y,z,t) & -c(0,0,z,t)\end{bmatrix}$$

$$\mathscr{M}_{ij12}(c(x,y,z,t))=\begin{bmatrix}c(x,y,z,0) & -c(x,0,z,0)\\ -c(0,y,z,0) & c(0,0,z,0)\end{bmatrix}$$

$$\mathscr{M}_{ij13}(c(x,y,z,t))=\begin{bmatrix}c(x,y,z,1) & -c(x,0,z,1)\\ -c(0,y,z,1) & c(0,0,z,1)\end{bmatrix}$$

$$\mathscr{M}_{ij21}(c(x,y,z,t))=\begin{bmatrix}c(x,y,1,t) & -c(x,0,1,t)\\ -c(0,y,1,t) & c(0,0,1,t)\end{bmatrix}$$

$$\mathscr{M}_{ij22}(c(x,y,z,t))=\begin{bmatrix}-c(x,y,1,0) & c(x,0,1,0)\\ c(0,y,1,0) & -c(0,0,1,0)\end{bmatrix}$$

$$\mathscr{M}_{ij23}(c(x,y,z,t))=\begin{bmatrix}-c(x,y,1,1) & c(x,0,1,1)\\ c(0,y,1,1) & -c(0,0,1,1)\end{bmatrix}$$

and

$$v_i(x)=\{1,1\}^{\mathrm{T}}\ v_j(y)=\{1,1\}^{\mathrm{T}}\ v_k(z)=\{1,1\}^{\mathrm{T}}\ v_l(t)=\{1,1-t,t\}^{\mathrm{T}}.$$

Constrained expression (expanded):

$$\begin{aligned}f(x,y,z,t)=&\,g(x,y,z,t)\\&+(1-t)(-g(x,y,z,0)+g(x,y,1,0)+g(x,0,z,0)\\&-g(x,0,1,0)+g(0,y,z,0)-g(0,y,1,0)-g(0,0,z,0)\\&+g(0,0,1,0)+xy^{3/2}z-xy^{3/2})+t(-g(x,y,z,1)+g(x,y,1,1)\\&+g(x,0,z,1)-g(x,0,1,1)+g(0,y,z,1)-g(0,y,1,1)\\&-g(0,0,z,1)+g(0,0,1,1)+xy^{3/2}z-xy^{3/2})-g(x,y,1,t)\\&-g(x,0,z,t)+g(x,0,1,t)-g(0,y,z,t)+g(0,y,1,t)\\&+g(0,0,z,t)-g(0,0,1,t)+t^2\sin(2\pi z)+\sin\left(x^2y\right)+xy^{3/2}\end{aligned}$$

For this problem, the free-function was chosen to be an ELM with 340 neurons that used `tanh` as the activation function. The problem was discretized over 5×5×5×5 training points that spanned the domain, and each iteration of the nonlinear least-squares was solved using NumPy's `lstsq` function.

The total execution time was 321.9 s, and the nonlinear least-squares, which needed 10 iterations, took 0.229 s. Additionally, the training set maximum error was 2.744×10^{-5}, and the training set average error was 6.641×10^{-7}. The test set maximum error was 2.978×10^{-5}, and the test set average error was 8.082×10^{-7}; the test set was a $10 \times 10 \times 10 \times 10$ grid of uniformly spaced points.

3.3. Limitations of X-TFC and Synergy with Deep-TFC

Although, at a first glance, X-TFC and Deep-TFC may appear to be similar methods, they are significantly different. Moreover, as we will show in this section, rather than being competing methods, they form a synergy that defines a PINN-TFC based method for the solution of linear and nonlinear ODEs and PDEs that overcomes the main limitation of the standard TFC [4,43] (e.g. curse of dimensionality) especially for the solution of large scale PDEs.

With Deep-TFC, for the first time TFC and DNNs were brought together, but not as a PINN framework; in this method, the DNNs are trained via quasi-Newton methods. In [41], the authors did not fully capture the entire potential of the Deep-TFC method. According to the definition of PINN framework, Deep-TFC belongs to the PINN methods family, and therefore, it can also be used for data-driven solutions and data-driven parameter discovery of DEs; however, in Ref. [41], the authors described it only as a method for approximating the solution of PDEs.

With X-TFC, for the first time, TFC and shallow NNs trained via the ELM algorithm are brought together as a PINN framework. This is not only new, but also demonstrates superior performance in terms of accuracy and computational speed when compared with competing methods such as the Finite Element Method (FEM), and some other methods using NNs for the solution of DEs like the ones presented in Refs. [20,31,58], especially for problems with smooth solutions such as the problems considered in the previous subsection. Moreover, for these kinds of problems, X-TFC outperforms the companion Deep-TFC both in terms of accuracy and computational time. However, for sufficiently complex problems, e.g., PDEs with stiff and/or quasi-discontinuous solutions, Deep-TFC outperforms X-TFC. In other words, while for problems with smooth solutions X-TFC is preferable to Deep-TFC both in term of accuracy and computational time; for sufficiently complex problems Deep-TFC performs better in terms of accuracy.

Below, two problems will be solved, (1) the 1D time-dependent Burger's equation, and (2) the Navier–Stokes (NS) equations. These problems show the limitations of X-TFC, and how, at this stage, in these cases, Deep-TFC is needed to achieve accurate solutions both quantitatively and qualitatively. The Burger's equation shows that when the problem is smooth X-TFC performs better than Deep-TFC both in terms of accuracy and computational speed. When the problem becomes sufficiently stiff, Deep-TFC performs better in terms of accuracy. These examples highlight how X-TFC and Deep-TFC complement each other, and together define an accurate and reliable PINN-TFC based framework for the solution of linear and nonlinear problems involving DEs.

3.3.1. Burgers' equation

Consider the one-dimensional Burgers' Equation,

$$\frac{\partial u}{\partial t} + \alpha u \frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}$$

where $x, t \in [-3, 3] \times [0, 1]$, subject to

$$u(-3,t) = \frac{c}{\alpha} - \frac{c}{\alpha}\tanh\left(\frac{c}{2\nu}(-3-ct)\right)$$
$$u(3,t) = \frac{c}{\alpha} - \frac{c}{\alpha}\tanh\left(\frac{c}{2\nu}(3-ct)\right)$$
$$u(x,0) = \frac{c}{\alpha} - \frac{c}{\alpha}\tanh\left(\frac{c}{2\nu}x\right),$$

which has the true solution $u(x,t) = \frac{c}{\alpha} - \frac{c}{\alpha}\tanh\left(\frac{c}{2\nu}(x-ct)\right)$. Furthermore, let $c = 1$ and $\alpha = 1$. The compact constrained expression for this problem is,

$$u(x,t) = g(x,t) + \mathscr{M}(c(x,t) - g(x,t))_{ij}\, v_i(x)\, v_j(t)$$

where,

$$\mathscr{M}_{ij}(c(x,y,t)) = \begin{bmatrix} 0 & c(x,0) \\ c(-3,t) & -c(-3,0) \\ c(3,t) & -c(3,0) \end{bmatrix}$$

and

$$v_i(x) = \left\{1, \quad \frac{3-x}{6}, \quad \frac{x+3}{6}\right\},$$
$$v_j(t) = \{1, \quad 1\}.$$

In expanded form, the constrained expression is,

$$\begin{aligned} u(x,t) = {} & g(x,t) + \frac{1}{6}(3-x)\left(-g(-3,t) + \frac{c}{\alpha} - \frac{c}{\alpha}\tanh\left(\frac{c(-ct-3)}{2\nu}\right)\right) \\ & + \frac{1}{6}(x+3)\left(-g(3,t) + \frac{c}{\alpha} - \frac{c}{\alpha}\tanh\left(\frac{c(3-ct)}{2\nu}\right)\right) - \frac{1}{6} \\ & \times (3-x)\left(-g(-3,0) + \frac{c}{\alpha}\tanh\left(\frac{3c}{2\nu}\right) + \frac{c}{\alpha}\right) - \frac{1}{6} \\ & \times (x+3)\left(-g(3,0) - \frac{c}{\alpha}\tanh\left(\frac{3c}{2\nu}\right) + \frac{c}{\alpha}\right) - g(x,0) + \frac{c}{\alpha} - \frac{c}{\alpha} \\ & \times \tanh\left(\frac{cx}{2\nu}\right). \end{aligned}$$

Deep-TFC and X-TFC were used to approximate the solution of Burgers' equation using different values of ν; Fig. 13 shows the analytical solution for the largest and smallest values of ν given in Table 7 to give the reader a sense of the depndency of the solution on ν. X-TFC used 601 neurons and the hyberbolic tangent activation function, and Deep-TFC used four hidden layers with 30 neurons per layer and the *hyperbolic tangent* activation function. Each method used a 30×30 grid of uniformly spaced training points, and a uniform grid of 100×100 test points. The maximum and average solution errors on the test set are tabulated for each method as a function of ν in Table 7. In addition, Table 7 shows the training time for each of the methods. The training time was calculated using the `perf_counter` function from the `time` package in Python. The authors note that this is not an ideal comparison as the X-TFC solution was heavily optimized and ran on a CPU, while the Deep-TFC solution was less optimized and ran on a GPU.

Table 7 shows that for larger values of ν, the X-TFC solution outperforms the Deep-TFC solution by as much as three orders of magnitude in terms of accuracy. However, as ν becomes small, Deep-TFC outperforms X-TFC by as much as three orders of magnitude in terms of accuracy. X-TFC always has a similar training time of approximately five seconds, whereas the Deep-TFC training time grows as ν shrinks.

It is clear then when the problem is smooth, i.e., for large values of ν, X-TFC outperforms Deep-TFC both in terms of accuracy and computational time. In contrast, when the problem becomes sufficiently stiff, i.e., for lowers values of ν, Deep-TFC outperforms X-TFC in terms of solution accuracy.

3.3.2. Navier–Stokes Equations

On more complex problems, Deep-TFC does better than X-TFC. To highlight Deep-TFC's performance on complex problems, consider low-speed, two-dimensional developing channel flow governed by the Navier–Stokes equations and the following boundary conditions:

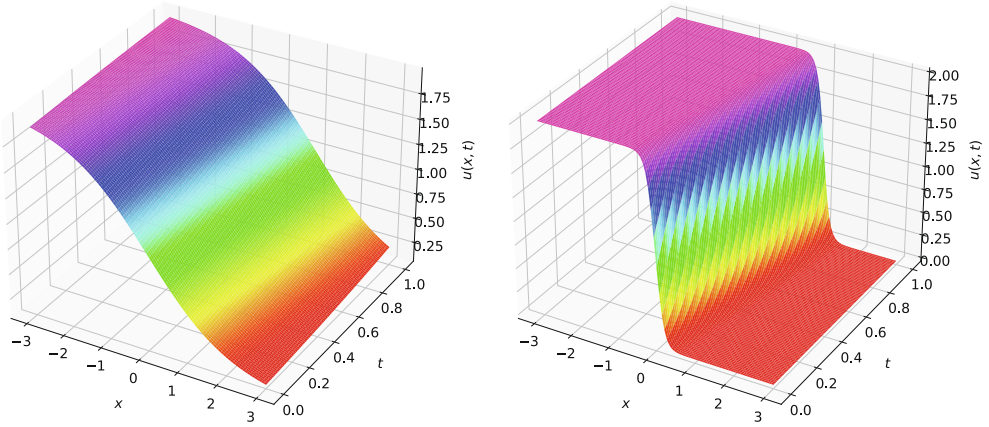


Fig. 13. Analytical solutions of Burgers' Equation. The left plot shows the solution for $\nu = 1.0$ and the right plot for $\nu = 0.1$.

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0$$
$$\rho\left(\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right) = -\frac{\partial p}{\partial x} + \mu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$$
$$\rho\left(\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y}\right) = \mu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right)$$

subjectto :

$$\begin{cases} u(0,y,t) = \frac{\partial u}{\partial x}(L,y,t) = u(x,y,0) = 0 \\ u(x,\frac{H}{2},t) = u(x,-\frac{H}{2},t) = 0 \\ v(0,y,t) = \frac{\partial v}{\partial x}(L,y,t) = v(x,y,0) = 0 \\ v(x,\frac{H}{2},t) = v(x,-\frac{H}{2},t) = 0 \end{cases}$$

where u and v are velocities in the x and y directions respectively, H is the height of the channel, P is the pressure, ρ is the density, and μ is the viscosity. For this problem, the values $H = 1$ m, $\rho = 1$ kg/m^3, $\mu = 1$ Pa·s, and $\frac{\partial p}{\partial x} = -5$ N/m^3 were chosen.

The u and v dependent variables each have the same constraints; therefore, their constrained expressions are the same. Hence, just the constrained expression for u will be shown. In compact form, the constrained expression for u is,

$$u(x,y,t) = g(x,y,t) + \mathscr{M}(c(x,y,t) - g(x,y,t))_{ijk}\, v_i(x)\, v_j(y)\, v_k(t)$$

where,

$$\mathscr{M}_{ij1}(c(x,y,t)) = \begin{bmatrix} 0 & c(x,-\frac{H}{2},t) & c(x,\frac{H}{2},t) \\ c(0,y,t) & -c(0,-\frac{H}{2},t) & -c(0,\frac{H}{2},t) \\ c_x(L,y,t) & -c_x(L,-\frac{H}{2},t) & -c_x(L,\frac{H}{2},t) \end{bmatrix}$$

$$\mathscr{M}_{ij2}(c(x,y,t)) = \begin{bmatrix} c(x,y,0) & -c(x,-\frac{H}{2},0) & -c(x,\frac{H}{2},0) \\ -c(0,y,0) & c(0,-\frac{H}{2},0) & c(0,\frac{H}{2},0) \\ -c_x(L,y,0) & c_x(L,-\frac{H}{2},0) & c_x(L,\frac{H}{2},0) \end{bmatrix}$$

and

$$v_i(x) = \{1, \quad 1, \quad x\},$$
$$v_j(y) = \left\{1, \quad \frac{H-2y}{2H}, \quad \frac{H+2y}{2H}\right\},$$
$$v_k(t) = \{1, \quad 1\}.$$

In expanded form, the constrained expression is,

$$\begin{aligned} u(x,y,t) = \; & g(x,y,t) + \frac{(H+2y)\left(xg_x\left(L,\frac{H}{2},t\right) - g\left(x,\frac{H}{2},t\right) + g\left(0,\frac{H}{2},t\right)\right)}{2H} \\ & + \frac{(H-2y)\left(xg_x\left(L,-\frac{H}{2},t\right) - g\left(x,-\frac{H}{2},t\right) + g\left(0,-\frac{H}{2},t\right)\right)}{2H} \\ & + \frac{(H+2y)\left(-xg_x\left(L,\frac{H}{2},0\right) + g\left(x,\frac{H}{2},0\right) - g\left(0,\frac{H}{2},0\right)\right)}{2H} \\ & + \frac{(H-2y)\left(-xg_x\left(L,-\frac{H}{2},0\right) + g\left(x,-\frac{H}{2},0\right) - g\left(0,-\frac{H}{2},0\right)\right)}{2H} \\ & - xg_x(L,y,t) + xg_x(L,y,0) - g(0,y,t) - g(x,y,0) + g(0,y,0). \end{aligned}$$

For Deep-TFC the training set chosen was 2,000 points independently and identically sampled (IID) from $x \in U(0,15), y \in U(-H/2, H/2)$, and $t \in U(0,3)$. For X-TFC, the training set was a grid of $10 \times 10 \times 10$ evenly spaced points. For each method, the test set consisted of a grid of 100×100 evenly spaced points in x and y at three different times: $t = 0.01, t = 0.1$ and $t = 3$. The test set can be analyzed in two different ways,

1. Qualitatively - The solution should be symmetric about the line $y = 0$, and the solution should develop spatially and temporally such that after a sufficient amount of time has passed and sufficiently far from the inlet, $x = 0$, the u-velocity will be equal, or very nearly equal, to the steady-state Poiseuille flow solution.
2. Quantitatively - The solution at $x = 15$ and $t = 3$ can be compared to the steady-state Poiseuille flow solution.

The neural network used for the Deep-TFC solution had four hidden layers and 30 neurons per layer, and the nonlinear activa-

Table 7
Comparing X-TFC and Deep-TFC on the 1-D Burgers' Equation.

	X-TFC			Deep-TFC		
ν	Maximum Error	Mean Error	Training Time	Maximum Error	Mean Error	Training Time
1.0	1.79×10^{-6}	4.67×10^{-7}	5.40	1.32×10^{-3}	4.30×10^{-4}	15.0
0.5	1.87×10^{-3}	3.95×10^{-4}	5.59	1.66×10^{-3}	5.48×10^{-4}	19.8
0.1	1.38	1.69×10^{-1}	5.48	3.90×10^{-3}	6.24×10^{-4}	81.5

tion function used was the hyperbolic tangent [41]. The X-TFC solution used 200 neurons, and the hyperbolic tangent as the nonlinear activation function: up to 300 neurons were tried, which nearly exceeded the available RAM on the authors' computers during training, but this did not improve the solution over the case with 200 neurons.

A quantitative comparison of the error of the two methods with respect to the steady-state Poiseuille flow solution at $x = 15$ and $t = 3$ is shown in Table 8. In addition, a qualitative comparisons is illustrated in Figs. 14–19: Figs. 14–16 correspond to the X-TFC solution, and Figs. 17–19 correspond to the Deep-TFC solution.

Table 8 shows that in terms of error the Deep-TFC solution is approximately an order of magnitude better than X-TFC. This is reflected qualitatively in the figures as well.

Table 8
Comparison of maximum and mean test set error for X-TFC and Deep-TFC.

Method	Test Set Maximum Error	Test Set Average Error
X-TFC	4.02×10^{-3}	1.89×10^{-3}
Deep-TFC	5.38×10^{-4}	3.12×10^{-4}

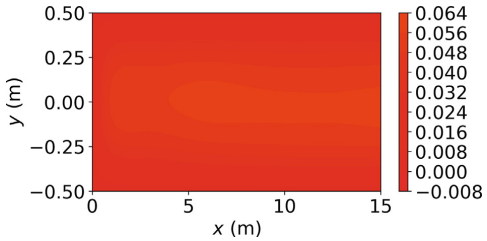


Fig. 14. X-TFC solution at $t = 0.01$.

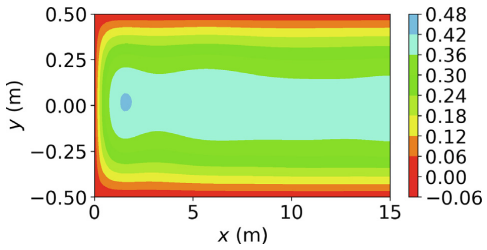


Fig. 15. X-TFC solution at $t = 0.1$.

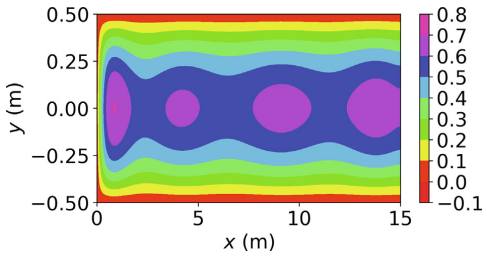


Fig. 16. Deep-TFC solution at $t = 3.0$.

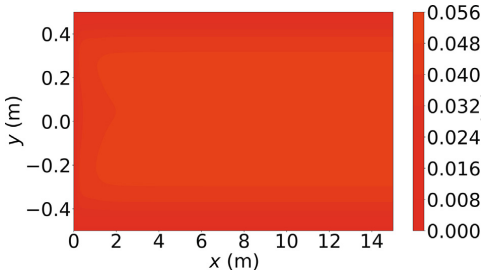


Fig. 17. Deep-TFC solution at $t = 0.01$.

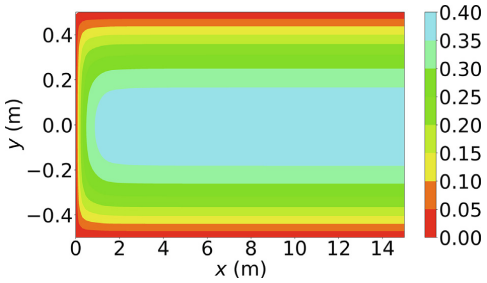


Fig. 18. Deep-TFC solution at $t = 0.1$.

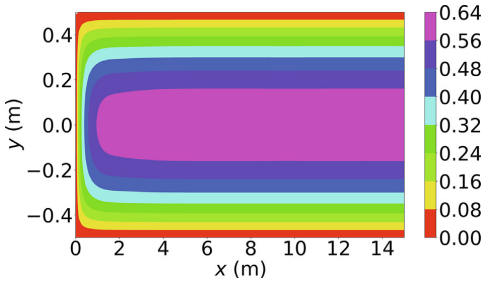


Fig. 19. Deep-TFC solution at $t = 3.0$.

The Deep-TFC figures, Figs. 17–19, match the qualitative expectation given earlier. In contrast, the figures for the X-TFC solutions do not: this difference is highlighted most in the figures for $t = 3.0$. In summation, this Navier–Stokes example demonstrates the utility of Deep-TFC as problems become sufficiently complex.

4. Conclusions

In this paper, a novel, accurate, fast, and robust physics-informed NN method for solving problems involving ODEs and PDEs called the *Extreme Theory of Functional Connection* (X-TFC) was developed. As with the Deep-TFC framework, X-TFC is a synergy of the standard TFC method developed by Mortari et al. [4–6], where the latent solution of the DE is approximated by a constrained expression, which analytically satisfies the constraints while maintaining a free-function that can be used to satisfy the DE, and the classic physics-informed neural network (PINN) meth-

ods proposed by Raissi et al. [7,39], where the free-function is chosen as a NN. In X-TFC, the NN used is not a DNN like in the Deep-TFC framework, but a shallow NN, that is trained based on the ELM learning algorithm proposed by Huang et al. [32]. Although we focused on the solution of *exact problems* (e.g. problems where the modeling error is negligible), X-TFC can also handle *data-driven solutions* and *data-driven parameters discovery* of DEs [61].

The results presented in this manuscript show that the proposed physics-informed NN method can solve several types of exact problems involving DEs with high accuracy and low computational time. For linear and nonlinear (ODEs) and systems of ODEs (SODEs) the proposed framework achieves machine level accuracy in milliseconds. This makes the X-TFC method as accurate or more accurate than all other state-of-the-art methods. Importantly, X-TFC extends the classical TFC with similar performance in terms of accuracy and computational time for higher-dimensional PDEs where the direct application of classical TFC is virtually impossible because it suffers from the curse of dimensionality. For linear and nonlinear bi-dimensional PDEs, the results achieved by the X-TFC framework are comparable with state-of-the-art methods in terms of speed and outperforms state-of-the-art methods in terms of accuracy by four to 11 orders of magnitude. Furthermore, X-TFC was tested on higher dimensional problems such as a 2D time-dependent nonlinear PDE and a 3D time-dependent nonlinear PDE. These problems showed that while the method's accuracy and computational time are affected by the increase in the number of dimensions, the method still generates high-quality solutions with relatively low computational time.

In fact, X-TFC is well suited for a variety of applications, even real-time applications that require accurate and fast solutions, such as optimal control problems in aerospace applications. To this end, the authors are currently applying X-TFC to optimal control problems such as energy optimal landing on planetary bodies, minimum time orbit transfer, maximum radius orbit transfer.

The authors are also working in the application of X-TFC to the solution of several problems arising in transport theory such as a series of Rarefied-Gas Dynamics problems such as Poiseuille Flow [62] and Thermal Creep Flow in a plane channel [63], and 3D time dependent Radiative Transfer problems [64], to name a few.

Moreover, future work will focus on the application of the X-TFC framework to the solution of ruin probabilities of some risk models such as the ones presented in [65–67], as well as series of elliptic and parabolic PDEs like the ones proposed in [68,69].

Clearly, there is still room to further improve the performance and capability of this new physics-informed NN method, as showed by the Burgers Navier–Stokes problems. For these cases, the authors showed how Deep-TFC complements X-TFC. However, the authors are currently investigating the possibility to use new activation functions in addition to the well known logistic, hyperbolic tangent, sinusoid, and Gaussian. Currently under investigation is the use of different probability distributions to sample input weights and biases, in addition to the uniform and normal distributions. Future work will attempt to create a Distributed X-TFC framework similar to the one proposed in Ref. [24] and/or to adapt different sampling of the training points, as proposed by Lu et al. [27]. The goal will be to target problems where the latent solution contains piece-wise continuous behaviour, sharp derivatives, and/or more complex such as the Navier–Stokes problem. In addition to exploring these areas further, future work will also focus on the extension of X-TFC to multi-dimensional problems with non-rectangular domains [7,24,70–72].

Finally, efforts are in progress to provide theoretical justification and establish a mathematical foundation for the proposed physics-informed X-TFC framework. Indeed, as previously mentioned, despite their empirical success, there is still little theoretical justification for the physics-informed NN methods.

CRediT authorship contribution statement

Enrico Schiassi: Methodology, Investigation, Software, Writing - original draft. **Roberto Furfaro:** Methodology, Supervision, Conceptualization, Investigation, Writing - review & editing. **Carl Leake:** Methodology, Investigation, Software, Writing - review & editing. **Mario De Florio:** Methodology, Software, Writing - review & editing. **Hunter Johnston:** Methodology, Software. **Daniele Mortari:** Supervision, Writing - review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was partially supported by a NASA Space Technology Research Fellowship, Leake [NSTRF 2019] Grant #: 80NSSC19K1152 and Johnston [NSTRF 2019] Grant #: 80NSSC19K1149.

In addition, the authors would like to acknowledge Professor Barry D. Ganapol and Mr. Andrea D'Ambrosio for their precious advise that helped to improve this manuscript, and for suggesting a few remote sensing and aerospace applications, where X-TFC can be tested, as future work topics.

Appendix A. Constrained Expression Derivation

This section gives a step-by-step derivation of multivariate TFC constrained expressions, which in general are $n+1$ dimensional as they depends on the vector $\mathbf{x}$ as previously defined. As mentioned earlier, the multivariate form of the constrained expression is written as follows,

$$f_{CE}(\mathbf{x}, g(\mathbf{x})) = \underbrace{\mathscr{M}_{i_1,i_2,\cdots,i_{n+1}}(c(\mathbf{x}))v_{i_1}(t)v_{i_2}(x_1)\cdots v_{i_{n+1}}(x_n)}_{A(\mathbf{x})} + \underbrace{g(\mathbf{x}) - \mathscr{M}_{i_1,i_2,\cdots,i_{n+1}}(g(\mathbf{x}))v_{i_1}(t)v_{i_2}(x_1)\cdots v_{i_{n+1}}(x_n)}_{B(\mathbf{x},g(\mathbf{x}))}$$

where $\mathbf{x} = \{t, x_1, \cdots, x_n\}^{\mathsf{T}}$ is a vector of the $n+1$ independent variables, $\mathscr{M}$ is an $(n+1)^{th}$ order tensor containing the boundary conditions $c(\mathbf{x})$, $v_{i_1}, \cdots, v_{i_{n+1}}$ are vectors whose elements are functions of the independent variables, $g(\mathbf{x})$ is the free-function that can be any function that is defined at the constraints [4]. According to Ref. [41], $A(\mathbf{x})$ analytically satisfies the constraints and $B(\mathbf{x}, g(\mathbf{x}))$ projects the free-function $g(\mathbf{x})$ onto the space of functions that vanish at the constraints. As already explained, in the X-TFC method, the free-chosen function is an ELM. A mathematical proof that this form of the constrained expression always analytically satisfies the constraints is given in Ref. [43].

Some preliminary mathematical notation is defined here that will assist in the derivation of the constrained expression. For consistency, this article uses the same notation as Ref. [41]. Let $k \in [1, n+1]$ be the index that refers to the k^{th} independent variable. Let ${}^{k}c_q^d := \frac{\partial^d c(\mathbf{x})}{\partial x_k^d}|_{x_k=q}$ be the constraint defined by taking the d^{th} order derivative of the constraint function $c(\mathbf{x})$ and evaluating the result at the $x_k = q$ hyperplane. Moreover, let ${}^{k}c_{\mathbf{q}_k}^{\mathbf{d}_k}$ be the vector of ℓ_k constraints defined at the $x_k = \mathbf{q}_k$ hyperplanes with derivative orders of $\mathbf{d}_k$, where $\mathbf{q}_k$ and $\mathbf{d}_k \in \mathbb{R}^{\ell_k}$. Finally, let ${}^{k}b_q^d$ denote the boundary condition operator, where,

$${}^{k}b_{q}^{d}[f(\mathbf{x})] = \frac{\partial^{d} f(\mathbf{x})}{\partial x_{k}^{d}}|_{x_k=q}.$$

This operator takes the d^{th} order derivative with respect to x_k of a function, and then evaluates the result at the $x_k = q$ hyperplane.

Now, the step-by-step process for building the constrained expressions is shown, beginning with the $\mathscr{M}$ tensor. The easiest way to explain this derivation is via an example. As in [41], the construction of this $\mathscr{M}$ tensor will be presented via a 3D example with Dirichlet boundary conditions in x_1 and initial conditions in x_2, and x_3 on the domain $x_1, x_2, x_3 \in [0,1] \times [0,1] \times [0,1]$. The $\mathscr{M}$ tensor is constructed in three steps.

First, the element $\mathscr{M}_{111} = 0$. Secondly, the first order sub-tensors of $\mathscr{M}$ specified by keeping one dimension's index free and setting all other dimension's indices to 1 consists of the value 0 and the boundary conditions for that dimension. That is,

$$\mathscr{M}_{1,\dots,1,i_k,1,\dots,1} = \left\{ 0, {}^{k}c_{\mathbf{q}_k}^{\mathbf{d}_k} \right\}. \tag{A.1}$$

For the example considered here, these first-order sub-tensors are,

$$\mathscr{M}_{i_1 11} = [0, c(0,x_2,x_3), c(1,x_2,x_3)]^{\mathrm{T}}$$
$$\mathscr{M}_{1 i_2 1} = [0, c(x_1,0,x_3), c_{x_2}(x_1,0,x_3)]^{\mathrm{T}}$$
$$\mathscr{M}_{11 i_3} = [0, c(x_1,x_2,0), c_{x_3}(x_1,x_2,0)]^{\mathrm{T}}.$$

Third, the remaining elements of the $\mathscr{M}$ tensor are those with at least two indices that are not equal to one. These elements are the geometric intersection of the boundary condition elements of the first order tensors given in Eq. (A.1), plus a sign (+ or −) that is determined by the number of elements being intersected. In general, this can be formally written as follows,

$$\mathscr{M}_{i_1 i_2 \dots i_{n+1}} = {}^{1}b_{\mathbf{q}_{i_1-1}^{1}}^{\mathbf{d}_{i_1-1}^{1}} \left[{}^{2}b_{\mathbf{q}_{i_2-1}^{2}}^{\mathbf{d}_{i_2-1}^{2}} \left[\dots \left[{}^{n+1}b_{\mathbf{q}_{i_n}^{n+1}}^{\mathbf{d}_{i_n}^{n+1}} [c(\mathbf{x})] \right] \dots \right] \right] (-1)^{m+1},$$

where m is the number of indices for the element that are not equal to one. Using the example constraints, some of these remaining elements are,

$$M_{133} = -c_{x_2 x_3}(x_1,0,0)$$
$$M_{221} = -c(0,0,x_3)$$
$$M_{332} = c_{x_2}(1,0,0).$$

Combining these steps results in the full $\mathscr{M}$ tensor; for the example constraints, the full $\mathscr{M}$ tensor is,

$$\mathscr{M}_{ij1} = \begin{bmatrix} 0 & c(0,x_2,x_3) & c(1,x_2,x_3) \\ c(x_1,0,x_3) & -c(0,0,x_3) & -c(1,0,x_3) \\ c_{x_2}(x_1,0,x_3) & -c_{x_2}(0,0,x_3) & -c_{x_2}(1,0,x_3) \end{bmatrix}$$
$$\mathscr{M}_{ij2} = \begin{bmatrix} c(x_1,x_2,0) & -c(0,x_2,0) & -c(1,x_2,0) \\ -c(x_1,0,0) & c(0,0,0) & c(1,0,0) \\ -c_{x_2}(x_1,0,0) & c_{x_2}(0,0,0) & c_{x_2}(1,0,0) \end{bmatrix}$$
$$\mathscr{M}_{ij3} = \begin{bmatrix} c_{x_3}(x_1,x_2,0) & -c_{x_3}(0,x_2,0) & -c_{x_3}(1,x_2,0) \\ -c_{x_3}(x_1,0,0) & c_{x_3}(0,0,0) & c_{x_3}(1,0,0) \\ -c_{x_2 x_3}(x_1,0,0) & c_{x_2 x_3}(0,0,0) & c_{x_2 x_3}(1,0,0) \end{bmatrix}$$

A standard step-by-step procedure also exists for constructing the v_{i_k} vectors. The general form of v_{i_k} vectors is given by,

$$v_{i_k} = \left\{ 1, \quad \sum_{i=1}^{\ell_k} \alpha_{i1} h_i(x_k), \quad \sum_{i=1}^{\ell_k} \alpha_{i2} h_i(x_k), \quad \dots, \quad \sum_{i=1}^{\ell_k} \alpha_{i\ell_k} h_i(x_k) \right\}^{\mathrm{T}},$$

where $h_i(x_k)$ can be any ℓ_k linearly independent functions that produce a nonsingular matrix in Eq. (A.2). As suggested in Refs. [4,41,43], the simplest set of linearly independent functions are monomials. That is,

$$h_i(x_k) = x_k^{i-1}$$

Finally, the $\ell_k \times \ell_k$ coefficients α_{ij} are simply computed via matrix inversion,

$$\begin{bmatrix} {}^{k}b_{q_1}^{d_1}[h_1] & {}^{k}b_{q_1}^{d_1}[h_2] & \dots & {}^{k}b_{q_1}^{d_1}[h_{\ell_k}] \\ {}^{k}b_{q_2}^{d_2}[h_1] & {}^{k}b_{q_2}^{d_2}[h_2] & \dots & {}^{k}b_{q_2}^{d_2}[h_{\ell_k}] \\ \vdots & \vdots & \ddots & \vdots \\ {}^{k}b_{q_{\ell_k}}^{d_{\ell_k}}[h_1] & {}^{k}b_{q_{\ell_k}}^{d_{\ell_k}}[h_2] & \dots & {}^{k}b_{q_{\ell_k}}^{d_{\ell_k}}[h_{\ell_k}] \end{bmatrix} \begin{bmatrix} \alpha_{11} & \alpha_{12} & \dots & \alpha_{1\ell_k} \\ \alpha_{21} & \alpha_{22} & \dots & \alpha_{2\ell_k} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{\ell_k 1} & \alpha_{\ell_k 2} & \dots & \alpha_{\ell_k \ell_k} \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}. \tag{A.2}$$

The interested reader can refer to [43,41] for further details, a mathematical proof that this procedure for generating the $\mathscr{M}$ tensor and the $\mathbf{v}$ vectors produces a valid constrained expression, and for more examples.

Appendix B. Extreme learning machine algorithm

According to the physics-informed method introduced in this article, the free-chosen function in the TFC constrained expression is chosen to be an ELM. The ELM used in this method is based on the ELM algorithm proposed by Huang et al. [32]. ELM is a learning algorithm for Single-hidden Layer Feed-forward Networks (SLFNs) that randomly selects hidden input weights and biases, and computes the output weights via least-squares. That is, input weights and biases are selected randomly and not tuned during the training. Thus, the output weights can be computed by least-squares. Consider N distinct training samples, $[\mathbf{x}_i, \mathbf{y}_i]_{i=1}^{N}$, where $\mathbf{x}_i = [x_{i1}, \dots, x_{in}]^{\mathrm{T}} \in \mathbb{R}^n$ and $\mathbf{y}_i = [y_{i1}, \dots, y_{im}]^{\mathrm{T}} \in \mathbb{R}^m$, were a standard SLFN with L hidden neurons and activation function $\sigma(\cdot)$ are used as follows,

$$\mathbf{y}_i = \sum_{j=1}^{L} \boldsymbol{\beta}_j \sigma\left(\mathbf{w}_j^{\mathrm{T}} \mathbf{x}_i + b_j\right), i = 1, \dots, N$$

where $\mathbf{w}_j = [w_{j1}, \dots, w_{jn}]^{\mathrm{T}} \in \mathbb{R}^n$ is the input weight vector connecting the j^{th} hidden neuron and the input nodes, $\boldsymbol{\beta}_j = [\beta_{j1}, \dots, \beta_{jm}]^{\mathrm{T}} \in \mathbb{R}^m$ is the output weight vector connecting the j^{th} hidden neuron and the output nodes, and b_j is the bias of the j^{th} hidden neuron. The N equations above can be rewritten in following compact form,

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{Y}, \tag{B.1}$$

where $\mathbf{H} \in \mathbb{R}^{N \times L}$ where $H_{ij} = \sigma\left(\mathbf{w}_j^{\mathrm{T}} \mathbf{x}_i + b_j\right)$ with $i = 1, \dots, N$ and $j = 1, \dots, L$, $\boldsymbol{\beta} \in \mathbb{R}^{L \times m}$ where $\boldsymbol{\beta}_i = \boldsymbol{\beta}_i^{\mathrm{T}}$ and $i = 1, \dots, L$, and $\mathbf{Y} \in \mathbb{R}^{N \times m}$ where $\mathbf{Y}_i = \mathbf{y}_i^{\mathrm{T}}$ and $i = 1, \dots, N$. As the input weights and biases of the ELM are not tuned, the only unknowns in Eq. (B.1) are $\boldsymbol{\beta}$. Thus, (B.1) reduces to a least-squares problem. In Ref. [32], $\boldsymbol{\beta}$ is computed as follows,

$$\boldsymbol{\beta} = \mathbf{H}^{\dagger}\mathbf{Y}$$

where $\mathbf{H}^{\dagger}$ is the Moore–Penrose generalized inverse of the matrix $\mathbf{H}$, which is computed via singular value decomposition (SVD) [32].

The ELM algorithm is based on Theorems 2.1 and 2.2 of [32], which state,

Theorem 4.1. *Given a standard single layer feed forward neural network with L hidden neurons and activation function σ which is infinitely differentiable in any interval, for N arbitrary distinct samples $[\mathbf{x}_i, \mathbf{y}_i]_{i=1}^N$, where $\mathbf{x}_i = [x_{i1}, \dots, x_{in}]^T \in \mathbb{R}^n$ and $\mathbf{y}_i = [y_{i1}, \dots, y_{im}]^T \in \mathbb{R}^m$, for any $\mathbf{w}_j = [w_{j1}, \dots, w_{jn}]^T \in \mathbb{R}^n$ and $b_j \in \mathbb{R}$ randomly chosen according to any continuous probability distribution, then with probability one, the hidden layer output matrix $\mathbf{H}$ of the neural network is invertible and $||\mathbf{HB} - \mathbf{Y}|| = 0$.*

Theorem 4.2. *Given any small positive value δ and activation function σ which is infinitely differentiable in any interval, there exists $L \leqslant N$ such that for N arbitrary distinct samples $[\mathbf{x}_i, \mathbf{y}_i]_{i=1}^N$, where $\mathbf{x}_i = [x_{i1}, \dots, x_{in}]^T \in \mathbb{R}^n$ and $\mathbf{y}_i = [y_{i1}, \dots, y_{im}]^T \in \mathbb{R}^m$, for any $\mathbf{w}_j = [w_{j1}, \dots, w_{jn}]^T \in \mathbb{R}^n$ and $b_j \in \mathbb{R}$ randomly chosen according to any continuous probability distribution, then with probability one,*

$||\mathbf{H}_{[N\times L]}\mathbf{B}_{[L\times m]} - \mathbf{Y}_{[N\times m]}|| < \delta$.

In [32], the interested reader can find the formal proofs of these two theorems.

References

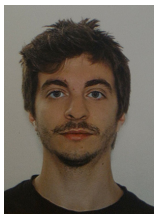
[1] J. Dormand, P. Prince, A Family of Embedded Runge-Kutta Formulae, J. Comp. Appl. Math. 6 (1980) 19–26.
[2] J.L. Junkins, A.B. Younes, R. Woollands, X. Bai, Picard Iteration, Chebyshev Polynomials, and Chebyshev Picard Methods: Application in Astrodynamics, J. Astronautical Sci. 60 (December 2015) 623–653.
[3] D. Gottlieb, S.A. Orszag, Numerical Analysis of Spectral Methods: Theory and Applications, Society Ind. Appl. Math. (1977).
[4] D. Mortari, "The Theory of Connections: Connecting Points," Mathematics, vol. 5, no. 57, 2017.
[5] D. Mortari, Least-squares Solution of Linear Differential Equations, Mathematics 5 (48) (2017) 1–18.
[6] D. Mortari, H. Johnston, L. Smith, High accuracy least-squares solutions of nonlinear differential equations, J. Comput. Appl. Math. 352 (2019) 293–307.
[7] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[8] A. Gil, J. Segura, N. Temme, Numerical Methods for Special Functions. Society Ind. Appl. Math., 1 2007.
[9] C. Lanczos, Applied Analysis, Dover Publications Inc, New York, 1957.
[10] R. Furfaro, D. Mortari, Least-squares solution of a class of optimal space guidance problems via Theory of Connections, Acta Astronaut. (2019).
[11] H. Johnston, E. Schiassi, R. Furfaro, D. Mortari, "Fuel-Efficient Powered Descent Guidance on Large Planetary Bodies via Theory of Functional Connections," The Journal of the Astronautical Sciences, under review.
[12] H. Johnston, D. Mortari, "Linear Differential Equations Subject to Multivalued, Relative and/or Integral Constraints with Comparisons to Chebfun," SIAM Journal of Numerical Analysis, 2018. Submitted.
[13] M. De Florio, E. Schiassi, R. Furfaro, B.D. Ganapol, D. Mostacci, Solutions of chandrasekhar's basic problem in radiative transfer via theory of functional connections, J. Quant. Spectrosc. Radiat. Transfer (2020) 107384.
[14] R. Garcia, C. Siewert, Benchmark results in radiative transfer, Transport Theory Stat. Phys. 14 (4) (1985) 437–483.
[15] B.D. Ganapol, The response matrix discrete ordinates solution to the 1D radiative transfer equation, J. Quant. Spectrosc. Radiat. Transfer 154 (2015) 72–90.
[16] J.N. Reddy, "An Introduction to the Finite Element Method," J. Pressure Vessel Technology, vol. 111, pp. 348–349, 08 1989.
[17] J. Argyris, S. Kelsey, Energy Theorems and Structural Analysis: A Generalized Discourse with Applications on Energy Principles of Structural Analysis Including the Effects of Temperature and Non-Linear Stress-Strain Relations, Aircraft Eng. Aerospace Technol. 26 (10) (1954) 347–356.
[18] M.J. Turner, R.W. Clough, H.C. Martin, L.J. Topp, "Stiffness and Deflection Analysis of Complex Structures," J. Aeronautical Sci., vol. 23, pp. 805–823, sep 1956.
[19] R.W. Clough, The finite element method in plane stress analysis, Am. Soc. Civil Eng. (1960).
[20] I.E. Lagaris, A. Likas, D.I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Networks 9 (Sept 1998) 987–1000.
[21] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control, Signals Syst. 2 (4) (1989) 303–314.
[22] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural Networks 4 (2) (1991) 251–257.
[23] G.-B. Huang, L. Chen, C.-K. Siew, Universal approximation using incremental constructive feedforward networks with random hidden nodes, IEEE Trans. Neural Networks 17 (4) (2006) 879–892.
[24] V. Dwivedi, B. Srinivasan, Physics informed extreme learning machine (pielm)–a rapid method for the numerical solution of partial differential equations, Neurocomputing 391 (2020) 96–118.
[25] T. Chen, H. Chen, Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems, IEEE Trans. Neural Networks 6 (4) (1995) 911–917.
[26] A. Pinkus, Approximation theory of the mlp model in neural networks, Acta numerica 8 (1999) 143–195.
[27] L. Lu, X. Meng, Z. Mao, G.E. Karniadakis, "DeepXDE: A deep learning library for solving differential equations," arXiv preprint arXiv:1907.04502, 2019.
[28] J. Sirignano, K. Spiliopoulos, Dgm: A deep learning algorithm for solving partial dif- ferential equations, J. Comput. Phys. 375 (2018) 1339–1364.
[29] P. Grohs, F. Hornung, A. Jentzen, P.V. Wurstemberger, "A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of black-scholes partial differential equations," arXiv preprint arXiv:1809.02362, 2018.
[30] J. Darbon, T. Meng, "On some neural network architectures that can represent viscosity solutions of certain high dimensional hamilton–jacobi partial differential equations," arXiv preprint arXiv:2002.09750, 2020.
[31] Y. Yang, M. Hou, J. Luo, A novel improved extreme learning machine algorithm in solving ordinary differential equations by Legendre neural network methods, Adv. Diff. Eq. 2018 (1) (2018) 469.
[32] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: Theory and applications, Neurocomputing 70 (May 2006) 489–501.
[33] Y. Shin, J. Darbon, G.E. Karniadakis, "On the convergence and generalization of physics-informed neural networks," arXiv preprint arXiv:2004.01806v1, 2020.
[34] G. Pang, L. Lu, G.E. Karniadakis, Fpinns: Fractional physics-informed neural networks, SIAM J. Sci. Computing 41 (2019) A2603–A2626.
[35] F. Song, G. Pange, C. Meneveau, G.E. Karniadakis, "Fractional physical-inform neural networks (fpinns) for turbulent flows," Bull. Am. Phys. Soc., 2019.
[36] D. Zhang, L. Guo, G.E. Karniadakis, "Learning in modal space: Solving time-dependent stochastic pdes using physics-informed neural networks," arXiv preprint arXiv:1905.01205, 2019.
[37] Z. Mao, A.D. Jagtap, G.E. Karniadakis, Physics-informed neural networks for high-speed flows, Comput. Methods Appl. Mech. Eng. 360 (2020) 112789.
[38] M. Raissi, A. Yazdani, G.E. Karniadakis, Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations, Science 367 (2020) 1026–1030.
[39] J. Sirignano, K. Spiliopoulos, "DGM: A deep learning algorithm for solving partial differential equations," September 2018.
[40] S.A. Coons, "Surfaces for computer-aided design of space forms," tech. rep., Massachusetts Institute of Technology, Cambridge, MA, USA, 1967.
[41] C. Leake, D. Mortari, Deep Theory of Functional Connections: A New Method for Estimating the Solutions of Partial Differential Equations, Machine Learn. Know. Extraction 2 (1) (2020) 37–55.
[42] C. Leake, H. Johnston, D. Mortari, The multivariate theory of functional connections: Theory, proofs, and application in partial differential equations, Mathematics 8 (8) (2020) 1303.
[43] D. Mortari, C. Leake, The Multivariate Theory of Connections, Mathematics 7 (3) (2019) 296.
[44] C. Leake, D. Mortari, "An Explanation and Implementation of Multivariate Theory of Connections via Examples," in 2019 AAS/AIAA Astrodynamics Specialist Conference, Portland, MN, August 11–15, 2019, AAS/AIAA, 2019.
[45] C. Leake, H. Johnston, L. Smith, D. Mortari, Analytically Embedding Differential Equation Constraints into Least Squares Support Vector Machines Using the Theory of Functional Connections, Mach. Learning Know. Extraction 1 (Oct. 2019) 1058–1083.
[46] R.C. Aster, B. Borchers, C.H. Thurber, Parameter estimation and inverse problems, Elsevier, 2018.
[47] S. Rogers, M. Girolami, A first course in machine learning, CRC Press, 2016.
[48] E. Schiassi, R. Furfaro, J.S. Kargel, C.S. Watson, D.H. Shugar, U.K. Haritashya, "GLAM Bio-Lith RT: A Tool for Remote Sensing Reflectance Simulation and Water Components Concentration Retrieval in Glacial Lakes," Frontiers in Earth Science, vol. 7, 2019.
[49] E. Schiassi, R. Furfaro, D. Mostacci, Bayesian inversion of coupled radiative and heat transfer models for asteroid regoliths and lakes, Radiat. Eff. Defects Solids 171 (9–10) (2016) 736–745.
[50] B. Hapke, Bidirectional reflectance spectroscopy: 1. Theory, J. Geophys. Res.: Solid Earth 86 (B4) (1981) 3039–3054.
[51] B. Hapke, A model of radiative and conductive energy transfer in planetary regoliths, J. Geophys. Res.: Planets 101 (E7) (1996) 16817–16831.
[52] A.S. Hale, B. Hapke, A time-dependent model of radiative and conductive thermal energy transport in planetary regoliths with applications to the Moon and Mercury, Icarus 156 (2) (2002) 318–334.
[53] F. Jacob, B. Ted, A first course in finite elements, Wiley, 2007.
[54] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind, "Automatic differentiation in machine learning: a survey," 2015.
[55] S. Carroll, Construction of neural networks using the radon transform, IEEE International Conference on Neural Networks, vol. 1, IEEE, 1989, pp. 607–611.
[56] T. Chen, H. Chen, Approximations of continuous functionals by neural networks with application to dynamic systems, IEEE Trans. Neural Networks 4 (6) (1993) 910–918.
[57] L. Lu, P. Jin, G.E. Karniadakis, "Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators," arXiv preprint arXiv:1910.03193, 2019.

