Journal of Scientific Computing (2026) 107:22
https://doi.org/10.1007/s10915-026-03220-z

Use of ELMs in Physics-Informed Machine Learning: Estimates on the Generalization Error in Learning ODE Solutions

Enrico Schiassi[1] · Francesco Calabrò[2] · Mario De Florio[3] · Davide Elia De Falco[4] · Guang-Bin Huang[5,6]

Received: 11 June 2024 / Revised: 21 January 2026 / Accepted: 7 February 2026

Abstract

Physics-Informed Neural Networks have been introduced to learn the solutions to Differential Problems. Extreme Learning Machines (ELMs), which are shallow feedforward architectures based on random projection, have been utilized along with other types of Neural Networks. Recently, ELM-based Physics-Informed Neural Networks have been combined with a functional interpolation technique called the Theory of Functional Connections, leading to an efficient method named the Extreme Theory of Functional Connections. The Extreme Theory of Functional Connections allows us to analytically satisfy the constraints, e.g., initial and/or boundary conditions that complete the differential problem removing one of the limitations of the traditional Physics-Informed Neural Networks frameworks: the need to learn the constraints alongside learning the solution within the domain. In this work, we provide upper bounds on the generalization error, in terms of the training error and number and choice of training points, of The Extreme Theory of Functional Connections in learning the solutions for the class of univariate Differential Problems. We fully cover three kinds of problems: first-order initial value problems, higher-order IVPs, and second-order two-point

✉ Francesco Calabrò
francesco.calabro@unina.it

Enrico Schiassi
enrico.schiassi2@unibo.it

Mario De Florio
mario_de_florio@brown.edu

Davide Elia De Falco
de.defalco@ssmeridionale.it

Guang-Bin Huang
egbhuang@ntu.edu.sg

1 Department of Industrial Engineering, University of Bologna, Bologna, Italy

2 Dipartimento di Matematica e Applicazioni “Renato Caccioppoli", Università degli Studi di Napoli “Federico II", Napoli, Italy

3 Division of Applied Mathematics, Brown University, Providence, RI, USA

4 Scuola Superiore Meridionale, Naples, Italy

5 School of Automation, Organization Southeast University, Nanjing 210096, China

6 Key Laboratory of Measurement and Control of Complex Systems of Engineering Organization Ministry of Education, Nanjing 210096, People’s Republic of China

Published online: 14 March 2026

boundary value problems. The theoretical results for these three cases are validated with numerical experiments on some benchmark linear and non-linear problems. Furthermore, the resolution using the proposed method is tested by comparing the results obtained with the ones computed by MATLAB's routines, and the proposed method outperforms the other solvers in terms of accuracy and computational efficiency.

Keywords Differential Equations · Generalization Error · Physics-Informed Neural Networks · Scientific Machine Learning · Extreme Learning Machine

Mathematics Subject Classification 65N35 · 65N75

1 Introduction

Differential Equations (DEs) are a powerful mathematical tool used to model many real-world problems in many different fields, such as physics, engineering, finance, biology, chemistry, oceanography, and epidemiology, to name a few. Therefore, there is a need to have accurate, efficient, robust, and reliable tools to solve problems modeled via DEs. The problems governed by DEs can be grouped into two macro-categories: forward problems and inverse problems. In the former, the task is to evaluate the function that solves the DEs (e.g., the heat flux in the heat transfer equations). In the latter, the task is to retrieve the parameters governing the DEs given data about the DE solution and/or its derivatives (e.g., to estimate the thermal conductivity in the heat equation, given some data on the heat flux).

There exist many DEs solvers. For Ordinary DE (ODEs), the most common and widely used solvers are the Runge-Kutta family methods [3, 25]. For Partial DEs (PDEs) the most common and widely used solvers are Finite Element Method (FEM) [29, 45, 53], Finite Difference Method (FDM) [33], Finite Volume Method (FVM) [22, 43], Isogeometric Analysis (IgA) [6], Spectral methods [5]. All these methods represent state-of-the-art DE solvers. They have been used for years for academic research and by industries and companies in many disciplines because they are accurate, robust, and reliable. Nonetheless, they present some non-negligible limitations. Some limitations of these state-of-the-art DEs solvers are that these methods are affected by the curse of dimensionality [2], and they are not designed to tackle inverse problems; see also [50] and references within.

Neural Networks (NNs) represent a solution to the limitations listed above. Employing NNs to tackle problems involving DEs has been introduced in the '90s. See, for instance, Ref. [19, 35]. However, only recently, thanks to the pioneering work by Raissi et al. [44], NNs have started to become widespread as a tool to tackle problems involving DEs. In [44], the authors introduced and formalized the concept of Physics-Informed Neural Networks (PINNs) for solving forward and inverse problems involving DEs. PINNs belong to the broader family of the Physics-Informed Machine Learning (PIML) frameworks [34]. PIML frameworks employ physics, usually modeled via DEs, and data to drive the training of Machine Learning (ML) algorithms. With PINNs, the data-physics-driven training aims to minimize the Mean Square Error (MSE). The total MSE is the sum of the MSE of the NN and the data, the MSE of the DE residuals (where the NN approximates the DE unknown solution), the MSE of the NN, and the DE constraints (e.g., initial and/or boundary conditions). When solving a forward problem, this training results in the NN parameters (e.g., weights and bias) that best approximate the DE unknown solution. When solving an inverse problem, the results of this training are the NN parameters along with the unknown DE governing parameters. PINNs

solve only forward problems when data are unavailable, and training is performed solely in a physics-driven fashion. Within this scenario, PINNs are numerical methods to solve DEs.

This work focuses only on PINNs trained in a physics-driven fashion. Our goal is to provide upper bounds on the generalization error in approximating the DEs solution of a novel PINN framework developed by the authors. We focus on three differential problems: first-order initial value problems (IVPs), higher-order IVPs, and second-order order two-point boundary value problems. This novel PINN framework combines PINNs with a functional interpolation technique called the Theory of Functional Connections (TFC). TFC has been introduced by Mortari in [42]; it is a general mathematical framework for linear functional interpolation. It derives functionals, called Constraint Expressions (CEs), containing a free function and a functional that always satisfies a set of specified linear constraints. These functionals reduce the solutions search space of constrained problems to just the subspace of functions that analytically satisfy the specified linear constraints. All the details on building these CEs are provided in [30, 37, 39] along with TFC applications, see also [9, 15, 31, 32, 38]. When solving DE, the original TFC employs orthogonal polynomials as free functions. This choice is ineffective when solving PDEs with more than two independent variables, as it makes the TFC suffer from the dimensionality curse. To avoid this limitation, PINN TFC-based methods use NNs as free functions. Two PINN TFC-based methods have been introduced: Deep-TFC and Extreme Theory of Functional Connections (X-TFC) [1, 50]. Deep-TFC [40] employs deep NN as a free function, while X-TFC employs shallow NN trained via the Extreme Learning Machine (ELM) algorithm. Both allow to satisfy analytically the DEs constraints removing one of the main limitations of the traditional PINN frameworks: the need to learn separately the DE constraints and the DE solution within the domain, see ad ex. [52] for a discussion on how this affects the overall determination of the solution. The ELM algorithm [27, 28] randomly samples input weights and bias. Thus, the training is needed solely for the output weights and is performed via least-squares. ELMs have been successfully applied for solving Ordinary Differential Equations (ODEs) and PDEs [4, 10, 20, 21, 23]. In all these, the constraints are not included in the formulation, as it is instead done in X-TFC.

Although the development of X-TFC was driven by the need to have an accurate, reliable, effective, and robust TFC-based method for large-scale PDE, X-TFC has resulted in being accurate, reliable, effective, and robust even for ODEs: it has already been employed to learn the solutions of several ODE classes [36]. In [16] and [17], the authors use X-TFC to learn the solution for a class of rarefied gas dynamics problems. In [49] and [14], they adapt and employ X-TFC to solve stiff nonlinear dynamical ODE systems for the Point Kinetic Equations (PKEs) with temperature feedback for nuclear reactor dynamics and stiff chemical kinetics, respectively. In [13], X-TFC is used for integro-DEs, and in [12] it is used to tackle sharp-gradient problems. In [48], X-TFC is employed for epidemiological compartmental models for parameter estimation. Finally, X-TFC has widely been used to learn the solutions of a class of optimal control problems via indirect method [7, 8, 46, 47], in particular for space applications and now in the groundfloor method for more challenging applications, such as black-box and gray-box systems identification [1, 11].