[58] H. Sun, M. Hou, Y. Yang, T. Zhang, F. Weng, F. Han, Solving Partial Differential Equation Based on Bernstein Neural Network and Extreme Learning Machine Algorithm, Neural Process. Lett. 50 (2) (2019) 1153–1172.
[59] D. Maclaurin, D. Duvenaud, M. Johnson, J. Townsend, "Autograd." URL:https://github.com/HIPS/autograd, 2013.
[60] S. Mall, S. Chakraverty, Single Layer Chebyshev Neural Network Model for Solving Elliptic Partial Differential Equations, Neural Process. Lett. 45 (3) (2017) 825–840.
[61] E. Schiassi, A. D'Ambrosio, M. De Florio, R. Furfaro, F. Curti, "Physics-informed extreme theory of functional connections applied to data-driven parameters discovery of epidemiological compartmental models," arXiv preprint arXiv:2008.05554, 2020.
[62] M. De Florio, E. Schiassi, R. Furfaro, "Physics-Informed Neural Networks for Rarefied-Gas Dynamics: Poiseuille Flow in the Bhatnagar–Gross–Krook approximation," In preparation, 2021.
[63] M. De Florio, E. Schiassi, B.D. Ganapol, R. Furfaro, Physics-informed neural networks for rarefied-gas dynamics: Thermal creep flow in the Bhatnagar–Gross–Krook approximation, Physics of Fluids 33 (4) (2021) 047110.
[64] S. Mishra, R. Molinaro, Physics informed neural networks for simulating radiative transfer, Journal of Quantitative Spectroscopy and Radiative Transfer 270 (2021) 107705.
[65] T. Zhou, X. Liu, M. Hou, C. Liu, Numerical solution for ruin probability of continuous time model based on neural network algorithm, Neurocomputing 331 (2019) 67–76.
[66] Y. Lu, G. Chen, Q. Yin, H. Sun, M. Hou, Solving the ruin probabilities of some risk models with legendre neural network algorithm, Digital Signal Processing 99 (2020) 102634.
[67] Y. Chen, C. Yi, X. Xie, M. Hou, Y. Cheng, Solution of ruin probability for continuous time model based on block trigonometric exponential neural network, Symmetry 12 (6) (2020) 876.
[68] Y. Yang, M. Hou, J. Luo, Z. Tian, Numerical solution of several kinds of differential equations using block neural network method with improved extreme learning machine algorithm, J. Intell. Fuzzy Systems (2020) 1–17.
[69] Y. Yang, M. Hou, H. Sun, T. Zhang, F. Weng, J. Luo, Neural network algorithm based on legendre improved extreme learning machine for solving elliptic partial differential equations, Soft. Comput. 24 (2) (2020) 1083–1096.
[70] J. Berg, N. Kaj, A unified deep artificial neural network approach to partial differential equations in complex geometries, Neurocomputing 317 (2018) 28–41.
[71] L. Isaac E., L. Aristidis C., P. Dimitris G., "Neural-network methods for boundary value problems with irregular boundaries.," IEEE Transactions on Neural Networks, vol. 11, no. 5, pp. 1041–1049, 2000.
[72] D. Mortari, D. Arnas, Bijective mapping analysis to extend the theory of functional connections to non-rectangular 2-dimensional domains, Mathematics 8 (9) (2020) 1593.

Enrico Schiassi earned his B.Sc. degree in Energy Engineering from the Alma Mater Studiorum – University of Bologna, Italy, in 2013, and his M.Sc. degree in Energy and Nuclear Engineering in 2016 from the same University. He started a Ph.D. in System Engineering at The University of Arizona in 2017. His main research interest is the application of machine learning to several fields such as Physics-Informed solutions of space guidance problems and general optimal control problems, remote sensing for Earth systems characterization and space exploration, and Nuclear Reactor Dynamics.

Roberto Furfaro earned his M.Sc. in Aerospace Engineering at Sapienza University of Rome, Italy, in 1998 and his Ph.D. in Aerospace Engineering, at The University of Arizona in 2004. He is currently Professor at the Department of Systems and Industrial Engineering, Department of Aerospace and Mechanical Engineering, University of Arizona. He is the Director of the Space Systems Engineering Laboratory (SSEL), the Director of the Space Situational Awareness Arizona (SSA-Arizona) Initiative, and currently the PI of the AFRL Cooperative Agreement. He is a technical member of the AIAA Astrodynamics Committee and of the AAS Space Surveillance Committee. In 2010–2016, he was the systems engineering lead for the Science Processing and Operations Center of the NASA OSIRIS REx Asteroid Sample Return Mission.

Carl Leake is an NSTRF fellow and PhD candidate at Texas A&M University working under the advisement of Dr. Daniele Mortari. Carl earned a B.S. in Aerospace Engineering from Embry-Riddle Aeronautical University, Prescott, AZ in 2017, and his research interests include the Theory of Functional Connections, flexible body dynamics, star identification algorithms, and n-dimensional range searching techniques.



Mario De Florio earned his B.Sc. degree in Energy Engineering from the Alma Mater Studiorum - University of Bologna, Italy, in 2016, and his M.Sc. degree in Energy and Nuclear Engineering in 2019 from the same University. He started a Ph.D. in Systems and Industrial Engineering at The University of Arizona in 2020. His main research interest is the application of machine learning to several fields, such as Physics-Informed solutions of Transport Theory Problems, Nuclear Reactor Dynamics, and Remote Sensing for Earth Systems characterization and Space Exploration.

Hunter Johnston earned his B.Sc. in Mechanical Engineering from the University of Dayton in 2017. Following this, he started his Ph.D. in Aerospace Engineering at Texas A&M University working with Dr. Daniele Mortari. His current research interests include the theoretical development of the Theory of Functional Connections and its applications in numerical methods, optimization, and optimal control.

Daniele Mortari is a Professor of Aerospace Engineering at Texas A&M University, working on the field of attitude and position estimation, satellite constellation design, sensor data processing, and various topics in linear algebra and numerical algorithms. In addition, he has taught at the School of Aerospace Engineering of Rome's University, and at Electronic Engineering of Perugia's University, both in Italy. He has been widely recognized for his work, including receiving three NASA's Group Achievement Awards, 2007 IEEE Judith A. Resnik Award, and 2016 AAS Dirk Brouwer Award. He is an IEEE and AAS Fellow, an Honorary Member of IEEE-AESS Space System Technical Panel, and a former IEEE Distinguished Speaker. He has also recently become a Full Member of the International Academy of Astronautics.

Appendix B

Schiassi et al. (2022): X-TFC for Nuclear Reactors Dynamics

Reproduced from Schiassi, E., De Florio, M., Ganapol, B.D., Picca, P. and Furfaro, R., 2022. Physics-informed neural networks for the point kinetics equations for nuclear reactor dynamics. *Annals of Nuclear Energy*, 167, p.108833. https://doi.org/10.1016/j.anucene.2021.108833 [62], with the permission of Elsevier.

Annals of Nuclear Energy 167 (2022) 108833

Contents lists available at ScienceDirect

Annals of Nuclear Energy

journal homepage: www.elsevier.com/locate/anucene

Physics-informed neural networks for the point kinetics equations for nuclear reactor dynamics

Enrico Schiassi [a], Mario De Florio [a], Barry D. Ganapol [b], Paolo Picca [a], Roberto Furfaro [a,b,*]

[a] Department of Systems & Industrial Engineering, The University of Arizona, Tucson, AZ, USA
[b] Department of Aerospace & Mechanical Engineering, The University of Arizona, Tucson, AZ, USA

ARTICLE INFO

Article history:
Received 18 May 2021
Accepted 10 November 2021
Available online 5 December 2021

Keywords:
Physics-informed neural networks
Functional interpolation
Extreme learning machine
Extreme theory of functional connections
Points kinetic equations
Nuclear reactor dynamics

ABSTRACT

The paper presents a novel approach based on Physics-Informed Neural Networks (PINNs) for the solution of Point Kinetics Equations (PKEs) with temperature feedback. The approach is based on a new framework developed by the authors, which combines PINNs with Theory of Functional Connections and Extreme Learning Machines in the so called Extreme Theory of Functional Connections (X-TFC). The accuracy of X-TFC is tested against a number of published benchmarks (including for non-linear PKEs), showing its performance both in terms of accuracy and computational time. One of the main advantages of the proposed framework is in its flexibility to adapt to a variety of problems with minimal changes in coding and, after the training of the network, in its ability to offer an analytical representation (by Neural Networks) of the solution at any desired time instant outside the initial discretization.

1. Introduction

An accurate and fast solution of the reactor kinetic equations is of paramount importance for the simulation of reactor transient in safety analyses. The general solution of the time-dependent problem requires the solution of (or an approximation of) the linear Boltzmann equation coupled with the precursors' equations, which are typically characterised by very different timescales (making stiff differential equations). The neutron-precursors equations then needs to be coupled with the temperature distribution through the thermal feedback, introducing non-linearity into the model. In order to make the problem computationally manageable, simplifications of the reactor kinetic equations have been developed in the past. A well-known approximation is the Point Kinetics Equations (PKEs), firstly derived by Henry (Henry, 1958) and still in use in many applications of existing codes. The basic idea behind PKE is to factorise the solution of the time-dependent neutron-precursor balance equations in amplitude (function of the time only) and shape (function of space, energy and angle), and project the equations onto a weight, to reduce the mathematical problem from a system of integro-differential equations to a system of ODEs. More refined approximations of the reactor kinetics equations such as quasi static approach (Ott and Meneley, 1969) consider the evolution of the shape on a longer timescale. However the resulting set of differential equations modelling the kinetics of the amplitude in the quasi-static approximation are still in the form of the PKEs. Similarly, other classical methods like multi-point kinetics or modal methods can be reduced to a mathematical problem which is equivalent to the PKEs (Picca et al., 2013).

As the analytical solution of PKEs is only possible in very idealised cases (e.g., step wise insertion of reactivity for the linear probnumerical techniques have been developed to simulate transients of interest in applications. Whilst there ian extensive literature on solution methods for PKEs, the topic is still attracting interest of various research groups, both for improving the efficiency of solution techniques and for deriving highly accurate benchmarks solutions to test commercial codes against. For example, recently in Cai et al. (2016) adapted the Magnus expansion for the solution of the nonlinear PKEs. Leite et al. in Leite et al. (2014) proposed an analytical representation of the solution of the PKEs with an adaptive time step. Leite et al. in Leite et al. (2016), used the same method to solve the PKEs in the integral formulation with an arbitrary number of delayed neutron groups and temperature feedback. In Hamada (2015) and Hamada (2018) Hamada introduced a new method based on Fourier series expansion using adaptive step size to solve the stiff system of the PKEs. In Nahla and Edress (2016), Nahla and Edress developed an analytical exponential model to solve the stochastic PKEs via eigenvalues and eigenvectors. Razak et al., in Razak et al. (2015), devised a modified

* Corresponding author at: Department of Systems & Industrial Engineering, The University of Arizona, Tucson, AZ, USA.
E-mail addresses: eschiassi@email.arizona.edu (E. Schiassi), mariodf@email.arizona.edu (M. De Florio), ganapol@cowboy.ame.arizona.edu (B.D. Ganapol), ppicca@email.arizona.edu (P. Picca), robertof@email.arizona.edu (R. Furfaro).

https://doi.org/10.1016/j.anucene.2021.108833
0306-4549/

exponential time differencing method. Picca et al. in Picca et al. (2013) proposed a method based on piecewise constant approximation to solve the nonlinear PKEs. In Da Silva et al. (2014), Wollmann da Silva et al. suggested removing the stiffness by splitting the matrix into a diagonal matrix plus a matrix containing the remaining terms. Hamada, in Hamada (2011) and Hamada (2013), introduced a solution of the PKEs based on the power series method both with constant and adaptive step integrations. In Hamada (2018), Hamada proposed a new method based on shifted Chebyshev series of the first kind (SCS) to solve stiff linear/nonlinear systems of PKEs, where constant step size to divide the time domain is employed. Finally, the two most accurate solution methods are the Converged Accelerated Taylor Series (CATS) Ganapol et al. (2012) and the Backward Euler Difference scheme (BEFD) (Ganapol, 2013), both proposed by Ganapol (et al.).

In the recent past, new methods to solve differential equations were developed within the Machine Learning community. A class of solution methods are called Physics-Informed Neural Networks (PINN). The basic idea behind these methods is to train Neural Networks (NNs) to derive the solution of Differential Equations (DE). More generally, PINNs are not only employed to solve DEs and can be generally considered as machine learning algorithms that embed physics into a data-driven representation of functional relationships underlying collections of input–output pairs. As described by Raissi et al. (2019), the term PINN defines NNs that use the physics as a regularizer in the loss function, which is used to define the optimisation problem equivalent to the solution of the original DEs. For example, let's assume that we want to perform a regression of a set of experimental data (that represent physical phenomena modeled via DEs) using a NN. In standard regression, an approximation of the data using a shallow or deep NN trained to minimize a Mean Squared Error (MSE) as the loss function can be used.. However, there is no guarantee that the physics phenomena governing the data would not be violated, especially when the data are noisy and over-fitting issues can arise. In these cases, there is a risk that the data would be accurately fitted (e.g., a negligible error between the NN approximation and the experimental data), but the relation between input–output, awared by the physics phenomena, could be poorly represented. PINNs are introduced to ensure that DEs, coming from first principles and modeling the experimental dataset's physics, are added as a penalty to the loss function. This additional term acts as a regularizer that penalizes the training when the DE and its constraints (e.g., Initial Conditions ICs and/or Boundary Conditions BCs) are violated. Overall, this approach, often referred to as data-driven solution of DEs, ensures that the physics underlying the process is not violated. In particular, from the physics perspective, this approach enables NNs to learn the solution of DEs in a data-physics-driven fashion and becomes particularly important when the DEs do not accurately describe the physics of the problem (e.g., for uncertain dynamical systems or when perturbations are present). When data are unavailable, and therefore the loss function only contains the residual of the DEs and its constraints, PINNs are used to find the solutions of problems involving Ordinary Differential Equations (ODEs) or Partial Differential Equations (PDEs) and solely in a physics-driven fashion.

This paper aims to develop and evaluate a PINNs-based methodology to solve linear and nonlinear PKEs, with and without temperature feedback. For the purpose, a particular PINN framework developed by the authors is implemented. This method combines NNs, and the Theory of Functional Connections (TFC) (Leake and Mortari, 2020; Schiassi et al., 2020), resulting in a PINN-TFC based method. TFC, developed by Mortari (2017), is a functional interpolation technique that derives functionals, called Constrained Expressions (CEs), containing a free function and always satisfying a set of specified linear constraints. In the proposed framework, shallow NNs are used as free functions within the TFC's CEs and trained using Extreme Learning Machine (ELM) algorithm (Huang et al., May 2006). The advantage of ELM algorithm is that input weights and bias are randomly selected and the training is only necessary for the output weights, with a significant reduction in the computational effort. The resulting approach, called Extreme Theory of Functional Connections (X-TFC) (Schiassi et al., 2020), aims to overcome the main limitations of the standard PINN and TFC methods, e.g. in terms of solution accuracy and curse of dimensionality. Among its many applications, TFC has been used to solve ODEs with great results, leading to machine-level accuracy solutions with very competitive computational time (Mortari, 2017; Mortari et al., 2019; De Florio et al., 2021). Recent examples of successful application of this framework includes its implementation for the solution of Chandrasekhar's basic problem in radiative transfer (De Florio et al., 2020), rarefied-gas dynamics problems (De Florio et al., 2021; De Florio et al., 1999), and optimal control problems for aerospace applications (Furfaro and Mortari, 2019; Johnston et al., 2020; Schiassi et al., 2020; Drozd et al., 2021).

The manuscript is organized as follows. In Section 2, the general PINN methodology is presented, clarifying how it can be efficiently combined with TFC and ELMs to obtain highly accurate solutions. In Section 3, the general form of PKEs with temperature feedback is introduced. In Section 4, the accuracy of the proposed PINN-based method is tested against several methods and benchmarks from literature both for linear and non-linear problems. Conclusions are drawn in Section 5.

2. Background on the solution method based on Physics-informed neural networks

PINNs have been proposed by Raissi et al. (2019), following the original idea introduced by Lagaris et al. (Sept 1998), to solve two main classes of problems: data-driven solution and data-driven discovery of DEs. In Raissi et al. (2019) PINN were introduced to tackle problems governed by Partial DEs (PDEs), but the same framework can be employed to solve problems involving ODEs. By contrast to classical method for the solution of DEs like Finite Difference Method and Finite Element Method (Reddy, 1989), the PINN approach identifies a loss (cost) function based on the physics of the problem and aims at minimizing the error on the solution by determining the weights and biases parameters in the NN.

The path taken in this work is to approach the initial value problem represented by the PKEs using a particular PINN framework named X-TFC. X-TFC is a PINN method that approximates the DE solutions using particular functionals, called CEs, defined within the TFC framework (Mortari, 2017). The basic principles of the application of this approach to first-order ODEs governing the reactor kinetics equations are presented below, starting from the PINN approach for solving first order ODEs (Section 2.1) and then continuing with the basic idea behind the X-TFC (Section 2.2). For further details on PINN and X-TFC implementation additional details are presented in Raissi et al. (2019), Lu et al. (2019) and Schiassi et al. (2020).

2.1. PINN to solving first order ODEs

Consider solving the following generic first-order initial value problem, that can be either linear or nonlinear:

$$F(t, y, \dot{y}) = 0 \quad \text{subjectto}: \quad y(t_0) = y_0 \tag{1}$$

The basic idea behind the PINN is to approximate the solution y of (1) by means of a NN (Lu et al., 2019), i.e..

$$y(t) \approx y_{NN}(t, \Theta) \tag{2}$$

where Θ are the NN hyperparameters (e.g., weights and biases) to be determined during training. Its derivative can be evaluated using automatic differentiation (Baydin et al., 2015; Lu et al., 2019) and formally written as:

$$\dot{y}(t) \approx \dot{y}_{NN}(t,\Theta) = \frac{d}{dt} y_{NN}(t,\Theta) \tag{3}$$

By substituting (2) and (3) into Eq. (1), one obtains the residuals:

$$\mathscr{R}_F(t) = F(t, y_{NN}, \dot{y}_{NN}) \tag{4}$$

It is worth noting that a residue must also be evaluated for the initial conditions

$$\mathscr{R}_{IC}(t) = y_{NN}(t_0,\Theta) - y_0 \tag{5}$$

where $y_{NN}(t_0,\Theta)$ is the NN evaluated on $t = t_0$. The loss function can be written as the sum of the contributions of the two residues in the form of a mean square error (MSE), i.e..

$$MSE = MSE_F + MSE_{IC} \tag{6}$$

where,

$$MSE_F = \frac{1}{N_F}\sum_{i=1}^{N_F} \mathscr{R}_F^2(t_i) \tag{7}$$

and

$$MSE_{IC} = \mathscr{R}_{IC}^2(t_0) \tag{8}$$

In the definitions above, N_F are the training points, and t_i is the independent variable computed in the *i-th* training point. In general, the training points can be sampled from any probability distribution, be equidistant, or sampled from a particular quadrature scheme (e.g., Legendre, Gauss, Gauss–Legendre, etc.) (Mishra et al., 2020).

The NN hyperparameters Θ in (2) are derived (or "learned" in the terminology of the Machine Learning community) by minimizing the MSE loss with gradient-based methods, such as stochastic gradient descent (e.g., Adam optimizer).

One of the major drawbacks of the standard PINN frameworks is that the DE constraints (here initial condition in (1)) are not analytically satisfied, and during the PINN training there are competing objectives: learning the DE solution within the domain, and satisfying the initial condition. This leads to unbalanced gradients during the network training via gradient-based methods that causes PINNs to often have difficulties to accurately approximate the solution of DE (Wang et al., 2020). Indeed, it is well known that gradient-based methods may get stuck in limit cycles or even diverge if multiple competing objectives are present (Mertikopoulos et al., 2018; Balduzzi et al., 2018). In Wang et al. (2020), to overcome this issue, the authors developed a learning rate annealing algorithm that employs gradient statistics to adaptively 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. The approach used here is different and is based on the TFC (Mortari, 2017) which allows to analytically satisfy the initial conditions, hence removing the competing objective in the NN training. Next section provides an overview of TFC and the proposed learning algorithm.

2.2. X-TFC to solving first order ODEs

As previously mentioned, the authors develop an alternative PINN framework, named X-TFC. In the proposed approach, we combine the standard PINN with the TFC method (Leake and Mortari, 2020; Schiassi et al., 2020).

Here, we present the step-by-step procedure for solving generic first-order initial value problems, that can be either linear or nonlinear, using the proposed PINN-TFC based method.

Consider solving the same first order initial value problem of Eq. (1). The unknown solution is approximated via the constrained expression, defined within the TFC framework (Mortari, 2017). That is:

$$y(t) = g(t) + \sum_{k=1}^{n} \eta_k p_k(t) = g(t) + \boldsymbol{\eta}^T \mathbf{p}(t) \tag{9}$$

where, $p_k(t)$ are n assigned linearly independent functions and the $g(t)$ is the free function, which must be linearly independent from the $p_k(t)$ functions. The values of the η_k coefficients are calculated by imposing a set of n linear constraints in $y(t)$. By doing so, the constraints for the differential equations are satisfied for any chosen $g(t)$.

For a first order ODE, the CE is the following:

$$y(t) = g(t) + \eta\, p(t) \tag{10}$$

where $p(t) = 1$ (see Mortari (2017) and Mortari et al. (2019) for the details of this decision). Thus the CE is now:

$$y(t) = g(t) + \eta \tag{11}$$

By enforcing the equation constraint, we get:

$$y(t_0) = y_0 = g_0 + \eta \qquad \Rightarrow \qquad \eta = y_0 - g_0 \tag{12}$$

which is plugged into Eq. (11) to get:

$$y(t) = g(t) + (y_0 - g_0) \tag{13}$$

which represents all possible functions satisfying the initial value constraint. The derivative then follows:

$$\dot{y}(t) = \dot{g}(t) \tag{14}$$

The CE defined by Eq. (13) and its derivative are plugged into our ODE which then becomes:

$$F(t, y, \dot{y}) = 0 \tag{15}$$

that is now an initial condition free ODE. By substituting Eq. (13) into Eq. (15), the differential equation F is transformed into a new differential equation $\widetilde{F}$, which is only a function of the independent variable t and the free-function $g(t)$, that is:

$$\widetilde{F}(t, g, \dot{g}) = 0 \tag{16}$$

This differential equation is now *unconstrained* and will always satisfy the initial-value due to Eq. (11). In this work, we choose the free function $g(t)$ to be a shallow NN. That is:

$$g(t) = \sum_{j=1}^{L} \beta_j \sigma(w_j t + b_j) = \begin{bmatrix} \sigma_1 \\ \vdots \\ \sigma_L \end{bmatrix}^T \boldsymbol{\beta} = \boldsymbol{\sigma}^T \boldsymbol{\beta} \tag{17}$$

where L is the number of hidden neurons, $w_j \in \mathbb{R}$ is the input weights vector connecting the jth hidden neuron and the input nodes, $\beta_j \in \mathbb{R}$ with $j = 1, \ldots, L$ is the jth output weight connecting the jth hidden neuron and the output node, and b_j is the bias of the jth hidden neuron, and $\sigma_j(\cdot)$ are activation functions.

The classical approach for NN is to train all of the parameters in Eq. 17. In the following, the ELM algorithm is used to train the network (hence the name of the algorithm X-TFC). According to the ELM algorithm Huang et al. (May 2006), input weights and biases are randomly selected and not tuned during the training, thus they are known parameters. The activation functions, $\sigma_j(\cdot)$, are chosen by the user, so they are also known. Therefore, the only unknowns to compute are the output weights $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_L]^T$. The interested

reader can find more details about the ELM and its convergence proofs in Huang et al. (May 2006). For the convenience of the reader, in Fig. (1) we show a representation of a general shallow NN used in this paper. It may happen that some activation functions are defined on an inconsistent domain $x \in [x_0, x_f]$. Thus, our independent variable $t \in [t_0, t_f]$ needs to be mapped in the x domain. Generally, to have better training performance, regardless of the activation function used, it is convenient to map the independent variable $t \in [t_0, t_f]$ in the x domain (usually $x \in [0, +1]$ or $x \in [-1, +1]$). This can be done using the following linear transformation:

$$x = x_0 + c(t - t_0) \qquad \leftrightarrow \qquad t = t_0 + \frac{1}{c}(x - x_0)$$

where c is the mapping coefficient:

$$c = \frac{x_f - x_0}{t_f - t_0}$$

By the derivative chain rule, the n^{th} derivative of Eq. (17) is,

$$\frac{\mathrm{d}^n g(t)}{\mathrm{d}t^n} = c^n \boldsymbol{\beta}^T \frac{\mathrm{d}^n \boldsymbol{\sigma}(x)}{\mathrm{d}x^n}, \tag{18}$$

which defines all mappings of the free-function. To determine the unknown parameters, i.e. β, the domain t must be discretized in n training points and then apply *unconstrained* optimization techniques to solve the following problem:

$$\widetilde{F}(\boldsymbol{\beta}) = 0 \tag{19}$$

For linear DEs, Eq. (19) is a system of linear algebraic equations with unknown output weights $\boldsymbol{\beta}$. By imposing $\mathscr{L}(\boldsymbol{x}, \boldsymbol{\beta})$ to be equal to $\boldsymbol{0}$, the terms in Eq. (19) can be rearranged to form a system of linear algebraic equations. This is possible as the ELM algorithm does not train the NN's input weights and biased, leaving the output weights $\boldsymbol{\beta}$ the only hyperparameters to be tuned. That is,

$$A\boldsymbol{\beta} = \boldsymbol{b} \tag{20}$$

which can solved via any least-squares technique (Mortari, 2017). For instance, using a classic least-squares, we have,

$$\boldsymbol{\beta} = (ATA)^{-1} AT\boldsymbol{b} \tag{21}$$

For nonlinear DEs, an iterative least-squares approach is used (Mortari et al., 2019). According to the iterative procedure, the output weights are updated at each iteration as follows,

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k + \Delta\boldsymbol{\beta}_k \tag{22}$$

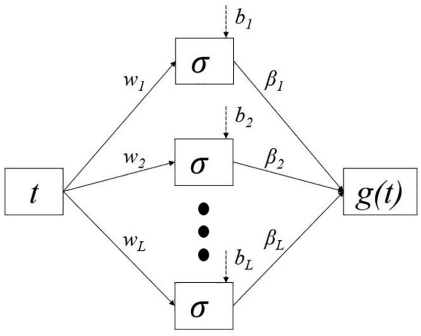


Fig. 1. Example of a shallow NN, trained via ELM, used as the free-function in the X-TFC method.

where the k subscript refers to the current iteration. In general, the $\Delta\boldsymbol{\beta}_k$ term can be evaluated by performing linear least-squares at each iteration of the iterative least-square procedure. For instance, if a classic linear least-square is used, then,

$$\Delta\boldsymbol{\beta}_k = -(\mathbb{J}\mathrm{T}(\boldsymbol{\beta}_k)\mathbb{J}(\boldsymbol{\beta}_k))^{-1}\mathbb{J}\mathrm{T}(\boldsymbol{\beta}_k)\mathscr{L}(\boldsymbol{\beta}_k) \tag{23}$$

where $\mathbb{J}$ is the Jacobian matrix containing the derivatives of the loss with respect to the output weights $\boldsymbol{\beta}$. One may compute the Jacobian analytically or by means of automatic differentiation. The iterative procedure is repeated until the following conditions is met,

$$L_2[\mathscr{L}(\boldsymbol{\beta}_k)] < \epsilon$$

where ϵ is some user prescribed tolerance.

A schematic that summarizes how X-TFC works for first order linear and nonlinear initial value problems is showed in Fig. 2 Throughout the body of the paper, the $\boldsymbol{\beta}$ output weights will be identified as ξ. This decision was made to keep the traditional notation in the physics of nuclear reactors of the precursor yield fraction $\boldsymbol{\beta}$, defined as:

$$\beta = \frac{\text{precursor atoms}}{\text{prompt neutrons} + \text{precursor atoms}} \tag{24}$$

3. The point kinetics equations with temperature feedback

The Point Kinetics Equations (Duderstadt and Hamilton, 1976) with temperature feedback and no neutron source with G delayed neutron groups, the system of equations takes the following form:

$$\begin{cases} \frac{dN(t)}{dt} = \left[\frac{\rho(t,N)-\beta}{\Lambda}\right]N(t) + \sum_{i=1}^{G}\lambda_i C_i(t) \\ \frac{dC_i(t)}{dt} = \frac{\beta_i}{\Lambda}N(t) - \lambda_i C_i(t) \\ \frac{dT(t)}{dt} = E(t)N(t) + U(t)T(t) \\ \rho(t) = F(t,N,T) \end{cases} \tag{25}$$

Where $N(t)\left[\#/cm^3\right]$ is the neutron flux, $C_i(t)\left[\#/cm^3\right]$ is the precursor concentration in the i^{th} group (with $i = 1, \ldots, G$), β_i is the i^{th} group delayed fraction (with $\beta = \sum_{i=1}^{G}\beta_i$), $\lambda_i\left[s^{-1}\right]$ is the i^{th} group decay constant, $\Lambda[s]$ is the neutron generation time, $\rho(t)[]isthereactivity(with\rho/\beta = [])$, and $T(t)[^o\ C]$ is the temperature of the reactor. The functions $E(t)$ and $U(t)$ can be both constants or functions of the independent variable. Likewise, $F(t, N, T)$ can be a constant, or a linear or nonlinear function of the time, neutron density, and reactor temperature. The system of Eqs. (25) must be solved subject to the following initial conditions:

$$\begin{cases} N(0) = N_0 \\ C_{0i}(0) = \frac{N_0\beta_i}{\Lambda\lambda_i} \\ T(0) = T_0 \\ \rho(0) = \rho_0 \end{cases} \tag{26}$$

For the convenience of the reader, the all process described in this section is summarized in the schematic of Fig. 3. In this paper we are facing problems in nuclear reactor dynamics with a relatively long time domain. Thus, to avoid a carryover of the error during the operation of the reactor, a subdivision of the time interval is made. As one can see in Fig. 4, the domain is divided into n time steps of equal size $h = t_i - t_{i-1}$, with $i = 1, \ldots, n$. Starting from the time step 1, we use the initial conditions given by the problem at time t_0. The time step is discretized into n_x points. By solving the DEs, we find the solutions at time t_1. These solutions become the initial constraints for the time step 2, and so on. Thus, for each time step X-TFC is applied recursively up to the last time step of the total time interval of operation of the reactor.

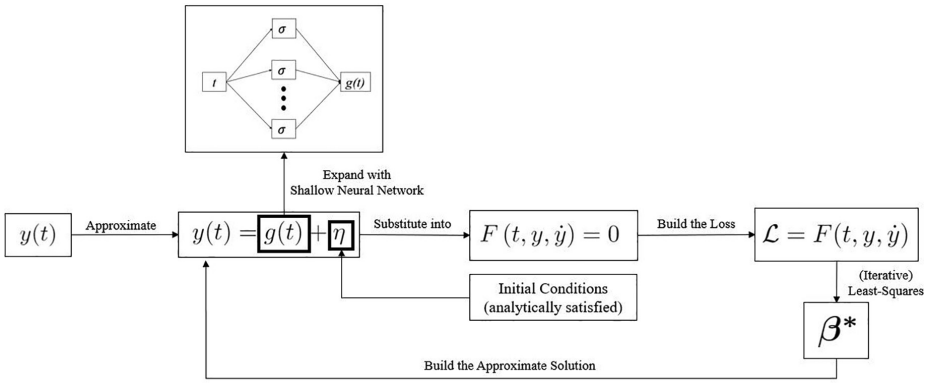


Fig. 2. Schematic of the general PINN-TFC Based framework to solve linear ODEs with one constraint in one point.

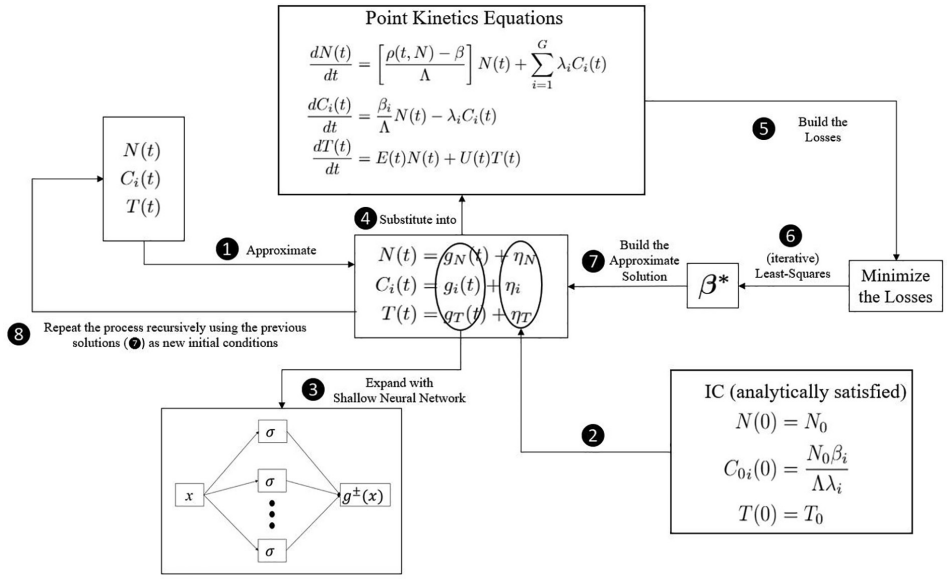


Fig. 3. Schematic of the X-TFC algorithm to solve the Point Kinetics Equations.

4. Results and discussions

In this section six test cases are analyzed, in which different conditions of reactivity, temperature feedback, and number of delayed neutron groups are considered. The parameters for the delayed neutron groups are reported in Table 1 according to Nóbrega da (1971). All the problems tackled in this manuscript have been coded in Matlab R2020a, and run with an Intel Core i7 - 9700 CPU PC with 64 GB of RAM.

In the X-TFC framework, several hyperparameters can be modified to obtain accurate solutions. These hyperparameters are the number of training points, n, the number of neurons, L, the type of activation function, and the probability distribution where input weights and bias are sampled. In Schiassi et al. (2020), an analysis was performed to study the X-TFC method's sensitivity to these hyperparameters. This analysis showed that, for the problems considered, the solution accuracy is not as sensitive to the type of activation function used and/or to the probability distribution used to sample the inputs weights and bias as it is to the number of training points and the number of neurons.

4.1. Test Case 1 – step reactivity insertion in thermal reactor

The firs test case is for step reactivity insertion in a thermal reactor with six delayed neutron groups. The reactivity is considered constant in time (e.g., $\rho(t) = \rho_0$) and there is no thermal feedback. The system of Eqs. (25) becomes the following:

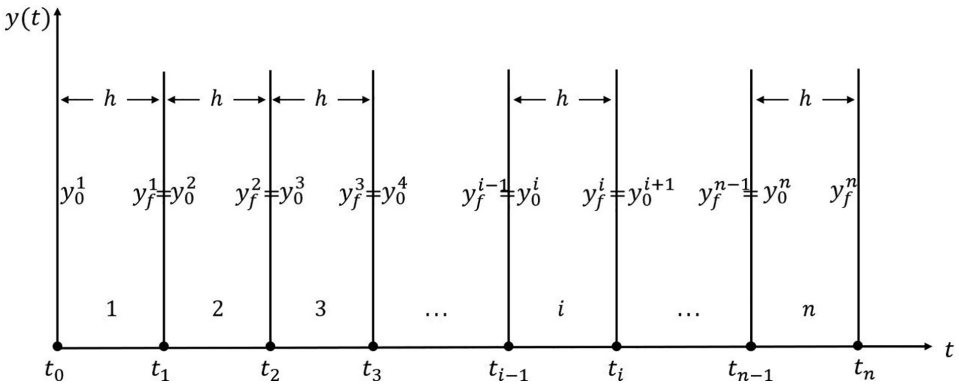


Fig. 4. Division of time domain in subintervals of length h.

Table 1
Six precursor group parameters for the test cases considered.

	Test Cases 1,2		Test Cases 4		Test Cases 5		Test Cases 6	
i	$\lambda_i[s^{-1}]$	β_i	$\lambda_i[s^{-1}]$	β_i	$\lambda_i[s^{-1}]$	β_i	$\lambda_i[s^{-1}]$	β_i
1	0.0127	2.6600E−4	0.0124	2.150E−4	0.0124	2.10E−4	0.0124	2.20E−4
2	0.0317	1.4910E−3	0.0305	1.424E−3	0.0305	1.41E−3	0.0305	1.42E−3
3	0.1150	1.3160E−3	0.1110	1.274E−3	0.1110	1.27E−3	0.1110	1.27E−3
4	0.3110	2.8490E−3	0.3010	2.568E−3	0.3010	2.55E−3	0.3010	2.57E−3
5	1.4000	8.9600E−4	1.1400	7.480E−4	1.1300	7.40E−4	1.1400	7.50E−4
6	3.8700	1.8200E−4	3.0100	2.730E−4	3.0000	2.70E−4	3.0100	2.70E−4
	$\beta = 0.007$ $\Lambda = 0.00002s$ $N_0 = 1$		$\beta = 0.006502$ $\Lambda = 0.0005s$ $N_0 = 1$		$\beta = 0.00654$ $\Lambda = 0.00005s$ $N_0 = 1$		$\beta = 0.0065$ $\Lambda = 0.00005s$ $N_0 = 0.01$	

$$\begin{cases} \frac{dN(t)}{dt} = \frac{\rho_0-\beta}{\Lambda}N(t) + \sum_{i=1}^{6}\lambda_i C_i(t) \\ \frac{dC_i(t)}{dt} = \frac{\beta_i}{\Lambda}N(t) - \lambda_i C_i(t) \end{cases} \tag{27}$$

$$\text{subject to : } \begin{cases} N(0) = N_0 \\ C_i(0) = C_{i0} = \frac{N_0\beta_i}{\Lambda\lambda_i} \end{cases}$$

The constrained expressions are:

$$N(t) = (\boldsymbol{\sigma} - \boldsymbol{\sigma}_0)\xi_N + N_0 \tag{28}$$

$$C_i(t) = (\boldsymbol{\sigma} - \boldsymbol{\sigma}_0)\xi_i + C_{i0} \qquad \forall i = 1,\dots,6 \tag{29}$$

$$\dot{N}(t) = c\boldsymbol{\sigma}'\xi_N \tag{30}$$

$$\dot{C}_i(t) = c\boldsymbol{\sigma}'\xi_i \qquad \forall i = 1,\dots,6 \tag{31}$$

which are placed into problem (27), that then becomes unconstrained:

$$\begin{cases} \left[c\boldsymbol{\sigma}' - \frac{\rho_0-\beta}{\Lambda}(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0)\right]\xi_N - \sum_{i=1}^{6}\lambda_i(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0)\xi_i = \frac{\rho_0-\beta}{\Lambda}N_0 + \sum_{i=1}^{6}\lambda_i C_{i0} \\ -\frac{\beta_i}{\Lambda}(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0)\xi_N + [c\boldsymbol{\sigma}' + \lambda_i(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0)]\xi_i = \frac{\beta_i}{\Lambda}N_0 - \lambda_i C_{i0} \end{cases} \tag{32}$$

Thus, the problem reduces to the following linear system of algebraic equations in the form $Ax = b$:

$$\begin{bmatrix} c\boldsymbol{\sigma}' - \frac{\rho_0-\beta}{\Lambda}(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) & -\lambda_1(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) & -\lambda_2(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) & \dots & -\lambda_6(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) \\ -\frac{\beta_1}{\Lambda}(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) & c\boldsymbol{\sigma}' + \lambda_1(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) & \mathbf{0} & \dots & \mathbf{0} \\ -\frac{\beta_2}{\Lambda}(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) & \mathbf{0} & c\boldsymbol{\sigma}' + \lambda_2(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) & \dots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -\frac{\beta_6}{\Lambda}(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) & \mathbf{0} & \mathbf{0} & \dots & c\boldsymbol{\sigma}' + \lambda_6(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) \end{bmatrix}$$

$$\begin{bmatrix} \xi_N \\ \xi_1 \\ \xi_2 \\ \vdots \\ \xi_6 \end{bmatrix} = \begin{bmatrix} \frac{\rho_0-\beta}{\Lambda}N_0 + \sum_{i=1}^{6}\lambda_i C_{i0} \\ \frac{\beta_1}{\Lambda}N_0 - \lambda_1 C_{10} \\ \frac{\beta_2}{\Lambda}N_0 - \lambda_2 C_{20} \\ \vdots \\ \frac{\beta_6}{\Lambda}N_0 - \lambda_6 C_{60} \end{bmatrix} \tag{33}$$

Finally, the NN's weights ξ are learned via Least-Squares method. By replacing them in the constrained expression (28), we obtain the neutron flux over all time interval of interest.

This first test case has been chosen to compare our numerical solutions with the SCS method (Hamada, 2018), and the analytical solutions available. The neutron group parameters used for this case are reported in Table 1, according to Nóbrega da (1971) for a thermal reactor.

Our results, compared with SCS method, BEFD method, and analytical solutions are reported in Table 2, and are obtained by using the parameters in Table 3. The training time is expressed in seconds and it is referred to one time step iteration. This test case has been computed for three different values of reactivity, each of

Table 2
Comparison of the X-TFC, SCS, and analytical solutions for Test Case 1 of step reactivity insertions.

ρ	$t(s)$	X-TFC	SCS	BEFD	Analytical
0.003	0.20	1.851268287511E+00	1.851268287511E+00	1.851268287511E+00	1.851268287511E+00
	0.40	1.947593411379E+00	1.947593411379E+00	1.947593411379E+00	1.947593411379E+00
	0.60	2.037922055925E+00	2.037922055925E+00	2.037922055925E+00	2.037922055925E+00
	0.80	2.124831636423E+00	2.124831636423E+00	2.124831636423E+00	2.124831636423E+00
	1.00	2.209840456983E+00	2.209840456983E+00	2.209840456983E+00	2.209840456983E+00
	5.00	4.111945003689E+00	4.111945003689E+00	4.111945003689E+00	4.111945003689E+00
	10.0	8.019199973227E+00	8.019199973227E+00	8.019199973227E+00	8.019199973227E+00
	20.0	2.829739978101E+01	2.829739978101E+01	2.829739978101E+01	2.829739978101E+01
	30.0	9.774957522848E+01	9.774957522848E+01	9.774957522848E+01	9.774957522848E+01
	40.0	3.364929691090E+02	3.364929691090E+02	3.364929691090E+02	3.364929691090E+02
	50.0	1.157602990957E+03	1.157602990957E+03	1.157602990957E+03	1.157602990957E+03
	100	5.573863220602E+05	5.573863220602E+05	5.573863220602E+05	5.573863220602E+05
0.007	0.01	4.508858486352E+00	4.508858486352E+00	4.508858486352E+00	4.508858486352E+00
	0.05	1.961041160367E+01	1.961041160367E+01	1.961041160367E+01	1.961041160367E+01
	0.10	4.522304458456E+01	4.522304458456E+01	4.522304458456E+01	4.522304458456E+01
	0.20	1.597257698629E+02	1.597257698629E+02	1.597257698629E+02	1.597257698629E+02
	0.30	5.188964648308E+02	5.188964648308E+02	5.188964648308E+02	5.188964648308E+02
	0.40	1.667287692551E+03	1.667287692551E+03	1.667287692551E+03	1.667287692551E+03
	0.50	5.345887612045E+03	5.345887612045E+03	5.345887612045E+03	5.345887612045E+03
	1.00	1.805731634228E+06	1.805731634228E+06	1.805731634228E+06	1.805731634228E+06
	5.00	3.053491380444E+26	3.053491380444E+26	3.053491380444E+26	3.053491380444E+26
	10.0	5.888108082190E+51	5.888108082190E+51	5.888108082190E+51	5.888108082190E+51
	50.0	1.125662952084E+254	1.125662952084E+254	1.125662952084E+254	1.125662952084E+254
0.008	0.01	6.202853575094E+00	6.202853575094E+00	6.202853575094E+00	6.202853575094E+00
	0.05	9.564079408383E+01	9.564079408383E+01	9.564079408383E+01	9.564079408383E+01
	0.10	1.410421803591E+03	1.410421803591E+03	1.410421803591E+03	1.410421803591E+03
	0.20	2.779627082462E+05	2.779627082462E+05	2.779627082462E+05	2.779627082462E+05
	0.30	5.460343932944E+07	5.460343932944E+07	5.460343932944E+07	5.460343932944E+07
	0.40	1.072625143566E+10	1.072625143566E+10	1.072625143566E+10	1.072625143566E+10
	0.50	2.107055258349E+12	2.107055258349E+12	2.107055258349E+12	2.107055258349E+12
	1.00	6.163333749908E+23	6.163333749908E+23	6.163333749908E+23	6.163333749908E+23
	5.00	3.303191632502E+115	3.303191632502E+115	3.303191632502E+115	3.303191632502E+115
	10.0	1.514716564142E+230	1.514716564142E+230	1.514716564142E+230	1.514716564142E+230

Table 3
n_x is the discretization order for x, L is the number of neurons in the NN, h is the time step, *LB and UB* are the lower and upper bounds of the uniform distribution where input weights and bias are sampled from.

ρ	x range	n_x	L	h	LB	UB	training time [s/step]	total training time [s]
0.003	[−1,1]	30	18	0.01	−1	1	0.001	10
0.007	[−1,1]	50	20	0.01	−1	1	0.0015	7.5
0.008	[−1,1]	100	20	0.01	−1	1	0.002	2

which with a different time interval considered. It can be seen that we are able to achieve all the 12 digits of accuracy according to the analytical solutions, even the three digits in the twelfth place that SCS and BEFD were unable to get, marked in bold. We use a time step $h = 0.01$ with an average training time of $0.003 s/step$, against the time step of $h = 0.001$ with an average training time of $0.0005 s/step$.

4.2. Test Case 2 – ramp reactivity insertion

The second taste case is for a ramp reactivity insertion of $\rho(t) = 0.1\beta t$ with the kinetic parameters given in Table (1) for thermal reactor II, again without thermal feedback.

The system of Eqs. (25) becomes:

$$\begin{cases} \frac{dN(t)}{dt} = \beta\frac{0.1t-1}{\Lambda}N(t) + \sum_{i=1}^{6}\lambda_i C_i(t) \\ \frac{dC_i(t)}{dt} = \frac{\beta_i}{\Lambda}N(t) - \lambda_i C_i(t) \end{cases} \quad (34)$$

$$\text{subjectto}: \quad \begin{cases} N(0) = N_0 \\ C_i(0) = C_{i0} = \frac{N_0\beta_i}{\Lambda\lambda_i} \end{cases}$$

By following the same procedure as in the test case 1, the linear system of algebraic equations $Ax = b$ to be solved is

$$\begin{bmatrix} c\boldsymbol{\sigma}' - \beta\frac{0.1t-1}{\Lambda}(\boldsymbol{\sigma}-\boldsymbol{\sigma}_0) & -\lambda_1(\boldsymbol{\sigma}-\boldsymbol{\sigma}_0) & -\lambda_2(\boldsymbol{\sigma}-\boldsymbol{\sigma}_0) & \dots & -\lambda_6(\boldsymbol{\sigma}-\boldsymbol{\sigma}_0) \\ -\frac{\beta_1}{\Lambda}(\boldsymbol{\sigma}-\boldsymbol{\sigma}_0) & c\boldsymbol{\sigma}' + \lambda_1(\boldsymbol{\sigma}-\boldsymbol{\sigma}_0) & \mathbf{0} & \dots & \mathbf{0} \\ -\frac{\beta_2}{\Lambda}(\boldsymbol{\sigma}-\boldsymbol{\sigma}_0) & \mathbf{0} & c\boldsymbol{\sigma}' + \lambda_2(\boldsymbol{\sigma}-\boldsymbol{\sigma}_0) & \dots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -\frac{\beta_6}{\Lambda}(\boldsymbol{\sigma}-\boldsymbol{\sigma}_0) & \mathbf{0} & \mathbf{0} & \dots & c\boldsymbol{\sigma}' + \lambda_6(\boldsymbol{\sigma}-\boldsymbol{\sigma}_0) \end{bmatrix} \begin{bmatrix} \xi_N \\ \xi_1 \\ \xi_2 \\ \vdots \\ \xi_6 \end{bmatrix} = \begin{bmatrix} \beta\frac{0.1t-1}{\Lambda}N_0 + \sum_{i=1}^{6}\lambda_i C_{i0} \\ \frac{\beta_1}{\Lambda}N_0 - \lambda_1 C_{10} \\ \frac{\beta_2}{\Lambda}N_0 - \lambda_2 C_{20} \\ \vdots \\ \frac{\beta_6}{\Lambda}N_0 - \lambda_6 C_{60} \end{bmatrix} \quad (35)$$

Our results have been compared with benchmarks produced by some of the most accurate methods so far, which are the SCS method (Hamada, 2018), CATS (Ganapol et al., 2012), and BEFD (Ganapol, 2013). Our results reported in Table 4 have been obtained using the following parameters: x-range $= [-1, 1]$, $n_x = 20$, and $L = 15$. It can be seen how all the methods achieve the same accuracy up to the 9th digit. The difference can be noticed for the time step, which is variable for CATS and BEFD methods (with the smallest time calculation per step of 0.0156 s), and $h = 0.01$ for SCS, with a time calculation per step of 0.16

Table 4
Solutions of the X-TFC, SCS, CATS, and BEFD methods for $0.1\beta t$ ramp reactivity (fast reactor).

$t(s)$	X-TFC $h = 0.1$	SCS $h = 0.01$	CATS h = variable	BEFD h = variable
0.00	1.000000000E+00	1.000000000E+00	1.000000000E+00	1.000000000E+00
2.00	1.338200050E+00	1.338200050E+00	1.338200050E+00	1.338200050E+00
4.00	2.228441897E+00	2.228441897E+00	2.228441897E+00	2.228441897E+00
6.00	5.582052449E+00	5.582052449E+00	5.582052449E+00	5.582052449E+00
8.00	4.278629573E+01	4.278629573E+01	4.278629573E+01	4.278629573E+01
10.0	4.511636239E+05	4.511636239E+05	4.511636239E+05	4.511636239E+05
11.0	1.792213607E+16	1.792213607E+16	1.792213607E+16	1.792213607E+16

s. With X-TFC we are able to use a greater step size ($h = 0.1$), with a smaller training time of 0.0007 s per step. Hence the total training time is 0.08 seconds. Furthermore, we present a new 12 digits benchmark for the Ramp Reactivity insertion case, achieved by X-TFC and BEFD (Quadruple Precision (QP)). The results are shown in Table 5, obtained with $h = 0.01, n_x = 20$, and $L = 15$ (training time of 0.0035 s per step). X-TFC overall computational time is 0.4*s*, whereas BEFD computational time is 77.6*s*.

4.3. Test Case 3 – sinusoidal reactivity

For the third case, and last one without thermal feedback, we consider a sinusoidal reactivity insertion $\rho(t) = \rho_0 \sin(\frac{\pi t}{T})$, where $\rho_0 = \sin\frac{8\beta}{8+\lambda T}$, for one delayed group in a fast reactor with $\beta = 0.0079$ and $\lambda = 0.077/s$. The value of the half period is set at $T = 50s$, the initial value of neutron density is $N_0 = 1$, and the value of the initial reactivity is truncated at $\rho_0 = 0.0053333$. We compute the neutron flux density for thermal reactor I (with neutron generation time $\Lambda = 10^{-6}$) and thermal reactor II (with neutron generation time $\Lambda = 10^{-8}$).

The system of Eqs. (25) becomes:

$$\begin{cases} \frac{dN(t)}{dt} = \frac{\rho_0 \sin\left(\frac{\pi t}{T}\right)-\beta}{\Lambda} N(t) + \lambda C(t) \\ \frac{dC(t)}{dt} = \frac{\beta}{\Lambda} N(t) - \lambda C(t) \end{cases} \quad \text{subject to :} \quad \begin{cases} N(0) = N_0 \\ C(0) = C_0 = \frac{N_0\beta}{\Lambda\lambda} \end{cases} \tag{36}$$

Again, following the procedure used in the previous two cases, the linear system of algebraic equations $Ax = b$ to be solved is:

$$\begin{bmatrix} c\boldsymbol{\sigma}' - \frac{\rho_0 \sin\left(\frac{\pi t}{T}\right)-\beta}{\Lambda}(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) & -\lambda(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) \\ -\frac{\beta}{\Lambda}(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) & c\boldsymbol{\sigma}' + \lambda(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) \end{bmatrix} \begin{bmatrix} \xi_N \\ \\ \xi_C \end{bmatrix} = \begin{bmatrix} \frac{\rho_0 \sin\left(\frac{\pi t}{T}\right)-\beta}{\Lambda} N_0 + \lambda C_0 \\ \frac{\beta}{\Lambda} N_0 - \lambda C_0 \end{bmatrix} \tag{37}$$

Our results and their comparison with existing benchmarks are shown in Tables 6 and 7. For fast reactor I a simulation up to $t = 50s$ has been computed, with a step size of $h = 0.01$, while SCS uses $h = 0.0001$. All the 9 digits of accuracy of CATS and EPCA methods have been achieved (except for the last CATS digit at $t = 50s$), however 12 figures have been printed in order to compare them with SCS and BEFD methods' results (differences are marked in bold). These results have been obtained using $n_x = 20$ and $L = 14$, with a training time of 0.0001 s per step. Only X-TFC and BEFD report the benchmark for a reactor activity of 100 s, with overall computational times of 1.1 s and 0.625 s, respectively. In Table 7 the value of neutron flux density for fast reactor II is reported up to $t = 100s$. The results are proven to be accurate up to the 11th digit according to the BEFD benchmark (overall computational time of 1.9 s). Good compatibility has been found with SCS 12 digits, as we can see for the times 70, 80, and 90 s, while the differences are marked in bold. The SCS method required a very small step size (h = 1E−6) consuming about 9.5E−5 *s/step*, while X-TFC required a step size of h = 1E−4 consuming about 3.5E−4 *s/step* (total training time of about 300 s), that involves a great improvement in the computational efforts. However, to reach the 9 digits of accuracy according to BEFD, we can use the same parameters used for fast reactor I, decreasing the computational efforts (around 1.2 seconds).

Table 5
Solutions of the X-TFC, and BEFD methods for $0.1\beta t$ ramp reactivity (moderately fast reactor) 12-digits benchmark.

$t(s)$	X-TFC $h = 0.1$	BEFD h = variable
0.00	1.000000000E+00	1.000000000E+00
2.00	1.338200050049E+00	1.338200050049E+00
4.00	2.228441896810E+00	2.228441896810E+00
6.00	5.582052448674E+00	5.582052448674E+00
8.00	4.278629573112E+01	4.278629573112E+01
10.0	4.511636239090E+05	4.511636239090E+05
11.0	1.792213607343E+16	1.792213607343E+16

The second benchmark aims to evaluate the accuracy of the method in determining the time to the first peak and the peak neutron density for fast reactor II, for four sinusoidal insertions, given in Table 8. It should be emphasized that the BEFD benchmark of Table 8 by Ganapol (2013) was the first true benchmark for peak times and neutron flux densities for a sinusoidal reactivity insertion, that is a case of interest for nearly 40 years. Comparison with the benchmark is appreciable up to the 6th digit, however 9 figures have been printed in order to be tested with further numerical methods. The step size used is $h = 0.0001$, with $n_x = 100$, and $L = 20$.

Fig. 5 shows the trend of the neutron flux densities for four cases of sinusoidal reactivity insertion (by varying T and ρ_0), while Fig. 6 has been plotted to compare our results with those obtained with SCS method for both fast reactors I and II, with a half period $T = 50s$. We can notice a difference only in the last digit of neutron flux density for fast reactor I.

4.4. Test Case 4 – the reactivity is a function of the neutron density

In this fourth case we consider a reactivity that is function of the neutron density, that is $\rho(t) = \pm 0.1\beta N(t)$. For simplicity we call $k = \pm 0.1$.

The system of Eq. (25) becomes:

$$\begin{cases} \frac{dN(t)}{dt} = \frac{k\beta}{\Lambda} N^2(t) - \frac{\beta}{\Lambda} N(t) + \sum_{i=1}^{6} \lambda_i C_i(t) \\ \frac{dC_i(t)}{dt} = \frac{\beta_i}{\Lambda} N(t) - \lambda_i C_i(t) \end{cases} \tag{38}$$

$$\text{subject to :} \quad \begin{cases} N(0) = N_0 \\ C_i(0) = C_{i0} = \frac{N_0\beta_i}{\Lambda\lambda_i} \end{cases}$$

As we are facing a non linear problem, we need to compute the losses and their derivatives:

Table 6
Solutions of the X-TFC, SCS, CATS, and EPCA methods for sinusoidal reactivity of fast reactor I ($\Lambda = 10^{-6}$).

$t(s)$	X-TFC	BEFD	SCS	CATS	EPCA
10	2.065311114128E+00	2.065311114128E+00	2.065311114128E+00	2.065311114128E+00	2.065311114128E+00
20	8.852831171282E+00	8.852831171282E+00	8.852831171282E+00	8.852831171282E+00	8.852831171282E+00
30	4.063328341665E+01	4.063328341665E+01	4.063328341665E+01	4.063328341665E+01	4.063328341665E+01
40	6.134134599167E+01	6.134134599167E+01	6.134134599167E+01	6.134134599167E+01	6.134134599167E+01
50	4.609559630885E+01	4.609559630885E+01	4.609559630885E+01	4.609559630885E+01	4.609559630885E+01
60	2.911965188074E+01	2.911965188074E+01	–	–	–
70	1.894742294991E+01	1.894742294991E+01	–	–	–
80	1.393509265414E+01	1.393509265414E+01	–	–	–
90	1.253063799200E+01	1.253063799200E+01	–	–	–
100	1.544449281926E+01	1.544449281926E+01	–	–	–

Table 7
Solutions of the X-TFC, SCS, and BEFD methods for sinusoidal reactivity of fast reactor II ($\Lambda = 10^{-8}$).

$t(s)$	X-TFC	SCS	BEFD
10	2.065383519140E+00	2.065383519140E+00	2.065383519140E+00
20	8.854133920588E+00	8.854133920588E+00	8.854133920588E+00
30	4.064354221747E+01	4.064354221747E+01	4.064354221747E+01
40	6.135607516739E+01	6.135607516739E+01	6.135607516739E+01
50	4.610628770207E+01	4.610628770207E+01	4.610628770207E+01
60	2.912634839579E+01	2.912634839579E+01	2.912634839579E+01
70	1.895177042303E+01	1.895177042303E+01	1.895177042303E+01
80	1.393829211020E+01	1.393829211020E+01	1.393829211020E+01
90	1.253353405687E+01	1.253353405687E+01	1.253353405687E+01
100	1.544816513659E+01	1.544816513659E+01	1.544816513659E+01

Table 8
Solutions of the X-TFC, SCS, CATS, and EPCA methods for sinusoidal reactivity of fast reactor I $\left(\Lambda = 10^{-6}\right)$.

		X-TFC		BEFD	
$T(s)$	$\rho_0 \times 10^3$	$t_p(s)$	N_p	$t_p(s)$	N_p
50	5.3333	3.910712E+01	6.15301460E+01	3.910712E+01	6.15301460E+01
150	3.2327	1.373198E+02	9.58115007E+01	3.910712E+01	6.15301460E+01
250	2.3193	2.371265E+02	1.13467887E+02	3.910712E+01	6.15301460E+01
350	1.8083	3.370713E+02	1.23820868E+02	3.910712E+01	6.15301460E+01

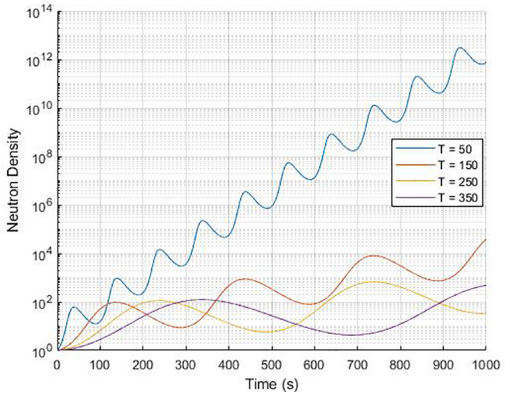


Fig. 5. Neutron density variation for all four cases of sinusoidal reactivity insertion.

$$\mathscr{L}_N = \dot{N} - \frac{k\beta}{\Lambda}N^2 + \frac{\beta}{\Lambda}N - \sum_{i=1}^{6}\lambda_i C_i \qquad \mathscr{L}_i = \dot{C}_i - \frac{\beta_i}{\Lambda}N + \lambda_i C_i \tag{39}$$

$$\frac{\partial \mathscr{L}_N}{\partial \boldsymbol{\xi}_N} = c\boldsymbol{\sigma}' - \frac{2k\beta}{\Lambda}N(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) \qquad \frac{\partial \mathscr{L}_N}{\partial \boldsymbol{\xi}_i} = -\lambda_i(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) \tag{40}$$

$$\frac{\partial \mathscr{L}_i}{\partial \boldsymbol{\xi}_N} = -\frac{\beta_i}{\Lambda}(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) \qquad \frac{\partial \mathscr{L}_i}{\partial \boldsymbol{\xi}_i} = c\boldsymbol{\sigma}' + \lambda_i(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) \tag{41}$$

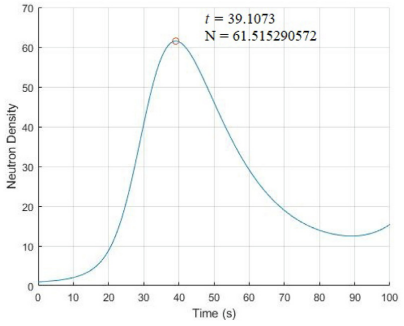
The linear system that we want to iteratively solve is $\mathscr{J}\Delta\xi = \mathscr{L}$, that is:

$$\begin{bmatrix} \frac{\partial \mathscr{L}_N}{\partial \boldsymbol{\xi}_N} & \frac{\partial \mathscr{L}_N}{\partial \boldsymbol{\xi}_1} & \frac{\partial \mathscr{L}_N}{\partial \boldsymbol{\xi}_2} & \frac{\partial \mathscr{L}_N}{\partial \boldsymbol{\xi}_3} & \frac{\partial \mathscr{L}_N}{\partial \boldsymbol{\xi}_4} & \frac{\partial \mathscr{L}_N}{\partial \boldsymbol{\xi}_5} & \frac{\partial \mathscr{L}_N}{\partial \boldsymbol{\xi}_6} \\ \frac{\partial \mathscr{L}_1}{\partial \boldsymbol{\xi}_N} & \frac{\partial \mathscr{L}_1}{\partial \boldsymbol{\xi}_1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \\ \frac{\partial \mathscr{L}_2}{\partial \boldsymbol{\xi}_N} & \mathbf{0} & \frac{\partial \mathscr{L}_2}{\partial \boldsymbol{\xi}_2} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \frac{\partial \mathscr{L}_3}{\partial \boldsymbol{\xi}_N} & \mathbf{0} & \mathbf{0} & \frac{\partial \mathscr{L}_3}{\partial \boldsymbol{\xi}_3} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \frac{\partial \mathscr{L}_4}{\partial \boldsymbol{\xi}_N} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \frac{\partial \mathscr{L}_4}{\partial \boldsymbol{\xi}_4} & \mathbf{0} & \mathbf{0} \\ \frac{\partial \mathscr{L}_5}{\partial \boldsymbol{\xi}_N} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \frac{\partial \mathscr{L}_5}{\partial \boldsymbol{\xi}_5} & \mathbf{0} \\ \frac{\partial \mathscr{L}_6}{\partial \boldsymbol{\xi}_N} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \frac{\partial \mathscr{L}_6}{\partial \boldsymbol{\xi}_6} \end{bmatrix} \begin{bmatrix} \boldsymbol{\Delta\xi}_N \\ \boldsymbol{\Delta\xi}_1 \\ \boldsymbol{\Delta\xi}_2 \\ \boldsymbol{\Delta\xi}_3 \\ \boldsymbol{\Delta\xi}_4 \\ \boldsymbol{\Delta\xi}_5 \\ \boldsymbol{\Delta\xi}_6 \end{bmatrix} = \begin{bmatrix} \mathscr{L}_N \\ \mathscr{L}_1 \\ \mathscr{L}_2 \\ \mathscr{L}_3 \\ \mathscr{L}_4 \\ \mathscr{L}_5 \\ \mathscr{L}_6 \end{bmatrix} \tag{42}$$

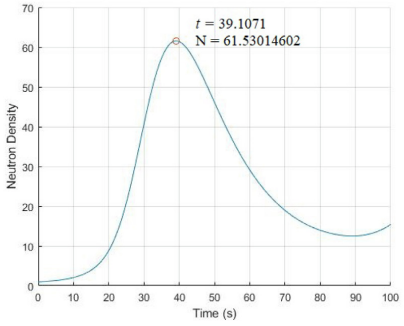
In Table 9, our results are compared with those obtained by SCS, TFS, and CATS methods, where 10 digits of accuracy are matched. However, we propose a new 12-digits benchmark obtained with X-TFC and BEFD, with $n_x = 20$ and $L = 15$. The time step used is $h = 0.1$, which allows us to achieve high accuracy with very small total training time (about 0.0008 seconds/step), reaching the second position for computational time among these methods.

4.5. Test Case 5 – temperature feedback reactivity

The fifth case is for the thermal reactor III, in which the Newtonian temperature feedback reactivity is considered in the point



(a) The time to the neutron peak and the neutron density at that peak for sinusoidal reactivity for fast reactor I with $\Lambda = 10^{-6}$



(b) The time to the neutron peak and the neutron density at that peak for sinusoidal reactivity for fast reactor II with $\Lambda = 10^{-8}$

Fig. 6. (a) The time to the neutron peak and the neutron density at that peak for sinusoidal reactivity for fast reactor I with $\Lambda = 10^{-6}$(b) The time to the neutron peak and the neutron density at that peak for sinusoidal reactivity for fast reactor II with $\Lambda = 10^{-8}$.

Table 9
Solutions of X-TFC, SCS, TFS, BEFD, and CATS methods for the case 4, with the reactivity as a function of neutron flux.

$\rho(\$)$		$\boldsymbol{t}(\boldsymbol{s})$	X-TFC $h = 0.1$	BEFD	TFS $h = 0.05$	SCS $h = 0.1$	CATS
$0.1\beta N(t)$	0.1	1.081784072596	1.081784072596		1.081784072596	1.081784072596	1.081784072596
	0.5	1.143562384399	1.143562384399	1.143562384399	1.143562384399	1.143562384399	1.143562384399
	1.0	1.167668808496	1.167668808496	1.167668808496	1.167668808496	1.167668808496	
	2.0	1.207921811316	1.207921811316	1.207921811316	1.207921811316	1.207921811316	
	5.0	1.319942464258	1.319942464258	1.319942464258	1.319942464258	1.319942464258	
	10	1.529609495280	1.529609495280	1.529609495280	1.529609495280	1.529609495280	
	15	1.817956359151	1.817956359151	1.817956359151	1.817956359151	1.817956359151	
$0.1\beta N(t)$	0.1	0.9334567978521	0.9334567978521	0.9334567978521	0.9334567978521	0.9334567978521	
	0.5	0.9072868920727	0.9072868920727	0.9072868920727	0.9072868920727	0.9072868920727	
	1.0	0.8981515494688	0.8981515494688	0.8981515494688	0.8981515494688	0.8981515494688	
	2.0	0.8840445693947	0.8840445693947	0.8840445693947	0.8840445693947	0.8840445693947	
	5.0	0.8529677646007	0.8529677646007	0.8529677646007	0.8529677646007	0.8529677646007	
	10	0.8157714690339	0.8157714690339	0.8157714690339	0.8157714690339	0.8157714690339	
	15	0.7874771648081	0.7874771648081	0.7874771648081	0.7874771648081	0.7874771648081	
Total training time			0.13s	4.1s	8.45s	1.12s	0.078s

kinetics equations. The reactivity is depending on temperature and their forms are given by the following:

$$\rho(t) = \rho_0 - \alpha[T(t) - T_0] \qquad \text{and} \qquad \frac{dT(t)}{dt} = k_c N(t) \tag{43}$$

Here, T_0 is the initial temperature of the reactor ($T_0 = 30^oC$), α is the temperature coefficient of the reactivity ($\alpha = 5 \times 10^{-5} K^{-1}$), and k_c is the reciprocal of the thermal capacity of the reactor ($k_c = 0.05 k/MWs$). In order to simplify the notation in the equations of this problem, we introduce the parameter $\theta = \frac{\rho_0 + \alpha T_0 - \beta}{\Lambda}$, which is constant. Then, the system of Eqs. (25) can be written as:

$$\begin{cases} \frac{dN(t)}{dt} = \frac{\theta - \alpha T(t)}{\Lambda} N(t) + \sum_{i=1}^{6} \lambda_i C_i(t) \\ \frac{dC_i(t)}{dt} = \frac{\beta_i}{\Lambda} N(t) - \lambda_i C_i(t) \\ \frac{dT(t)}{dt} = k_c N(t) \end{cases} \tag{44}$$

$$\text{subject to :} \quad \begin{cases} N(0) = N_0 \\ C_i(0) = C_{i0} = \frac{N_0 \beta_i}{\Lambda \lambda_i} \\ T(0) = T_0 \end{cases}$$

The constrained expressions are:

$$N = (\boldsymbol{\sigma} - \boldsymbol{\sigma}_0)\xi_N + N_0 \tag{45}$$

$$C_i = (\boldsymbol{\sigma} - \boldsymbol{\sigma}_0)\xi_i + C_{i0} \qquad \forall i = 1, \dots, 6 \tag{46}$$

$$T = (\boldsymbol{\sigma} - \boldsymbol{\sigma}_0)\xi_T + T_0; \tag{47}$$

$$\dot{N} = c\boldsymbol{\sigma}'\xi_N \tag{48}$$

$$\dot{C}_i = c\boldsymbol{\sigma}'\xi_i \qquad \forall i = 1, \dots, 6 \tag{49}$$

$$\dot{T} = c\boldsymbol{\sigma}'\xi_T \tag{50}$$

Again, we are facing a non linear problem, thus we need to compute the losses and their derivatives:

$$\mathscr{L}_N = \dot{N} - \theta N + \frac{\alpha}{\Lambda} TN - \sum_{i=1}^{6} \lambda_i C_i \quad \mathscr{L}_i = \dot{C}_i - \frac{\beta_i}{\Lambda} N + \lambda_i C_i \quad \mathscr{L}_T = \dot{T} - k_c N \tag{51}$$

$$\frac{\partial \mathscr{L}_N}{\partial \xi_N} = c\boldsymbol{\sigma}' + \left(\frac{\alpha}{\Lambda} T - \theta\right)(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) \quad \frac{\partial \mathscr{L}_N}{\partial \xi_i} = -\lambda_i(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) \frac{\partial \mathscr{L}_N}{\partial \xi_T} = \frac{\alpha}{\Lambda} N(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) \tag{52}$$

$$\frac{\partial \mathscr{L}_i}{\partial \xi_N} = -\frac{\beta_i}{\Lambda}(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) \qquad \frac{\partial \mathscr{L}_i}{\partial \xi_i} = c\boldsymbol{\sigma}' + \lambda_i(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) \qquad \frac{\partial \mathscr{L}_i}{\partial \xi_T} = \mathbf{0} \tag{53}$$

$$\frac{\partial \mathscr{L}_T}{\partial \boldsymbol{\xi}_N} = -k_c(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) \qquad \frac{\partial \mathscr{L}_T}{\partial \boldsymbol{\xi}_i} = \mathbf{0} \qquad \frac{\partial \mathscr{L}_T}{\partial \boldsymbol{\xi}_T} = c\boldsymbol{\sigma}' \tag{54}$$

The linear system that we want to iteratively solve is $\mathscr{J}\Delta\tilde{\xi} = \mathscr{L}$, as reported in (42) in the previous case.

Our results are reported in Tables 10 for different values of ρ_0. The comparisons are made with SCS, CATS, and BEFD methods.

The Table 10a shows the neutron flux density for an initial value of reactivity of $\rho_0 = 1$. These results have been obtained using $n_x = 50$, $L = 20$, and a time step relatively large ($h = 0.01$), computing the solution for each times step in 0.002s (total computational time of 20 seconds), achieving all the 12 digits of accuracy.

Table 10b reports the neutron density flux for $\rho_0 = 1.5\$$. For this case, we decreased the length of time step ($h = 0.005$), and used the parameters $n_x = 60$ and $L = 35$. All the 12 digits of the other methods are achieved, with a step training time of 0.012s (total computational time of 245 s). Finally, in Table 10c the reaction insertion is $\rho_0 = 2$, and the parameters used are $n_x = 65$ and $L = 14$, with a time step $h = 0.00025$. The training time for each time step is 0.002s (total computational time of 1050 s). Again, all the 12 digits of accuracy of BEFD and SCS methods are achieved. To complete this example, Fig. 7 displays the neutron density fluxes for several step reactivity insertions with temperature feedback up to $10^3 s$. The delay in time in which feedback is activated is clearly evident.

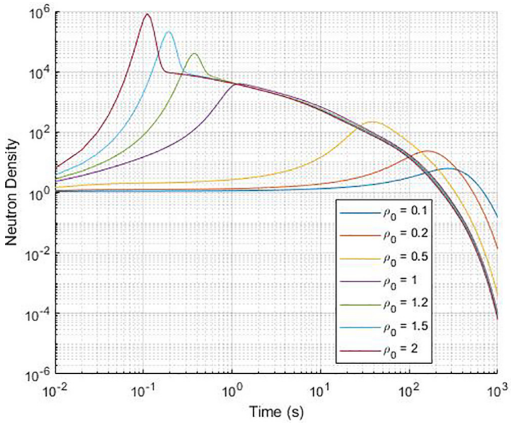


Fig. 7. Neutron density trace for step insertion with feedback expressed in.$.

4.6. Test Case 6 – thermal hydraulics feedback

The last case is derived from an old kinetics solution methodology from 1984 (SKINATH Dodds and Westfall, 1984) and demonstrates the generality and the accuracy of the X-TFC method. This

Table 10

(a) Neutron density for step reactivity feedback $\rho_0 = 1\$$ of X-TFC, BEFD, SCS, and CATS methods for the case 5.

ρ_0	t(s)	X-TFC	BEFD	SCS	CATS
1$	10	1.320385964312E+02	1.320385964312E+02	1.320385964312E+02	1.320385964312E+02
	20	5.169986094674E+01	5.169986094674E+01	5.169986094674E+01	5.169986094674E+01
	30	2.817468536129E+01	2.817468536129E+01	2.817468536129E+01	2.817468536129E+01
	40	1.814632999521E+01	1.814632999521E+01	1.814632999521E+01	1.814632999521E+01
	50	1.277957703240E+01	1.277957703240E+01	1.277957703240E+01	1.277957703240E+01
	60	9.474932500982E+00	9.474932500982E+00	9.474932500982E+00	9.474932500982E+00
	70	7.244477493764E+00	7.244477493764E+00	7.244477493764E+00	7.244477493764E+00
	80	5.646289700168E+00	5.646289700168E+00	5.646289700168E+00	5.646289700168E+00
	90	4.456834254635E+00	4.456834254635E+00	4.456834254635E+00	4.456834254635E+00
	100	3.550102765997E+00	3.550102765997E+00	3.550102765997E+00	3.550102765997E+00

(b) Neutron density for step reactivity feedback $\rho_0 = 1.5\$$ of X-TFC, BEFD, SCS, and CATS methods for the case 5.

ρ_0	t(s)	X-TFC	BEFD	SCS	CATS
1.5$	10	1.079116832006E+02	1.079116832006E+02	1.079116832006E+02	1.079116832006E+02
	20	4.160428127533E+01	4.160428127533E+01	4.160428127533E+01	4.160428127533E+01
	30	2.329893149652E+01	2.329893149652E+01	2.329893149652E+01	2.329893149652E+01
	40	1.530342749492E+01	1.530342749492E+01	1.530342749492E+01	1.530342749492E+01
	50	1.089014314534E+01	1.089014314534E+01	1.089014314534E+01	1.089014314534E+01
	60	8.101031858998E+00	8.101031858998E+00	8.101031858998E+00	8.101031858998E+00
	70	6.182690459164E+00	6.182690459164E+00	6.182690459164E+00	6.182690459164E+00
	80	4.793307819639E+00	4.793307819639E+00	4.793307819639E+00	4.793307819639E+00
	90	3.755614629341E+00	3.755614629341E+00	3.755614629341E+00	3.755614629341E+00
	100	2.966074951555E+00	2.966074951555E+00	2.966074951555E+00	2.966074951555E+00

(c) Neutron density for step reactivity feedback $\rho_0 = 2\$$ of X-TFC, BEFD, SCS, and CATS methods for the case 5.

ρ_0	t(s)	X-TFC	BEFD	SCS	CATS
2$	10	1.033808535461E+02	1.033808535461E+02	1.033808535461E+02	1.033808535461E+02
	20	3.913886902604E+01	3.913886902604E+01	3.913886902604E+01	3.913886902604E+01
	30	2.200377720638E+01	2.200377720638E+01	2.200377720638E+01	2.200377720638E+01
	40	1.449367192746E+01	1.449367192746E+01	1.449367192746E+01	1.449367192746E+01
	50	1.031861108271E+01	1.031861108271E+01	1.031861108271E+01	1.031861108271E+01
	60	7.663319203219E+00	7.663319203219E+00	7.663319203219E+00	7.663319203219E+00
	70	5.829395377947E+00	5.829395377947E+00	5.829395377947E+00	5.829395377947E+00
	80	4.499427072767E+00	4.499427072767E+00	4.499427072767E+00	4.499427072767E+00
	90	3.507422662628E+00	3.507422662628E+00	3.507422662628E+00	3.507422662628E+00
	100	2.755126886406E+00	2.755126886406E+00	2.755126886406E+00	2.755126886406E+00

Table 11

(a) Reactor power (W): solutions of X-TFC, SCS, BEFD, and SKINATH for the case 6.

t (min)	X-TFC	SCS	BEFD	SKINATH
0.000	1.00000000000E−02	1.00000000000E−02	1.00000000000E−02	1.000E−02
1.000	1.39066439993E−02	1.39066439992E−02	1.39066439993E−02	1.391E−02
10.00	1.03307456949E−01	1.03307456949E−01	1.03307456949E−01	1.033E−01
100.0	7.94613738185E−01	7.94613738201E−01	7.94613738173E−01	7.946E−01
250.0	2.11868053192E−03	2.11868053231E−03	2.11868053196E−03	2.118E−03
500.0	9.81885437276E−01	9.81885437265E−01	9.81885437257E−01	9.819E−01
750.0	4.03347200419E+00	4.03347200416E+00	4.03347200415E+00	4.033E+00
1000	7.68095079082E+00	7.68095079127E+00	7.68095079091E+00	7.680E+00
1500	15.0242166933E+00	1.50242166936E+01	1.50242166935E+01	–
2000	13.5885192814E+01	1.35885192813E+01	1.35885192814E+01	–
2500	13.4825032230E+01	1.34825032231E+01	1.34825032230E+01	–
3000	13.5975570942E+01	1.35975570943E+01	1.35975570943E+01	–
4000	13.5721402958E+01	1.35721402958E+01	1.35721402958E+01	–
5000	13.5731363004E+01	1.35731363004E+01	1.35731363004E+01	–

(b) Reactivity $: solutions of X-TFC, SCS, BEFD, and SKINATH for the case 6.

0.000	4.30000000000E−02	4.30000000000E−02	4.30000000000e-02	4.300E−02
1.000	4.29998260709E−02	4.29998260709E−02	4.29998260709e-02	4.230E−02
10.00	4.29941606543E−02	4.29941606543E−02	4.29941606543e-02	4.299E−02
100.0	−2.94953040873E−02	−2.94953040871E−02	−2.94953040874e-02	−2.950E−02
250.0	6.38846746721E−03	6.38846746738E−03	6.38846746726e-03	6.388E−03
500.0	−4.82517310769E−03	−4.82517310699E−03	−4.82517310754e-03	−4.826E−03
750.0	−1.85728671295E−03	−1.85728671222E−03	−1.85728671278e-03	−1.858E−03
1000	1.37914848552E−03	1.37914848596E−03	1.37914848566e-03	1.379E−03
1500	4.96259074581E−04	4.96259074364E−04	4.96259074524e-04	–
2000	−2.10499597542E−0	−2.10499597521E−04	−2.10499597540e-04	–
2500	3.09424979671E−05	3.09424979793E−05	3.09424979722e-05	–
3000	3.81971801108E−06	3.81971800210E−06	3.81971800786e-06	–
4000	5.94739153984 E−07	5.94739152658E−07	5.94739151924e-07	–
5000	−3.71867590307E−08	−3.71867594560E−08	−3.71867600357e-08	–

Temperature (°C): solutions of X-TFC, SCS, BEFD, and SKINATH for the case 6.

0.000	2.00000000000E+01	2.00000000000E+01	2.00000000000E+01	2.000E+01
1.000	2.00000568396E+01	2.00000568396E+01	2.00000568396E+01	2.000E+01
10.00	2.00019082829E+01	2.00019082829E+01	2.00019082829E+01	2.000E+01
100.0	4.36912758455E+01	4.36912758455E+01	4.36912758455E+01	4.369E+01
250.0	3.19645531153E+01	3.19645531153E+01	3.19645531153E+01	3.196E+01
500.0	3.56291415385E+01	3.56291415385E+01	3.56291415385E+01	3.563E+01
750.0	3.46592440238E+01	3.46592440238E+01	3.46592440238E+01	3.466E+01
1000	3.36015854623E+01	3.36015854623E+01	3.36015854623E+01	3.360E+01
1500	3.38901114135E+01	3.38901114135E+01	3.38901114135E+01	–
2000	3.41210783300E+01	3.41210783300E+01	3.41210783300E+01	–
2500	3.40421756543E+01	3.40421756543E+01	3.40421756543E+01	–
3000	3.40510393078E+01	3.40510393078E+01	3.40510393078E+01	–
4000	3.40520932225E+01	3.40520932225E+01	3.40520932225E+01	–
5000	3.40522997342E+01	3.40522997342E+01	3.40522997342E+01	–

problem is a stiff nonlinear system with a simple thermal hydraulic model that adds the following overall energy balance in terms of an average temperature:

$$\frac{dT(t)}{dt} = \frac{1}{C_p}\left[\varepsilon N(t) - AHD^{0.75}(T(t)-T_c)^{1.25}T(t)^{-0.25}\right] \tag{55}$$

where $C_p = 13006.193586 Erg^oC$ is the heat capacity of the reactor, $\varepsilon = 1$ is the fraction of the energy from fission deposited as heat in the system, $A = 17.52$ is a physical constant for air at STP, $H = 0.23m$ and $D = 0.2m$ are the height and diameter of the reactor respectively, and $T_c = 20^oC$ is the coolant temperature. The reactivity is given by the following function:

$$\rho(t) = \rho_0 + \beta_T[T(t) - T_c] \tag{56}$$

where $\beta_T = -0.00306\beta/^oC$ is the reactivity feedback coefficient and the initial reactivity is $\rho_0 = 0.043\beta$. For the sake of simplicity we make use of the following parameters:

$$\gamma = AHD^{0.75} \qquad \text{and} \qquad \theta = \frac{\rho_0 - \beta_T T_c - \beta}{\Lambda} \tag{57}$$

Then, the system of Eqs. (25) can be written as:

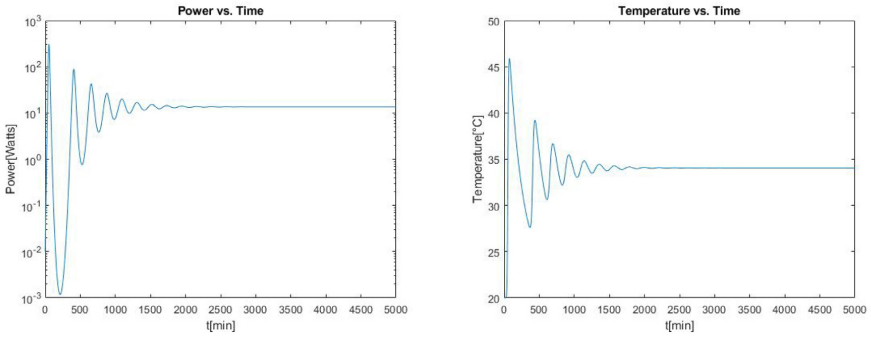
$$\begin{cases} \frac{dN(t)}{dt} = \theta N(t) + \frac{\beta_T}{\Lambda}T(t)N(t) + \sum_{i=1}^{6}\lambda_i C_i(t) \\ \frac{dC_i(t)}{dt} = \frac{\beta_i}{\Lambda}N(t) - \lambda_i C_i(t) \\ \frac{dT(t)}{dt} = \frac{1}{C_p}\left[\varepsilon N(t) - AHD^{0.75}(T(t)-T_c)^{1.25}T(t)^{-0.25}\right] \end{cases} \tag{58}$$

$$\text{subject to}: \quad \begin{cases} N(0) = N_0 \\ C_i(0) = C_{i0} = \frac{N_0\beta_i}{\Lambda\lambda_i} \\ T(0) = T_c \end{cases}$$

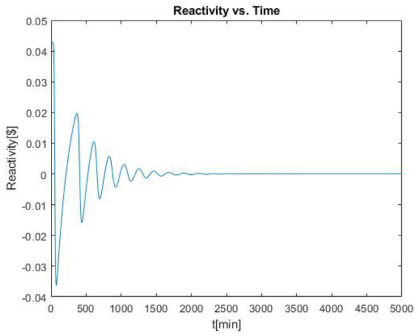
The losses and their derivatives are:

$$\begin{aligned} &\mathscr{L}_N = \dot{N} - \theta N - \frac{\beta_T}{\Lambda}TN - \sum_{i=1}^{6}\lambda_i C_i \qquad \mathscr{L}_i = \dot{C}_i - \frac{\beta_i}{\Lambda}N + \lambda_i C_i \\ &\mathscr{L}_T = \dot{T} - \frac{\varepsilon}{C_p}N + \frac{\gamma}{C_p}(T-T_c)^{1.25}T^{-0.25} \end{aligned} \tag{59}$$

$$\frac{\partial \mathscr{L}_N}{\partial \xi_N} = c\boldsymbol{\sigma}' + \left(\frac{\beta_T}{\Lambda}T - \theta\right)(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) \quad \frac{\partial \mathscr{L}_N}{\partial \xi_i} = -\lambda_i(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) \quad \frac{\partial \mathscr{L}_N}{\partial \xi_T} = \frac{\beta_T}{\Lambda}N(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) \tag{60}$$



(a) Power vs. time for thermal hydraulic feedback for case 6. (b) Temperature vs. time for thermal hydraulic feedback for case 6.



(c) Reactivity vs. time for thermal hydraulic feedback for case 6.