Despite all these excellent results on the efficient application of the method, it still lacks theoretical explanations, and only a few convergence estimates are available. One of the main results of this work is to present estimates for the so-called generalization error by following the procedure proposed by Mishra and Molinaro [41] to be applied to X-TFC to fill this gap and make X-TFC trustworthy. In the three classes of problems considered, these results are proved and then tested in benchmark cases. Our test always confirms the available bounds and emphasizes that, in some cases, these are too pessimistic.

The manuscript is organized as follows. In Section 2, we introduce the use of X-TFC in the different classes and discuss how the network is trained, namely fixing the training set, the residuals, and the loss function. In the same section, Theorems 1, 2, and 3 give the generalization error estimates for the DE classes mentioned above. In Section 3, the algorithm is described and numerical results are presented and discussed for the three classes of problems. In summary, the numerical results show that the proposed method has a wide range of applicability and that the error bounds–although sometimes pessimistic–are confirmed by the tests, while for stiff problems the approach needs to be enhanced with adaptive strategies. Extended conclusions are given in Section 4.

2 Estimate on the Generalization Error of X-TFC for DEs

In the present section, we detail the training of the proposed method and introduce the generalization error estimate in the different classes. We are interested in estimating the generalization error $\varepsilon_G \in \mathbb{R}$ for X-TFC, which is defined, according to [41], as

$$\varepsilon_G = \left(\int_{t_0}^{T} ||\boldsymbol{u}(t) - \boldsymbol{u}^*(t)||^2 dt \right)^{1/2} \tag{1}$$

where $\boldsymbol{u}(t)$ denotes the true solution and $\boldsymbol{u}^*(t)$ the one computed via X-TFC, within in the domain $t_0 \leq t \leq T$. It has been previously established in [50] that randomly chosen parameters in the activation functions lead to a neural network that falls within the ELM-type training framework [28]. This allows for universal approximation and interpolation-type results [27]. We recall that ELMs are Random projection NN that are widely used in various applications. See the review papers [18, 26, 51]

2.1 First Order Initial Value Problems

In this section, we derive an estimate of the generalization error when using X-TFC for learning the solution of a general system of non-linear first-order ODEs.

A general system of n first-order non-linear ODEs is posed as follows,

$$\begin{cases} \dfrac{d\boldsymbol{u}}{dt} = \boldsymbol{f}(\boldsymbol{u}, t) & \forall t \in [t_0, T], T \geq t_0 \\ \boldsymbol{u}(t_0) = \boldsymbol{u}_0 \end{cases} \tag{2}$$

Here, $\boldsymbol{u}_0 \in \mathbb{R}^n$ is the initial conditions, $\boldsymbol{u} \in \mathbb{R}^n$ defined in $[t_0, T] \subset \mathbb{R}$ is the solution of (2), and $\boldsymbol{f}$ is the non-linear forcing term. We assume that $\boldsymbol{f}$ is globally Lipschitz i.e., there exists a constant $C_f \geq 0$ (independent of t) such that,

$$||\boldsymbol{f}(\boldsymbol{v}) - \boldsymbol{f}(\boldsymbol{w})|| \leq C_f ||\boldsymbol{v} - \boldsymbol{w}||, \qquad \boldsymbol{v}, \boldsymbol{w} \in [t_0, T]. \tag{3}$$

In order to apply the X-TFC algorithm to learn the solution of (2), we need to specify the training set $\mathcal{S}$ and define the residuals.

Training set Consider the time domain $[t_0, T] \in \mathbb{R}$. We choose the training set $\mathcal{S} = \{t_i\}_{1 \leq i \leq N} \subset [t_0, T]$, based on suitable quadrature (training) points. According to Ref. [41], these points can be any quadrature points, corresponding to a suitable time grid-based composite Gauss quadrature rule or randomly selected points.