Fig. 8. (a) Power vs. time for thermal hydraulic feedback for case 6. (b) Temperature vs. time for thermal hydraulic feedback for case 6. (c) Reactivity vs. time for thermal hydraulic feedback for case 6.

$$\frac{\partial \mathscr{L}_i}{\partial \boldsymbol{\xi}_N} = -\frac{\beta_i}{\Lambda}(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) \quad \frac{\partial \mathscr{L}_i}{\partial \boldsymbol{\xi}_i} = c\boldsymbol{\sigma}' + \lambda_i(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) \quad \frac{\partial \mathscr{L}_i}{\partial \boldsymbol{\xi}_T} = \mathbf{0} \tag{61}$$

$$\frac{\partial \mathscr{L}_T}{\partial \boldsymbol{\xi}_N} = -\frac{\varepsilon}{C_p}(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) \quad \frac{\partial \mathscr{L}_T}{\partial \boldsymbol{\xi}_i} = \mathbf{0} \tag{62}$$

$$\frac{\partial \mathscr{L}_T}{\partial \boldsymbol{\xi}_T} = c\boldsymbol{\sigma}' + \frac{\gamma}{C_p}\Big(1.25(T - T_c)^{0.25}T^{-0.25} - 0.25(T - T_c)^{1.25}T^{-1.25}\Big)(\boldsymbol{\sigma} - \boldsymbol{\sigma}_0) \tag{63}$$

The linear system that we want to iteratively solve is $\mathscr{J}\Delta\xi = \mathscr{L}$, as reported in (42) in cases 4 and 5.

In Tables 11a, 11b, and 11c the numerical results for neutron density flux, reactivity, and temperature are reported up to 5000 min of reactor operation, obtained with constant step size $h = 0.1$, consuming training time $0.035s$ per step (total computational time of about 30 h). The parameters used are $n_x = 50$ and $L = 50$. The comparison with SCS ($h = 0.001$ and training time of $0.0052s$ per step, thus a total computational time of about 433 h), BEFD (overall computational time of a most remarkable 67 s), and SKINATH ($h = 0.01$) methods are reported. The digits marked in bold represent the X-TFC and SCS digits that differ from those obtained with BEFD method. As one can see, for neutron density and temperature, 10 digits match for almost all the time intervals. A slight discrepancy is more visible for the reactivity for the last 2 or 3 digits from 2500 min to above. To complete this example, in Figs. 8a, 8b, and 8c the Neutron flux, reactivity and temperature functions are plotted for 5000 min or reactor operation.

5. Conclusions

This paper presents a novel and highly accurate algorithm for solving the PKEs for nuclear reactor dynamics. The basic idea behind this approach is to use a Physics-Informed Neural Network approach for the approximation of the unknowns, combining it with the Theory of Functional Connection to naturally satisfy the initial conditions. The single layer Neural Network is trained via the ELM algorithm, to achieve high computational performances. One of the advantages of using PINN-based methods is that once the output weights of the NN are determined for the training points, we have an analytical representation (by NNs) of PKEs' solutions. Therefore, they can be evaluated at any desired time instant with no additional computational effort. An unintended consequence of this work shows the robustness of BEFD as a benchmark, now requiring QP.

As for other future works, the authors will consider the application of convergence acceleration techniques to further improve the accuracy and reduce the computational effort required for training. Further work is also planned to design the network training in a data-physics-driven manner, adding additional terms in the PINN-based method's loss function to better drive the learning with a data-physics driven fashion. This could be achieved in a

number of different ways, including using data generated from other models or real data to account for physical features not accounted for in the PKEs.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors would like to acknowledge Mr. Andrea D'Ambrosio for providing computational resources.

References

Henry, A.F., 1958. The application of reactor kinetics to the analysis of experiments. Nucl. Sci. Eng. 3 (1), 52–70.
Ott, K., Meneley, D., 1969. Accuracy of the quasistatic treatment of spatial reactor kinetics. Nucl. Sci. Eng. 36 (3), 402–411.
Picca, P., Furfaro, R., Ganapol, B.D., 2013. A highly accurate technique for the solution of the non-linear point kinetics equations. Ann. Nucl. Energy 58, 43–53.
Cai, Y., Peng, X., Li, Q., Wang, K., 2016. A numerical solution to the nonlinear point kinetics equations using magnus expansion. Ann. Nucl. Energy 89, 84–89.
Leite, S.Q.B., Palma, D.A., de Vilhena, M.T., Bodmann, B.E., 2014. Analytical representation of the solution of the point reactor kinetics equations with adaptive time step. Prog. Nucl. Energy 70, 112–118.
Leite, S.Q.B., de Vilhena, M.T., Bodmann, B.E., 2016. Solution of the point reactor kinetics equations with temperature feedback by the its2 method. Prog. Nucl. Energy 91, 240–249.
Hamada, Y.M., 2015. Trigonometric fourier-series solutions of the point reactor kinetics equations. Nucl. Eng. Des. 281, 142–153.
Hamada, Y.M., 2018. Generalized trigonometric fourier-series method with automatic time step control for non-linear point kinetics equations. Computat. Appl. Math. 37 (3), 3473–3502.
Nahla, A., Edress, A., 2016. Analytical exponential model for stochastic point kinetics equations via eigenvalues and eigenvectors. Nucl. Sci. Tech. 27 (1), 20.
Razak, M.M.A., Devan, K., Sathiyasheela, T., 2015. The modified exponential time differencing (etd) method for solving the reactor point kinetics equations. Ann. Nucl. Energy 76, 193–199.
Da Silva, M.W., Leite, S.B., Vilhena, M., Bodmann, B., 2014. On an analytical representation for the solution of the neutron point kinetics equation free of stiffness. Ann. Nucl. Energy 71, 97–102.
Hamada, Y.M., 2011. Generalized power series method with step size control for neutron kinetics equations. Nucl. Eng. Design 241 (8), 3032–3041.
Hamada, Y.M., 2013. Confirmation of accuracy of generalized power series method for the solution of point kinetics equations with feedback. Ann. Nucl. Energy 55, 184–193.
Hamada, Y.M., 2018. A new accurate numerical method based on shifted chebyshev series for nuclear reactor dynamical systems. Sci. Technol. Nucl. Install. 2018.
Ganapol, B., Picca, P., Previti, A., Mostacci, D., 2012. The solution of the point kinetics equations via converged accelerated taylor series (cats). In: Proceedings of PHYSOR, 2012.
Ganapol, B.D., 2013. A highly accurate algorithm for the solution of the point kinetics equations. Ann. Nucl. Energy 62, 564–571.
Raissi, M., Perdikaris, P., Karniadakis, G.E., 2019. Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J. Comput. Phys. 378, 686–707.
Leake, C., Mortari, D., 2020. Deep theory of functional connections: a new method for estimating the solutions of partial differential equations. Mach. Learn. Knowl. Extract. 2 (1), 37–55.
E. Schiassi, C. Leake, M. De Florio, H. Johnston, R. Furfaro, and D. Mortari, "Extreme Theory of Functional Connections: A Physics-Informed Method For Solving Parametric Differential Equations," arXiv, 2020.
Mortari, D., 2017. The theory of connections: connecting points. Mathematics 5 (4), 57.
Huang, G.-B., Zhu, Q.-Y., Siew, C.-K., May 2006. Extreme learning machine: theory and applications. Neurocomputing 70, 489–501.
Mortari, D., 2017. Least-squares solution of linear differential equations. Mathematics 5 (4), 48.
Mortari, D., Johnston, H., Smith, L., 2019. High accuracy least-squares solutions of nonlinear differential equations. J. Comput. Appl. Math. 352, 293–307.
De Florio, M., Schiassi, E., D'Ambrosio, A., Mortari, D., Furfaro, R., 2021. Theory of functional connections applied to linear odes subject to integral constraints and linear ordinary integro-differential equations. Math. Comput. Appl. 26 (3), 65.
De Florio, M., Schiassi, E., Furfaro, R., Ganapol, B.D., Mostacci, D., 2020. Solutions of chandrasekhar' s basic problem in radiative transfer via theory of functional connections. J. Quant. Spectrosc. Radiat. Transf. 107384
De Florio, M., Schiassi, E., Ganapol, B.D., Furfaro, R., 2021. Physics-informed neural networks for rarefied-gas dynamics: Thermal creep flow in the bhatnagar–gross–krook approximation. Phys. Fluids 33, (4) 047110.
M. De Florio, E. Schiassi, R. Furfaro, and B.D. Ganapol, 1999. A Physics-Informed Solution for Poiseuille Flow in a Plane Channel via Extreme Theory of Functional Connections. In preparation.
Furfaro, R., Mortari, D., 2019. Least-squares solution of a class of optimal space guidance problems via Theory of Connections. Acta Astronaut.
H. Johnston, E. Schiassi, R. Furfaro, and D. Mortari, "Fuel-efficient powered descent guidance on large planetary bodies via theory of functional connections," arXiv preprint arXiv:2001.03572, 2020.
E. Schiassi, A. D'Ambrosio, H. Johnston, R. Furfaro, F. Curti, and D. Mortari, "Complete Energy Optimal Landing on Planetary Bodies via Theory of Functional Connections," Acta Astronautica - in preparation, 2020.
Drozd, K., Furfaro, R., Schiassi, E., Johnston, H., Mortari, D., 2021. Energy-optimal trajectory problems in relative motion solved via theory of functional connections. Acta Astronaut.
Lagaris, I.E., Likas, A., Fotiadis, D.I., Sept 1998. Artificial neural networks for solving ordinary and partial differential equations. IEEE Trans. Neural Networks 9, 987–1000.
J.N. Reddy, "An Introduction to the Finite Element Method," J. Pressure Vessel Technol., 111 (1989) 348–349.
Lu, L., Meng, X., Mao, Z., Karniadakis, G.E., 2019. DeepXDE: A deep learning library for solving differential equations. arXiv preprint arXiv:1907.04502.
A.G. Baydin, B.A. Pearlmutter, A.A. Radul, and J.M. Siskind, "Automatic differentiation in machine learning: a survey," 2015.
S. Mishra and R. Molinaro, "Estimates on the generalization error of physics informed neural networks (pinns) for approximating pdes ii: A class of inverse problems," arXiv preprint arXiv:2007.01138, 2020.
Wang, S., Teng, Y., Perdikaris, P., 2020. Understanding and mitigating gradient pathologies in physics-informed neural networks. In: arXiv preprint arXiv:2001.04536.
P. Mertikopoulos, C. Papadimitriou, and G. Piliouras, "Cycles in adversarial regularized learning," in Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 2703–2717, SIAM, 2018.
Balduzzi, D., Racaniere, S., Martens, J., Foerster, J., Tuyls, K., Graepel, T., 2018. The mechanics of n-player differentiable games. In: International Conference on Machine Learning PMLR, pp. 354–363.
Duderstadt, J.J., Hamilton, L.J., 1976. Nuclear reactor analysis. John Wiley & Sons Inc., New York.
J. Nóbrega da, "A new solution of the point kinetics equations," Nucl. Sci. Eng., vol. 46, no. 3, pp. 366–375, 1971.
H. Dodds Jr and R. Westfall, "Skinath-a computer program for solving the reactor point kinetics equations with simple thermal-hydraulic feedback," tech. rep., Tennessee Univ., Knoxville (USA). Dept. of Nuclear Engineering; Oak Ridge, 1984.

Appendix C

Schiassi et al. (2022): X-TFC for Optimal Planar Orbit Transfer

Reproduced from Schiassi, E., D'Ambrosio, A., Drozd, K., Curti, F. and Furfaro, R., 2022. Physics-Informed Neural Networks for Optimal Planar Orbit Transfers. Journal of Spacecraft and Rockets, pp.1-16. https://doi.org/10.2514/1.A35138 [63], with the permission of ARC.

JOURNAL OF SPACECRAFT AND ROCKETS
Vol. 59, No. 3, May–June 2022

Physics-Informed Neural Networks for Optimal Planar Orbit Transfers

Enrico Schiassi*
University of Arizona, Tucson, Arizona 85721
Andrea D'Ambrosio†
University of Rome "La Sapienza," 00138 Rome, Italy
Kristofer Drozd‡
University of Arizona, Tucson, Arizona 85721
Fabio Curti§
University of Rome "La Sapienza," 00138 Rome, Italy
and
Roberto Furfaro¶
University of Arizona, Tucson, Arizona 85721

https://doi.org/10.2514/1.A35138

This paper presents a novel framework, combining the indirect method and Physics-Informed Neural Networks (PINNs), to learn optimal control actions for a series of optimal planar orbit transfer problems. According to the indirect method, the optimal control is retrieved by directly applying the Pontryagin minimum principle, which provides the first-order necessary optimality conditions. The necessary conditions result in a two-point boundary value problem (TPBVP) in the state–costate pair, constituting a system of ordinary differential equations, representing the physics constraints of the problem. More precisely, the goal is to model a neural network (NN) representation of the state–costate pair for which the residuals of the TPVBP are as close to zero as possible. This is done using PINNs, which are particular NNs where the training is driven by the problem's physics constraints. A particular PINN method will be used, named Extreme Theory of Functional Connections (X-TFC), which is a synergy of the classic PINN and the Theory of Functional Connections. With X-TFC, the TPBVP's boundary conditions are analytically satisfied. This avoids having unbalanced gradients during the network training. The results show the feasibility of employing PINNs to tackle this class of optimal control problems for space applications.

Nomenclature

A	=	solar sail area
b	=	square root of mapping coefficient
b_q	=	bias vector component
c	=	mapping coefficient
c_L	=	speed of light
F	=	differential equation in implicit form
$\boldsymbol{f}$	=	right-hand side of dynamics equation
g	=	free function
H	=	Hamiltonian
$\mathcal{J}$	=	cost function
$\mathbb{J}$	=	Jacobian matrix
L	=	Lagrangian cost
L	=	number of hidden neurons
$\mathcal{L}$	=	loss associated to each variable
$\mathbb{L}$	=	loss vector
m	=	spacecraft mass
m_0	=	spacecraft initial mass
N	=	number of discretization points
$\mathcal{N}_{\text{eq}}$	=	number of ordinary differential equations
$\hat{\boldsymbol{n}}$	=	solar sail normal unit vector
R_E	=	sun–Earth mean distance
R_M	=	sun–Mars mean distance
R_0	=	constant low-thrust magnitude
r	=	sun–spacecraft radial distance
s	=	support function
t	=	time instant
t_f	=	final time instant
t_0	=	initial time instant
$\mathcal{U}$	=	uniform distribution
u	=	thrust direction angle
v_r	=	radial velocity
v_θ	=	tangential velocity
W_E	=	solar constant
w_q	=	input weights vector component
$\boldsymbol{x}$	=	state vector
x_i	=	state vector components
y	=	unknown function
z	=	input variable
α	=	solar sail cone angle
β	=	output weights vector
Γ	=	weighting parameter in cost function
γ	=	gravitational attraction of sun (dimensionless)
$\Delta\boldsymbol{\Xi}$	=	augmented unknowns vector variation
ϵ	=	user-defined tolerance
η	=	coefficients of constrained expression
θ	=	angular displacement about attracting center
$\boldsymbol{\theta}_{\text{NN}}$	=	neural network parameters vector
κ_{ss}	=	solar sail mass-to-area ratio
$\boldsymbol{\Xi}$	=	augmented unknowns vector

Part of this work was presented as Paper 20-524 at the 2020 AAS/AIAA Astrodynamics Specialist Conference, Lake Tahoe, CA, August 9-12, 2020; received 30 March 2021; revision received 15 October 2021; accepted for publication 23 October 2021; published online 3 January 2022.

*Ph.D. Student, System & Industrial Engineering, Department of System & Industrial Engineering, 1127 East James E. Rogers Way.
†Ph.D. Candidate, School of Aerospace Engineering, Via Salaria 851.
‡Ph.D. Student, System & Industrial Engineering, Department of System & Industrial Engineering, 1127 East James E. Rogers Way.
§Associate Professor, School of Aerospace Engineering, School of Aerospace Engineering, Via Salaria 851.
¶Professor, Department of System & Industrial Engineering, 1127 East James E. Rogers Way (Corresponding Author).

$\boldsymbol{\lambda}$ = costate vector
μ_S = gravitational attraction of sun
σ = activation function
Φ = Meyer cost (boundary conditions vector)
ϕ = instantaneous acceleration vector norm
Ω_i = switching functions for constrained expression

I. Introduction

OPTIMAL orbit transfer problems are essential to accurately design space exploration missions. There are many orbit transfers that can be studied, for example, from a planet to another planet, between two orbits around the same planet, and from a planet to another celestial body, such as asteroids or comets. All these problems belong to the class of optimal control problems (OCPs). Thus, they can be tackled by following two different approaches: direct methods and indirect methods [1]. Both the methods have been widely applied in the literature to study optimal orbit transfers. For instance, the direct method has been employed in [2] to study optimal planar interplanetary transfers toward Mars and Mercury in combination with a Genetic Algorithm. In [3], Sequential Quadratic Programming is exploited to solve the constrained parameter optimization problem related to low-thrust Earth-orbit transfers. On the other hand, the indirect method is used in [4] to compute low-thrust trajectories with a homotopy approach, which allows us to achieve good accuracy in solving fuel-optimal orbit transfers. In this case, a shooting method is employed to solve the two-point boundary value problem (TPBVP), and the initial guess is provided through shape based methods. The same method is also employed in [5], in which a smoothing technique is exploited to make the fuel-optimal orbit transfer converge to a feasible solution. Moreover, optimization of minimum-time low-thrust orbit transfers can also be studied via convex programming, as shown in [6]. As can be seen, OCPs for orbit transfers are mostly tackled via shooting methods, pseudospectral methods, convex programming, and several numerical techniques to solve nonlinear programming problems (NLPs). In addition, many works in the literature also employ the commercial software GPOPS-II [7] to compare their performances, as GPOPS-II is a commonly adopted and reliable tool for nonlinear OCPs. In particular, this software uses variable-order Gaussian quadrature orthogonal collocation methods and an adaptive mesh refinement method to determine the number of mesh intervals and the degree of the approximating polynomial within each mesh interval to achieve a specified accuracy. Hence, the resulting continuous-time OCP is transcribed to a large sparse NLP that is solved via quasi-Newton or Newton NLP solvers.

In general, direct and indirect methods are based on different theoretical and mathematical concepts. In the direct method, the state and control of the system are discretized, and thus the problem is transcribed into a nonlinear optimization problem. These NLPs can be solved using well-known optimization techniques, such as the trust region method, Nelder–Mead method, or interior point methods [8]. According to these techniques, the optimal solution is found by transcribing an optimization problem from infinite dimensional to finite dimensional. Conversely, indirect methods are based on the first-order necessary conditions retrieved by direct application of the Pontryagin Minimum Principle (PMP). The necessary conditions result in a 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 [9,10], orthogonal collocation [11], or pseudospectral methods [12]). Generally, the indirect method provides a solution which 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 very difficult to obtain because of how sensitive it is to an initial guesses, which is required for a numerical solution. Moreover, it is also very difficult to provide good initial guesses for the costates because 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 generally preferred by the aerospace community. Furthermore, path and control constraints can be more easily be 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 the class of optimal planar orbit transfer problems, applying the PMP and modeling a neural network (NN) representation of the state–costate pair for which the residuals of the TPVBP are as close to zero as possible. Although standard deep learning frameworks have already been employed in the literature to tackle OCPs, such as for real-time optimal control for spacecraft orbit transfer [13], real-time guidance for low-thrust transfers [14], optimal control for landing trajectories [15–17], and onboard generation of optimal trajectories for hypersonic vehicles [18], to the best of our knowledge, Physics-Informed Neural Networks (PINNs) are employed for the first time in this paper to solve OCPs. More specifically, the goal of this paper is to show the effectiveness of PINN frameworks at solving optimal orbit transfer problems, achieving results comparable with the other state-of-the-art methods such as the ones mentioned previously. In addition, we will show how the proposed method can represent an alternative to the state-of-the-art methods as well as be used in synergy with those methods. PINNs are machine-learning methods that insert physics into data-driven functional representations of input–output paring collections. The term PINN, as defined by Raissi et al. [19], describes NNs that embed the physics, modeled via differential equations (DEs), as a regularization term in the loss function. Moreover, initial conditions (ICs) and/or boundary conditions (BCs) are also added in the loss function as additional penalty terms. Thus, one can guarantee that the physics underlying the process (e.g., DEs and the corresponding constraints) is not violated.

In this paper, we employ a novel and efficient PINN model, developed by the authors, that merges NN and Theory of Functional Connections (TFC) [20,21]. That is, we will use a PINN–TFC based model, called Extreme Theory of Functional Connections (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 key ingredient of the TFC, developed by Mortari [22], which is a method for functional interpolation where functions are approximated using the CEs. A constrained expression is a functional that is a sum of a free function and a functional that analytically satisfies the constraints regardless to the choice of the free function [22,23]. 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 [24–26]. 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 [27,28], fuel optimal landing on large planetary bodies [29], and energy optimal relative motion problems subject to Clohessy–Wiltshire dynamics [30]. For solving DEs, the standard TFC method uses a linear combination of orthogonal polynomials, such as Legendre or Chebyshev polynomials [24,25], 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 partial differential equations (PDEs). To overcome this limitation, X-TFC uses a single-layer NN (SLNN) trained via the Extreme Learning Machine (ELM) algorithm [31] 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 physic-driven fashion, being a PINN framework, it can also be used for data-driven solution of DEs, data-driven DE parameter discovery, and data-driven discovery of DEs [32]. The manuscript is organized as follows. First is how PINNs, and in particular X-TFC, are employed to learn control actions for optimal control problems formulated using the indirect method. This approach is then used to tackle three typical optimal planar orbit transfer problems: the

maximum radius orbit transfer, the minimum time orbit transfer, and the minimum time orbit transfer using a solar sail. Finally, the results will be discussed in the remaining sections.

II. PINNs for Optimal Control Problems

In this section, a brief review of how to approach OCPs using the indirect method is given. In particular, we show how the TPBVP is derived from the application of PMP. Afterward, how PINN methods are employed to learn the state–costate pair, which is the solution of the TPBVP, is presented in detail.

A. Optimal Control Problems via Indirect Method

Optimal control problems can be formulated as a set of differential equations, describing the paths of the control variables, that satisfies an optimal criterion, which is usually expressed as the minimization or maximization of a cost function. Generally, this cost function depends on state and control variables as

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

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,

$$\boldsymbol{\Phi}(\boldsymbol{x}(t_0), t_0) = \boldsymbol{\Phi}_0 \tag{3}$$

$$\boldsymbol{\Phi}(\boldsymbol{x}(t_f), t_f) = \boldsymbol{\Phi}_f \tag{4}$$

where $\boldsymbol{x}(t)$ is the state vector, $\boldsymbol{u}(t)$ is the control vector, t is the independent variable (usually time), t_0 is the initial time, and t_f is the final time. The terms $\boldsymbol{\Phi}$ and L, appearing in Eq. (1) are the endpoint cost and Lagrangian, respectively [8]. Hereafter, the authors will only refer to problems without path constraints and/or bounds on the control. As previously mentioned, optimal control problems can be solved via either direct or indirect methods. The approach proposed in this work can be labeled within the indirect methods category. Following these methods, the optimal control problems are solved by applying the Pontryagin Minimum Principle (PMP) [33]. The use of PMP requires the definition of the Hamiltonian. That is,

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

where $\boldsymbol{\lambda}$ represent the costate (or adjoint variables). According to the first-order optimality conditions of the PMP, the optimal control can be retrieved by taking the derivative of the Hamiltonian with respect to the control vector and setting it equal to zero,

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

Once Eq. (6) is solved for $\boldsymbol{u}$, the result is plugged into Eq. (5). Now, applying the first-order necessary conditions for the state and costate variables, we get the following system of ordinary differential equations (ODEs):

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

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

Additionally, the transversality conditions, if any (e.g., when the corresponding state variable is free), should be imposed. For the sake of completeness, the possible transversality conditions (not considering the constraints) are listed in 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 boundary value problem (BVP), whose solutions will be learned via the PINN–TFC based framework X-TFC. One should note that the costate's transversality conditions will be a priori satisfied with the proposed approach, as we will explain in the following. Thus, they do not explicitly appear in the BVP.

B. Physics-Informed Neural Networks

PINNs have been defined by Raissi et al. [19], following the original idea introduced by Lagaris et al. [34], to tackle two main classes of problems: data-driven solution and data-driven discovery of problems involving DEs. As an example, assume that one wants to perform a regression of an experimental data set using a NN and that such collected data are a representation of some physical phenomena modeled via a set of DEs. In standard regression, one would approximate the data using a NN (either shallow or deep) trained to minimize a mean squared error (MSE) as a loss function. However, doing so, there is no guarantee that the physics phenomena governing the data set would not be violated. Therefore, PINNs are introduced to ensure that DEs, coming from first principles and modeling the physics of the experimental data, are added as a penalty to the loss function. This additional term acts as a regularizator that penalizes the training when the DE and its constraints (e.g., the ICs and/or BCs) are violated. Thus, one can guarantee that the physics underlying the process is not violated. This approach is defined as a data-driven solution of DEs. More specifically, from the physics prospective, this approach enables the training of NNs in order to learn the solution of DEs in a data-physics-driven fashion. This becomes of extreme importance if the DEs do not exactly model the physics of the problem, for instance, when perturbations are present and/or when uncertain dynamical systems are considered. Conversely, when the goal is to estimate parameters governing some physical phenomena modeled through a DE (e.g., for orbit determination), one conventionally talks about data-driven parameters discovery of DEs (e.g., inverse problems) [19,32]. When data are not available, and therefore the loss function contains only the residual of the DEs, and its constraints, PINNs learn the solutions of problems involving ODEs and PDEs and solely in a physics-driven fashion.

Standard PINNs use a single deep neural network (DNN) to fully represent the DE solution. In particular, the training set consists of randomly parsed data points from the full high-resolution data set, and randomly sampled collocation points for enforcing BCs and to enforce the DE, on the boundary and inside the solution domain, respectively. Once the output of the DNN is computed, it is plugged into the residuals of the DEs and the BCs. The training of the NN is used to learn the weights and biases parameters that minimize the physics based cost function via gradient based optimization techniques. Thus, the key feature of the PINN is the embedding information pertaining to the physics of the problem, derived from the DEs and BCs, in their cost functions. To better clarify how standard PINNs and X-TFC work, the mathematical formulation to solve a generic TPBVP with both the approaches is reported in the following.

1. PINN for TPBVPs

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

$$F_i(t, y_j(t), \dot{y}_j(t), \ddot{y}_j(t)) = 0 \quad \text{subject to:} \begin{cases} y_j(t_0) = y_{0_j} \\ y_j(t_f) = y_{f_j} \\ \dot{y}_j(t_0) = \dot{y}_{j_0} \\ \dot{y}_j(t_f) = \dot{y}_{f_j} \end{cases} \tag{13}$$

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 (13) are approximated with a NN, which can be shallow or deep. Following [35], we use a single-input and multiple-outputs NN to approximate the solution of the system of ODEs. More precisely, the input is represented by the time t, and the outputs are represented by the y_j. That is,

$$y_j(t) = y_j^{\text{NN}}(t, \boldsymbol{\theta}_{\text{NN}}) \tag{14}$$

where $\boldsymbol{\theta}_{\text{NN}}$ are the NN parameters which are trained via gradient based methods. To train the network, the MSE is to be minimized with respect to the NN parameters,

$$\min_{\boldsymbol{\theta}_{\text{NN}}} \text{MSE} \tag{15}$$

where

$$\text{MSE} = \text{MSE}_{\mathcal{F}} + \text{MSE}_{\text{BC}_0} + \text{MSE}_{\text{BC}_f} \tag{16}$$

with

$$\text{MSE}_{\mathcal{F}} = \frac{1}{N_{\mathcal{F}}} \sum_{i=1}^{\mathcal{N}_{\text{eq}}} \sum_{k=1}^{N_{\mathcal{F}}} \left| F_i(t_{\mathcal{F}}^k, y_j^{\text{NN}}(t_{\mathcal{F}}^k), \dot{y}_j^{\text{NN}}(t_{\mathcal{F}}^k), \ddot{y}_j^{\text{NN}}(t_{\mathcal{F}}^k)) \right|^2 \tag{17}$$

$$\text{MSE}_{\text{BC}_0} = \sum_{j=1}^{N_y} \left| y_{0_j} - y_j^{\text{NN}}(t_0) \right|^2 + \sum_{j=1}^{N_y} \left| \dot{y}_{0_j} - \dot{y}_j^{\text{NN}}(t_0) \right|^2 \tag{18}$$

$$\text{MSE}_{\text{BC}_f} = \sum_{j=1}^{N_y} \left| y_{f_j} - y_j^{\text{NN}}(t_f) \right|^2 + \sum_{j=1}^{N_y} \left| \dot{y}_{f_j} - \dot{y}_j^{\text{NN}}(t_f) \right|^2 \tag{19}$$

Here, $\text{MSE}_{\mathcal{F}}$, MSE_{BC_0}, and MSE_{BC_f} are the MSEs that take into account the dynamics, the initial conditions, and the final conditions, respectively. $\mathcal{N}_{\text{eq}}$ is the number of ODEs in the system (13), N_y is the number of latent solutions and y_j, and $\{t_{\mathcal{F}}^k\}_{k=1}^{N_{\mathcal{F}}}$ represent the training points for each $\{F_i\}_{i=1}^{\mathcal{N}_{\text{eq}}}$. Finally, the derivatives of the $y_j^{\text{NN}}(t, \boldsymbol{\theta}_{\text{NN}})$'s with respect to time are computed via automatic differentiation.

The major drawback of the standard PINN framework, as introduced by Raissi et al. [19], 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, during the PINN training, we deal with competing objectives: 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 methods that cause PINNs to often struggle to accurately learn the underlying DE solution [36]. It is well known that gradient based methods may get stuck in limit cycles or even diverge in the presence of multiple competing objectives [37,38]. When using PINNs for learning the solutions of TPBVPs arising from the application of the PMP, one deals with a system of ODEs. Thus, the number of competing objectives increases; learning the ODE latent solution within the domain for all the equations of the systems and learning the ODE latent solution on the boundaries for all the unknowns. In [36], to overcome this issue, the authors developed a learning rate annealing algorithm that employs gradient statistics to adaptively 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. As previously mentioned, in this research, this competing objective issue is removed thanks to X-TFC.

2. X-TFC for TPBVPs

Here, how to apply X-TFC to solve a generic TPBVP in the time domain is presented in detail. Considering the system expressed by Eq. (13), the first step of the X-TFC method follows the derivation of the constrained expressions, and their derivatives, as developed by TFC,

$$y_j^{(\ell)}(t) = g_j^{(\ell)}(t) + \sum_{k=1}^{n_j} \eta_{k_j} s_k^{(\ell)}(t) \tag{20}$$

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 [22], the functions $s_k(t)$, called support functions, can be selected as follows:

$$s_k(t) = t^{k-1} \tag{21}$$

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, that is then solved for the η_{k_j} coefficients. Once the η_{k_j} coefficients are calculated, the boundary constraints of Eq. (13) are analytically embedded within the 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 only a function of the independent variable t, the free function $g_j(t)$, and their derivatives. That is,

$$\tilde{F}_i(t, g_j(t), \dot{g}_j(t), \ddot{g}_j(t)) = 0 \tag{22}$$

This vector differential equation is unconstrained because the boundary conditions are embedded within it through the derived η_{k_j} values. To solve Eq. (22), X-TFC uses as free functions $g_j(t)$, a single-layer NN trained via the ELM algorithm [31]. That is,

$$g_j(z) = \sum_{q=1}^{L} \beta_{j,q} \sigma(w_q z + b_q) = \begin{bmatrix} \sigma_1 \\ \vdots \\ \sigma_L \end{bmatrix}^T \boldsymbol{\beta}_j = \boldsymbol{\sigma}^T(z) \boldsymbol{\beta}_j \tag{23}$$

where L is the number of hidden neurons, $w_q \in \mathbb{R}$ 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, b_q is the bias of the qth hidden neuron, and $\sigma_j(\cdot)$ are activation functions for the $g_j(z)$ free chosen function. Equation (23) 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 a 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 a PINN method. As we are using the ELM algorithm to train the NN, the only unknowns to compute are the output weights $\boldsymbol{\beta}_j = [\beta_{j,1}, \ldots, \beta_{j,L}]^T$. 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,

$$z = z_0 + c(t - t_0) \leftrightarrow t = t_0 + \frac{1}{c}(z - z_0) \tag{24}$$

where c is a mapping coefficient that is

$$c = b^2 = \frac{z_f - z_0}{t_f - t_0} \tag{25}$$

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

$$\frac{\mathrm{d}^n g_j}{\mathrm{d}t^n} = \boldsymbol{\beta}_j^T \frac{\mathrm{d}^n \boldsymbol{\sigma}_j(z)}{\mathrm{d}z^n} \left(\frac{\mathrm{d}z}{\mathrm{d}t}\right)^n = \boldsymbol{\beta}^T \frac{\mathrm{d}^n \boldsymbol{\sigma}_j(z)}{\mathrm{d}z^n} (b^2)^n \tag{26}$$

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 the $\boldsymbol{\beta}_j$. Hence, the transformation of the free function and its derivatives between the t domain to the z domain can be summarized as

$$\begin{cases} g_j(t) = \boldsymbol{\sigma}_j^T(z)\boldsymbol{\beta}_j \\ \dot{g}_j(t) = b^2 \boldsymbol{\sigma}_j'^T(z)\boldsymbol{\beta}_j \\ \ddot{g}_j(t) = b^4 \boldsymbol{\sigma}_j''^T(z)\boldsymbol{\beta}_j \end{cases} \tag{27}$$

where $\boldsymbol{\sigma}_{j'}(z)$ is the abbreviation for $d\boldsymbol{\sigma}_j(z)/dz$. Equation (22) in the z domain then becomes

$$\tilde{F}_i(z, \boldsymbol{\beta}_j) = 0 \tag{28}$$

Now, to solve this TPBVP numerically, we must discretize the z domain into n points. In this work, we discretize z with equally spaced points, but one can use any desired quadrature scheme. The unconstrained set of differential equations in Eq. (28) can then be expressed as loss functions at each discretization point,

$$\mathcal{L}_i(\boldsymbol{\beta}_j) = \begin{Bmatrix} \tilde{F}_i(z_0, \boldsymbol{\beta}_j) \\ \vdots \\ \tilde{F}_i(z_d, \boldsymbol{\beta}_j) \\ \vdots \\ \tilde{F}_i(z_n, \boldsymbol{\beta}_j) \end{Bmatrix} \tag{29}$$

Now, by combining the differential equation for each dimension, an augmented loss function is formed as

$$\mathbb{L} = \{ \mathcal{L}_1, \quad \ldots, \quad \mathcal{L}_i, \quad \ldots, \quad \mathcal{L}_{\mathcal{N}_{\text{eq}}} \}^T \tag{30}$$

and imposing that to be a true solution, this vector should be equal to $\mathbf{0}$. This allows the $\boldsymbol{\beta}_j$ coefficients to be solved via different optimization schemes, for example, least square for linear problems [24] and iterative least squares for nonlinear problems [25]. If the iterative least-square method is required, the estimation of the unknowns is updated at each iteration as

$$\boldsymbol{\Xi}_{k+1} = \boldsymbol{\Xi}_k + \Delta\boldsymbol{\Xi}_k \tag{31}$$

where $\boldsymbol{\Xi}$ 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 $\Delta\boldsymbol{\Xi}_k$ term can be defined by performing classic linear least square at each iteration of the iterative least-square procedure,

$$\Delta\boldsymbol{\Xi}_k = -(\mathbb{J}(\boldsymbol{\Xi}_k)^T \mathbb{J}(\boldsymbol{\Xi}_k))^{-1} \mathbb{J}(\boldsymbol{\Xi}_k)^T \mathbb{L}(\boldsymbol{\Xi}_k) \tag{32}$$

where $\mathbb{J}$ is the Jacobian matrix containing the derivatives of the losses with respect to all the unknowns. One can consider computing the Jacobian either by hand or by means of computing tools, such as the Symbolic or the Automatic Differentiation routines available within MATLAB® 2020a. The iterative process is repeated until either of the following conditions is met,

$$L_2[\mathcal{L}(\boldsymbol{\Xi}_k)] < \varepsilon \tag{33}$$

where ϵ defines some user-prescribed tolerance. Once we are sure that the convergence is achieved by meeting the criteria $L_2[\mathcal{L}(\boldsymbol{\beta}_k)] < \epsilon$, the nonlinear DE can be solved using the following criteria: $L_2[\mathcal{L}(\boldsymbol{\beta}_{k+1})] > L_2[\mathcal{L}(\boldsymbol{\beta}_k)]$. Doing so, the convergence is caused by roundoff error, 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).

For the convenience of the reader in Fig. 1, a schematic summarizing how the proposed X-TFC based framework works for solving generic OCPs is presented.

In addition, we provide a step-by-step summary of the proposed framework. That is as follows:

1) Once the OCPs to tackle are properly formulated, apply the PMP to derive the TPBVP (e.g., systems of ODEs subject to some boundary conditions).

2) Approximate all the unknowns variables (e.g., states and costates) with the CEs (made by the sum of a free function, which is a shallow NN with random features, and a functional that analytically satisfied the TPBVP boundary conditions).

3) Plug the CEs into the TPBVP, and build the losses (which are represented by the residuals of the DEs forming the original TPBVP).

4) Train the shallow NN with random features, via (iterative) least squares, to learn the best possible approximation of the unknowns variables.

5) Once the NN is trained, plug the learned networks parameters (e.g., input weights and bias, which are randomly selected and not tuned, and the output weights learned via least squares) back into the CEs to derive the approximated unknowns variables.

III. Optimal Planar Orbit Transfer Problems

In this section, three typical optimal control problems for planar orbit transfer will be solved with our newly proposed PINN–TFC based method: 1) the maximum radius orbit transfer, 2) the minimum time orbit transfer, and 3) the solar sail minimum time orbit transfer. The problems have been coded in MATLAB® R2020a and run with an Intel Core i7 - 9700 CPU PC with 64 GB of RAM.

A. Maximum Radius Orbit Transfer

The orbit transfer is one of the best known trajectory optimization problems. The problem considered here takes into account circular and coplanar orbits, along with a two-body gravitational model (i.e., the sun is the primary body, and the spacecraft is the secondary body). Let x_1 be the radial distance toward the spacecraft from the sun, and then have x_2 and x_3 be the corresponding radial and tangential velocity, respectively. The dynamics related to the orbit transfer are expressed by the equations [39]

$$\dot{x}_1 = x_2 \tag{34}$$

$$\dot{x}_2 = \frac{x_3^2}{x_1} - \frac{\gamma}{x_1^2} + \frac{R_0 \sin(u)}{m_0 + \dot{m}t} \tag{35}$$

$$\dot{x}_3 = -\frac{x_2 x_3}{x_1} + \frac{R_0 \cos(u)}{m_0 + \dot{m}t} \tag{36}$$

with $t_0 \le t \le t_f$. It is important to note that Eqs. (35) and (36) are nonlinear. Thus, the time-variant dynamical system represented by Eqs. (34–36) is nonlinear. With the dynamical system defined previously, we must specify the cost function. Because our purpose is to maximize the final radius, the cost function to be minimized can be written as follows:

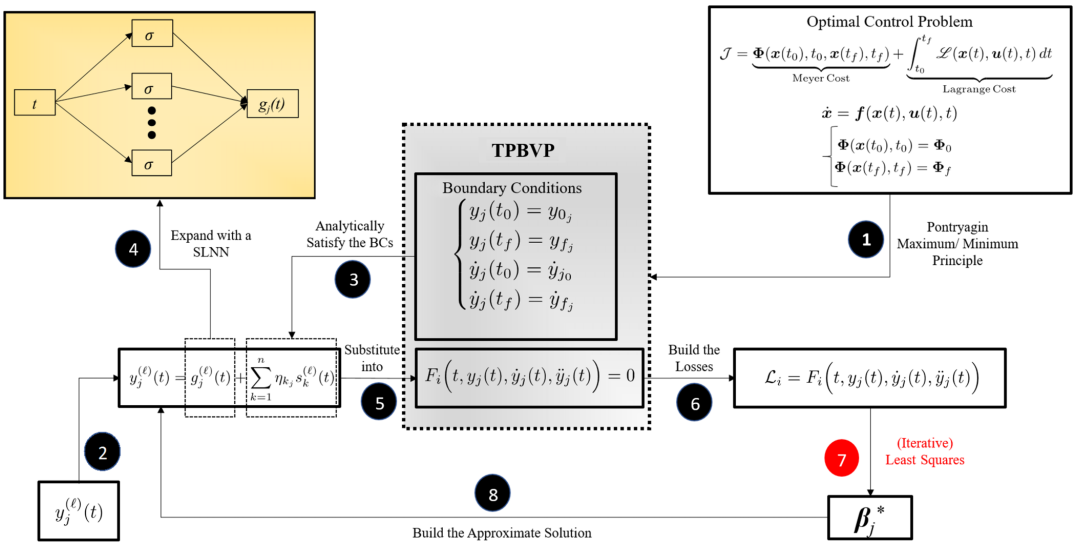


Fig. 1 Schematic summarizing how the proposed X-TFC based framework works for solving generic OCPs (SLNN = Single Layer Neural Network).

$$J = -x_{1,f} \tag{37}$$

In the previous equations, γ represents the gravitational attraction of the sun, R_0 is the constant low thrust magnitude, u is the thrust direction angle measured from the local horizon (i.e., the line perpendicular to the radial direction), m_0 is the initial mass, and $\dot{m}$ is a negative value corresponding to the mass flow rate. By calling $\phi(t) = R_0/m_0 + \dot{m}t$, the optimal control problem to solve is

$$\min \mathcal{J} = -x_{1,f} \quad \text{subject to:} \begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = \frac{x_3^2}{x_1} - \frac{\gamma}{x_1^2} + \phi(t)\sin(u) \\ \dot{x}_3 = -\frac{x_2 x_3}{x_1} + \phi(t)\cos(u) \\ t_0 \le t \le t_f \\ x_1(t_0) = x_{1,0} \\ \dot{x}_{1,0} = x_2(t_0) = x_{2,0} \\ x_3(t_0) = x_{3,0} \\ x_1(t_f) = x_{1,f} \\ \dot{x}_{1,f} = x_2(t_f) = x_{2,f} \\ x_{3,f} = (x_{1,f})^{-1/2} \end{cases} \tag{38}$$

The Hamiltonian is as follows:

$$H(t) = \lambda_1 x_2 + \lambda_2\left(\frac{x_3^2}{x_1} - \frac{\gamma}{x_1^2} + \phi\sin(u)\right) + \lambda_3\left(-\frac{x_2 x_3}{x_1} + \phi\cos(u)\right) \tag{39}$$

The optimal control is then retrieved from the Hamiltonian by taking the partial derivative with respect to the control u,

$$\frac{\partial H}{\partial u} = \lambda_2\cos(u) - \lambda_3\sin(u) = 0 \rightarrow \tan(u) = \frac{\lambda_2}{\lambda_3} \tag{40}$$

By plugging Eq. (40) into Eq. (39) and recalling the tangent identities $\cos(u) = 1/\sqrt{1+\tan^2(u)}$, and $\sin(u) = \tan(u)/\sqrt{1+\tan^2(u)}$, the Hamiltonian is reformulated as

$$H(t) = \lambda_1 x_2 + \lambda_2\left(\frac{x_3^2}{x_1} - \frac{\gamma}{x_1^2} + \frac{\phi\lambda_2}{(\lambda_2^2+\lambda_3^2)^{1/2}}\right) + \lambda_3\left(-\frac{x_2 x_3}{x_1} + \frac{\phi\lambda_3}{(\lambda_2^2+\lambda_3^2)^{1/2}}\right) \tag{41}$$

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

$$\dot{x}_1 = \frac{\partial H}{\partial \lambda_1} = x_2 \tag{42}$$

$$\dot{x}_2 = \frac{\partial H}{\partial \lambda_2} = \frac{x_3^2}{x_1} - \frac{\gamma}{x_1^2} + \frac{\phi\lambda_2}{(\lambda_2^2+\lambda_3^2)^{1/2}} \tag{43}$$

$$\dot{x}_3 = \frac{\partial H}{\partial \lambda_3} = -\frac{x_2 x_3}{x_1} + \frac{\phi\lambda_3}{(\lambda_2^2+\lambda_3^2)^{1/2}} \tag{44}$$

$$\dot{\lambda}_1 = -\frac{\partial H}{\partial x_1} = -\lambda_2\left[-\frac{x_3^2}{x_1^2} + \frac{2\gamma}{x_1^3}\right] - \frac{\lambda_3 x_2 x_3}{x_1^2} \tag{45}$$

$$\dot{\lambda}_2 = -\frac{\partial H}{\partial x_2} = -\lambda_1 + \frac{\lambda_3 x_3}{x_1} \tag{46}$$

$$\dot{\lambda}_3 = -\frac{\partial H}{\partial x_3} = -\frac{2\lambda_2 x_3}{x_1} + \frac{\lambda_3 x_2}{x_1} \tag{47}$$

According to the X-TFC framework, we approximate the state–costate pair with the constrained expressions,

$$x_1 = (\boldsymbol{\sigma} - \Omega_1\boldsymbol{\sigma}_0 - \Omega_2\boldsymbol{\sigma}_f - \Omega_3\boldsymbol{\sigma}_0' - \Omega_4\boldsymbol{\sigma}_f')^T\boldsymbol{\beta}_1 + \Omega_1 x_{1,0} + \Omega_2 x_{1,f} + \frac{\Omega_3\dot{x}_{1,0}}{b^2} + \frac{\Omega_4\dot{x}_{1,f}}{b^2}$$

$$\dot{x}_1 = b^2\left[(\boldsymbol{\sigma}\prime - \Omega_1'\boldsymbol{\sigma}_0 - \Omega_2'\boldsymbol{\sigma}_f - \Omega_3'\boldsymbol{\sigma}_0' - \Omega_4'\boldsymbol{\sigma}_f')^T\boldsymbol{\beta}_1 + \Omega_1' x_{1,0} + \Omega_2' x_{1,f} + \frac{\Omega_3'\dot{x}_{1,0}}{b^2} + \frac{\Omega_4'\dot{x}_{1,f}}{b^2}\right]$$

$$\ddot{x}_1 = b^4\left[(\boldsymbol{\sigma}'' - \Omega_1''\boldsymbol{\sigma}_0 - \Omega_2''\boldsymbol{\sigma}_f - \Omega_3''\boldsymbol{\sigma}_0' - \Omega_4''\boldsymbol{\sigma}_f')^T\boldsymbol{\beta}_1 + \Omega_1'' x_{1,0} + \Omega_2'' x_{1,f} + \frac{\Omega_3''\dot{x}_{1,0}}{b^2} + \frac{\Omega_4''\dot{x}_{1,f}}{b^2}\right]$$

$$x_3 = (\boldsymbol{\sigma} - \Omega_1\boldsymbol{\sigma}_0 - \Omega_2\boldsymbol{\sigma}_f)^T\boldsymbol{\beta}_3 + \Omega_1 x_{3,0} + \Omega_2 x_{3,f}$$

$$\dot{x}_3 = b^2[(\boldsymbol{h}' - \Omega_1'\boldsymbol{\sigma}_0 - \Omega_2'\boldsymbol{\sigma}_f)^T\boldsymbol{\beta}_3 + \Omega_1' x_{3,0} + \Omega_2' x_{3,f}]$$

$$\lambda_j = \boldsymbol{\sigma}^T\boldsymbol{\beta}_{\lambda_{x_j}}$$

$$\dot{\lambda}_j = b^2\boldsymbol{\sigma}'^T\boldsymbol{\beta}_{\lambda_{x_j}}$$

where $j = 1, 2, 3$ and the $\Omega_k(z)$ terms are the switching functions and are defined in Table A6, for x_1, and in Table A3, for x_3. One can note that the variable x_2 does not appear as a constrained expression, because it is the derivative of x_1. This also causes Eq. (42) to be removed because it is implicitly satisfied by the constrained expressions on x_1. Thus, the unknowns for the state–costate pair are

$$\boldsymbol{\beta} = \{\boldsymbol{\beta}_1 \quad \boldsymbol{\beta}_3 \quad \boldsymbol{\beta}_{\lambda_{x_1}} \quad \boldsymbol{\beta}_{\lambda_{x_2}} \quad \boldsymbol{\beta}_{\lambda_{x_3}}\}^T$$

If the final radial distance were known, an iterative least-square approach would be used to find the unknowns and minimize the following losses, which are the residuals of the arising TPBVP,

$$\mathcal{L}_{x,2} = \dot{x}_2 - \frac{x_3^2}{x_1} + \frac{\gamma}{x_1^2} - \frac{\phi\lambda_2}{(\lambda_2^2 + \lambda_3^2)^{1/2}} \tag{48}$$

$$\mathcal{L}_{x,3} = \dot{x}_3 + \frac{x_2 x_3}{x_1} - \frac{\phi\lambda_3}{(\lambda_2^2 + \lambda_3^2)^{1/2}} \tag{49}$$

$$\mathcal{L}_{\lambda_1} = \dot{\lambda}_1 + \lambda_2\left[-\frac{x_3^2}{x_1^2} - \frac{2\gamma}{x_1^3}\right] + \frac{\lambda_3 x_2 x_3}{x_1^2} \tag{50}$$

$$\mathcal{L}_{\lambda_2} = \dot{\lambda}_2 + \lambda_1 - \frac{\lambda_3 x_3}{x_1} \tag{51}$$

$$\mathcal{L}_{\lambda_3} = \dot{\lambda}_3 + \frac{2\lambda_2 x_3}{x_1} - \frac{\lambda_3 x_2}{x_1} \tag{52}$$

The $\mathcal{L}_{x,1}$ loss related to Eq. (42) is not considered because it is redundant. Because the final radial distance is free, the following transversality condition has to be imposed on the costate λ_1,

$$\lambda_1(t_f) + 1 = 0 \tag{53}$$

Thus, the constrained expressions remain the same as those expressed before, apart from the expression of λ_1, which now becomes

$$\lambda_1 = (\boldsymbol{\sigma} - \boldsymbol{\sigma}_f)^T\boldsymbol{\beta}_{\lambda_1} + \lambda_1(t_f) \tag{54}$$

Thus, the complete set of unknowns becomes

$$\boldsymbol{\Xi} = \{\boldsymbol{\beta}_1 \quad \boldsymbol{\beta}_3 \quad \boldsymbol{\beta}_{\lambda_1} \quad \boldsymbol{\beta}_{\lambda_2} \quad \boldsymbol{\beta}_{\lambda_3} \quad x_1(t_f)\}^T$$

An iterative least-square approach is used to find the unknowns and minimize the losses, which are the same as Eqs. (48–52).

1. Numerical Results

Using the normalized quantities proposed in [39], we have $\gamma = 1$, $R_0 = 0.1405$, $m_0 = 1$, and $\dot{m} = -0.07487$. According to this normalization, the time unit is 58.18 days. The fixed final time is equal to $t_f = 3.32$ (193.1576 days according to the normalization time unit). The boundary conditions are

$$x_1(0) = 1.0, \qquad x_2(0) = 0.0, \qquad x_3(0) = 1.0 \tag{55}$$

$$x_2(t_f) = 0.0, \qquad x_3(t_f) = x_1(t_f)^{-1/2} \tag{56}$$

A logistic activation function was used to solve this OCP with X-TFC. Input weights and bias were sampled from a uniform distribution $\mathcal{U}[-1, 1]$. Because the problem is nonlinear, an initial guess on the unknowns is required to initialize the iterative least square. As a first guess, all the unknowns $\boldsymbol{\Xi}$ have been randomly sampled from $\mathcal{U}[0; 1]$. The solution learned by the X-TFC for this type of problem when the number of training points N is 50 and the amount of neurons L is 15 is shown in Figs. 2 and 3. The subfigures within Fig. 2 show time history of the solution's states and control, while Fig. 3 shows the entire trajectory with thrusting direction. In Table 1, we report the performances of our algorithm for different values of N and L, while all the other hyperparameters (e.g., activation function and probability distribution from where input weights and biases are sampled) are kept fixed. From Table 1, it can be noticed that the results are not very sensitive to the number of N and L. The fact that the accuracy of the solution is not to machine level, as in other works where the authors employ TFC based algorithm to face OCPs for space applications, can be attributed to the strong nonlinear nature of the dynamical system governing this problem.

In Ref. [39], the same problem is solved via a direct method, where state and control variables are expanded in the Chebyshev series. The maximum final radius found in [39] is 1.521261 with a dynamical error of $\mathcal{O}(10^{-2})$, with the computational time of approximately 9 s. From Table 1, it is evident that our results are better for any choice of N and L than the results presented in [39] regarding the precision of the dynamics. Moreover, in [39], the authors report that the maximum error on the boundary conditions is of $\mathcal{O}(10^{-13})$, while in our case, it is analytically satisfied because of the properties of the X-TFC methodology. Additionally, the computational time needed to solve the problem with our algorithm is three orders of magnitude lower than the computational time needed in [39]. For this reason, our algorithm could be potentially suitable for real-time applications for this kind of orbit transfer problem. Moreover, as a further comparison, the software GPOPS-II has been run for this problem. The computed maximum radius was 1.5189, which is lower than the values found with the proposed method and reported in Table 1.

B. Minimum Time Orbit Transfer

For this problem, because just the final time is free, we have no transversality conditions on the costate, but we have the following transversality condition on the Hamiltonian at final time:

$$H(t_f) + 1 = 0 \tag{57}$$

Thus, the unknowns are

$$\boldsymbol{\Xi} = \{\boldsymbol{\beta}_1 \quad \boldsymbol{\beta}_3 \quad \boldsymbol{\beta}_{\lambda_{x_1}} \quad \boldsymbol{\beta}_{\lambda_{x_2}} \quad \boldsymbol{\beta}_{\lambda_{x_3}} \quad b\}^T$$

1. Numerical Results

Using the normalized quantities proposed in [39], we have $\gamma = 1$, $R_0 = 0.1405$, $m_0 = 1$, and $\dot{m} = -0.07487$. According to this normalization, the time unit is 58.18 days. For what concerns the initial and final conditions, the spacecraft is assumed to depart from Earth's orbit with velocity only in the tangential direction and to arrive at Mars's orbit with velocity only in the tangential direction, too. This leads to the following boundary conditions:

$$x_1(0) = 1.0, \qquad x_2(0) = 0.0, \qquad x_3(0) = 1.0 \tag{58}$$

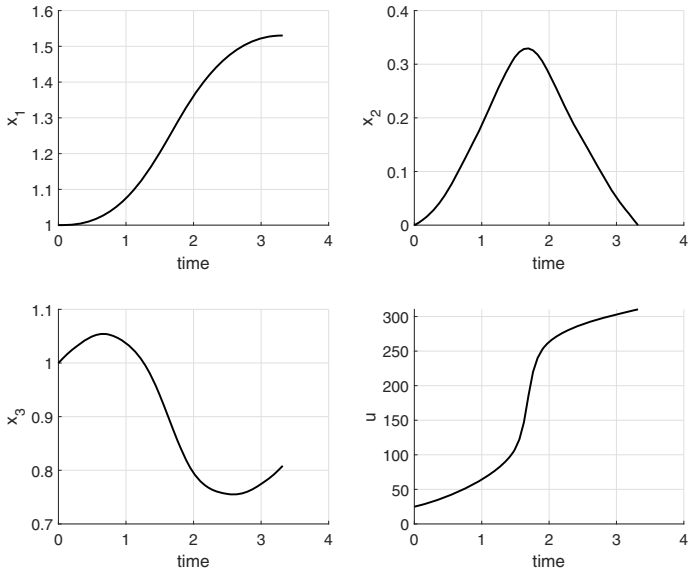


Fig. 2 Time history of states and control for the maximum radius orbit transfer problem computed with $N = 50$, and $L = 15$. The quantities in the plots are dimensionless.

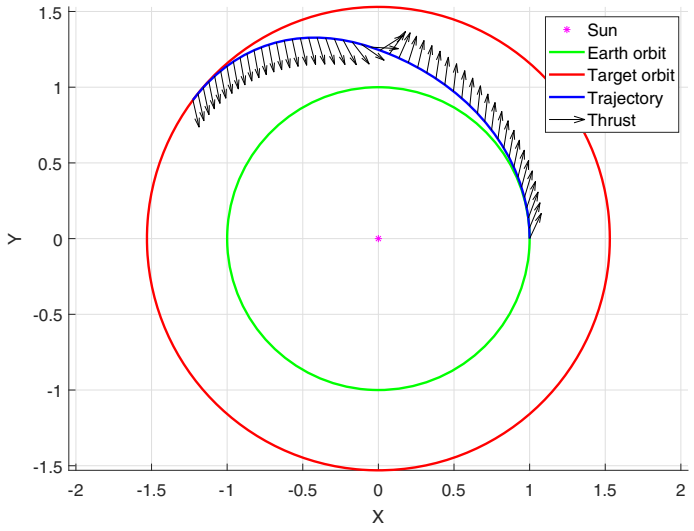


Fig. 3 Trajectory of the maximum radius orbit transfer problem computed with $N = 50$, and $L = 15$ (the thrust is represented in the heliocentric frame). The quantities in the plot are dimensionless.

$$x_1(t_f) = 1.525, \qquad x_2(t_f) = 0.0, \qquad x_3(t_f) = 0.8098 \tag{59}$$

The activation function used is a logistic. Input 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. For this problem, as a first guess, the unknowns $\boldsymbol{\beta}$ of all the constrained expressions have been randomly sampled from $\mathcal{U}[0; 1]$, while the guess on the final time is set equal to 1 and then transformed into the guess on the b parameter thanks to the mapping coefficient. Similar to the previous subsection, Figs. 4 and 5 show the X-TFC solver's solution for the minimum time orbit transfer problem. The selection of N is still 50, but the selection of L was changed to 20. In Table 2, we report the performances of our algorithm for different values of N and L, while all the other hyperparameters are kept fixed. As can be seen, there is no evidence of an improvement of the solution accuracy with greater numbers of neurons and/or discretization points. Indeed, the problem seems to be better solved with low number of N and L. From Table 2, it can be observed that the results are not very sensitive to the number of N and L. The fact that the accuracy of the solution is not to machine level can be attributed even for this case to the strong nonlinear nature of the dynamical system

Table 1 Summary of the performances of the proposed PINN based algorithm in solving the maximum radius orbit transfer for different values of N and L (the final radial distance is dimensionless)

N	L	No. of iterations	CPU time, s	Mean($\lvert\mathbb{L}\rvert$)	$x_1(t_f)$
20	7	52	0.014	1.1×10^{-2}	1.5316
20	10	40	0.014	6.0×10^{-3}	1.5200
20	12	39	0.015	5.5×10^{-3}	1.5306
30	12	45	0.020	5.6×10^{-3}	1.5297
30	15	27	0.014	5.3×10^{-3}	1.5318
50	15	34	0.023	5.4×10^{-3}	1.5303

governing the problem, just like for the maximum radius orbit transfer problem.

In [39], the same problem is solved via a direct method, where state and control variables are expanded in the Chebyshev series. The minimum time found in [39] is 3.3187 (193.0843 days according to the normalization time unit) with a dynamical error of $\mathcal{O}(10^{-2})$. The computational time is around 9 s. From Table 2, it can be noticed that our results are in each case better than the results presented in [39]. Moreover, in [39], the authors report that the maximum error on the boundary conditions is of $\mathcal{O}(10^{-13})$, while in our case, it is analytically satisfied because of the properties of the X-TFC methodology. Additionally, the computational time needed to solve the problem with our algorithm is three orders of magnitude lower than the

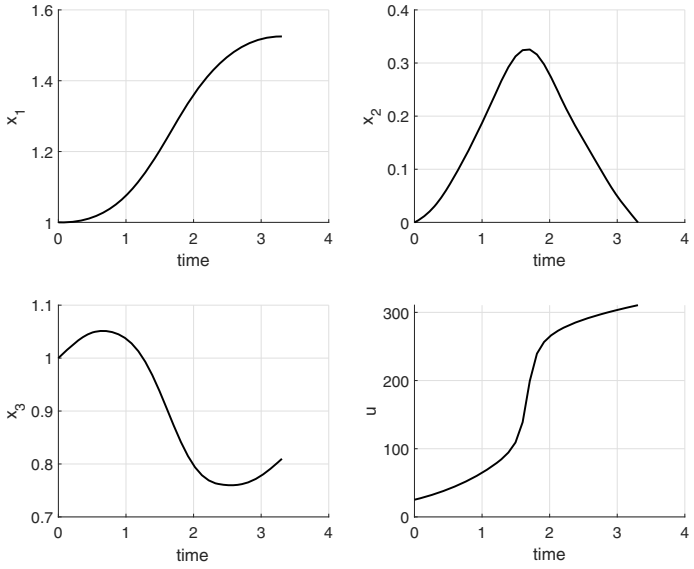


Fig. 4 Time history of states and control for the minimum time orbit transfer problem computed with $N = 50$, and $L = 20$. The quantities in the plots are dimensionless.

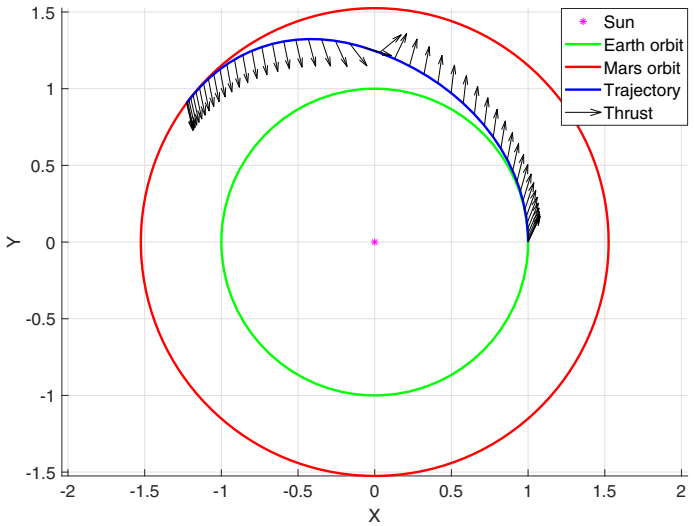


Fig. 5 Trajectory of the minimum time orbit transfer problem computed with $N = 50$, and $L = 20$ (the thrust is represented in the heliocentric frame). The quantities in the plot are dimensionless.

Table 2 Summary of the performances of the proposed PINN based algorithm in solving the minimum time orbit transfer for different values of N and L (the final time is dimensionless)

N	L	No. of iterations	CPU time, s	Mean($\lvert\mathbb{L}\rvert$)	$H(t_f)+\Gamma$	t_f
20	7	44	0.012	1.5×10^{-2}	2.0×10^{-2}	3.2913
20	10	39	0.014	5.5×10^{-3}	5.4×10^{-4}	3.3088
20	12	28	0.012	4.3×10^{-3}	1.1×10^{-4}	3.3061
30	12	43	0.019	4.6×10^{-3}	2.1×10^{-4}	3.3047
30	15	35	0.018	4.5×10^{-3}	1.7×10^{-4}	3.3058
50	20	30	0.031	4.5×10^{-3}	2.6×10^{-4}	3.3069

computational time needed in [39]. For this reason, for this kind of orbit transfer problems, our algorithm could be potentially suitable for real-time applications. To validate our results, we have compared them with the ones generated with the software GPOPS-II [7]. The results computed with GPOPS-II are $t_f = 3.3194$, $H(t_f) = 6.5\times10^{-3}$, and CPU time of approximately 30 s. As can be seen, our results are comparable with those of GPOPS-II, proving the reliability of the proposed PINN based method.

C. Solar Sail Minimum Time Orbit Transfer

In this section, a solar sail is considered for an optimal orbit transfer between two circular and coplanar orbits. Beause of these assumptions, even in this case, it is convenient to express the equations of motion in radial and tangential components (as it was for the two previous examples) rather than classical Cartesian components. There are multiples models to represent a solar sail. Here, the ideal solar sail model is employed, which means that it is considered as a perfect mirror with a flat surface. The following preliminary considerations have to be made:

1) Solar radiation pressure (SRP) acts on the spacecraft's center of mass.

2) The perfect mirror model assumes only specular reflection of sunlight (the value of tangential acceleration due to SRP is supposed always zero).

3) The change in the orientation of the solar sail is neglected and not considered in this work (the change of the acceleration is abrupt).

4) The spacecraft is only subject to the solar gravity and SRP.

A representation of the ideal solar sail is provided in Fig. 6. As can be seen, the control of the solar sail is due to the cone angle α, which is the angle between the radial direction of the sunlight and the normal $\boldsymbol{n}$ of the sail [40]. Through computation, this angle can take values within the range $[-\pi/2, \pi/2]$, where the limit values correspond to the sail aligned with the radial direction. Note that the cone angle aligned exactly with the radial direction (i.e., $-\pi/2$ or $\pi/2$) is not considered feasible because then the employed model of the sail would not provide any acceleration. It is also important to understand that a positive value of α causes an outward spiralling, whereas a negative value of α causes an inward spiralling with respect to the departing point [41].

The SRP acceleration acting on a solar sail is given by [41]

$$\boldsymbol{a}_{\mathrm{SRP}} = 2\frac{W_E}{c_L}\frac{R_E^2}{r^2}\frac{(\cos\alpha)^2}{\kappa_{\mathrm{ss}}}\hat{\boldsymbol{n}} = \frac{\beta_s}{r^2}(\cos\alpha)^2\hat{\boldsymbol{n}} \tag{60}$$

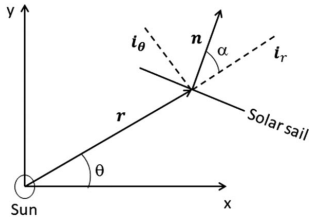


Fig. 6 Solar sail representation.

where W_E is the solar constant (1367 W/m^2), c_L is the speed of light (300,000 km/s), R_E is the mean distance from the sun to the Earth ($149.6\cdot10^6$ km), r is the actual distance of the sail from the sun, and κ_{ss} is the mass-to-area ratio of the sailcraft (the mass is meant to be the total mass of the spacecraft, whereas the area is referred to just the area of the sail). Considering $x_1 = r$, $x_2 = \theta$, $x_3 = v_r$ (radial velocity), and $x_4 = v_\theta$ (tangential velocity), the equations of motion are

$$\dot{x}_1 = x_3 \tag{61}$$

$$\dot{x}_2 = \frac{x_4}{x_1} \tag{62}$$

$$\dot{x}_3 = \frac{x_4^2}{x_1} - \frac{\mu_S}{x_1^2} + \frac{\beta_s}{x_1^2}(\cos\alpha)^3 \tag{63}$$

$$\dot{x}_4 = -\frac{x_3x_4}{x_1} + \frac{\beta_s}{x_1^2}(\cos\alpha)^2\sin\alpha \tag{64}$$

where μ_{s} is the gravitational parameter of the sun ($1.327\cdot10^{11}$ km^3/s^2). Note that Eqs. (61–64) are very similar to Eqs. (34–36). The only differences between the two equations of motion are the control terms because of the different propulsion systems. The goal for the solar sail problem is to minimize the time of flight. Therefore, the cost function is

$$J = \Gamma t_f \tag{65}$$

with Γ representing a weight parameter. The boundary conditions are

$$\begin{cases} x_1(t_0) = x_{1,0} \\ x_2(t_0) = x_{2,0} \\ x_3(t_0) = \dot{x}_1(t_0) = x_{3,0} \\ x_4(t_0) = x_{4,0} \\ x_1(t_f) = x_{1,f} \\ x_3(t_f) = x_{3,f} \\ x_4(t_f) = x_{4,f} \end{cases} \tag{66}$$

The Hamiltonian is

$$H(t) = \lambda_1x_3 + \lambda_2\frac{x_4}{x_1} + \lambda_3\left(\frac{x_4^2}{x_1} - \frac{\mu_s}{x_1^2} + \frac{\beta_s}{x_1^2}(\cos\alpha)^3\right) + \lambda_4\left(-\frac{x_3x_4}{x_1} + \frac{\beta_s}{x_1^2}(\cos\alpha)^2\sin\alpha\right) \tag{67}$$

The optimal control is then retrieved from the Hamiltonian by taking the partial derivative with respect to the control variable α,

$$\frac{\partial H}{\partial\alpha} = -3\frac{\lambda_3\beta_s}{x_1^2}(\cos\alpha)^2\sin\alpha + \frac{\lambda_4\beta_s}{x_1^2}(-2\cos\alpha(\sin\alpha)^2 + (\cos\alpha)^3) = 0 \tag{68}$$

$$(\tan\alpha)^2 + \frac{3\lambda_3}{2\lambda_4}\tan\alpha - \frac{1}{2} = 0 \tag{69}$$

To compute the expression for the cone angle, a division by $(\cos\alpha)^3$ has to be performed. This division can only occur if $\cos\alpha \neq 0$. Hence, as previously stated, $\alpha = -\pi/2$ or $\alpha = \pi/2$ is not feasible. Thus, we do not consider these values of α. The general solution for the optimal value of α which minimizes the Hamiltonian is given by [42],

$$\alpha = \begin{cases} \arctan\left(\frac{-3\lambda_3 - \sqrt{9\lambda_3^2+8\lambda_4^2}}{4\lambda_4}\right) & \text{if } \lambda_4 \neq 0 \\ 0 & \text{if } \lambda_4 = 0, \lambda_3 \leq 0 \\ \pm\pi/2 & \text{if } \lambda_4 = 0, \lambda_3 > 0 \end{cases} \tag{70}$$

The first-order necessary conditions of optimality can then be obtained from the Hamiltonian and are as follows:

$$\dot{x}_1 = \frac{\partial H}{\partial \lambda_1} = x_3 \tag{71}$$

$$\dot{x}_2 = \frac{\partial H}{\partial \lambda_2} = \frac{x_4}{x_1} \tag{72}$$

$$\dot{x}_3 = \frac{\partial H}{\partial \lambda_3} = \frac{x_4^2}{x_1} - \frac{\mu_s}{x_1^2} + \frac{\beta_s}{x_1^2}(\cos\alpha)^3 \tag{73}$$

$$\dot{x}_4 = \frac{\partial H}{\partial \lambda_4} = -\frac{x_3 x_4}{x_1} + \frac{\beta_s}{x_1^2}(\cos\alpha)^2 \sin\alpha \tag{74}$$

$$\dot{\lambda}_1 = -\frac{\partial H}{\partial x_1} = \frac{\lambda_2 x_4}{x_1^2} - \lambda_3\left(-\frac{x_4^2}{x_1^2} + \frac{2\mu_s}{x_1^3} - \frac{2\beta_s}{x_1^3}(\cos\alpha)^3\right) - \lambda_4\left(\frac{x_3 x_4}{x_1^2} - \frac{2\beta_s}{x_1^3}(\cos\alpha)^2 \sin\alpha\right) \tag{75}$$

$$\dot{\lambda}_2 = -\frac{\partial H}{\partial x_2} = 0 \tag{76}$$

$$\dot{\lambda}_3 = -\frac{\partial H}{\partial x_3} = -\lambda_1 + \frac{\lambda_4 x_4}{x_1} \tag{77}$$

$$\dot{\lambda}_4 = -\frac{\partial H}{\partial x_4} = -\frac{\lambda_2}{x_1} - \frac{2\lambda_3 x_4}{x_1} + \frac{\lambda_4 x_3}{x_1} \tag{78}$$

Because the final time is free and the orbit rendezvous problem is considered (e.g., the final anomaly θ is free), the following transversality conditions have to be imposed:

$$H(t_f) + \Gamma = 0 \tag{79}$$

$$\lambda_2(t_f) = 0 \tag{80}$$

Now, according to the X-TFC method, the constrained expressions can be written as

$$x_1 = (\boldsymbol{\sigma} - \Omega_1\boldsymbol{\sigma}_0 - \Omega_2\boldsymbol{\sigma}_f - \Omega_3\boldsymbol{\sigma}_0' - \Omega_4\boldsymbol{\sigma}_f')^T\boldsymbol{\beta}_1 + \Omega_1 x_{1,0} + \Omega_2 x_{1,f} + \frac{\Omega_3\dot{x}_{1,0}}{b^2} + \frac{\Omega_4\dot{x}_{1,f}}{b^2}$$

$$\dot{x}_1 = x_3 = b^2\left[(\boldsymbol{\sigma}' - \Omega_1'\boldsymbol{\sigma}_0 - \Omega_2'\boldsymbol{\sigma}_f - \Omega_3'\boldsymbol{\sigma}_0' - \Omega_4'\boldsymbol{\sigma}_f')^T\boldsymbol{\beta}_1 + \Omega_1' x_{1,0} + \Omega_2' x_{1,f} + \frac{\Omega_3'\dot{x}_{1,0}}{b^2} + \frac{\Omega_4'\dot{x}_{1,f}}{b^2}\right]$$

$$\ddot{x}_1 = \dot{x}_3 = b^4\left[(\boldsymbol{\sigma}'' - \Omega_1''\boldsymbol{\sigma}_0 - \Omega_2''\boldsymbol{\sigma}_f - \Omega_3''\boldsymbol{\sigma}_0' - \Omega_4''\boldsymbol{\sigma}_f')^T\boldsymbol{\beta}_1 + \Omega_1'' x_{1,0} + \Omega_2'' x_{1,f} + \frac{\Omega_3''\dot{x}_{1,0}}{b^2} + \frac{\Omega_4''\dot{x}_{1,f}}{b^2}\right]$$

$$x_2 = (\boldsymbol{\sigma} - \Omega_1\boldsymbol{\sigma}_0)^T\boldsymbol{\beta}_2 + \Omega_1 x_{2,0}$$

$$\dot{x}_2 = b^2[(\boldsymbol{\sigma}' - \Omega_1'\boldsymbol{\sigma}_0)^T\boldsymbol{\beta}_2 + \Omega_1' x_{2,0}]$$

$$x_4 = (\boldsymbol{\sigma} - \Omega_1\boldsymbol{\sigma}_0 - \Omega_2\boldsymbol{\sigma}_f)^T\boldsymbol{\beta}_4 + \Omega_1 x_{4,0} + \Omega_2 x_{4,f}$$

$$\dot{x}_4 = b^2[(\boldsymbol{\sigma}' - \Omega_1'\boldsymbol{\sigma}_0 - \Omega_2'\boldsymbol{\sigma}_f)^T\boldsymbol{\beta}_4 + \Omega_1' x_{4,0} + \Omega_2' x_{4,f}]$$

$$\lambda_j = \boldsymbol{\sigma}^T\boldsymbol{\beta}_{\lambda_j}$$

$$\dot{\lambda}_j = b^2\boldsymbol{\sigma}'^T\boldsymbol{\beta}_{\lambda_j}$$

where $j = 1, 3, 4$ and the $\Omega_k(z)$ terms are the switching functions and are defined in the Appendix according to the number and the type of imposed constraints. One can note that the variable x_3 does not appear as a constrained expression because it is the derivative of x_1. This also causes Eq. (71) to be removed because it is implicitly satisfied by the constrained expressions on x_1. We also know, from Eq. (76), that λ_2 is a constant value for this problem. Thus, λ_2 is set to 0 by the transversality condition at final time shown in Eq. (80), which is then removed from the equations constituting the TPBVP. We therefore do not need to introduce a constrained expression for λ_2. Hence, the unknowns are

$$\boldsymbol{\Xi} = \{\boldsymbol{\beta}_1 \quad \boldsymbol{\beta}_2 \quad \boldsymbol{\beta}_4 \quad \boldsymbol{\beta}_{\lambda_1} \quad \boldsymbol{\beta}_{\lambda_3} \quad \boldsymbol{\beta}_{\lambda_4} \quad b\}^T$$

An iterative least-square approach can be used to find the unknowns and minimize the losses, which can be retrieved from Eqs. (72–75) and (77–79), as already explained in the previous examples.

1. Numerical Results

To perform the orbit transfer, a value of κ_{ss} equal to 15 $\mathrm{g/m^2}$ was chosen so that the parameter β_s can be computed. For what concerns the initial and final conditions, the spacecraft was assumed to depart from the orbit of the Earth, supposedly circular, with only tangential velocity and to arrive at the circular orbit of Mars with the only tangential velocity. This leads to the boundary conditions

$$x_1(0) = R_E, \quad x_2(0) = 0, \quad x_3(0) = 0, \quad x_4(0) = \sqrt{\frac{\mu_s}{R_E}} \tag{81}$$

$$x_1(t_f) = R_M, \qquad x_3(t_f) = 0, \qquad x_4(t_f) = \sqrt{\frac{\mu_s}{R_M}} \tag{82}$$

where $R_M = 227.94 \cdot 10^6$ km corresponds to the radius of the Mars orbit (supposed circular and coplanar with the orbit of Earth). If dimensionless quantities are employed, we obtain $\mu_s = 1$, a length unit equal to 1 Astronomical Unit (AU) (R_E), a time unit equal to 58.1364 days, and consequently $\beta_s = 0.10246564$. The activation function used for this problem is a logistic, whereas input weights and biases are sampled from $\mathcal{U}[-10, 10]$. The first guesses of the unknowns $\boldsymbol{\beta}$ of all the constrained expressions have been randomly sampled from $\mathcal{U}[0; 1]$. One will note that, despite the random initialization of the coefficients, the algorithm is still able to converge to a feasible solution. However, because of the complexity of the problem, the methodology employed previously for the minimum time orbit transfer problem, in other words, to solve for the final time together with the other unknowns in one same loop, did not lead to any feasible solutions. For this reason, and considering that accurate solutions were found if the time of flight was fixed, a two-loop based procedure is exploited for this case. A similar optimization approach based on two loops has already been exploited in [29], in which TFC was used within a nonlinear equation solver. Even in this case, the inner loop solves the fixed-time problem via X-TFC, whereas the outer loop, based on the Unified Particle Swarm Optimization (UPSO) algorithm [43], takes into account the optimization of the time of flight. Therefore, the transversality condition of Eq. (79) was no longer considered in the inner loop, but it was actually considered in the cost function to minimize within the UPSO. In particular, the UPSO cost function is represented by the sum between the norm of the entire loss vector and the norm of the Hamiltonian function plus Gamma $[H(t) + \Gamma]$ considering all the time instants. Indeed, it has been noticed that enforcing the algorithm to minimize the norm of the Hamiltonian, instead of simply considering the transversality condition on the Hamiltonian at final time [shown in Eq. (79)], helps the convergence of the algorithm to the solution. For the UPSO, the number of particles and maximum iterations were set to 10 and 100, respectively. Moreover, the UPSO based optimization ends when the relative error between the best cost functions of two successive iterations is below a user-defined threshold (here set to 10^{-6}) or when it does not change for ten successive iterations. For what concerns the search space on the final time, it was set equal to the interval [480,520] days and then transformed into the the b parameter thanks to the mapping coefficient. By setting the parameter Γ in the cost

function equal to 1, $N = 100$ and $L = 100$, the algorithm finds the solution reported in Figs. 7–11. The optimal solution led to a time of flight of 499.77 days and the following performances: $|H(t_f) + \Gamma| = 6.13 \cdot 10^{-4}$ and mean$|\mathbb{L}| = 2.64 \cdot 10^{-5}$. Figure 11 also reports the losses and the function $|H(t) + \Gamma|$ associated to each discretization point, which demonstrate the precision and optimality of the obtained results. Moreover, as can be seen from Table 3, the results obtained by running the same problems with the commercial software GPOPS-II are very similar, thus proving the reliability of the proposed framework. It is important to highlight again that the boundary conditions are here perfectly matched, unlike the other works in literature in which some tolerances on final position and velocity have to be introduced.

IV. Discussions

The results presented in this paper show that the proposed PINN based framework is able to effectively learn the optimal control for the class of planar orbit transfer problems. In particular, the framework converges to the solution even for a random initialization of the unknowns, which is a significant advantage because we know how the results of the optimization can really be affected by the choice of the initial guesses when more traditional deterministic algorithms are used.

Importantly, once the optimal control is learned on the training points, for a specified target, we have an analytical representation (by NNs) of it. Therefore, it can be evaluated in points unseen during the

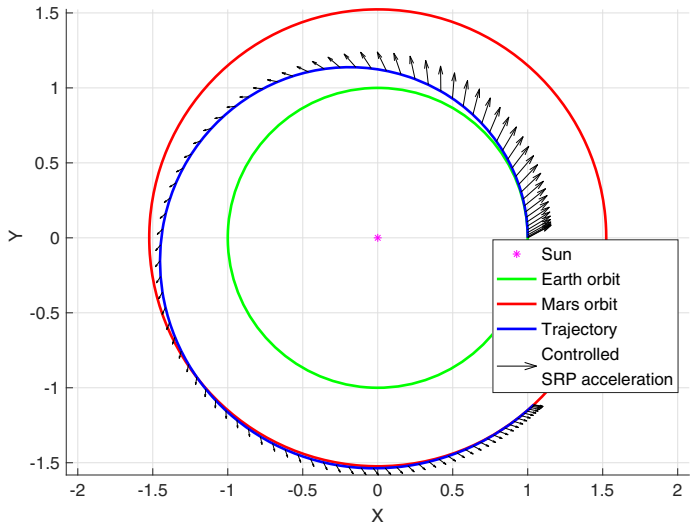


Fig. 7 Solar sail trajectory computed with $N = 100$, and $L = 100$ (dimensionless units are employed).

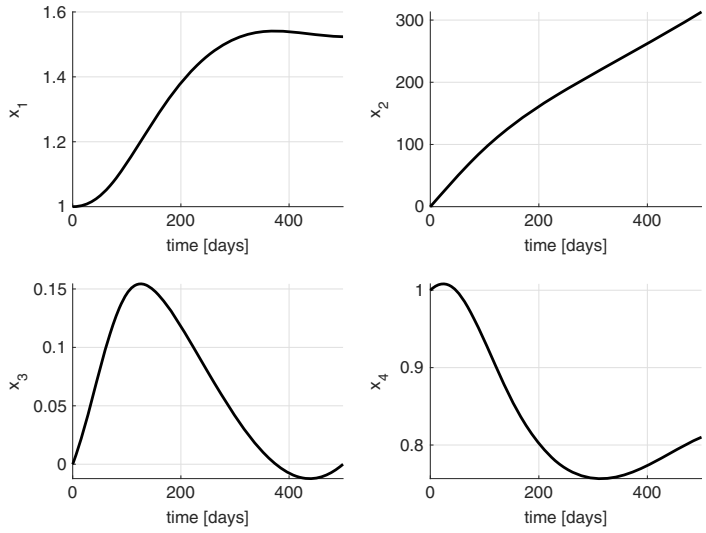


Fig. 8 Time history of states for the solar sail minimum time orbit transfer problem computed with $N = 100$, and $L = 100$ (dimensionless units are employed for the states).

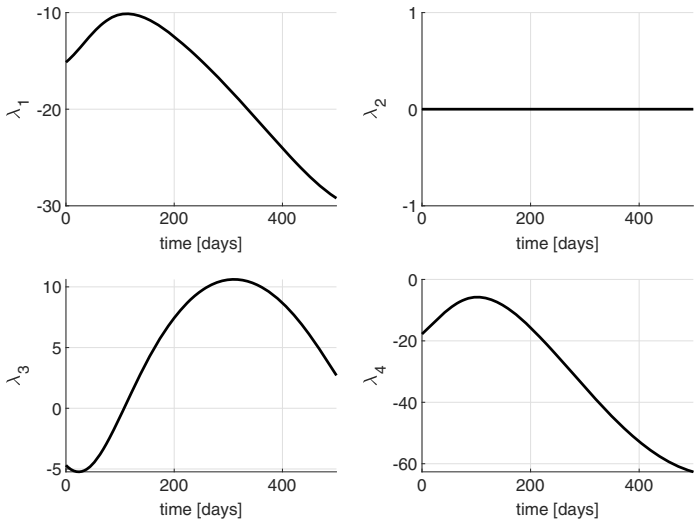


Fig. 9 Time history of costates for the solar sail minimum time orbit transfer problem computed with $N = 100$ and $L = 100$.

training (e.g., test points) without recurring to interpolation techniques and/or with no additional computational efforts. Nevertheless, if the target changes, retraining is required. However, the proposed approach features a low training time, which may be conducive to real-time guidance; in other words, guidance occurs by closing the loop via a sequence of open loops. As demonstrated previously, the approach is very efficient in terms of computational time and cost, making the methodology potentially suitable for real-time applications.

Furthermore, the results are comparable with GPOPS-II, which is a commonly adopted tool for optimization of space problems, thus proving the reliability and efficiency of the proposed method.

Although the NN was trained solely in a physics-driven fashion to generate the results, the training can be carried out in a data-physics-driven manner, as previously mentioned. That is, thanks to this unique feature, the proposed PINN based framework could be used solo or in synergy with other state-of-the-art methods for OCPs. For instance, one in principle could use our method in synergy with GPOPS-II, where the problem to tackle is sufficiently complex and both the methods alone would struggle. For example, consider a case where both methods converge to a solution whose accuracy is not excellent.

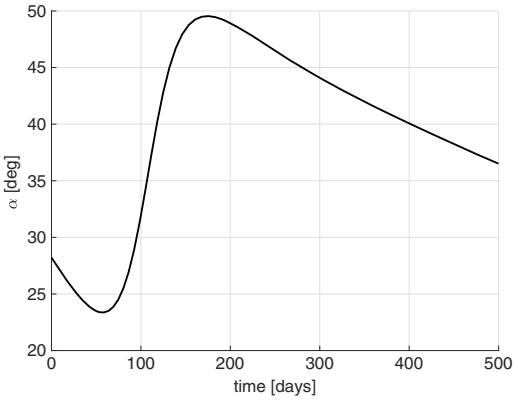


Fig. 10 Time history of the control angle for the solar sail minimum time orbit transfer problem computed with $N = 100$, and $L = 100$.

The GPOPS-II converged solution's accuracy may be higher than the X-TFC one on its own. In this scenario, the solution generated by GPOPS-II can be sampled as data, which can be added as additional terms in the PINN based method's loss function to better drive the learning with a data-physics-driven fashion. In doing so, one could obtain a X-TFC solution whose accuracy is better than the accuracy of the solutions obtained using the methods on their own. Furthermore, 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.

V. Conclusions

We introduced a novel, accurate, and reliable PINN based algorithm for solving optimal planar orbit transfer problems via the indirect method. This algorithm uses the newly developed PINN–TFC based method named Extreme Theory of Functional Connections. More specifically, the goal of this method is to model a NN representation of the optimal control via learning the state–costate pair, which is the solution of the TPBVP arising from the application of the PMP. The states and costates are approximated via the TFC's CEs, where the free function is a single-layer NN trained via the ELM algorithm. An advantage of using the proposed PINN based algorithm is the convergence to the optimal solution even for random initial guesses. This is particularly true for fixed-time solutions, whereas it is not generally true for free final time problems, as shown by the solar sail minimum time orbit transfer problem, where a two-loop optimization procedure has been introduced to achieve the convergence to the optimal solution. However, as the numerical results clearly show, the proposed algorithm is able to solve all the proposed problems and learn the optimal control with good accuracy. This is also proved by the comparison with the results obtained via GPOPS-II. Another significant advantage of the proposed framework is that the network training can be carried out in a data-physics-driven manner. This unique feature could further increase the robustness of this new PINN based method. Indeed, optimal trajectories could be generated with any other state-of-the-art method for solving OCPs and then sampled to generate data, which can be added as additional terms in the PINN based method's loss function to better drive the learning with a data-physics-driven fashion. Moreover, 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. As for other future applications of PINNs for OCPs, the authors are currently working on

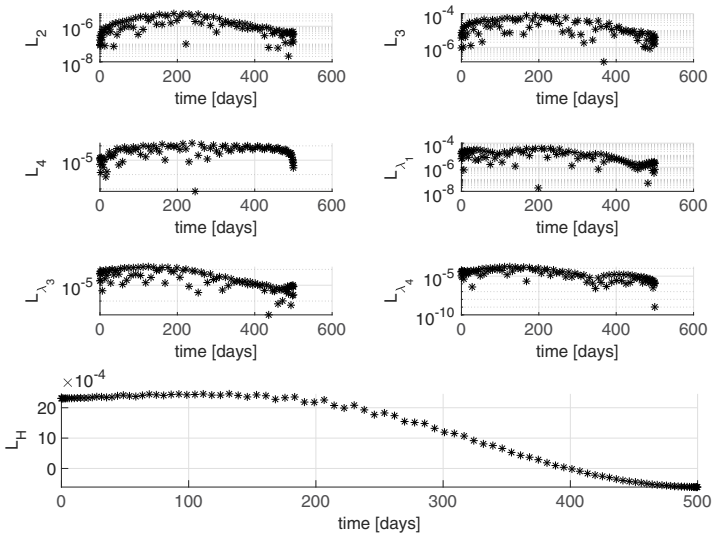


Fig. 11 Time history of the losses for the solar sail minimum time orbit transfer problem computed with $N = 100$, and $L = 100$.

Table 3 Comparisons of the performances between the proposed PINN algorithm and GPOPS-II in solving the solar sail minimum time orbit transfer with $\Gamma = 1$, $N = 100$, and $L = 100$

Method	t_f, days	λ_0	λ_f
GPOPS-II	498.917	[−15.1581, 0, −4.6952, −17.8337]	[−29.2142, 0, 2.7797, −62.7025]
X-TFC	499.777	[−15.1375, 0, −4.6920, −17.7784]	[−29.2106, 0, 2.7077, −62.6414]

making the framework more general, for example, by extending this framework to other three-dimensional space applications, including also state and control constraints. In addition, other OCPs with cost functions leading to discontinuous optimal control, such as fuel-optimal problems, will be considered.

Appendix A: Switching Functions

The detailed derivation of the switching functions is reported in the work of Johnston et al. [29]. 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

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

	Initial/final value
	$\Omega_1(z)$
$(\cdot)$	1
$\frac{d}{dz}(\cdot)$	0

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

	Initial/final value
	$\Omega_1(z)$
$(\cdot)$	z
$\frac{d}{dz}(\cdot)$	1

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

	Initial value	Final value
	$\Omega_1(z)$	$\Omega_2(z)$
$(\cdot)$	$\frac{z_f - z}{\Delta z}$	$\frac{z - z_0}{\Delta z}$
$\frac{d}{dz}(\cdot)$	$-\frac{1}{\Delta z}$	$\frac{1}{\Delta z}$
$\frac{d^2}{dz^2}(\cdot)$	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	Initial derivative value
	$\Omega_1(z)$	$\Omega_2(z)$
$(\cdot)$	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 the domain of $z \in [z_0, z_f]$

	Initial value	Final value	Initial derivative
	$\Omega_1(z)$	$\Omega_2(z)$	$\Omega_3(z)$
$(\cdot)$	$\frac{(z_f-z)(z-2z_0+z_f)}{\Delta z^2}$	$\frac{(z-z_0)^2}{\Delta z^2}$	$\frac{(z-z_0)(z_f-z)}{\Delta z}$
$\frac{d}{dz}(\cdot)$	$\frac{-2(z-z_0)}{\Delta z^2}$	$\frac{2(z-z_0)}{\Delta z^2}$	$\frac{-2z+z_0+z_f}{\Delta z}$
$\frac{d^2}{dz^2}(\cdot)$	$\frac{-2}{\Delta z^2}$	$\frac{2}{\Delta z^2}$	$\frac{-2}{\Delta z}$

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

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

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 f and two constraints on f', are given in Table A6.

Acknowledgments

The authors would like to acknowledge Hunter Johnston for providing computational resources and precious insights that have helped for the realization of this work.

References

[1] Rao, A. V., "A Survey of Numerical Methods for Optimal Control," *Advances in the Astronautical Sciences*, Vol. 135, No. 1, 2009, pp. 497–528.

[2] Rauwolf, G. A., and Coverstone-Carroll, V. L., "Near-Optimal Low-Thrust Orbit Transfers Generated by a Genetic Algorithm," *Journal of Spacecraft and Rockets*, Vol. 33, No. 6, 1996, pp. 859–862. https://doi.org/10.2514/3.26850

[3] Kluever, C. A., and Oleson, S. R., "Direct Approach for Computing Near-Optimal Low-Thrust Earth-Orbit Transfers," *Journal of Spacecraft and Rockets*, Vol. 35, No. 4, 1998, pp. 509–515. https://doi.org/10.2514/2.3360

[4] Jiang, F., Tang, G., and Li, J., "Improving Low-Thrust Trajectory Optimization by Adjoint Estimation with Shape-Based Path," *Journal of Guidance, Control, and Dynamics*, Vol. 40, No. 12, 2017, pp. 3282–3289. https://doi.org/10.2514/1.G002803

[5] Taheri, E., and Junkins, J. L., "Generic Smoothing for Optimal Bang-Off-Bang Spacecraft Maneuvers," *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 11, 2018, pp. 2470–2475. https://doi.org/10.2514/1.G003604

[6] Wang, Z., and Grant, M. J., "Optimization of Minimum-Time Low-Thrust Transfers Using Convex Programming," *Journal of Spacecraft and Rockets*, Vol. 55, No. 3, 2018, pp. 586–598. https://doi.org/10.2514/1.A33995

[7] Patterson, M. A., and 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 Transactions on Mathematical Software (TOMS)*, Vol. 41, No. 1, 2014, pp. 1–37. https://doi.org/10.1145/2558904

[8] Poe, W. A., and Mokhatab, S., *Modeling, Control, and Optimization of Natural Gas Processing Plants*, Gulf Professional, Elsevier, 2016.

[9] Keller, H. B., *Numerical Solution of Two Point Boundary Value Problems*, Vol. 24, SIAM, 1976.

[10] Stoer, J., and Bulirsch, R., *Introduction to Numerical Analysis*, Vol. 12, Springer Science & Business Media, Berlin, 2013.

[11] Oh, S., and Luus, R., "Use of Orthogonal Collocation Method in Optimal Control Problems," *International Journal of Control*, Vol. 26, No. 5, 1977, pp. 657–673. https://doi.org/10.1080/00207177708922339

[12] Fahroo, F., and Ross, I., "Trajectory Optimization by Indirect Spectral Collocation Methods," *Astrodynamics Specialist Conference*, AIAA Paper 2000-4028, 2000.

[13] Cheng, L., Wang, Z., Jiang, F., and Zhou, C., "Real-Time Optimal Control for Spacecraft Orbit Transfer via Multiscale Deep Neural Networks," *IEEE Aerospace and Electronic Systems Magazine*, Vol. 55, No. 5, 2018, pp. 2436–2450. https://doi.org/10.1109/TAES.2018.2889571

[14] Izzo, D., and Öztürk, E., "Real-Time Guidance for Low-Thrust Transfers Using Deep Neural Networks," *Journal of Guidance, Control, and Dynamics*, Vol. 44, No. 2, 2021, pp. 315–327. https://doi.org/10.2514/1.G005254

[15] Cheng, L., Wang, Z., Jiang, F., and Li, J., "Fast Generation of Optimal Asteroid Landing Trajectories Using Deep Neural Networks," *IEEE Aerospace and Electronic Systems Magazine*, Vol. 56, No. 4, 2019, pp. 2642–2655. https://doi.org/10.1109/TAES.2019.2952700

[16] Cheng, L., Wang, Z., Song, Y., and Jiang, F., "Real-Time Optimal Control for Irregular Asteroid Landings Using Deep Neural Networks," *Acta Astronautica*, Vol. 170, May 2020, pp. 66–79. https://doi.org/10.1016/j.actaastro.2019.11.039

[17] Sánchez-Sánchez, C., and Izzo, D., "Real-Time Optimal Control via Deep Neural Networks: Study on Landing Problems," *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 5, 2018, pp. 1122–1135. https://doi.org/10.2514/1.G002357

[18] Shi, Y., and Wang, Z., "Onboard Generation of Optimal Trajectories for Hypersonic Vehicles Using Deep Learning," *Journal of Spacecraft and Rockets*, Vol. 58, No. 2, 2021, pp. 400–414. https://doi.org/10.2514/1.A34670

[19] Raissi, M., Perdikaris, P., and Karniadakis, G. E., "Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations," *Journal of Computational Physics*, Vol. 378, Feb. 2019, pp. 686–707. https://doi.org/10.1016/j.jcp.2018.10.045

[20] Leake, C., and Mortari, D., "Deep Theory of Functional Connections: A New Method for Estimating the Solutions of Partial Differential Equations," *Machine Learning and Knowledge Extraction*, Vol. 2, No. 1, 2020, pp. 37–55.

[21] Schiassi, E., Furfaro, R., Leake, C., De Florio, M., Johnston, H., and Mortari, D., "Extreme Theory of Functional Connections: A Fast Physics-Informed Neural Network Method for Solving Ordinary and Partial Differential Equations," *Neurocomputing*, Vol. 457, 2021, pp. 334–356. https://doi.org/10.1016/j.neucom.2021.06.015

[22] Mortari, D., "The Theory of Connections: Connecting Points," *Mathematics*, Vol. 5, No. 4, 2017, p. 57. https://doi.org/10.3390/math5040057

[23] Mortari, D., and Leake, C., "The Multivariate Theory of Connections," *Mathematics*, Vol. 7, No. 3, 2019, p. 296. https://doi.org/10.3390/math7030296

[24] Mortari, D., "Least-Squares Solution of Linear Differential Equations," *Mathematics*, Vol. 5, No. 4, 2017, p. 48. https://doi.org/10.3390/math5040048

[25] Mortari, D., Johnston, H., and Smith, L., "High Accuracy Least-Squares Solutions of Nonlinear Differential Equations," *Journal of Computational and Applied Mathematics*, Vol. 352, May 2019, pp. 293–307. https://doi.org/10.1016/j.cam.2018.12.007

[26] Leake, C., Johnston, H., and Mortari, D., "The Multivariate Theory of Functional Connections: Theory, Proofs, and Application in Partial Differential Equations," *Mathematics*, Vol. 8, No. 8, 2020, p. 1303. https://doi.org/10.3390/math8081303

[27] Furfaro, R., and Mortari, D., "Least-Squares Solution of a Class of Optimal Space Guidance Problems via Theory of Connections," *Acta Astronautica*, Vol. 168, March 2019, pp. 92–103. https://doi.org/10.1016/j.actaastro.2019.05.050

[28] Schiassi, E., D'Ambrosio, A., Johnston, H., Furfaro, R., Curti, F., and Mortari, D., "Complete Energy Optimal Landing on Small and Large Planetary Bodies via Theory of Functional Connections," *2020 AAS/AIAA Astrodynamics Specialist Conference*, Univelt, Inc., San Diego, CA, 2020.

[29] Johnston, H., Schiassi, E., Furfaro, R., and Mortari, D., "Fuel-Efficient Powered Descent Guidance on Large Planetary Bodies via Theory of Functional Connections," *Journal of the Astronautical Sciences*, Vol. 67,

No. 4, 2020, pp. 1521–1552.
https://doi.org/10.1007/s40295-020-00228-x

[30] Drozd, K., Furfaro, R., Schiassi, E., Johnston, H., and Mortari, D., "Energy-Optimal Trajectory Problems in Relative Motion Solved via Theory of Functional Connections," *Acta Astronautica*, Vol. 182, May 2021, pp. 361–382.
https://doi.org/10.1016/j.actaastro.2021.01.031

[31] Huang, G.-B., Zhu, Q.-Y., and Siew, C.-K., "Extreme Learning Machine: Theory and Applications," *Neurocomputing*, Vol. 70, No. 2006, 2006, pp. 489–501.
https://doi.org/10.1016/j.neucom.2005.12.126

[32] Schiassi, E., De Florio, M., D'Ambrosio, A., Mortari, D., and Furfaro, R., "Physics-Informed Neural Networks and Functional Interpolation for Data-Driven Parameters Discovery of Epidemiological Compartmental Models," *Mathematics*, Vol. 9, No. 17, 2021, p. 2069.
https://doi.org/10.3390/math9172069

[33] Ross, I. M., *A Primer on Pontryagin's Principle in Optimal Control*, Collegiate, 2009.

[34] Lagaris, I. E., Likas, A., and Fotiadis, D. I., "Artificial Neural Networks for Solving Ordinary and Partial Differential Equations," *IEEE Transactions on Neural Networks*, Vol. 9, No. 5, 1998, pp. 987–1000.
https://doi.org/10.1109/72.712178

[35] Lu, L., Meng, X., Mao, Z., and Karniadakis, G. E., "DeepXDE: A Deep Learning Library for Solving Differential Equations," *SIAM Review*, Vol. 63, No. 1, 2021, pp. 208–228.
https://doi.org/10.1137/19M1274067

[36] Wang, S., Teng, Y., and Perdikaris, P., "Understanding and Mitigating Gradient Flow Pathologies in Physics-Informed Neural Networks," *SIAM Journal on Scientific Computing*, Vol. 43, No. 5, 2021, pp. A3055–A3081.
https://doi.org/10.1137/20M1318043

[37] Mertikopoulos, P., Papadimitriou, C., and Piliouras, G., "Cycles in Adversarial Regularized Learning," *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, 2018, pp. 2703–2717.
https://doi.org/10.1137/1.9781611975031

[38] Balduzzi, D., Racaniere, S., Martens, J., Foerster, J., Tuyls, K., and Graepel, T., "The Mechanics of n-Player Differentiable Games," *International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2018, pp. 354–363.

[39] Vlassenbroeck, J., and Van Dooren, R., "A Chebyshev Technique for Solving Nonlinear Optimal Control Problems," *IEEE Transactions on Automatic Control*, Vol. 33, No. 4, 1988, pp. 333–340.
https://doi.org/10.1109/9.192187

[40] McInnes, C. R., *Solar Sailing: Technology, Dynamics and Mission Applications*, Springer Science & Business Media, Berlin, 2004.

[41] Dachwald, B., "Optimal Solar Sail Trajectories for Missions to the Outer Solar System," *Journal of Guidance, Control, and Dynamics*, Vol. 28, No. 6, 2005, pp. 1187–1193.
https://doi.org/10.2514/1.13301

[42] Wood, L. J., Bauer, T. P., and Zondervan, K. P., ""Comment on" Time-Optimal Orbit Transfer Trajectory for Solar Sail Spacecraft," *Journal of Guidance, Control, and Dynamics*, Vol. 5, No. 2, 1982, pp. 221–224.
https://doi.org/10.2514/3.56160

[43] Parsopoulos, K. E., and Vrahatis, M. N., "UPSO: A Unified Particle Swarm Optimization Scheme," *International Conference of Computational Methods in Sciences and Engineering 2004 (ICCMSE 2004)*, CRC Press, 2019.

Y. Xu
Associate Editor

Appendix D

Schiassi et al. (2022): Bellman Neural Networks

Reproduced from Schiassi, E., D'Ambrosio, A. and Furfaro, R., 2022. Bellman Neural Networks for the Class of Optimal Control Problems With Integral Quadratic Cost. IEEE Transactions on Artificial Intelligence (Early Access). https://doi.org/10.1109/TAI.2022.3206735 [64], with the permission of IEEE.

This article has been accepted for publication in IEEE Transactions on Artificial Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TAI.2022.3206735

Bellman Neural Networks for the Class of Optimal Control Problems with Integral Quadratic Cost

Enrico Schiassi, Andrea D'Ambrosio, and Roberto Furfaro.

***Abstract*—This work introduces Bellman Neural Networks (BeNNs) and employs them to learn the optimal control actions for the class of optimal control problems (OCPs) with integral quadratic cost. BeNNs represent a particular family of Physics-Informed Neural Networks (PINNs) specifically designed and trained to tackle OCPs via applying the Bellman Principle of Optimality (BPO). The BPO provides necessary and sufficient optimality conditions, which result in a nonlinear partial differential equation known as the Hamilton-Jacobi-Bellman (HJB) equation. BeNNs learn the optimal control actions from the unknown solution of the arising HJB equation (i.e., the value function), where the unknown solution is modeled using a Neural Network. Additionally, the paper shows how to estimate the upper bounds on the generalization error of BeNNs while learning the solutions for the OCP class under consideration. The generalization error estimate is provided in terms of the choice and number of the training points as well as the training error. Numerical studies show that BeNNs can be successfully applied to learn the feedback control actions for the class of optimal control problems considered and, after the training is completed, deployed to control the system in a closed-loop fashion.**

***Impact Statement*—The proposed research improves our understanding of how to solve optimal control problems with closed-loop solutions and has potentially a countless number of applications in several different areas. The study is at the intersection between optimal control theory and artificial intelligence connected with mathematical tools for functional interpolation. This advances the ability to implement a higher level of autonomy in decision-making for practical applications with a beneficial impact on our society.**

***Index Terms*—Bellman Neural Networks, Close-loop Control, Extreme Learning Machines, Functional Interpolation, Hamilton-Jacobi-Bellman Equation, Physics-Informed Neural Networks**

I. Introduction

During the last century, optimal control theory has garnered interest in many fields of study. In particular, Optimal Control Problems (OCPs) play a crucial role in several scientific areas, such as trajectory optimization and tracking [1, 2, 3]. In many practical OCPs, it is not trivial and most likely impossible to find an analytical solution. Hence, one has to rely on numerical methods. The solutions of OCPs can be open-loop or closed-loop. One way to derive closed-loop solutions is to exploit the *Bellman's Principle of Optimality (BPO)*, ensuring the optimality sufficient and necessary conditions, which returns the Hamilton–Jacobi–Bellman (HJB) partial differential equation [4]. That is, by solving the HJB equation, feedback optimal control actions can be synthesized and then directly deployed to optimally control the system. In many practical applications, the HJB PDE must be solved numerically. The HJB equation is affected by the *curse of dimensionality*, as the computational cost increases exponentially with the system dimension number. Hence, traditional numerical methods to solve PDE, such as FEM, becomes impracticable [5, 6, 7]. Many methods, relying on the so-called Adaptive Dynamic Programming (ADP), have been proposed to solve HJB PDEs for OCPs [8, 9, 10, 11, 12]. ADP can be directly employed to solve the HJB equation using an actor-critic structure. ADP is generally implemented via two most popular methods: 1) policy iteration and 2) value iteration. Examples include Vrabie and Lewis [13] who devised an adaptive critic design using a policy iteration algorithm, and Al-Tamimi et al. [14] who employed two NNs to model actor and critic and solve a discrete optimal control problem using value iteration. In contrast, the method we propose attempts to directly solve the HJB using Physics-Informed Neural Network (PINN [15]). It builds a loss function with HJB residual. Using PINNs allows learning closed-loop optimal controllers that directly satisfy the HJB PDE. We will specifically solve HJB PDEs with two different PINN frameworks. The first one is the classic PINNs, as introduced by Raissi et al. [15]. The second one is an improved and more robust Physics-Informed Neural Networks (PINNs) framework that combines Neural Networks (NNs) and a functional interpolation technique called *Theory of Functional Connections (TFC)* [16], forming a PINN TFC-based framework [17, 18]. For the problems considered in this work, PINNs are specifically designed and trained to learn optimal control actions by satisfying the BPO. For this reason, we name these PINNs *Bellman Neural Networks (BeNNs)*. More specifically, this paper focuses on the class of OCPs with integral quadratic cost, both for finite and infinite horizon problems. The main contributions and goals of this work are: 1) to show the feasibility in using PINN-based frameworks to directly learn the HJB PDE solutions and thus closed-loop optimal control, 2) the estimation of the generalization error generated via the proposed methodology, and 3) the comparison of three different PINN-based frameworks in learning the HJB PDE solutions, to select the most suitable choice to tackle this class of OCPs. PINNs represent a machine learning framework that exploits the physics as a regularization term in

This paragraph of the first footnote will contain the date on which you submitted your paper for review.
The corresponding author is Roberto Furfaro (e-mail: robertof@email.arizona.edu)

Enrico Schiassi is a PhD candidate at the Systems and Industrial Engineering Department, The University of Arizona, Tucson, AZ 85721 (e-mail: eschiassi@email.arizona.edu).

Andrea D'Ambrosio is a Postdoctoral Associate at the Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA 02139 (e-mail: andreada@mit.edu).

Roberto Furfaro (corresponding author) is a professor at the Aerospace and Mechanical Engineering Department and Systems and Industrial Engineering Department, The University of Arizona, Tucson, AZ 85721 (e-mail: robertof@email.arizona.edu).

This paragraph will include the Associate Editor who handled your paper.

the loss function for training NNs [15]. PINNs are introduced to consider the DEs describing the physics behind the collected data. Indeed, DEs residuals are embedded within the loss function as additional terms, which penalize the training when the DEs and their constraints (e.g., the initial conditions or ICs and/or boundary conditions or BCs) are violated. In the classic PINN framework the DE constraints are part of the loss function along with the DE residuals within the domain. This represents the major drawback for the classic PINN framework [19]. The PINN TFC-based methods used in this paper substantially improve the classic PINN frameworks, removing their main limitation. The key ingredients of the PINN-TFC methods are the Constrained Expressions (CEs) introduced in the Theory of Functional Connections (TFC) [16]. The general expression of the CE is,

$$f(\mathbf{x}) \simeq f_{CE}(\mathbf{x}, g(\mathbf{x})) = A(\mathbf{x}) + B(\mathbf{x}, g(\mathbf{x}))$$

where $f(\mathbf{x})$ is the unknown function to be interpolated, $f_{CE}(\mathbf{x}, g(\mathbf{x}))$ is the CE, $A(\mathbf{x})$ is a functional that analytically satisfies any given linear constraint, and $B(\mathbf{x}, g(\mathbf{x}))$ projects the free-function $g(\mathbf{x})$, which is a real function that must exist on the constraints, onto the space of functions that vanish at the constraints [18]. TFC is a functional interpolation technique which can be applied to many mathematical problems. In particular, TFC is applied for approximating the solution of DEs [20], as well as for OCPs via the PMP approach. For solving DE, the original (or classic) TFC employs a linear combination of orthogonal polynomials as free-function [20]. Although employing orthogonal polynomials as free-function allows to compute highly accurate results, it affects the original TFC framework with the curse of dimensionality. In particular when solving large-scale PDEs, as the computational cost increases exponentially with the number of bases required to compute accurate solutions [20]. To overcome these issues, PINN TFC-based methods use NNs as free-function. PINN TFC-based methods have been already applied to solve a class of OCPs via indirect method [21, 22].

In this manuscript, first, we explain how BeNNs are modeled and trained to learn control actions for the class of finite and infinite horizon optimal control problems with integral quadratic cost. Then we will provide an estimate on the upper bounds of the generalization error of BeNNs in learning the solutions for the OCP class considered in this work in terms of the training error, number, and choice of the training points. Two benchmarks OCPs are then tackled to show the effectiveness of BeNNs in learning optimal control actions for the class of OCPs considered and to test the theoretical findings. Finally, the results will be discussed in the remaining section.

II. Bellman Neural Networks for Optimal Control Problems

BeNNs represent a particular family of PINNs specifically designed and trained to learn the solutions of OCPs via the application of the BPO. In this work, we will focus on the class of OCPs with integral quadratic cost. The authors here coin the term *"Bellman Neural Networks (BeNNs)"* to refer in a more compact form to PINNs designed and trained to learn optimal control via the application of the BPO. The NN representation of the HJB unknown solution is done via two different PINN frameworks: classic PINNs and PINNs combined with TFC. According to PINN TFC-based frameworks, a generic differential equation's unknown solutions are modeled via the TFC's CEs using NNs as free-functions. When Deep NNs (DNNs) are used as free-function, the PINN TFC-based framework is called *Deep-TFC* [18]. In the this framework, gradient-based method, such as ADAM optimizer [23] or the L-BFGS optimizer [24], are employed to optimize the free-function parameters. When shallow NNs, trained via Extreme Learning Machine (ELM) algorithm [25], are used as free-function, the PINN TFC-based framework is called *Extreme-TFC (X-TFC)* [17]. The ELM algorithm randomly samples input weights and bias. These are kept fixed during the training procedure. Thus, the output weights are the only trainable parameters [25]. Thanks to this feature, the training is performed with a robust and fast least-squares. When X-TFC is used to learn the solution of linear DEs, a single pass least-squares is employed to optimize the free-function parameters (e.g., the NN output weights). Conversely, iterative least-squares are required to optimize the free-function parameters when X-TFC is used to learn the solution of nonlinear DEs (such as the HJB PDEs). The details regarding the iterative least-squares procedure can be found in [17]. This section shows how to design and train BeNN to learn the optimal control actions for two classes of problems with integral quadratic cost: finite horizon problems and infinite horizon problems. Moreover, following Ref. [26], we derive an estimate on the upper bounds of the generalization error of BeNNs in learning the solutions for the OCP class considered in this work.

A. Finite Horizon Problems

The typical finite horizon OCP is posed as follows,

$$\min_{\boldsymbol{u}\in\mathcal{U}} \mathcal{J}(t, \boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{\phi}(\boldsymbol{x}(t_f)) + \int_{t_0}^{t_f} \mathcal{L}(t, \boldsymbol{x}, \boldsymbol{u}) \tag{1}$$

subject to

$$\begin{aligned} &\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}, \boldsymbol{u}) \\ &\boldsymbol{x}(t_0) = \boldsymbol{x}_0 \\ &\boldsymbol{x}(t_f) \in \mathcal{C} \\ &t \in [t_0, t_f] \end{aligned} \tag{2}$$

where t, $\boldsymbol{x} \in \Omega \subseteq \mathbb{R}^n$, $\boldsymbol{u} \in \mathcal{U} \subseteq \mathbb{R}^m$, and $\mathcal{C} \subset \Omega$ are the time variable, states, control, and terminal conditions respectively. In general the final time t_f may be fixed or unknown (e.g. for the time-free problems). The system dynamics $\boldsymbol{f} \in \mathbb{R}^n$ is assumed to be known. The main characteristic of the finite horizon problems is the presence of the Meyer term in the cost function, $\boldsymbol{\phi}(\boldsymbol{x}(t_f))$, which results to be a temporal boundary condition for the arising HJB equation. In this manuscript we will consider only fixed time problems. We consider the value function $V(t, \boldsymbol{x})$ defined as follows,

$$V(t, \boldsymbol{x}) = \inf_{\boldsymbol{u}\in\mathcal{U}} \mathcal{J}(t, \boldsymbol{x}, \boldsymbol{u}) \tag{3}$$

This article has been accepted for publication in IEEE Transactions on Artificial Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TAI.2022.3206735

which is the unique solution of the following HJB equation [27],

$$-V_t + \sup_{\boldsymbol{u}\in\mathcal{U}}\{-V_{\boldsymbol{x}}^T\dot{\boldsymbol{x}} - \mathcal{L}(t,\boldsymbol{x},\boldsymbol{u})\} = 0, \quad \forall t \in [t_0, t_f], \forall \boldsymbol{x} \in \Omega \tag{4}$$

subject to

$$V(t_f, \boldsymbol{x}) = \phi(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \Omega \tag{5}$$

The first step is to derive the optimal control, $\boldsymbol{u}_{\text{opt}}$, as a closed analytical form with respect to the value function V. That is,

$$\boldsymbol{u}_{\text{opt}} = \sup_{\boldsymbol{u}\in\mathcal{U}}\{-V_{\boldsymbol{x}}^T\dot{\boldsymbol{x}} - \mathcal{L}(t,\boldsymbol{x},\boldsymbol{u})\} \tag{6}$$

Once $\boldsymbol{u}_{\text{opt}}$ is obtained, its closed analytical form is plugged back into (4). Then, (4) reduces to a nonlinear PDE that is solved for V,

$$\mathcal{D}(V(t,\boldsymbol{x})) = \mathcal{F}(t,\boldsymbol{x}), \quad \forall t \in [t_0, t_f], \forall \boldsymbol{x} \in \Omega \tag{7}$$

subject to

$$V(t_f, \boldsymbol{x}) = \phi(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \Omega \tag{8}$$

where $\mathcal{D}(\cdot)$ is a nonlinear differential operator acting on V, and $\mathcal{F}(t,\boldsymbol{x})$ is the known forcing term. Our goal is to train a BeNN to learn the solution of problem (7)-(8). Two different training algorithms are proposed. One employs classic PINNs and the other one PINN TFC-based. The training algorithms are given below. The main challenge in solving finite horizon problems with PINN-based frameworks is that the nonlinear PDE (7) must be learned simultaneously with the temporal boundary condition (8). The goal of the Algorithm 1 is to train a BeNN to learn the solution of problem (7)-(8) employing classic PINNs. Therefore, V is modeled as,

$$V(t,\boldsymbol{x}) \simeq V(t,\boldsymbol{x};\theta) = V^{\theta}(t,\boldsymbol{x}) \tag{9}$$

where θ are the NN parameters that are learned during the training. To train the NN, the following residuals need to be defined. The interior (int) residual is,

$$\mathcal{R}_{\text{int},\theta}(t,\boldsymbol{x}) = \mathcal{D}(V^{\theta}(t,\boldsymbol{x})) - \mathcal{F}(t,\boldsymbol{x}), \quad \forall t \in [t_0, t_f], \forall \boldsymbol{x} \in \Omega \tag{10}$$

The temporal boundary (tb) residual is,

$$\mathcal{R}_{tb,\theta}(\boldsymbol{x}) = V^{\theta}(t_f,\boldsymbol{x}) - \phi(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \Omega \tag{11}$$

The following physics-driven loss function, $J(\theta)$, is formed by collocating the residuals on the training points,

$$J(\theta) = \lambda\sum_{n=1}^{N_{\text{int}}}||\mathcal{R}_{\text{int},\theta}(t_n,\boldsymbol{x}_n)||^2 + \sum_{n=1}^{N_{\text{tb}}}||\mathcal{R}_{tb,\theta}(\boldsymbol{x}_n)||^2 \tag{12}$$

where, N_{int} are the interior training points (e.g., the points sampled from $[t_0, t_f]$ and Ω), N_{tb} are the temporal boundary training points, (e.g., the points sampled from Ω), and λ is a hyperparameter for balancing the interior residual over the temporal boundary residual.

The algorithm to train a BeNN to learn the solution of problem (7)-(8) by employing PINN TFC-based is Algorithm 2, where V is modeled with the following CE,

$$V(t,\boldsymbol{x}) \simeq V(t,\boldsymbol{x};\theta) = V^{\theta}(t,\boldsymbol{x}) = g^{\theta}(t,\boldsymbol{x}) + \Big(\phi(\boldsymbol{x}) - g^{\theta}(t_f,\boldsymbol{x})\Big) \tag{13}$$

Algorithm 1: PINN based algorithm for BeNN training

Input: Domain $[t_0, t_f] \times \Omega$, differential operator $\mathcal{D}$, forcing term $\mathcal{F}$, training points (for interior and temporal boundary), hyperparameter for balancing the two terms in the total loss function, and the optimization algorithm to minimize the loss with respect to the parameters θ.

Goal: To find BeNN, $V^* = V^{\theta^*}$, to approximate the true solution V of Eqs. (7)-(8) as best as possible via solving the following minimization problem

$$\theta^* = \min_{\theta\in\Theta} J(\theta)$$

where Θ is the search space of the NN parameters, whose formal definition is given in Ref. [26]

Step 1: Approximate the real, unknown, V with a NN, which is called BeNN, V^{θ}.

Step 2: For an initial guess of the vector $\theta^0 \in \Theta$, evaluate V^{θ^0}, the residuals, the loss, and its gradients to initialize the BeNN training (e.g., the optimization algorithm).

Step 3: Run the BeNN training until the optimal parameters, θ^*, are computed. The map $V^* = V^{\theta^*}$ is the desired BeNN for approximating the true solution V of (7)-(8).

where $g^{\theta}(t,\boldsymbol{x}) = g(t,\boldsymbol{x};\theta)$ is a NN with parameters θ, that are learned during the training. To train the NN, the residuals need to be defined. The Interior (int) residual is,

$$\mathcal{R}_{\text{int},\theta}(t,\boldsymbol{x}) = \mathcal{D}(V^{\theta}(t,\boldsymbol{x})) - \mathcal{F}(t,\boldsymbol{x}), \quad \forall t \in [t_0, t_f], \forall \boldsymbol{x} \in \Omega \tag{14}$$

The temporal boundary (tb) residual is,

$$\mathcal{R}_{tb,\theta}(\boldsymbol{x}) = V^{\theta}(t_f,\boldsymbol{x}) - \phi(\boldsymbol{x}) = 0, \quad \forall \boldsymbol{x} \in \Omega \tag{15}$$

The following physics-driven loss function, $J(\theta)$, is formed by collocating the residuals on the training points,

$$J(\theta) = \sum_{n=1}^{N_{\text{int}}}||\mathcal{R}_{\text{int},\theta}(t_n,\boldsymbol{x}_n)||^2 \tag{16}$$

where, N_{int} are the interior training points (e.g., the points sampled from $[t_0, t_f]$ and Ω),

Algorithm 2: PINN TFC-based algorithm for BeNN training

Input: see Alg. 1, but only the interior training points are needed.

Goal: see Alg. 1, but the loss to minimize is the one of Eq. (16)

Step 1: Approximate the real, unknown, V with the functional V^{θ} given in Eq. (13), which is the BeNN.

Step 2: see Alg. 1

Step 3: see Alg. 1

As previously stated, using a PINN TFC-based framework removes one of the main limitations of the traditional PINN methods. Competing objectives during the PINN training, which are minimizing the DE residuals within the domain (unsupervised learning task) and learning the equation constraints (supervised learning task), arise because the constraints

This article has been accepted for publication in IEEE Transactions on Artificial Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TAI.2022.3206735

of the equation are not analytically fulfilled. Thus, unbalanced gradients can occur during the PINN training via gradient-based methods. This may cause the PINN failure in learning the correct DE solution [28]. Indeed, when multiple competing objectives are present, gradient-based techniques may diverge or get stuck in limit cycles [29, 30]. Moreover, when using gradient-based methods the unknown NN parameters appear non-linearly. Thus, it becomes prohibitive to provide any meaningful initial guess for nonlinear DEs. Two different PINN TFC-based approaches are employed: Deep-TFC and X-TFC. Within Deep-TFC the CE free-function is a deep NN, whose parameters $\boldsymbol{\theta}$ are learned via gradient-based methods. Deep-TFC removes the issue of having competing objectives in the loss function, but it does not remove the issue of providing the initial guess for sensitive nonlinear DEs. Indeed, as the NN parameters appear non-linearly in the functional V^{θ}, it is prohibitive to provide an initial guess for nonlinear DEs when it is needed. Within X-TFC the CE free-function is represented by a shallow NN trained with ELM algorithm, which randomly samples input weights and bias. These are kept fixed during the training procedure. Thus, the output weights are the only trainable parameters [25, 31]. Thanks to this feature, the training is performed with a robust and fast least-squares. Hence, X-TFC eliminates all the limitations previously mentioned. In particular, as the unknowns (e.g., output weights) appear linearly in the functional V^{θ}, it is possible to provide a meaningful initial guess when required. For OCPs, the HJB equation is always nonlinear. In many cases, when needed, a good initial guess would be a quadratic value function dependent on the state vector. As an example, the following expression could be used: $V_{\text{guess}}(\boldsymbol{x}) = \boldsymbol{x}^T P \boldsymbol{x}$, where P is a positive definite matrix. Although X-TFC does not, in principle, need a stabilizing control, the latter can be alternative as an informative initial guess. As X-TFC solves a non-linear optimization via a sequence of approximation that converges toward a (local) minimum, one can select as starting point for the optimization, an initial stabilizing controller with a defined region of attraction and use the theorems from Beard et al. [32] to conclude that the region of attraction is at least as large as the one for the initial stabilizing controller. Although the extension is straightforward, we believe that stability analysis requires further investigation which is out of the scope of this work. However, an informative initial guess is not always needed, especially if the X-TFC framework is used, for which even a random initialization of the value function can allow the BeNN to learn the correct optimal control actions.

B. Infinite Horizon Problems

Considering a typical LQR integral cost, these problems are posed as following,

$$\min_{\boldsymbol{u}\in\mathcal{U}} \mathcal{J}(\boldsymbol{x},\boldsymbol{u}) = \int_0^{\infty} \left(Q(\boldsymbol{x}) + \boldsymbol{u}^T R \boldsymbol{u}\right) \mathrm{dt} \tag{17}$$

subject to

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{f}(t,\boldsymbol{x},\boldsymbol{u}) \\ \boldsymbol{x}(0) &= \boldsymbol{x}_0 \end{aligned} \tag{18}$$

where $\boldsymbol{x} \in \Omega \subseteq \mathbb{R}^n$, $\boldsymbol{u} \in \mathcal{U} \subseteq \mathbb{R}^m$ are the states, and control respectively. The system dynamics $\boldsymbol{f} \in \mathbb{R}^n$ is assumed to be known. The function $Q(\boldsymbol{x})$ is positive definite, and $R \in \mathbb{R}^{n\times m}$ is a symmetric positive definite matrix. The main characteristic of the infinite horizon problems is the presence of the matrices Q and R which can affect and regulate the time when the system converges to equilibrium. If these matrices are not properly set, the desired final condition near the equilibrium will not be reached. The value function V is,

$$V(\boldsymbol{x}) = \inf_{\boldsymbol{u}\in\mathcal{U}} \mathcal{J}(\boldsymbol{x},\boldsymbol{u}) \tag{19}$$

which is the unique solution of the following HJB equation,

$$\sup_{\boldsymbol{u}\in\mathcal{U}} \{-V_{\boldsymbol{x}}^T \dot{\boldsymbol{x}} - \left(Q(\boldsymbol{x}) + \boldsymbol{u}^T R \boldsymbol{u}\right)\} = 0, \quad \forall \boldsymbol{x} \in \Omega \tag{20}$$

subject to

$$V(\boldsymbol{0}) = 0 \tag{21}$$

Even for infinite horizon OCPs, the proposed procedure can be applied to a generic dynamical system. However, if the dynamics is affine in the control (i.e., $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) + g(\boldsymbol{x})\boldsymbol{u}$), the optimal control $\boldsymbol{u}_{\text{opt}}$ can be derived in a closed analytical form. Since most of the infinite horizon OCPs of interest for real-world problems belong to this class, affine systems are considered and derived in the following of this work without loss of generality. Hence:

$$\boldsymbol{u}_{\text{opt}} = \underset{\boldsymbol{u}\in\mathcal{U}}{\text{argsup}} \{-V_{\boldsymbol{x}}^T \dot{\boldsymbol{x}} - \left(Q(\boldsymbol{x}) + \boldsymbol{u}^T R \boldsymbol{u}\right)\} = -\frac{1}{2} R^{-1} g^T(x) V_{\boldsymbol{x}} \tag{22}$$

Once $\boldsymbol{u}_{\text{opt}}$ is derived, its closed analytical form is plugged back into (20). Then equation (20) reduces to a nonlinear PDE that is solved for V,

$$Q(\boldsymbol{x}) + V_{\boldsymbol{x}}^T \boldsymbol{f}(\boldsymbol{x}) - \frac{1}{4} V_{\boldsymbol{x}}^T g(\boldsymbol{x}) R^{-1} g^T(\boldsymbol{x}) V_{\boldsymbol{x}} = 0, \quad \forall \boldsymbol{x} \in \Omega \tag{23}$$

subject to

$$V(\boldsymbol{0}) = 0 \tag{24}$$

Equation (23) can be rewritten in the following form,

$$\mathcal{D}(V(\boldsymbol{x})) = \mathcal{F}(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \Omega \tag{25}$$

subject to

$$V(\boldsymbol{0}) = 0 \tag{26}$$

where $\mathcal{D}(\cdot)$ is a nonlinear differential operator acting on V, and $\mathcal{F}(t,\boldsymbol{x}) = -Q(\boldsymbol{x})$ is the known forcing term. The problem (25)-(26) is in the same form of problem (7)-(8). Even for the solution of infinite horizon OCPs via PINN-based frameworks, the main challenge is the selection of the matrices Q and R and the fact that the nonlinear PDE (25) must be learned simultaneously with the temporal boundary condition (26). Thus algorithms 1 and 2 are applied to learn $V(\boldsymbol{x})$. For the type of infinite horizon problems considered in this work, the CE for the PINN TFC-based framework is,

$$V^{\theta}(\boldsymbol{x}) = g^{\theta}(\boldsymbol{x}) - g^{\theta}(\boldsymbol{0}) \tag{27}$$

The considerations about the convenience in using PINN TFC-based methods made for the finite horizon problems hold also for the infinite horizon ones.

This article has been accepted for publication in IEEE Transactions on Artificial Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TAI.2022.3206735

C. Estimate on the Generalization Error

In this section, following the steps in Ref. [26], we will estimate the error generated by BeNN trained both via algorithm 1 and by algorithm 2 in learning the solution V of the problems (7)-(8) and (25)-(26), respectively. This error is the *total error*, also known as the *generalization error* [26, 33]. For classic PINN we have,

$$\varepsilon_G = \varepsilon_G(\theta^*; \mathcal{S}) = ||V - V^*||, \tag{28}$$

for PINN TFC-based we have,

$$\varepsilon_{G,\text{tfc}} = \varepsilon_{G,\text{tfc}}(\theta^*; \mathcal{S}) = ||V - V^*_{\text{tfc}}|| \tag{29}$$

That is, the generalization error is the absolute error between the real, V, solution and the approximated solution V^* (or V^*_{tfc}). From Eqs. (28) and (29) it can be seen how the generalization error is a function of the training set $\mathcal{S}$ and the trained BeNN with parameters θ^*, obtained via algorithms 1 and 2 respectively. Nevertheless, for convenience, we will omit the generalization error dependency on the trained parameters and the training set.

As the real solution V is not always available, it is not always possible to evaluate the generalization error, neither during nor after the training process. Conversely, the so-called *training* error can be monitored and evaluated a-posteriori once the trained is completed. In the following subsections, we will defined the training error for both classic PINN and PINN TFC-based training algorithms, and show how the generalization error is bounded by the training error and the quadrature error, which is the error associated to the choice of the training points.

For classic PINN, the *training error*, ε_T, is given by,

$$\varepsilon_T = J(\theta^*)^{\frac{1}{2}} = \left(\lambda \sum_{n=1}^{N_{\text{int}}} ||\mathcal{R}_{\text{int},\theta^*}(t_n, \boldsymbol{x}_n)||^2 + \sum_{n=1}^{N_{\text{tb}}} ||\mathcal{R}_{tb,\theta^*}(\boldsymbol{x}_n)||^2 \right)^{\frac{1}{2}} \tag{30}$$

That is, ε_T can be evaluated once the training is completed and ε_G can be estimated in terms of ε_T. Before stating the theorem showing the ε_G estimate in term of ε_T, several hypothesis need to made.

Let X, Y be separable Banach spaces with norms $||\cdot||_X$ and $||\cdot||_Y$, respectively. Let $X^* \subset X$ and $Y^* \subset Y$ be closed sub-spaces with norms $||\cdot||^*_X$ and $||\cdot||^*_Y$, respectively. In problems (7)-(8) and (25)-(26), the differential operator $\mathcal{D}$ is mapping, $\mathcal{D} : X^* \mapsto Y^*$, and the forcing term $\mathcal{F} \in Y^*$, such that,

$$\begin{cases} (H1): \quad ||\mathcal{D}(V)||_{Y^*} < \infty, \quad \forall V \in X^*, \quad \text{with} \quad ||V||_{X^*} < \infty \\ (H2): \quad ||\mathcal{F}||_{Y^*} < \infty \end{cases} \tag{31}$$

In addition, we make the assumption that $\forall \mathcal{F} \in Y^*$, $\exists!\ V \in X^*$ such that (7)-(8) and (25)-(26) hold. Furthermore, we assume that the solutions of (7)-(8) and (25)-(26) fulfill the stability bound: let $Z \subset X^* \subset X$ be a closed sub-space with norm $||\cdot||_Z$.

Assumption II.1. *For any $V, U \in Z$, the differential operator $\mathcal{D}$ satisfies*

$$(H3): \quad ||V - U||_X \leq C_{\text{pde}}\left(||V||_Z, ||U||_Z\right) ||\mathcal{D}(V) - \mathcal{D}(U)||_Y, \tag{32}$$

where C_{DE} is a positive constant the explicitly depends on $|\boldsymbol{u}|_Z$ and $|\boldsymbol{v}|_Z$. That is,

$$C_{\text{pde}} = C_{\text{pde}}\left(||V||_Z, ||U||_Z\right) > 0.$$

Based on the assumptions above, that according to Ref. [26] require that both the differential operator and the forcing term are finite and bounded (H1 and H2, respectively), and that the DE is subject to a *conditional stability estimate* (H3), the following theorem holds.

Theorem 1. *Let $V \in Z \subset X^*$ be the unique solution of the DE (7)-(8) (and (25)-(26)) and assume that the stability hypothesis (H3) holds. Let $V^* \in Z \subset X^*$ be the BeNN computed via algorithm 1, based on the chosen training set $\mathcal{S}$. The following estimate on the generalization error holds,*

$$\varepsilon_G \leq C_{\text{pde}} \varepsilon_T + C_{\text{pde}} C^{\frac{1}{2}}_{\text{train}} N^{-\frac{\alpha}{2}} \tag{33}$$

with $C_{\text{pde}} = C_{\text{pde}}(||V||_Z, ||V^*||_Z)$ and $C_{\text{train}} = C_{\text{train}}(||\mathcal{R}_{\theta^*}||^2)$ (with $\mathcal{R}^*_\theta$ including interior and temporal boundary) stemming from (H3). It is worth to notice that the constants C_{pde}, C_{train} depend on the BeNN, V^*, and on the number of training points $N = N_{\text{int}} + N_{\text{tb}}$.

Proof: For this proof, we indicate as $\mathcal{R} = \mathcal{R}_{\theta^*}$ the residual corresponding to the trained BeNN, V^*. Here, in the residual term $\mathcal{R}$ we consider all the components (e.g., interior and temporal boundary). The residual is the following,

$$\mathcal{R} = \mathcal{D}(V^*) - \mathcal{F} \tag{34}$$

As V solves the DE (7)-(8) (and (25)-(26)) the following holds true,

$$\mathcal{D}(V) - \mathcal{F} = 0 \tag{35}$$

Thus, equation (34) can be rewritten as follows,

$$\mathcal{R} = (\mathcal{D}(V^*) - \mathcal{F}) - (\mathcal{D}(V) - \mathcal{F}) = \mathcal{D}(V^*) - \mathcal{D}(V) \tag{36}$$

We have, by (28),

$$\varepsilon_G = ||V - V^*|| \tag{37}$$

Because of the stability bound H3, the following holds true,

$$\varepsilon_G = ||V - V^*|| \leq C_{\text{pde}} ||\mathcal{D}(V^*) - \mathcal{D}(V)|| \tag{38}$$

Therefore, as by (36) $||\mathcal{D}(V^*) - \mathcal{D}(V)|| = ||\mathcal{R}||$, we have,

$$\varepsilon_G \leq C_{\text{pde}} ||\mathcal{R}|| \tag{39}$$

As the residuals are collocated on the training points, we have the following bound (due to the quadrature error, see [26] and references within for further details),

$$||\mathcal{R}||^2 \leq \varepsilon^2_T + C_{\text{train}} N^{-\alpha}, \tag{40}$$

where the term $C_{\text{train}} N^{-\alpha}$ is the error associated to the choice of the training points. Then,

$$\varepsilon^2_G \leq C^2_{\text{pde}} |\mathcal{R}|^2 \leq C^2_{\text{pde}} \left(\varepsilon^2_T + C_{\text{train}} N^{-\alpha}\right)$$

Finally,

$$\varepsilon^2_G \leq C^2_{\text{pde}} \left(\varepsilon^2_T + C_{\text{train}} N^{-\alpha}\right) = C^2_{\text{pde}} \varepsilon^2_T + C^2_{\text{pde}} C_{\text{train}} N^{-\alpha}$$

Thus,

$$\varepsilon_G \leq C_{\text{pde}}\varepsilon_T + C_{\text{pde}}C_{\text{train}}^{\frac{1}{2}}N^{-\frac{\alpha}{2}}$$

which is the estimate (33). This completes the proof. ■

The estimate (33) suggests that the generalization error when using algorithm 1 is small under several conditions. The BeNN must be *well-trained*. The training error must be sufficiently small (e.g., $\varepsilon_T << 1$). Moreover, the training error can only be computed a posteriori, since we do not have any prior control on it. The error associated with the training points depends on the choice and the number of training points N as well as on the training constant C_{train}. The latter also depends on the residual of the BeNN, V^*, and thus, indirectly, on the number of training points N (where $N = N_{\text{int}} + N_{\text{tb}}$). That is, N needs to be large enough such that $C_{\text{train}}^{\frac{1}{2}}N^{-\frac{\alpha}{2}} << 1$. Importantly, the constant C_{train} also depends on the architecture of the underlying BeNN. The C_{train} evaluation depends on the features of the governing DEs, and the training point collocation can not be worked out in the set up of the Theorem 1 [34]. The constant C_{pde} encodes the stability of the DEs and depends on both the exact (unknown) solution V and the trained BeNN, V^*, that needs to be bounded.

The main point of the estimate (33) is to provide an upper bound of the generalization error in terms of the training and quadrature errors, regardless of the choice of the BeNN architecture. The estimate (33) holds for any expansion of V, including NNs (of any types), and so does the algorithm 1. There is no guarantee that the training error is small if NNs are used to approximate V. However, it can be expected that if the DE is stable and the training points collocation scheme is accurate (e.g., there is some control on C_{pde} and C_{train}) the training error will be small, leading to a small generalization error.

For PINN TFC-based, the *training error* ε_T is given by,

$$\varepsilon_T = J(\theta^*)^{\frac{1}{2}} = \left(\sum_{n=1}^{N_{\text{int}}} w_n^{\text{int}} ||\mathcal{R}_{\text{int},\theta^*}(t_n, \boldsymbol{x}_n)||^2\right)^{\frac{1}{2}} \tag{41}$$

That is, ε_T can be evaluated once the training is completed and $\varepsilon_{G,\text{tfc}}$ can be estimated in terms of ε_T according to the following theorem, under the same assumptions made for the previous theorem.

Theorem 2. *Let $V \in Z \subset X^*$ be the unique solution of the DE (7)-(8) (and (25)-(26)) and assume that the stability hypothesis (H3) holds. Let $V_{\text{tfc}}^* \in Z \subset X^*$ be the BeNN computed via algorithm 2, based on the chosen training set $\mathcal{S}$. The following estimate on the generalization error holds,*

$$\varepsilon_{G,\text{tfc}} \leq C_{\text{pde}}\varepsilon_T + C_{\text{pde}}C_{\text{train}}^{\frac{1}{2}}N^{-\frac{\alpha}{2}} \tag{42}$$

with $C_{\text{pde}} = C_{\text{pde}}(||V||_Z, ||V_{\text{tfc}}^*||_Z)$ and $C_{\text{train}} = C_{\text{train}}(||\mathcal{R}_{\theta^*}^2||)$ (with $\mathcal{R}_\theta^*$ including solely the interior residual) stemming from (H3). Note that these constants C_{pde}, C_{train} depend on the BeNN, V_{tfc}^*, and on the number of training points $N = N_{\text{int}}$.

Proof: The proof of theorem 2 is equivalent to the proof of theorem 1. Thus it will not be repeated. ■

The considerations made for classic PINN holds true for PINN TFC-based.

III. Numerical Results

A finite and an infinite horizon OCPs with integral quadratic cost, both with linear and nonlinear dynamics, have been solved using the proposed algorithms. The problems selected are benchmarks problems with analytical solutions. We selected these problems to allow the readers to better appreciate the performances, understand the utility of the proposed PINN-based framework, and guide them through the step-by-step formulation using all the PINN frameworks. Moreover, having the analytical solutions allows us to compute the generalization error, and therefore to numerically test the theoretical findings. The formulation and the results are showed and analyzed below. The selected OCPs have been coded in Python 3.7 and ran with an Intel Core i7 - 9700 CPU PC with 64 GB of RAM. The training points are evenly spaced along all the dimensions, forming a uniform grid of points.

A. Problem 1: Finite Horizon Problem with Linear Dynamics

Consider the following finite horizon problem with linear dynamics (example 1 from [27]),

$$\min_{\boldsymbol{u}\in\mathcal{U}} \mathcal{J}(\boldsymbol{x}) = (x(t_f))^2 + \int_0^{t_f} u^2\, dt \tag{43}$$

subject to

$$\dot{x} = x + u \tag{44}$$

with t_f given, in particular $t_f = 1$.

The HJB equation for the problem is,

$$-V_t + \sup_u \left\{-V_x(x+u) - u^2\right\} = 0, \quad \forall t \in [0,1], \forall x \in [-1,1] \tag{45}$$

subject to,

$$V(1,x) = x^2, \quad \forall x \in [-1,1] \tag{46}$$

The optimal control is,

$$u = -\frac{1}{2}V_x \tag{47}$$

Thus, the resulting HJB PDE, which solution is learned using both algorithms 1 and 2, is,

$$-V_t - V_x x + \frac{1}{4}V_x^2 = 0, \quad \forall t \in [0,1], \forall x \in [-1,1] \tag{48}$$

subject to,

$$V(1,x) = x^2, \quad \forall x \in [-1,1] \tag{49}$$

The analytical expression of the value function is,

$$V_{\text{exact}}(t,x) = \frac{2x^2}{1 + \exp\{2(t-1)\}} \tag{50}$$

thus the exact optimal control is,

$$u_{\text{exact}}(t,x) = -\frac{2x}{1 + \exp\{2(t-1)\}} \tag{51}$$

To generate the results, different PINN frameworks have been considered for comparison, such as classic PINNs, Deep-TFC, and X-TFC. All the NNs employ the hyperbolic tangent

This article has been accepted for publication in IEEE Transactions on Artificial Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TAI.2022.3206735

as the activation function. For classic PINN and Deep-TFC, the unknown parameters θ have been randomly initialized. For X-TFC, input weights and bias have been sampled from $\mathcal{U}[-1,1]$, and the output weights have been initialized by setting them all equal to zero. When classic PINN and Deep-TFC are employed for the BeNNs training, the ADAM optimizer with initial learning rate equal to 0.001 has been adopted [23]. For the training with X-TFC, iterative least-squares with tolerance equal to 10^{-8} has been used (note that the tolerance is defined according to the losses of two subsequent iterations). The results are reported in Tables I-II, where ε_G and ε_T represent the generalization error and the training error as defined in section 2, respectively. Table I shows the BeNNs training with $N_{\text{int}} = 400$ internal training points, and $N_{\text{tb}} = 200$ for classic PINNs. As can be seen, X-TFC generally outperforms the other two methods, both in terms of accuracy and computational times. Moreover, it is worth to notice that, in order to obtain the same accuracy of classic PINN and Deep-TFC, X-TFC requires just one layer and also a lower number of neurons (as illustrated by the case with $L = 20$). The corresponding training time, in the order of 16 milliseconds, can also suggest a potential online (and real-time) training of the BeNNs. This is very promising for future applications. For Table II, the same hyperparameters of Table I are used, apart from the internal training points ($N_{\text{int}} = 4900$ for Table II), and the temporal boundary training points for classic PINNs ($N_{\text{tb}} = 2450$ for Table II). As expected, the training time for the BeNNs training increases as a function of the higher number of training points. However, especially for classic PINNs and Deep-TFC, more training points do not lead to obtain more accurate solutions for this example. This is also valid for X-TFC, for which an accurate solution was already found for fewer training points. The attentive reader can also observe that the generalization error ε_G is lower than the training error ε_T, proving numerically theorems 1 and 2. For the case of X-TFC with $N_{\text{int}} = 400$ and $L = 200$, another simulation has been performed to test the deployment of the network. Indeed, the optimal control actions obtained from the value function learned during the training have been exploited to propagate (44) forward in time, starting from the initial condition $x_0 = -1$. The corresponding results are reported in Fig. 1. The left panels show the integrated state with the BeNN-based optimal control action. The right panel reports the absolute error with respect to the integrated analytical solution. The absolute error is reported to be in the order of 10^-11, proving that the proposed methodology is capable of providing highly accurate solutions. A Monte Carlo analysis has also been performed, considering 1000 different initial conditions sampled randomly from a uniform distribution within the range [-1,1]. The distribution of the errors between the state at final time, obtained after the propagation, and the corresponding analytical solutions is reported in the histogram of Fig. 2. The error in the order of 10^{-10} demonstrates that BeNNs can robustly and accurately learn the optimal control actions even with perturbed initial conditions.

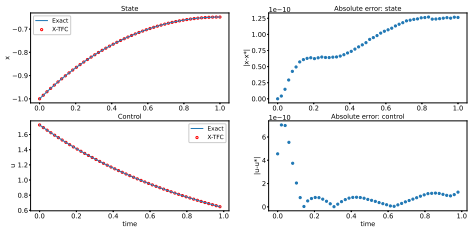


Figure 1. States, control and the corresponding error during the integration for the finite horizon problem ($N_{\text{int}} = 400$, $L = 20, x_0 = -1$). L is the number of neurons.

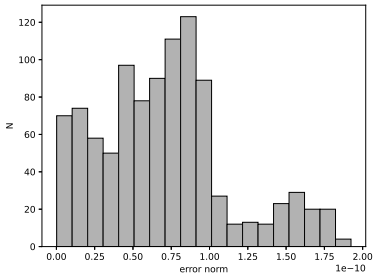


Figure 2. Histogram of the error between the final approximated and true states in the Monte Carlo analysis ($N_{\text{int}} = 400$, $L = 20$)

B. Problem 2: Infinite Horizon Problem with Nonlinear Dynamics

Consider the following infinite horizon problem with nonlinear dynamics (example 2 from [35]),

$$\min_{\boldsymbol{u}\in\mathcal{U}} \mathcal{J} = \int_0^\infty (\boldsymbol{x}^T\boldsymbol{x} + u^2)\, dt \tag{52}$$

subject to

$$\begin{aligned} \dot{x_1} &= -x_1 + x_2 \\ \dot{x_2} &= -\frac{1}{2}x_1 - \frac{1}{2}x_2 + \frac{1}{2}x_1^2 x_2 + x_1\, u \end{aligned} \tag{53}$$

Using equation (22), the optimal control is,

$$u = -\frac{1}{2}x_1 V_{x_2} \tag{54}$$

Thus the resulting HJB PDE, which solution is learned using both algorithms 1 and 2, is,

$$x_1^2 + x_2^2 + (-x_1 + x_2)\, V_{x_1} + \left(-\frac{1}{2}x_1 - \frac{1}{2}x_2 + \frac{1}{2}x_1^2 x_2\right) V_{x_2} - \frac{1}{4}x_1^2 V_{x_2}^2 = 0 \tag{55}$$

subject to the following condition,

$$V(\mathbf{0}) = 0 \tag{56}$$

in $x_1 \in [-1, 1]$ and $x_2 \in [-1, 1]$.

Table I
RESULTS FOR PROBLEM 1, WITH $N_{\text{int}} = 400$

PINN Framework	Layers	Neurons	Epochs	Training time [s]	ε_G	ε_T
Classic PINN	2	200	15000	348.87	6.89×10^{-4}	2.00×10^{-3}
Classic PINN	4	200	15000	668.80	6.41×10^{-4}	1.17×10^{-2}
Classic PINN	10	200	15000	1647.90	1.22×10^{-3}	4.01×10^{-3}
Deep-TFC	2	200	15000	320.78	2.48×10^{-5}	1.47×10^{-4}
Deep-TFC	4	200	15000	662.10	4.87×10^{-5}	1.60×10^{-4}
Deep-TFC	10	200	15000	1666.71	1.16×10^{-4}	4.16×10^{-4}
X-TFC	1	20	8	0.016	3.09×10^{-4}	1.44×10^{-3}
X-TFC	1	200	4	3.95	2.91×10^{-11}	8.56×10^{-11}
X-TFC	1	2000	4	15.95	2.04×10^{-11}	3.79×10^{-11}

Table II
RESULTS FOR PROBLEM 1, WITH $N_{\text{int}} = 4900$

PINN Framework	Layers	Neurons	Epochs	Training time [s]	ε_G	ε_T
Classic PINN	2	200	15000	1048.82	6.13×10^{-4}	3.35×10^{-3}
Classic PINN	4	200	15000	2356.83	5.40×10^{-4}	2.46×10^{-3}
Classic PINN	10	200	15000	4231.86	3.50×10^{-3}	5.12×10^{-2}
Deep-TFC	2	200	15000	1671.83	3.12×10^{-5}	1.71×10^{-4}
Deep-TFC	4	200	15000	2349.12	4.13×10^{-5}	2.38×10^{-4}
Deep-TFC	10	200	15000	2856.98	6.42×10^{-5}	2.31×10^{-4}
X-TFC	1	20	8	6.05	2.36×10^{-4}	1.23×10^{-3}
X-TFC	1	200	4	9.56	4.78×10^{-11}	5.82×10^{-10}
X-TFC	1	2000	4	292.22	1.80×10^{-11}	1.03×10^{-10}

The exact solution for the value function and the optimal control are,

$$V_{exact}(\boldsymbol{x}) = \frac{1}{2}x_1^2 + x_2^2 \tag{57}$$

$$u_{\text{exact}}(t, x) = -x_1 x_2 \tag{58}$$

For what concerns the numerical results, the same considerations carried out for the previous example are still valid. For this problem, one training point in $\boldsymbol{x} = \mathbf{0}$ has been used for classic PINNs, whereas the tolerance for X-TFC iterative least-squares has been set to 10^{-5}. Even for this case, it is possible to observe from Tables III-IV that the best accuracy and computational times are obtained with X-TFC. Moreover, Fig. 3 shows the results of the integration of Eqs. (53) under the application of the deployed optimal control actions learned during the training of the BeNNs. The comparison with respect to the analytical solution and the corresponding absolute errors between the real and approximated states and control prove the accuracy of the BeNNs training. The results of a Monte Carlo analysis for 1000 simulations with initial conditions sampled from a uniform random distribution within the range $[-\mathbf{1}, \mathbf{1}]$ are reported in Fig. 4. The errors between the approximated and analytical final states are on the order of 10^{-10}, proving the robustness of the BeNNs to compute accurate solutions starting from different initial states within the domain.

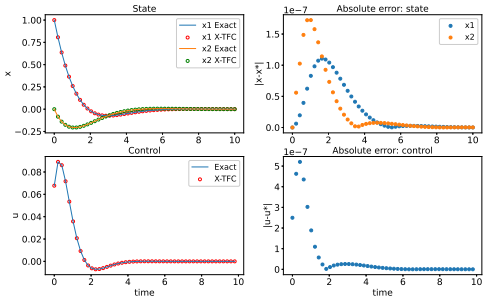


Figure 3. States, control and the corresponding error during the integration for the infinite horizon problem 2 ($N_{\text{int}} = 400$, $L = 20, X_0 = -1$)

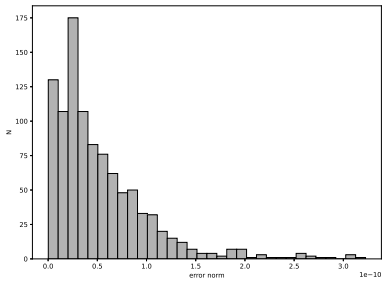


Figure 4. Histogram of the error between the final approximated and true states in the Monte Carlo for the infinite horizon problem 2 analysis ($N_{\text{int}} = 400$, $L = 20$)

IV. CONCLUSIONS

In this paper, we focused on the development of the BeNNs and used them to learn the HJB PDE solutions for OCPs with integral quadratic cost. Using BeNNs allows to retrieve the optimal control in an analytical form, represented by the NNs. Indeed, once the optimal control is learned on the training points, no further interpolation (usually needed for traditional methods) is required to evaluate it on the test/query

This article has been accepted for publication in IEEE Transactions on Artificial Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TAI.2022.3206735

Table III
RESULTS FOR PROBLEM 2, WITH $N_{\text{int}} = 400$

PINN Framework	Layers	Neurons	Epochs	Training time [s]	ε_G	ε_T
Classic PINN	2	200	15000	199.69	1.06×10^{-3}	2.03×10^{-3}
Classic PINN	4	200	15000	300.86	1.20×10^{-3}	1.01×10^{-2}
Classic PINN	10	200	15000	553.83	3.74×10^{-3}	2.79×10^{-1}
Deep-TFC	2	200	15000	210.02	2.48×10^{-5}	3.41×10^{-5}
Deep-TFC	4	200	15000	459.02	1.98×10^{-5}	2.03×10^{-5}
Deep-TFC	10	200	15000	1236.61	2.21×10^{-5}	3.38×10^{-5}
X-TFC	1	20	8	0.016	5.30×10^{-4}	6.91×10^{-4}
X-TFC	1	200	5	4.02	1.44×10^{-7}	2.71×10^{-7}
X-TFC	1	2000	5	16.47	8.16×10^{-7}	1.06×10^{-6}

Table IV
RESULTS FOR PROBLEM 2, WITH $N_{\text{int}} = 4900$

PINN Framework	Layers	Neurons	Epochs	Training time [s]	ε_G	ε_T
Classic PINN	2	200	15000	571.16	3.60×10^{-4}	1.24×10^{-3}
Classic PINN	4	200	15000	1074.71	1.84×10^{-3}	1.20×10^{-2}
Classic PINN	10	200	15000	2380.45	3.49×10^{-3}	3.07×10^{-2}
Deep-TFC	2	200	15000	541.1	2.14×10^{-5}	3.08×10^{-5}
Deep-TFC	4	200	15000	1099.01	2.96×10^{-5}	3.14×10^{-5}
Deep-TFC	10	200	15000	2806.87	2.18×10^{-5}	3.14×10^{-5}
X-TFC	1	20	7	4.02	2.35×10^{-4}	5.48×10^{-4}
X-TFC	1	200	5	88.11	2.98×10^{-6}	3.00×10^{-6}
X-TFC	1	2000	5	289.89	9.78×10^{-7}	1.10×10^{-6}

points. This results in a closed-loop optimal control that can be deployed to compute optimal trajectories in real-time, as showed in Fig. 5. Indeed, the deployment time is in order of milliseconds. Three PINN frameworks have been tested and compared. The main goal of this work is to show the feasibility in using PINN-based methods to learn closed-loop optimal control, by directly solving the HJB equation, and to select the best PINN method for the class of OCPs considered. The results show that X-TFC is the best option to tackle this class of OCPs. Our theoretical findings on the estimation of the generalization error generated by BeNNs in learning the solution of the HJB PDEs has been tested through benchmark OCPs. The low computational time required for the training of BeNNs also suggests a possible use for online training. Further investigations will be carried out in the future about this possibility and its applications. The proposed approach can be applied in all the research fields where OCPs with integral quadratic cost must be solved. The authors are currently focusing on the space engineering applications, because of the significant importance that generating real-time closed-loop optimal control brings to spacecraft autonomy. In particular, satellite optimal attitude control and trajectory planning are currently under investigation. In addition, efforts are in progress to solve large-scale OCPs belonging to the class considered here and for other classes of OCPs, such as problems with discontinuous control. Future works will also focus on the stability analysis of the closed-loop system obtained with the control generated from the solution of the HJB equation [36]. Finally, future works will focus on the comparison of BeNNs against some of the most common and used methods to solve HJB PDEs.

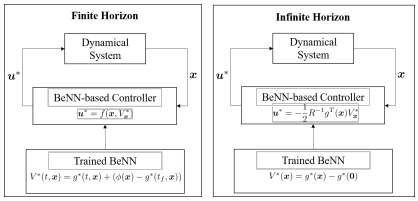


Figure 5. Schematic on how to deploy trained BeNNs to control a dynamical system

REFERENCES

[1] E. Schiassi, A. D'Ambrosio, A. Scorsoglio, R. Furfaro, and F. Curti, "Class of optimal space guidance problems solved via indirect methods and physics-informed neural networks," in *31st AAS/AIAA Space flight Mechanics Meeting*, 2021.

[2] L. Kong, W. He, Y. Dong, L. Cheng, C. Yang, and Z. Li, "Asymmetric bounded neural control for an uncertain robot by state feedback and output feedback," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 51, no. 3, pp. 1735–1746, 2019.

[3] Z. Li, X. Li, Q. Li, H. Su, Z. Kan, and W. He, "Human-in-the-loop control of soft exosuits using impedance learning on different terrains," *IEEE Transactions on Robotics*, 2022.

[4] B. R. E., "Dynamic programming," *Princeton University, Press, Princeton, NJ*, 1957.

[5] C. M. Chilan and B. A. Conway, "Optimal nonlinear control using hamilton–jacobi–bellman viscosity solutions on unstructured grids," *Journal of Guidance, Control, and Dynamics*, vol. 43, no. 1, pp. 30–38, 2020.

[6] E. Cristiani and P. Martinon, "Initialization of the shooting method via the hamilton-jacobi-bellman approach," *Journal of Optimization Theory and Applications*, vol. 146, no. 2, pp. 321–346, 2010.

[7] J. N. Reddy, "An Introduction to the Finite Element Method," *Journal of Pressure Vessel Technology*, vol. 111, no. 3, pp. 348–349, 08 1989. [Online]. Available: https://doi.org/10.1115/1.3265687

[8] H.-G. Zhang, X. Zhang, L. Yan-Hong, and Y. Jun, "An overview of research on adaptive dynamic programming," *Acta Automatica Sinica*, vol. 39, no. 4, pp. 303–311, 2013.

[9] Y. Li, J. Zhang, W. Liu, and S. Tong, "Observer-based adaptive optimized control for stochastic nonlinear systems with input and state constraints," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[10] S. Baldi, I. Michailidis, E. B. Kosmatopoulos, A. Papachristodoulou, and P. A. Ioannou, "Convex design control for practical nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 59, no. 7, pp. 1692–1705, 2014.

[11] E. B. Kosmatopoulos, "Control of unknown nonlinear systems with efficient transient performance using concurrent exploitation and exploration," *IEEE Transactions on Neural Networks*, vol. 21, no. 8, pp. 1245–1261, 2010.

[12] I. Michailidis, S. Baldi, E. B. Kosmatopoulos, and P. A. Ioannou, "Adaptive optimal control for large-scale nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 62, no. 11, pp. 5567–5577, 2017.

[13] F. L. Lewis and D. Vrabie, "Reinforcement learning and adaptive dynamic programming for feedback control," *IEEE circuits and systems magazine*, vol. 9, no. 3, pp. 32–50, 2009.

[14] A. Al-Tamimi, F. L. Lewis, and M. Abu-Khalaf, "Model-free q-learning designs for linear discrete-time zero-sum games with application to h-infinity control," *Automatica*, vol. 43, no. 3, pp. 473–481, 2007.

[15] 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*, vol. 378, pp. 686–707, 2019.

[16] D. Mortari, "The Theory of Connections: Connecting Points," *MDPI Mathematics*, vol. 5, no. 57, 2017.

[17] E. Schiassi, R. Furfaro, C. Leake, M. De Florio, H. Johnston, and D. Mortari, "Extreme theory of functional connections: A fast physics-informed neural network method for solving ordinary and partial differential equations," *Neurocomputing*, 2021.

[18] C. Leake and D. Mortari, "Deep theory of functional connections: A new method for estimating the solutions of partial differential equations," *Machine learning and knowledge extraction*, vol. 2, no. 1, pp. 37–55, 2020.

[19] S. Wang, Y. Teng, and P. Perdikaris, "Understanding and mitigating gradient flow pathologies in physics-informed neural networks," *SIAM Journal on Scientific Computing*, vol. 43, no. 5, pp. A3055–A3081, 2021.

[20] C. Leake, H. Johnston, and D. Mortari, "The multivariate theory of functional connections: Theory, proofs, and application in partial differential equations," *Mathematics*, vol. 8, no. 8, p. 1303, 2020.

[21] A. D'Ambrosio, E. Schiassi, F. Curti, and R. Furfaro, "Pontryagin neural networks with functional interpolation for optimal intercept problems," *Mathematics*, vol. 9, no. 9, p. 996, 2021.

[22] E. Schiassi, A. D'Ambrosio, K. Drozd, F. Curti, and R. Furfaro, "Physics-informed neural networks for optimal planar orbit transfers," *Journal of Spacecraft and Rockets*, pp. 1–16, 2022.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[24] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM Journal on scientific computing*, vol. 16, no. 5, pp. 1190–1208, 1995.

[25] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, " Extreme learning machine: Theory and applications ," *Neurocomputing*, vol. 70, no. 2006, pp. 489–501, May 2006.

[26] S. Mishra and R. Molinaro, "Estimates on the generalization error of physics-informed neural networks for approximating pdes," *IMA Journal of Numerical Analysis*, 2022.

[27] M. Mehrali-Varjani, M. Shamsi, and A. Malek, "Solving a class of hamilton-jacobi-bellman equations using pseudospectral methods," *Kybernetika*, vol. 54, no. 4, pp. 629–647, 2018.

[28] D. Y. Wang, *Study Guidance and Control for Lunar Soft Landing (Ph.D. Dissertation)*. School of Astronautics, Harbin Institute of Technology, Harbin, China, 2000.

[29] P. Mertikopoulos, C. Papadimitriou, and G. Piliouras, "Cycles in adversarial regularized learning," in *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2018, pp. 2703–2717.

[30] D. Balduzzi, S. Racaniere, J. Martens, J. Foerster, K. Tuyls, and T. Graepel, "The mechanics of n-player differentiable games," in *International Conference on Machine Learning*. PMLR, 2018, pp. 354–363.

[31] G.-B. Huang, L. Chen, and C.-K. Siew, "Universal approximation using incremental constructive feedforward networks with random hidden nodes," *IEEE Transactions on Neural Networks*, vol. 17, no. 4, p. 879–892, 2006.

[32] R. W. Beard, G. N. Saridis, and J. T. Wen, "Approximate solutions to the time-invariant hamilton–jacobi–bellman equation," *Journal of Optimization theory and Applications*, vol. 96, no. 3, pp. 589–626, 1998.

[33] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[34] T. De Ryck and S. Mishra, "Error analysis for physics informed neural networks (pinns) approximating kolmogorov pdes," *arXiv preprint arXiv:2106.14473*, 2021.

[35] W. Tang and P. Daoutidis, "Distributed adaptive dynamic programming for data-driven optimal control," *Systems & Control Letters*, vol. 120, pp. 36–43, 2018.

[36] T. Nakamura-Zimmerer, Q. Gong, and W. Kang, "Neural network optimal feedback control with enhanced closed loop stability," *arXiv preprint arXiv:2109.07466*, 2021.

Appendix E

Schiassi et al. (2021): X-TFC for Epidemiological Compartmental Models

Reproduced from Schiassi, E., De Florio, M., D'ambrosio, A., Mortari, D. and Furfaro, R., 2021. Physics-informed neural networks and functional interpolation for data-driven parameters discovery of epidemiological compartmental models. *Mathematics*, 9(17), p.2069. https://doi.org/10.3390/math9172069 [128], with the permission of MDPI.

mathematics

MDPI

Article

Physics-Informed Neural Networks and Functional Interpolation for Data-Driven Parameters Discovery of Epidemiological Compartmental Models

Enrico Schiassi [1], Mario De Florio [1], Andrea D'Ambrosio [1,2], Daniele Mortari [3] and Roberto Furfaro [1,4,*]

[1] Systems & Industrial Engineering, University of Arizona, Tucson, AZ 85721, USA; e.schiassi@email.arizona.edu (E.S.); mariodf@email.arizona.edu (M.D.F.); andrea.dambrosio@uniroma1.it (A.D.)
[2] School of Aerospace Engineering, Sapienza University of Rome, 00138 Rome, Italy
[3] Aerospace Engineering, Texas A&M University, College Station, TX 77843-3141, USA; mortari@tamu.edu
[4] Aerospace & Mechanical Engineering, University of Arizona, Tucson, AZ 85721, USA
* Correspondence: robertof@arizona.edu; Tel.: +1-520-621-2525

Abstract: In this work, we apply a novel and accurate Physics-Informed Neural Network Theory of Functional Connections (PINN-TFC) based framework, called Extreme Theory of Functional Connections (X-TFC), for data-physics-driven parameters' discovery of problems modeled via Ordinary Differential Equations (ODEs). The proposed method merges the standard PINNs with a functional interpolation technique named Theory of Functional Connections (TFC). In particular, this work focuses on the capability of X-TFC in solving inverse problems to estimate the parameters governing the epidemiological compartmental models via a deterministic approach. The epidemiological compartmental models treated in this work are Susceptible-Infectious-Recovered (SIR), Susceptible-Exposed-Infectious-Recovered (SEIR), and Susceptible-Exposed-Infectious-Recovered-Susceptible (SEIRS). The results show the low computational times, the high accuracy, and effectiveness of the X-TFC method in performing data-driven parameters' discovery systems modeled via parametric ODEs using unperturbed and perturbed data.

Keywords: Physics-Informed Neural Networks; functional interpolation; Theory of Functional Connections; Extreme Learning Machine; epidemiological compartmental models; COVID-19

Citation: Schiassi, E.; De Florio, M.; D'Ambrosio, A.; Mortari, D.; Furfaro, R. Physics-Informed Neural Networks and Functional Interpolation for Data-Driven Parameters Discovery of Epidemiological Compartmental Models. *Mathematics* **2021**, *9*, 2069. https://doi.org/10.3390/math9172069

Academic Editor: Massimiliano Ferrara

Received: 29 July 2021
Accepted: 25 August 2021
Published: 27 Auguts 2021

1. Introduction

The concern for viruses' spread has been in the researchers' spotlight for many years [1–4]. Particularly, in the last year and a half, due to the COVID-19 pandemic, this concern has become a hot topic in many research fields [5–13]. Many models exist to study the spread of viruses. The first categorization between these models can be made for deterministic and stochastic models [14–17].

Deterministic models are the simplest, with fixed input variables. They are also known as compartmental models because the individuals in the population are assigned to different subgroups, or compartments, each of which represents a specific condition of the individual in the epidemic situation [18]. Derivatives in time are used to express the transition rates of individuals from a compartment to another. Thus, the model is constructed as a system of Ordinary Differential Equations (ODEs).

Stochastic models take into account variations in input variables and provide results in terms of probability. Unlike a deterministic one, a stochastic model allows random variations in one or more inputs over time. Therefore, an estimation of the probability distributions of the outcomes can be carried out. Specifically, the variables changing in time can be the exposure risk, recovery rate, and other disease dynamics. Being able to insert the variability of the input data, the stochastic models have a more complex structure than the deterministic ones but manage to be more adherent to reality [19]. A second categorization

between the models can be made by taking into account (or not taking into account) the vital dynamics. The vital dynamics represent the demography dynamics, in which the naturally occurring births and deaths are included [20].

In this paper, deterministic models with vital dynamics are studied. Precisely, the Susceptible-Infectious-Recovered (SIR), Susceptible-Exposed-Infectious-Recovered (SEIR), and Susceptible-Exposed-Infectious-Recovered-Susceptible (SEIR) models are considered, with the vaccination factor for one of those [21,22].

This work aims to estimate various epidemiological model parameters by using the newly developed framework called *Extreme Theory of Functional Connections* (X-TFC) [23], which merges the Physics-Informed Neural Networks (PINNs) method, introduced by Raissi et al. [24], and the Theory of Functional Connections (TFC), proposed by Mortari [25]. This method aims to solve forward problems and inverse problems (*data-driven parameters discovery*) involving DEs in different perturbation scenarios. A typical field where solving inverse problems is of interest is remote sensing [26–29]. For instance, in Reference [30], the authors combine radiative and heat transfer equations to create a set of parametric DEs. The solutions of this system of equations are compared with real data to retrieve the grain size and the thermal inertia of planetary regoliths, which are the parameters governing the physics of the problem. There are two main approaches to solving mathematical and physical inverse problems: deterministic and probabilistic. The deterministic approach tackles inverse problems using standard optimization techniques. According to these techniques, a set of optimal parameters is found, which minimizes the difference between simulated and real data. However, inverse problems are known to be ill-posed [31] and hence, it becomes challenging to determine the uncertainty in the retrieved quantities mainly due to the noise in the observed data and the uncertainty in the real values of the input parameter that are not tuned.

As stated in [32], inverse problems to parameters' estimation are in general ill-posed because the problem is non-unique due to the higher number of unknowns than data/measurements and the stability of the solution to noise in the data and modeling errors is generally not guaranteed. Standard optimization techniques consider the tuning quantities to be deterministic. Therefore, the inverse problems' outputs are fixed quantities. However, these quantities are affected by uncertainties that need to be estimated. The issue is that uncertainty quantification (usually done via regularization techniques) is not trivial to perform, and it can lead to poor results, mainly when the problem is ill-posed. Moreover, nonlinear or non-convex inverse problems have more local minimum solutions. Thus, more than one acceptable solution can be computed, and it becomes challenging to select the best one via the classical optimization framework [33].

To overcome this issue, the probabilistic approach can be used, in particular Bayesian inversion techniques. In the Bayesian inversion framework, the quantities to be estimated are considered random variables. Thus, the output of the inverse modeling is the probability distribution for each of those parameters. Therefore, with the probabilistic approach, the degree of uncertainty of the quantities' values to be retrieved is included in their probability distributions [26].

Nevertheless, in this work, we tackle the inverse problem for data-driven parameters' discovery of epidemiological models via a deterministic approach. We show that solving these problems via Physics-Informed Neural Network (PINN) methods, such as X-TFC, mitigates the ill-posedness of the inverse problems toward modeling errors and noisy data. This is possible because the physics of the problem, modeled via a DE, acts as a regulator during the search of the optimal parameters (i.e., the NN training). That is, the network training is carried in a data-physics-driven fashion.

This manuscript is organized as follows. In Section 2, PINNs are introduced with particular regard to the X-TFC framework. In Section 3, the application of the X-TFC for the data-driven discovery of the parameters governing a few of the most common epidemiological compartmental models is presented. In Section 3, the results are presented and discussed. Finally, the concluding remarks are given in the last section.

2. Physics-Informed Neural Network and Functional Interpolation

PINNs are machine learning methods that include physics into a data-driven functional representation of input–output pairing collections. As defined by Raissi et al. [24], the term PINN describes NNs that embed the physics as a regularization term in the loss function. For instance, suppose that one aims to do a regression of an experimental dataset employing an NN and that the collected data represents some physical events modeled via a set of DEs. In conventional regression, one would approximate the data utilizing a NN trained to minimize a Mean Squared Error (MSE) as a loss function. Nevertheless, there is no guarantee that the physics phenomena governing the dataset would not be violated. To mitigate this issue, PINNs are introduced to ensure that the physics, modeled via DEs, is added as a penalty to the loss function. This extra term serves as a regularizator that penalizes the training when the DE and its constraints (e.g., Boundary Conditions BCs, and eventually Initial Conditions ICs) are violated. Thus, one can guarantee that the physics underlying the process is not violated. This method is defined as a data-physics-driven solution of DEs. More specifically, from the physics perspective, this approach enables the training of NNs to learn the solution of DEs in a data-physics-driven fashion. This becomes critical if the DEs do not precisely model the physics of the problem, for example, when uncertain dynamical systems are considered and/or when perturbations are nonnegligible. Conversely, when the purpose is to retrieve parameters governing some physical phenomena modeled via DEs (e.g., single scattering albedo in the radiative transfer equation), one usually refers to data-physics-driven parameters discovery of DEs (i.e., inverse problems) [24]. When data is not available, and consequently the loss function contains the residual of the DEs and its constraints solely, PINNs learn the solutions of problems involving DEs only in a physics-driven fashion.

The major shortcoming of the standard PINNs, as presented by Raissi et al. [24], is that the DE constraints are not analytically satisfied, and consequently, they need to be concurrently learned with the DE solution within the domain. Hence, during the PINN training, we deal with competing objectives: learning the DE hidden solution within the domain and the DE hidden solution on the boundaries. This drives to unbalanced gradients during the network training via gradient-based techniques that prompt PINNs to struggle frequently to accurately learn the underlying DE solution [34]. Gradient-based optimization methods may get stuck in limit cycles or diverge if several competing objectives are present [35,36]. In [34], to surmount this issue, the authors proposed a learning rate annealing algorithm that uses gradient statistics to adaptive assign proper weights to different terms (e.g., DE residuals within the domain and DE residuals on the boundaries) in the PINNs loss function during the training. In this work, we propose to employ a different and more robust PINN model, the Extreme Theory of Functional Connections (X-TFC), that merges NNs and the Theory of Functional Connections (TFC) [23,37]. X-TFC exploits the constrained expressions (CEs) introduced within the TFC to satisfy the boundary constraints analytically.

TFC, elaborated by Mortari [25], is a mathematical framework for functional interpolation where functions are approximated using these CEs. A CE is a functional that is a sum of a free function and a functional that analytically satisfies the constraints despite the choice of the free function [25,38]. TFC has multiple applications. Primarily, TFC is used for the solution of DEs because the CEs eliminate the "curse of the equation constraints" [39–41]. Moreover, TFC has already been used to solve different classes of optimal control space guidance problems such as energy optimal landing on large and small planetary bodies [42,43], fuel optimal landing on large planetary bodies [44], energy optimal relative motion problems subject to Clohessy-Wiltshire dynamics [45], and classes of transport theory problems, such as radiative transfer [46] and rarefied-gas dynamics [47]. For tackling DEs, the standard (or Vanilla as defined in [48,49]) TFC method employs a linear combination of orthogonal polynomials, such as Legendre or Chebyshev polynomials [39,40], as a free function. However, using orthogonal polynomials as a free function makes the standard TFC framework suffer from the curse of dimensionality, particularly

when solving large-scale PDEs. To overcome this limitation, X-TFC employs a shallow NN trained via the Extreme Learning Machine (ELM) algorithm [50] to represent the free function.

Being a PINN method, X-TFC can solve forward and inverse problems involving parametric DEs with high precision and low computational time. The method for solving direct problems involving parametric DEs is introduced and presented by Schiassi et al. [23]. As previously stated, the focus of this work is to apply the X-TFC for data-driven parameters discovery of compartmental epidemiological models such as SIR, SEIR, and SEIRS. In the remainder of this section, we will explain how the X-TFC is applied to tackle these problems. Such models are systems of ODEs, where the constraints are on the initial values of these systems' solutions. That is, these problems are initial value problems (IVPs). Therefore, in this section, we will also present the step-by-step derivation of the constrained expressions for these problems.

2.1. Generality on Neural Networks

Neural Networks (NNs) are one of the key components of the X-TFC framework that will be used to tackle the problems considered in this work. Therefore, for the convenience of the reader, before diving into the detailed explanation on how X-TFC works, we will give some generalities about NNs.

NNs are powerful mathematical tools, inspired by the biological neurosystems of the human brain, originally developed as function approximators for machine learning applications [51–53].

NNs are made by artificial neurons and their mutual connections. The output of every neuron is a non-linear function of the weighted sum of its inputs. The neurons are typically arranged into layers, whose number defines the type of NN. NNs with only a single layer of neurons are known as single-layer or shallow NNs. NNs with more than one layer of neurons are called Deep NNs (DNNs). Neurons and layers are not necessarily all connected among them. When all neurons and layers are connected, we generally talk about fully connected NNs (shallow or deep depending on the number of layers). Every layer of a fully connected NN can be mathematically represented as follows

$$\boldsymbol{z} = \sigma(W\boldsymbol{x} + \boldsymbol{b}) \tag{1}$$

where W is the weight matrix, $\boldsymbol{x}$ and $\boldsymbol{z}$ are the input and output vectors, respectively, $\boldsymbol{b}$ is the bias vector, and $\sigma(\cdot)$ is the activation function, which can be either different or the same for every layer.

As previously stated, NNs were originally introduced as function approximators thanks to their ability in interpolation and fitting. An NN function approximator works in a supervised manner, where the training set $\mathcal{T} = \{(x_i, y_i)\}_{i=1,\dots,N}$ consists in x_i input points and y_i output points (which can be affected or not by noise). First, a trial function $\hat{y}(x)$ is randomly initialized and the loss function can be defined as

$$\mathcal{L} = \sum_{i=1}^{N} |\hat{y}(x_i) - y_i|^2 \tag{2}$$

The training process consists in solving an optimization problem, where the loss function ($\mathcal{L}$) is the objective function that needs to be minimized, and the decision variables are the weights and biases of each layer. Usually, the training is performed via stochastic gradient based methods such as Adam Optimizer [54].

2.2. Extreme Theory of Functional Connections (X-TFC)

In this work, we will focus on systems of ODEs (SODEs) used to describe epidemiological compartmental models. In general, we can express ODEs, in their implicit form as,

$$\mathcal{N}[f; \boldsymbol{\lambda}] + \varepsilon - \mathcal{R} = 0 \tag{3}$$

subject to constraints given by initial conditions (IC) and/or boundary conditions (BC). In Equation (3), $f = f(x; \boldsymbol{\lambda}(x))$ is the unknown (or latent) solution, with $x \in \mathbb{D} \subseteq \mathbb{R}$, $\boldsymbol{\lambda} = \boldsymbol{\lambda}(x) \in \mathbb{L} \subseteq \mathbb{R}^m$ are the parameters governing the ODE (In general, even if it is not reported in the notation, f is a function of x, and it is parametrized by $\boldsymbol{\lambda}$, that in general can be x dependent as well.), $\mathcal{N}[\cdot; \boldsymbol{\lambda}]$ is a linear or non-linear operator acting on f and parametrized by $\boldsymbol{\lambda}$, ε is the modeling error that is negligible when solving problems where the physics is exactly modeled by the underlying DE, and $\mathcal{R}$ is a known term that in general can be x dependent and parametrized by $\boldsymbol{\lambda}$ as well.

The first step in our PINN-TFC based framework is to approximate the latent solution f with a constrained expression, defined within the TFC [25],

$$f(x; \boldsymbol{\lambda}) = f_{CE}(x, g(x); \boldsymbol{\lambda}) = A(x; \boldsymbol{\lambda}) + B(x, g(x); \boldsymbol{\lambda}) \tag{4}$$

where $A(x; \boldsymbol{\lambda})$ analytically satisfies the DE constraints, and $B(x, g(x); \boldsymbol{\lambda})$ projects the free function $g(x)$, which is a real valued function, onto the space of functions that vanish at the constraints [37]. In the X-TFC method we chose the free function, $g(x)$, to be a shallow NN, trained via ELM algorithm [50]. That is,

$$g(x) = \sum_{j=1}^{L} \xi_j \sigma(w_j x + b_j) = \begin{bmatrix} \sigma_1 \\ \vdots \\ \sigma_L \end{bmatrix}^{\mathrm{T}} \boldsymbol{\xi} = \boldsymbol{\sigma}^{\mathrm{T}} \boldsymbol{\xi} \tag{5}$$

where L is the number of hidden neurons, $w_j \in \mathbb{R}$ is the input weights vector connecting the jth hidden neuron and the input nodes, $\xi_j \in \mathbb{R}$ with $j = 1, ..., L$ is the jth output weight connecting the jth hidden neuron and the output node, and b_j is the bias of the jth hidden neuron, $\sigma(\cdot)$ are activation functions, and $\boldsymbol{\sigma} = [\sigma_1, \cdots, \sigma_L]^{\mathrm{T}}$. According to the ELM algorithm [50], biases and input weights are randomly selected and not tuned during the training, thus they are known hyperparameters. The activation functions, $\sigma(\cdot)$, are also known, as they are user selected. Thus, the only unknown NN hyperparameters to compute are the output weights $\boldsymbol{\xi} = [\xi_1, \cdots, \xi_L]^{\mathrm{T}}$. Hence we can write,

$$f(x; \boldsymbol{\lambda}) = f_{CE}(x, g(x); \boldsymbol{\lambda}) = f_{CE}(x, \boldsymbol{\xi}; \boldsymbol{\lambda}).$$

The step-by-step process to derive the constrained expression is provided in Section 2.2.1. Once f is approximated with a NN, the second step of the X-TFC method is to define the loss functions,

$$\mathcal{L}_{\text{DATA}} = f_{\text{DATA}} - f_{CE} \tag{6}$$

$$\mathcal{L}_{\text{DE}} = \mathcal{N}[f_{CE}; \boldsymbol{\lambda}] + \varepsilon - \mathcal{R} \tag{7}$$

where f_{DATA} are the real data that eventually can be perturbed. Once the losses are defined, we need to define the vectors with all the unknowns, that are the $\boldsymbol{\xi}$ coefficients and the parameters governing the equations $\boldsymbol{\lambda}$,

$$\boldsymbol{\Xi} = \{\boldsymbol{\xi}, \quad \boldsymbol{\lambda}\}$$

Now, by combining the losses, an augmented loss function vector is formed as follows,

$$\mathbb{L} = \{\mathcal{L}_{\text{DATA}}, \quad \mathcal{L}_{\text{DE}}\}^{\mathrm{T}} \tag{8}$$

and enforcing that for a true solution, this vector should be equal to 0. This allows the unknowns to be solved via different optimization schemes, e.g., least-square for linear problems [39] and iterative least-squares for non-linear problems [40]. When solving

inverse problems for parameter estimation, the iterative least-square method is required. Thus, the estimation of the unknowns is updated at each iteration as follows,

$$\Xi_{k+1} = \Xi_k + \Delta\Xi_k \tag{9}$$

where the k subscript refers to the current iteration. In general, the $\Delta\Xi_k$ term can be defined by performing classic linear least-square at each iteration of the iterative least-square procedure as follows,

$$\Delta\Xi_k = -\left(\mathbb{J}^{\mathrm{T}}(\Xi_k)\mathbb{J}(\Xi_k)\right)^{-1}\mathbb{J}(\Xi_k)^{\mathrm{T}}\mathbb{L}(\Xi_k) \tag{10}$$

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

$$L_2[\mathbb{L}(\Xi_k)] < \epsilon \qquad \text{or} \qquad L_2[\mathbb{L}(\Xi_{k+1})] > L_2[\mathbb{L}(\Xi_k)]. \tag{11}$$

where ϵ defines some user prescribed tolerance.

In Figure 1, a schematic that summarizes how the X-TFC algorithm works for solving inverse problems is shown. The main steps are also reported here:

1. Approximate the latent solution(s) with the CE;
2. Analytically satisfy the ICs/BCs;
3. Expand with the single layer NN (trained via ELM);
4. Substitute into the DE (that can be also a system of DEs);
5. Build the DE losses (that drive the training of the network, informing it with the physics of the problem);
6. Build the data losses (the data can be provided on the solutions and/or on their derivatives);
7. Train the network;
8. Build the approximate solution (with the estimated optimal parameters).

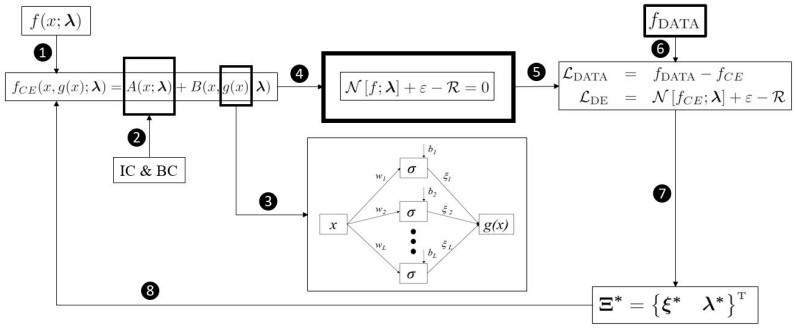


Figure 1. Schematic of the X-TFC framework for solving inverse problems.

2.2.1. Constrained Expression Derivation

Since this paper focuses on IVPs, for the convenience of the reader, we will present the step-by-step derivation of the constrained expression for these kinds of problems. The interested reader can find the general derivation for a $n+1$ dimensional constrained expression either in [37] or [23]. Given a parametric ODE where we have a constraint on

the initial value of the solution (i.e., $f(0) = f_0$), the constrained expression for f is the following [25],

$$f_{CE}(x, g(x)) = g(x) + \eta = \boldsymbol{\sigma}^{\intercal}\boldsymbol{\xi} + \eta \tag{12}$$

By imposing the constraint f_0 into Equation (12) we get the following,

$$\eta = f_0 - g_0 = f_0 - \boldsymbol{\sigma}_0^{\intercal}\boldsymbol{\xi} \tag{13}$$

Now by plugging this results back into Equation (12) we get,

$$f_{CE} = \left[\boldsymbol{\sigma} - \Omega_1\boldsymbol{\sigma}_0\right]^{\intercal}\boldsymbol{\xi} + \Omega_1 f_0, \tag{14}$$

where Ω_1 is called switching function. For an IVP with one constraint on f, we have $\Omega_1(z) = \Omega_1 = 1$.

In general f is defined in $x \in [x_0, x_f]$, which can be inconsistent with the domain where the activation function are defined. Thus we need to map it into the $z \in [z_0, z_f]$ domain as follows,

$$z = z_0 + c(x - x_0) \quad \leftrightarrow \quad x = x_0 + \frac{1}{c}(z - z_0), \tag{15}$$

where the mapping coefficient c is,

$$c = \frac{\mathrm{d}f}{\mathrm{d}x} = \frac{\Delta z}{x_f - x_0} \tag{16}$$

According to the chain rule of the derivative we then have,

$$\frac{\mathrm{d}^{\mathrm{n}}f}{\mathrm{d}x^n} = c^n\frac{\mathrm{d}^{\mathrm{n}}f}{\mathrm{d}z^n} \tag{17}$$

3. Epidemiological Models Formulation

In this section, the X-TFC formulation for the data-driven parameters discovery of a series of epidemiological compartmental models is explained in detail. The presented models are the SIR [55], SEIR [56], and SEIRS [57], taking into account the vital dynamics and the vaccination (for the SEIR model) [58]. As already mentioned, the goal is to estimate the parameters of our interest through solving inverse problems via a deterministic approach.

Given fixed parameters, by integration, we solve the systems of ODEs to create a synthetic dataset (with and without noise), through which the parameters that govern the physics of the problem can be retrieved. After building the constrained expressions and the loss functions, the Jacobian matrix (the matrix containing the derivatives of the losses with respect to the unknowns) is computed in order to perform the iterative least-squares and estimate the unknowns.

3.1. SIR Model

As first problem, we consider the system of differential equations that govern the classic deterministic **SIR** (**S**usceptible-**I**nfectious-**R**ecovered) compartmental model, in which individuals in the recovered state gain total immunity to the pathogen, with vital dynamics to take into account the births (that can provide an increase in susceptible individuals) and natural death rates. The DEs governing the SIR model are the following,

$$\begin{cases} \frac{dS}{dt} &= \mu N - \beta\frac{SI}{N} - \mu S \\ \frac{dI}{dt} &= \beta\frac{SI}{N} - \gamma I - \mu I \\ \frac{dR}{dt} &= \gamma I - \mu R \end{cases} \qquad \text{subject to} \qquad \begin{cases} S(t_0) &= S_0 \\ I(t_0) &= I_0 \\ R(t_0) &= R_0 \end{cases} \tag{18}$$

where $N = S + I + R$ is the total population, μ is the birth and natural death rate (considered equal to maintain a constant population), β is the infectious rate, and γ is the recovery rate. An important parameter to consider is the basic reproduction number $\mathcal{R}_0$, which represents the ratio between β and γ. If $\mathcal{R}_0 > 1$, an outbreak is going to occur.

According to the TFC framework, the latent solutions are approximated with the constrained expressions. That is,

$$\begin{cases} S &= (\boldsymbol{\sigma} - \Omega_1 \boldsymbol{\sigma_0})^{\mathrm{T}} \boldsymbol{\xi}_1 + \Omega_1 S_0 \\ I &= (\boldsymbol{\sigma} - \Omega_1 \boldsymbol{\sigma_0})^{\mathrm{T}} \boldsymbol{\xi}_2 + \Omega_1 I_0 \\ R &= (\boldsymbol{\sigma} - \Omega_1 \boldsymbol{\sigma_0})^{\mathrm{T}} \boldsymbol{\xi}_3 + \Omega_1 R_0 \end{cases} \tag{19}$$

The first three loss functions we present take into account the regression over the data. The last three losses drive the training of the NN, informing it with the physics governing the problem. The Loss functions are reported below,

$$\begin{cases} \mathcal{L}_1 &= \tilde{S} - S \\ \mathcal{L}_2 &= \tilde{I} - I \\ \mathcal{L}_3 &= \tilde{R} - R \\ \mathcal{L}_4 &= \dot{S} - \mu N + \beta \dfrac{SI}{N} + \mu S \\ \mathcal{L}_5 &= \dot{I} - \beta \dfrac{SI}{N} + \gamma I + \mu I \\ \mathcal{L}_6 &= \dot{R} - \gamma I + \mu R \end{cases} \tag{20}$$

To construct the Jacobian matrix $\mathcal{J}$ we need to compute the derivatives of the losses with respect to the $\boldsymbol{\xi}$ to compute the approximate solutions of the state variables, whereas the other derivatives are essential to estimate the parameters (in this case β and γ) appearing in the system of Equation (18). The resultant Jacobian matrix has the following form,

$$\mathcal{J} = \begin{bmatrix} \dfrac{\partial \mathcal{L}_1}{\partial \boldsymbol{\xi}_1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \dfrac{\partial \mathcal{L}_2}{\partial \boldsymbol{\xi}_2} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dfrac{\partial \mathcal{L}_3}{\partial \boldsymbol{\xi}_3} & \mathbf{0} & \mathbf{0} \\ \dfrac{\partial \mathcal{L}_4}{\partial \boldsymbol{\xi}_1} & \dfrac{\partial \mathcal{L}_4}{\partial \boldsymbol{\xi}_2} & \dfrac{\partial \mathcal{L}_4}{\partial \boldsymbol{\xi}_3} & \dfrac{\partial \mathcal{L}_4}{\partial \beta} & \mathbf{0} \\ \dfrac{\partial \mathcal{L}_5}{\partial \boldsymbol{\xi}_1} & \dfrac{\partial \mathcal{L}_5}{\partial \boldsymbol{\xi}_2} & \dfrac{\partial \mathcal{L}_5}{\partial \boldsymbol{\xi}_3} & \dfrac{\partial \mathcal{L}_5}{\partial \beta} & \dfrac{\partial \mathcal{L}_5}{\partial \gamma} \\ \mathbf{0} & \dfrac{\partial \mathcal{L}_6}{\partial \boldsymbol{\xi}_2} & \dfrac{\partial \mathcal{L}_6}{\partial \boldsymbol{\xi}_3} & \mathbf{0} & \dfrac{\partial \mathcal{L}_6}{\partial \gamma} \end{bmatrix} \tag{21}$$

3.2. SEIR Model

The second problem that we aim to solve is the **SEIR** (**S**usceptible-**E**xposed-**I**nfectious-**R**ecovered) compartmental model. This model, compared to the previous one, takes into account the incubation period of a virus, i.e., the time in which a subject comes into contact with the virus but still does not develop its symptoms. Therefore, the subject is infected but is not yet considered among the infectious. In addition, a vaccination parameter which

moves people from the Susceptible to Recovered directly is added. The following is the ODEs system describing the model,

$$\begin{cases} \frac{dS}{dt} &= \mu(N-S) - \beta\frac{SI}{N} - \nu S \\ \frac{dE}{dt} &= \beta\frac{SI}{N} - (\mu+\phi)E \\ \frac{dI}{dt} &= \phi E - \gamma I - \mu I \\ \frac{dR}{dt} &= \gamma I - \mu R + \nu S \end{cases} \quad \text{subject to} \quad \begin{cases} S(t_0) &= S_0 \\ E(t_0) &= E_0 \\ I(t_0) &= I_0 \\ R(t_0) &= R_0 \end{cases} \tag{22}$$

where $N = S + E + I + R$ is the total population, μ is the birth and natural death rate (considered equal to maintain a constant population), ν is the vaccination rate, β is the infectious rate, ϕ is the rate at which an Exposed person becomes Infectious, and γ is the recovery rate.

According to the TFC framework, the latent solutions are approximated with the constrained expressions. That is,

$$\begin{cases} S &= (\boldsymbol{\sigma} - \Omega_1\boldsymbol{\sigma_0})^{\mathrm{T}}\boldsymbol{\xi}_1 + \Omega_1 S_0 \\ E &= (\boldsymbol{\sigma} - \Omega_1\boldsymbol{\sigma_0})^{\mathrm{T}}\boldsymbol{\xi}_2 + \Omega_1 E_0 \\ I &= (\boldsymbol{\sigma} - \Omega_1\boldsymbol{\sigma_0})^{\mathrm{T}}\boldsymbol{\xi}_3 + \Omega_1 I_0 \\ R &= (\boldsymbol{\sigma} - \Omega_1\boldsymbol{\sigma_0})^{\mathrm{T}}\boldsymbol{\xi}_4 + \Omega_1 R_0 \end{cases} \tag{23}$$

The first four loss functions we present take into account the regression over the data. The last four losses drive the NN, informing it with the physics governing the problem. The loss functions are reported below,

$$\begin{cases} \mathcal{L}_1 &= \tilde{S} - S \\ \mathcal{L}_2 &= \tilde{E} - E \\ \mathcal{L}_3 &= \tilde{I} - I \\ \mathcal{L}_4 &= \tilde{R} - R \\ \mathcal{L}_5 &= \dot{S} - \mu(N-S) + \beta\frac{SI}{N} + \nu S \\ \mathcal{L}_6 &= \dot{E} - \beta\frac{SI}{N} + (\mu+\phi)E \\ \mathcal{L}_7 &= \dot{I} - \phi E + (\gamma+\mu)I \\ \mathcal{L}_8 &= \dot{R} - \gamma I + \mu R - \nu S \end{cases} \tag{24}$$

To construct the Jacobian matrix $\mathcal{J}$ we need to compute the derivatives of the losses in respect of the $\boldsymbol{\xi}$ to compute the approximate solutions of the state variables, whereas the other derivatives are essential to estimate the parameters (in this case β, γ, and ϕ) appearing in the system of Equation (18). The resultant Jacobian matrix has the following form,

$$\mathcal{J} = \begin{bmatrix} \frac{\partial \mathcal{L}_1}{\partial \xi_1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \frac{\partial \mathcal{L}_2}{\partial \xi_2} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \frac{\partial \mathcal{L}_3}{\partial \xi_3} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \frac{\partial \mathcal{L}_4}{\partial \xi_4} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \frac{\partial \mathcal{L}_5}{\partial \xi_1} & \frac{\partial \mathcal{L}_5}{\partial \xi_2} & \frac{\partial \mathcal{L}_5}{\partial \xi_3} & \frac{\partial \mathcal{L}_5}{\partial \xi_4} & \frac{\partial \mathcal{L}_5}{\partial \beta} & \mathbf{0} & \mathbf{0} \\ \frac{\partial \mathcal{L}_6}{\partial \xi_1} & \frac{\partial \mathcal{L}_6}{\partial \xi_2} & \frac{\partial \mathcal{L}_6}{\partial \xi_3} & \frac{\partial \mathcal{L}_6}{\partial \xi_4} & \frac{\partial \mathcal{L}_6}{\partial \beta} & \mathbf{0} & \frac{\partial \mathcal{L}_6}{\partial \phi} \\ \mathbf{0} & \frac{\partial \mathcal{L}_7}{\partial \xi_2} & \frac{\partial \mathcal{L}_7}{\partial \xi_3} & \mathbf{0} & \mathbf{0} & \frac{\partial \mathcal{L}_7}{\partial \gamma} & \frac{\partial \mathcal{L}_7}{\partial \phi} \\ \frac{\partial \mathcal{L}_8}{\partial \xi_1} & \mathbf{0} & \frac{\partial \mathcal{L}_8}{\partial \xi_3} & \frac{\partial \mathcal{L}_8}{\partial \xi_4} & \mathbf{0} & \frac{\partial \mathcal{L}_8}{\partial \gamma} & \mathbf{0} \end{bmatrix} \tag{25}$$

3.3. SEIRS Model

The last problem we present here, is the **SEIRS** (**S**usceptible-**E**xposed-**I**nfectious-**R**ecovered-**S**usceptible) compartmental model. This model is used in the case when the immunity of recovered individuals wane, and they return to exist in the category of Susceptibles. No vaccination is considered here. This model is governed by the following system of ODEs:

$$\begin{cases} \frac{dS}{dt} &= \mu N - \beta \frac{SI}{N} + \zeta R - \nu S \\ \frac{dE}{dt} &= \beta \frac{SI}{N} - \phi E - \nu E \\ \frac{dI}{dt} &= \phi E - \gamma I - \mu I \\ \frac{dR}{dt} &= \gamma I - \nu R - \zeta R \end{cases} \quad \text{subject to} \quad \begin{cases} S(t_0) &= S_0 \\ E(t_0) &= E_0 \\ I(t_0) &= I_0 \\ R(t_0) &= R_0 \end{cases} \tag{26}$$

where $N = S + E + I + R$ is the total population, μ is the natural deaths rate, ν is the new births rate, β is the infectious rate, ϕ is the rate at which an Exposed person becomes Infectious, ζ is the rate which Recovered individuals return to the Susceptible statue due to loss of immunity, and γ is the recovery rate.

According to the TFC framework, the latent solutions are approximated with the constrained expressions. That is,

$$\begin{cases} S &= (\boldsymbol{\sigma} - \Omega_1 \boldsymbol{\sigma_0})^{\mathrm{T}} \boldsymbol{\xi}_1 + \Omega_1 S_0 \\ E &= (\boldsymbol{\sigma} - \Omega_1 \boldsymbol{\sigma_0})^{\mathrm{T}} \boldsymbol{\xi}_2 + \Omega_1 E_0 \\ I &= (\boldsymbol{\sigma} - \Omega_1 \boldsymbol{\sigma_0})^{\mathrm{T}} \boldsymbol{\xi}_3 + \Omega_1 I_0 \\ R &= (\boldsymbol{\sigma} - \Omega_1 \boldsymbol{\sigma_0})^{\mathrm{T}} \boldsymbol{\xi}_4 + \Omega_1 R_0 \end{cases} \tag{27}$$

The first four loss functions we present take into account the regression over the data. The last four losses drive the NN informing it with the physics governing the problem. The loss functions are reported below,

$$\begin{cases} \mathcal{L}_1 &= \tilde{S} - S \\ \mathcal{L}_2 &= \tilde{E} - E \\ \mathcal{L}_3 &= \tilde{I} - I \\ \mathcal{L}_4 &= \tilde{R} - R \\ \mathcal{L}_5 &= \dot{S} - \mu N + \beta \frac{SI}{N} - \zeta R + \nu S \\ \mathcal{L}_6 &= \dot{E} - \beta \frac{SI}{N} + (\phi + \nu) E \\ \mathcal{L}_7 &= \dot{I} - \phi E + (\gamma + \mu) I \\ \mathcal{L}_8 &= \dot{R} - \gamma I + (\nu + \zeta) R \end{cases} \tag{28}$$

To construct the Jacobian matrix $\mathcal{J}$ we need to compute the derivatives of the losses in respect of the $\boldsymbol{\xi}$ to compute the approximate solutions of the state variables, whereas the other derivatives are essential to estimates the parameters (in this case β, γ, and ϕ) appearing in the system of Equation (18). The resultant Jacobian matrix has the following form:

$$\mathcal{J} = \begin{bmatrix} \frac{\partial \mathcal{L}_1}{\partial \boldsymbol{\xi}_1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \frac{\partial \mathcal{L}_2}{\partial \boldsymbol{\xi}_2} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \frac{\partial \mathcal{L}_3}{\partial \boldsymbol{\xi}_3} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \frac{\partial \mathcal{L}_4}{\partial \boldsymbol{\xi}_4} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \frac{\partial \mathcal{L}_5}{\partial \boldsymbol{\xi}_1} & \frac{\partial \mathcal{L}_5}{\partial \boldsymbol{\xi}_2} & \frac{\partial \mathcal{L}_5}{\partial \boldsymbol{\xi}_3} & \frac{\partial \mathcal{L}_5}{\partial \boldsymbol{\xi}_4} & \frac{\partial \mathcal{L}_5}{\partial \beta} & \mathbf{0} & \mathbf{0} & \frac{\partial \mathcal{L}_5}{\partial \zeta} \\ \frac{\partial \mathcal{L}_6}{\partial \boldsymbol{\xi}_1} & \frac{\partial \mathcal{L}_6}{\partial \boldsymbol{\xi}_2} & \frac{\partial \mathcal{L}_6}{\partial \boldsymbol{\xi}_3} & \frac{\partial \mathcal{L}_6}{\partial \boldsymbol{\xi}_4} & \frac{\partial \mathcal{L}_6}{\partial \beta} & \mathbf{0} & \frac{\partial \mathcal{L}_6}{\partial \phi} & \mathbf{0} \\ \mathbf{0} & \frac{\partial \mathcal{L}_7}{\partial \boldsymbol{\xi}_2} & \frac{\partial \mathcal{L}_7}{\partial \boldsymbol{\xi}_3} & \mathbf{0} & \mathbf{0} & \frac{\partial \mathcal{L}_7}{\partial \gamma} & \frac{\partial \mathcal{L}_7}{\partial \phi} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \frac{\partial \mathcal{L}_8}{\partial \boldsymbol{\xi}_3} & \frac{\partial \mathcal{L}_8}{\partial \boldsymbol{\xi}_4} & \mathbf{0} & \frac{\partial \mathcal{L}_8}{\partial \gamma} & \mathbf{0} & \frac{\partial \mathcal{L}_9}{\partial \zeta} \end{bmatrix} \tag{29}$$

4. Results and Discussion

To test the ability of the X-TFC in performing data-driven parameters discovery of epidemiological compartmental models, we have created synthetic datasets according to the three models presented above (SIR, SEIR, and SEIRS). In particular, for each model, a no-noisy synthetic dataset (here called original dataset $\tilde{f}_{orig}$) has been generated by simply propagating the dynamics equations of the model using the MatLab function ODE113. In addition, to simulate a more realistic example, perturbed synthetic datasets ($\tilde{f}_{pert}$) have been created by adding noise to the original dataset. That is,

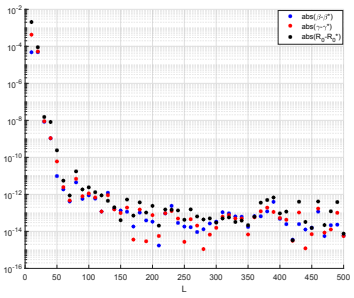
$$\tilde{f}_{pert} = \tilde{f}_{orig} + \delta \mathcal{U}(-1, -1) \tag{30}$$

where δ is the perturbation coefficient (equal to 0 for the original dataset) and $\mathcal{U}(\cdot, \cdot)$ represents a uniform distribution. The real values of the parameters governing the synthetic dataset are known, so that the accuracy of the results is measured by the absolute error between the real and estimated values of the parameters.

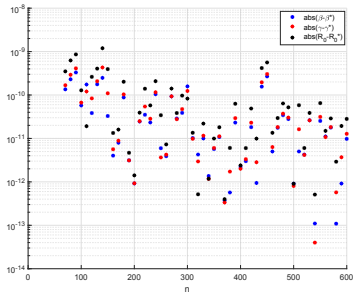
Additionally, the X-TFC method involves several hyperparameters that can be modified to obtain accurate solutions. These hyperparameters are the number of training points, n, the number of neurons, L, the type of activation function, and the probability distribution where input weights and bias are sampled from. Therefore, a sensitivity analysis

has been performed to study the behavior of the X-TFC method as these hyperparameters vary. The sensitivity analysis is only shown for the SIR model with no-noisy data, as a similar behavior has been encountered for all the other models considered. First of all, the sensitivity analysis has demonstrated that, for the models analyzed, the solution accuracy is not as sensitive to the type of activation function used or to the probability distribution used to sample the inputs weights and bias as it is to the number of training points and the number of neurons, confirming the results found from the sensitivity analysis reported in [23]. Hence, the two parameters that strongly influence the performances of the X-TFC are n and L. Figure 2a,b refer to the analysis with the original dataset ($\delta = 0$). As illustrated, high values of L ($L > 150$), with a fixed n, do not lead to an improvement of the accuracy, since Figure 2a presents an asymptotic-like behavior. The same considerations are valid varying n and keeping fixed L (Figure 2b). Indeed, the solution does not significantly improve increasing the number of discretization points. This result is also obtained if a perturbed dataset is considered (see Figure 2d). On the other hand, Figure 2c shows an interesting behavior. The accuracy of the solution gets worse by increasing the number of neurons L. This trend is probably due to the fact that X-TFC tries to overfit the perturbed data so that to diverge too much from the real curves and thus obtaining an inaccurate estimation of the parameters. The rest of this section focuses on the results obtained for each model presented previously. For these problems, the *ArcTan* activation function and a uniform random distribution ranging within [-10,10] are employed for the ELM.

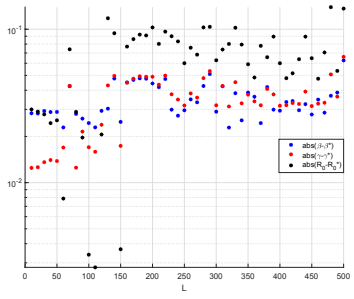
All the models tackled in this manuscript have been coded in MATLAB R2020a and ran with an Intel Core i7 - 9700 CPU PC with 64 GB of RAM.



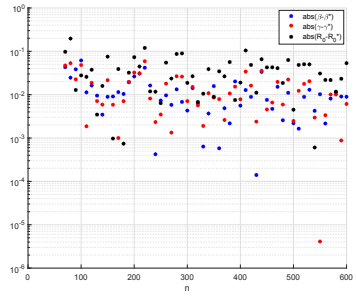
(**a**) $\delta = 0$, $n = 100$, and L varies.



(**b**) $\delta = 0$, $L = 100$, and n varies.



(**c**) $\delta = 1$, $n = 100$, and L varies.



(**d**) $\delta = 1$, $L = 100$, and n varies.

Figure 2. Monte Carlo simulations for SIR model with an *ArcTan* activation function.

4.1. SIR Model

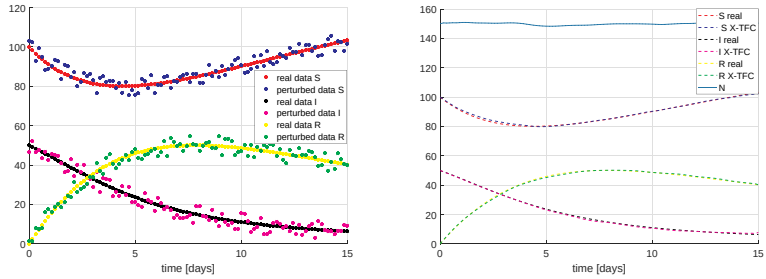
Here, the results and the performances for SIR problem are shown. The outputs are obtained by setting the following parameters:

- Natural mortality rate: $\mu = 0.1$ (set equal to the birth rate, to simulate a constant number of population);
- Effective contact rate (possibility to be infected): $\beta = \frac{1}{2}$;
- Removal rate (how often infected people become recovered): $\gamma = \frac{1}{3}$;
- Initial conditions: $S_0 = 100$; $I_0 = 5$; $R_0 = 0$;
- Analysis time: 15 days.

Several simulations are carried out by varying the intensity of the noise, and the outputs are reported in Table 1. While we could find the exact values of parameters with the original dataset, a slight deviation of these values occurs by increasing the perturbation coefficient δ. However, the absolute errors on the parameters result to have at least two digits of accuracy. Figure 3a,b report the perturbed and real dataset and the solution of the problem for the case of $\delta = 5$, respectively. As it can be seen, the X-TFC is able to obtain an accurate solution avoiding the overfitting on the data, as it could be expected by a simple regression on the perturbed dataset. This is due to the information about the physics of the problem, which acts as a regulator, that are embedded in the physics-informed training framework. The accuracy of the inversion with perturbed dataset is also proved by the constant value of the population N, as it has to be from the theory.

Table 1. Performances of the proposed physics-informed framework in the data-driven discovery of the SIR model with different noise on the data, with $n = 100$ and $L = 50$.

Noise	Iterations	CPU Time [s]	β	$\lvert\mathbf{err}(\beta)\rvert$	γ	$\lvert\mathbf{err}(\gamma)\rvert$	$\mathcal{R}_0$	$\lvert\mathbf{err}(\mathcal{R}_0)\rvert$
0	2	0.002	0.5000	0	0.3333	0	1.500	0
0.1	4	0.036	0.4999	4.2×10^{-5}	0.3334	4.2×10^{-5}	1.4997	3.2×10^{-4}
1	7	0.049	0.4996	4.4×10^{-4}	0.3338	4.2×10^{-4}	1.4968	3.2×10^{-3}
5	8	0.051	0.4978	2.2×10^{-3}	0.3353	2.1×10^{-3}	1.4884	1.56×10^{-2}



(**a**) Real and perturbed data. (**b**) Results with perturbed data.

Figure 3. Results for SIR model with $\delta = 5$, $n = 100$, and $L = 50$.

4.2. SEIR Model

Here, the results and the performances for SEIR problem are shown. The outputs were obtained by setting the following parameters:

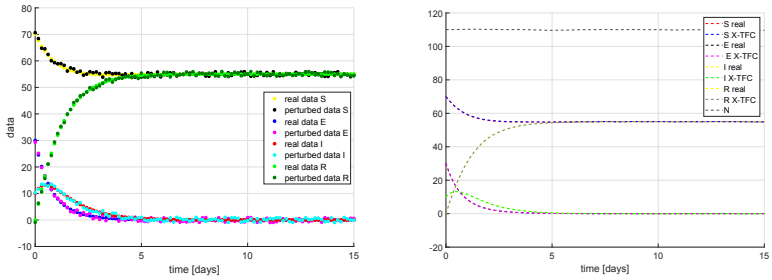
- Natural mortality rate $\mu = 0.5$ (set equal to the birth rate, to simulate a constant number of population);
- Vaccine rate $\nu = 0.5$;
- Effective contact rate (possibility to be infected) $\beta = 0.3$;
- Removal rate (how often infected people become recovered) $\gamma = 0.6$;
- Progression rate from exposed to infected $\phi = 0.9$;

- Initial conditions: $S_0 = 70$; $E_0 = 30$; $I_0 = 10$; $R_0 = 0$;
- Days = 15.

Several simulations are carried out by varying the intensity of the noise, and the outputs are reported in Table 2. While we could find the exact values of parameters with the original dataset, a slight deviation of these values occurs by increasing the perturbation coefficient δ. However, the absolute errors on the parameters result to have at least one digit of accuracy. Figure 4a,b report the perturbed and real dataset and the solution of the problem for the case of $\delta = 3$, respectively. Again, the X-TFC is able to obtain an accurate solution avoiding the overfitting on the data, as it could be expected by a simple regression on the perturbed dataset.

Table 2. Performances of the proposed physics-informed framework in the data-driven discovery of the SEIR model with different noise on the data, with $n = 100$ and $L = 80$, $\mu = \nu = 0.5$ (the training time is on the order of milliseconds).

Noise	β	$\lvert\mathbf{err}(\beta)\rvert$	γ	$\lvert\mathbf{err}(\gamma)\rvert$	ϕ	$\lvert\mathbf{err}(\phi)\rvert$	$\mathcal{R}_0$	$\lvert\mathbf{err}(\mathcal{R}_0)\rvert$
0	0.3	0	0.6	0	0.9	0	0.5	0
0.1	0.2971	2.9×10^{-3}	0.5996	3.7×10^{-4}	0.9005	4.9×10^{-4}	0.4955	4.5×10^{-3}
1	0.2711	2.9×10^{-2}	0.5962	3.8×10^{-3}	0.9048	4.8×10^{-3}	0.4547	4.5×10^{-2}
3	0.2130	8.7×10^{-2}	0.5878	1.2×10^{-2}	0.9141	1.4×10^{-2}	0.3624	1.4×10^{-1}



(**a**) Real and perturbed data. (**b**) Results with perturbed data.

Figure 4. Results for SEIR model with $\delta = 3$, $n = 100$, and $L = 80$.

4.3. SEIRS Model

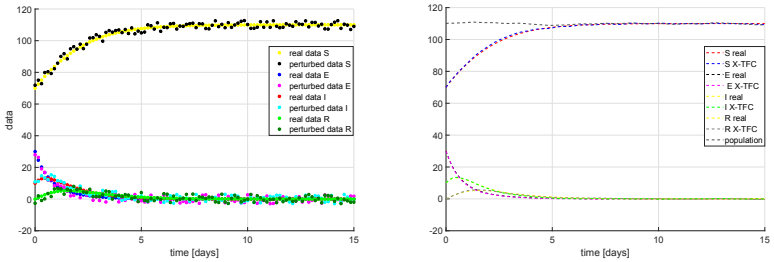
Here, the results and the performances for SEIRS problem are shown. The outputs were obtained by setting the following parameters:

- Natural mortality rate $\mu = 0.5$ (set equal to the birth rate, to simulate a constant number of population);
- Effective contact rate (possibility to be infected) $\beta = 0.3$;
- Removal rate (how often infected people become recovered) $\gamma = 0.6$;
- Progression rate from exposed to infected $\phi = 0.9$;
- Rate which recovered individuals return to the susceptible statue (due to loss of immunity) $\zeta = 0.5$;
- initial conditions: $S_0 = 70$; $E_0 = 30$; $I_0 = 10$; $R_0 = 0$;
- days = 15.

Several simulations are carried out by varying the intensity of the noise, and the outputs are reported in Table 3. While we could find the exact values of parameters with the original dataset, a slight deviation of these values occurs by increasing the perturbation coefficient δ. However, the absolute errors on the parameters result in having at least two digits of accuracy. Figure 5a,b report the perturbed and real dataset and the solution of the problem for the case of $\delta = 3$, respectively. Again, the X-TFC is able to obtain an accurate solution avoiding the overfitting on the data, as it could be expected by a simple regression on the perturbed dataset.

Table 3. Performances of the proposed physics-informed framework in the data-driven discovery of the SEIRS model with different noise on the data, with $n = 100$ and $L = 80$, $\mu = 0.5$ (the training time is on the order of milliseconds).

Noise	β	$\lvert\mathbf{err}(\beta)\rvert$	γ	$\lvert\mathbf{err}(\gamma)\rvert$	ϕ	$\lvert\mathbf{err}(\phi)\rvert$	ζ	$\lvert\mathbf{err}(\zeta)\rvert$	$\mathcal{R}_0$	$\lvert\mathbf{err}(\mathcal{R}_0)\rvert$
0	0.3	0	0.6	0	0.9	0	0.5	0	0.5	0
0.1	0.3011	1.1×10^{-3}	0.6020	2.0×10^{-3}	0.9026	2.6×10^{-3}	0.5028	2.8×10^{-3}	0.500	1.7×10^{-4}
1	0.3093	9.3×10^{-3}	0.6183	1.8×10^{-2}	0.9249	2.5×10^{-2}	0.5251	2.5×10^{-2}	0.5002	2.1×10^{-4}
3	0.3174	1.7×10^{-2}	0.6465	4.7×10^{-2}	0.9680	6.8×10^{-2}	0.5576	5.8×10^{-2}	0.4909	9.1×10^{-3}



(**a**) Real and perturbed data. (**b**) Results with perturbed data.

Figure 5. Results for SEIRS model with $\delta = 3$, $n = 100$, and $L = 80$.

5. Conclusions

In this work, the new PINN framework X-TFC has been employed to solve data driven discovery of DEs, also called inverse problems, via a deterministic approach. In particular, compartmental epidemiological models (SIR, SEIR, SEIRS) have been taken into account as test problems. The goal was to retrieve the parameters governing the dynamics equations considering unperturbed and perturbed data, to better simulate the reality. The tests have shown fairly accurate results even when a significant noise was added to the data. Furthermore, the information about the physics of the problem (considered for the training of the X-TFC) has allowed to avoid the overfitting and thus to obtain good estimations of parameters with noisy data. The low computational times obtained are extremely important to process data as soon as they are acquired, so that the results can be updated in real time. Moreover, the good estimations of parameters allow one to make predictions about the imminent future: this makes it possible to take actions in the short term (as it should be in emergency scenarios, like the COVID-19 pandemic). Future works involve the inversion of models with non-constant parameters (i.e., parameters that follow mathematical laws) as well as probabilistic parameters estimation (via Bayesian inversion) in different research fields, such as business, biology, space, and nuclear engineering.

Author Contributions: Conceptualization: E.S., A.D., M.D.F.; methodology: A.D., M.D.F. and E.S.; software: A.D., M.D.F. and E.S.; validation: A.D., E.S. and M.D.F.; formal analysis: A.D., E.S. and M.D.F.; investigation: A.D., M.D.F. and E.S.; resources: M.D.F., A.D. and E.S.; writing—original draft preparation: M.D.F., A.D. and E.S.; writing—review and editing: D.M. and R.F.; visualization: A.D., E.S. and M.D.F.; supervision: D.M. and R.F. 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.

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

References

1. Sharp, P.M.; Bailes, E.; Robertson, D.L.; Gao, F.; Hahn, B.H. Origins and evolution of AIDS viruses. *Biol. Bull.* **1999**, *196*, 338–342. [CrossRef]
2. Lowen, A.C.; Mubareka, S.; Tumpey, T.M.; García-Sastre, A.; Palese, P. The guinea pig as a transmission model for human influenza viruses. *Proc. Natl. Acad. Sci. USA* **2006**, *103*, 9988–9992. [CrossRef]
3. Geoghegan, J.L.; Senior, A.M.; Di Giallonardo, F.; Holmes, E.C. Virological factors that increase the transmissibility of emerging human viruses. *Proc. Natl. Acad. Sci. USA* **2016**, *113*, 4170–4175. [CrossRef]
4. Nelson, M.I.; Gramer, M.R.; Vincent, A.L.; Holmes, E.C. Global transmission of influenza viruses from humans to swine. *J. Gen. Virol.* **2012**, *93*, 2195. [CrossRef] [PubMed]
5. Kıymet, E.; Böncüoğlu, E.; Şahinkaya, Ş.; Cem, E.; Çelebi, M.Y.; Düzgöl, M.; Kara, A.A.; Arıkan, K.Ö.; Aydın, T.; İşgüder, R.; et al. Distribution of spreading viruses during COVID-19 pandemic: Effect of mitigation strategies. *Am. J. Infect. Control* **2021**. [CrossRef] [PubMed]
6. Galbadage, T.; Peterson, B.M.; Gunasekera, R.S. Does COVID-19 spread through droplets alone? *Front. Public Health* **2020**, *8*, 163. [CrossRef] [PubMed]
7. Arti, M.; Bhatnagar, K. Modeling and predictions for COVID 19 spread in India. *ResearchGate* **2020**. [CrossRef]
8. Castro, M.C.; Kim, S.; Barberia, L.; Ribeiro, A.F.; Gurzenda, S.; Ribeiro, K.B.; Abbott, E.; Blossom, J.; Rache, B.; Singer, B.H. Spatiotemporal pattern of COVID-19 spread in Brazil. *Science* **2021**, *372*, 821–826. [CrossRef] [PubMed]
9. Varotsos, C.A.; Krapivin, V.F. A new model for the spread of COVID-19 and the improvement of safety. *Saf. Sci.* **2020**, *132*, 104962. [CrossRef] [PubMed]
10. Caspi, G.; Shalit, U.; Kristensen, S.L.; Aronson, D.; Caspi, L.; Rossenberg, O.; Shina, A.; Caspi, O. Climate effect on COVID-19 spread rate: An online surveillance tool. *MedRxiv* **2020**. [CrossRef]
11. Aabed, K.; Lashin, M.M. An analytical study of the factors that influence COVID-19 spread. *Saudi J. Biol. Sci.* **2021**, *28*, 1177–1195. [CrossRef]
12. Piccolomiini, E.L.; Zama, F. Monitoring Italian COVID-19 spread by an adaptive SEIRD model. *MedRxiv* **2020**. [CrossRef]
13. Al-Kindi, K.M.; Alkharusi, A.; Alshukaili, D.; Al Nasiri, N.; Al-Awadhi, T.; Charabi, Y.; El Kenawy, A.M. Spatiotemporal assessment of COVID-19 spread over Oman using GIS techniques. *Earth Syst. Environ.* **2020**, *4*, 797–811. [CrossRef]
14. Adak, D.; Majumder, A.; Bairagi, N. Mathematical perspective of COVID-19 pandemic: Disease extinction criteria in deterministic and stochastic models. *Chaos Solitons Fractals* **2021**, *142*, 110381. [CrossRef] [PubMed]
15. Petrovskii, S.V.; Malchow, H.; Hilker, F.M.; Venturino, E. Patterns of patchy spread in deterministic and stochastic models of biological invasion and biological control. *Biol. Invasions* **2005**, *7*, 771–793. [CrossRef]
16. Perera, N.C. Deterministic and Stochastic Models of Virus Dynamics. Ph.D. Thesis, Texas Tech University, Lubbock, TX, USA, 2003.
17. Sazonov, I.; Grebennikov, D.; Kelbert, M.; Bocharov, G. Modelling stochastic and deterministic behaviours in virus infection dynamics. *Math. Model. Nat. Phenom.* **2017**, *12*, 63–77. [CrossRef]
18. Breda, D.; Diekmann, O.; De Graaf, W.; Pugliese, A.; Vermiglio, R. On the formulation of epidemic models (an appraisal of Kermack and McKendrick). *J. Biol. Dyn.* **2012**, *6*, 103–117. [CrossRef]
19. Britton, T. Stochastic epidemic models: A survey. *Math. Biosci.* **2010**, *225*, 24–35. [CrossRef]
20. Hethcote, H.W. Three basic epidemiological models. In *Applied Mathematical Ecology*; Springer: Berlin/Heidelberg, Germany, 1989; pp. 119–144.
21. Huang, G.; Takeuchi, Y.; Ma, W.; Wei, D. Global stability for delay SIR and SEIR epidemic models with nonlinear incidence rate. *Bull. Math. Biol.* **2010**, *72*, 1192–1207. [CrossRef] [PubMed]
22. Trawicki, M.B. Deterministic SEIRs epidemic model for modeling vital dynamics, vaccinations, and temporary immunity. *Mathematics* **2017**, *5*, 7. [CrossRef]
23. Schiassi, E.; Furfaro, R.; Leake, C.; De Florio, M.; Johnston, H.; Mortari, D. Extreme Theory of Functional Connections: A Fast Physics-Informed Neural Network Method for Solving Ordinary and Partial Differential Equations. *Neurocomputing* **2021**, *457*, 334–356. [CrossRef]
24. 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]
25. Mortari, D. The Theory of Connections: Connecting Points. *Mathematics* **2017**, *5*, 57. [CrossRef]
26. Schiassi, E.; Furfaro, R.; Kargel, J.S.; Watson, C.S.; Shugar, D.H.; Haritashya, U.K. GLAM Bio-Lith RT: A Tool for Remote Sensing Reflectance Simulation and Water Components Concentration Retrieval in Glacial Lakes. *Front. Earth Sci.* **2019**, *7*, 267. [CrossRef]
27. Schiassi, E.; Furfaro, R.; Mostacci, D. Bayesian inversion of coupled radiative and heat transfer models for asteroid regoliths and lakes. *Radiat. Eff. Defects Solids* **2016**, *171*, 736–745. [CrossRef]
28. Hapke, B. Bidirectional reflectance spectroscopy: 1. Theory. *J. Geophys. Res. Solid Earth* **1981**, *86*, 3039–3054. [CrossRef]
29. Hapke, B. A model of radiative and conductive energy transfer in planetary regoliths. *J. Geophys. Res. Planets* **1996**, *101*, 16817–16831. [CrossRef]
30. Hale, A.S.; Hapke, B. A time-dependent model of radiative and conductive thermal energy transport in planetary regoliths with applications to the Moon and Mercury. *Icarus* **2002**, *156*, 318–334. [CrossRef]

31. Kimes, D.S.; Knyazikhin, Y.; Privette, J.; Abuelgasim, A.; Gao, F. Inversion methods for physically-based models. *Remote Sens. Rev.* **2000**, *18*, 381–439. [CrossRef]
32. Kolehmainen, V. *Introduction to Bayesian Methods in Inverse Problems*; Department of Applied Physics, University of Eastern Finland: Kuopio, Finland, 2013.
33. Aster, R.C.; Borchers, B.; Thurber, C.H. *Parameter Estimation and Inverse Problems*; Elsevier: Amsterdam, The Netherlands, 2018.
34. Wang, S.; Teng, Y.; Perdikaris, P. Understanding and mitigating gradient pathologies in physics-informed neural networks. *arXiv* **2020**, arXiv:2001.04536.
35. Mertikopoulos, P.; Papadimitriou, C.; Piliouras, G. Cycles in adversarial regularized learning. In Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, New Orleans, LA, USA, 7–10 January 2018; pp. 2703–2717.
36. 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, PMLR, Stockholm, Sweden, 10–15 July 2018; pp. 354–363.
37. 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]
38. Mortari, D.; Leake, C. The Multivariate Theory of Connections. *Mathematics* **2019**, *7*, 296. [CrossRef] [PubMed]
39. Mortari, D. Least-squares solution of linear differential equations. *Mathematics* **2017**, *5*, 48. [CrossRef]
40. Mortari, D.; Johnston, H.; Smith, L. High accuracy least-squares solutions of nonlinear differential equations. *J. Comput. Appl. Math.* **2019**, *352*, 293–307. [CrossRef]
41. 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]
42. Furfaro, R.; Mortari, D. Least-squares solution of a class of optimal space guidance problems via Theory of Connections. *Acta Astronaut.* **2019**. [CrossRef]
43. 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. In Proceedings of the Astrodynamics Specialist Conference, AAS, South Lake Tahoe, CA, USA, 9–12 August 2020.
44. Johnston, H.; Schiassi, E.; Furfaro, R.; Mortari, D. Fuel-Efficient Powered Descent Guidance on Large Planetary Bodies via Theory of Functional Connections. *J. Astronaut. Sci.* under review. [CrossRef] [PubMed]
45. 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]
46. De Florio, M.; Schiassi, E.; Furfaro, R.; Ganapol, B.D.; Mostacci, D. Solutions of Chandrasekhar's basic problem in radiative transfer via theory of functional connections. *J. Quant. Spectrosc. Radiat. Transf.* **2021**, *259*, 107384. [CrossRef]
47. De Florio, M.; Schiassi, E.; Ganapol, B.; Furfaro, R. Physics-Informed Neural Networks for Rarefied-Gas Dynamics: Poiseuille Flow in the Bhatnagar–Gross–Krook approximation. *Phys. Fluids* **2021**, *33*, 047110. [CrossRef]
48. Johnston, H. The Theory of Functional Connections: A journey from theory to application. *arXiv* **2021**, arXiv:2105.08034.
49. Leake, C. The Multivariate Theory of Functional Connections: An n-Dimensional Constraint Embedding Technique Applied to Partial Differential Equations. *arXiv* **2021**, arXiv:2105.07070.
50. Huang, G.B.; Zhu, Q.Y.; Siew, C.K. Extreme learning machine: Theory and applications. *Neurocomputing* **2006**, *70*, 489–501. [CrossRef]
51. McCulloch, W.S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.* **1943**, *5*, 115–133. [CrossRef]
52. Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychol. Rev.* **1958**, *65*, 386. [CrossRef]
53. Rosenblatt, F. *Principles of Neurodynamics. Perceptrons and the Theory of Brain Mechanisms*; Technical Report; Cornell Aeronautical Lab Inc.: Buffalo, NY, USA, 1961.
54. Zhang, Z. Improved adam optimizer for deep neural networks. In Proceedings of the 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS), Banff, AB, Canada, 4–6 June 2018; pp. 1–2.
55. Brauer, F.; Driessche, P.; Wu, J. *Lecture Notes in Mathematical Epidemiology*; Springer: Berlin, Germany, 2008; Volume 75, pp. 3–22.
56. Röst, G. SEIR epidemiological model with varying infectivity and infinite delay. *Math. Biosci. Eng.* **2008**, *5*, 389–402.
57. Nakata, Y.; Kuniya, T. Global dynamics of a class of SEIRS epidemic models in a periodic environment. *J. Math. Anal. Appl.* **2010**, *363*, 230–237. [CrossRef]
58. Martcheva, M. *An Introduction to Mathematical Epidemiology*; Springer: Berlin/Heidelberg, Germany, 2015; Volume 61.