Residuals In order to apply the X-TFC algorithm, we need to define the residuals. We approximate the solution of (2) with the following CEs,

$$u_i(t;\boldsymbol{\beta}_i) = g(t;\boldsymbol{\beta}_i) + s(t)\eta_i \tag{4}$$

where $g(t;\boldsymbol{\beta}_i)$ is a single layer NN with random features, $s(t)$ is called a supporting function and can be defined by the user, see [42]. Finally, $\boldsymbol{\eta} \in \mathbb{R}^n$ is a vector of constant coefficients that are found by imposing the initial conditions.

In this trivial example we use $s(t) = 1$, from which we get $\eta_i = u_{i,0} - g(t_0;\boldsymbol{\beta}_i)$ and substituting in 4 we get the CEs

$$u_i(t;\boldsymbol{\beta}_i) = g(t;\boldsymbol{\beta}_i) + u_{i,0} - g(t_0;\boldsymbol{\beta}_i) \tag{5}$$

and their derivatives,

$$\frac{du_i(t;\boldsymbol{\beta}_i)}{dt} = \frac{dg(t;\boldsymbol{\beta}_i)}{dt} \tag{6}$$

Thus, we can write the CE in a vectorial form,

$$\boldsymbol{u}_{\boldsymbol{\beta}}(t) = \boldsymbol{u}(t;\boldsymbol{\beta}) = \boldsymbol{g}_{\boldsymbol{\beta}}(t) + \boldsymbol{u}_0 - \boldsymbol{g}_{\boldsymbol{\beta}}(t_0) \tag{7}$$

and their derivatives,

$$\frac{d\boldsymbol{u}_{\boldsymbol{\beta}}(t)}{dt} = \frac{d\boldsymbol{g}_{\boldsymbol{\beta}}(t)}{dt}\,. \tag{8}$$

Finally, we define the following residuals:

$$\boldsymbol{\mathcal{R}}(t;\boldsymbol{\beta}) = \boldsymbol{\mathcal{R}}_{\boldsymbol{\beta}}(t) = [\mathcal{R}_{1,\boldsymbol{\beta}} \dots \mathcal{R}_{n,\boldsymbol{\beta}}]^{\mathrm{T}} = \frac{d}{dt}\boldsymbol{u}_{\boldsymbol{\beta}}(t) - f(\boldsymbol{u}_{\boldsymbol{\beta}}(t), t) \tag{9}$$

Loss function To learn the unknown solution, we need to minimize the following vectorial loss function,

$$\boldsymbol{\mathcal{L}}(\boldsymbol{\beta}) = \begin{Bmatrix} \mathcal{R}_1(t_1;\boldsymbol{\beta}) \\ \vdots \\ \mathcal{R}_1(t_N;\boldsymbol{\beta}) \\ \vdots \\ \\ \vdots \\ \mathcal{R}_n(t_1;\boldsymbol{\beta}) \\ \vdots \\ \mathcal{R}_n(t_N;\boldsymbol{\beta}) \end{Bmatrix} \in \mathbb{R}^{n\cdot N} \tag{10}$$

The X-TFC algorithm is now completely specified and it can be run for learning the solution of the non-linear first-order ODE system of Eq. (2), which we denote as $\boldsymbol{u}^* = \boldsymbol{u}_{\boldsymbol{\beta}^*}$, where $\boldsymbol{\beta}^*$ are the computed optimal output weights.

We are interested in estimating the generalization error for X-TFC (1), that is the error emerging from the approximation of the true solution, $\boldsymbol{u}(t)$, via X-TFC, within the domain $t_0 \le t \le T$. We estimate the generalization error in terms of the *training error* $\varepsilon_T \in \mathbb{R}$ defined as,

$$\varepsilon_T^2 = \sum_{i=1}^{N} w_i ||\boldsymbol{\mathcal{R}}_{\boldsymbol{\beta}^*}(t_i)||^2 \tag{11}$$

where w_i are the quadrature weights. The reader can note that Eq. (11) is computed a posteriori once the training is completed.

According to Ref. [41], we assume the following for the quadrature error: for any function $\boldsymbol{y} \in C^l([t_0, T])$, the quadrature rule corresponding to quadrature weights w_i at points $t_i \in \mathcal{S}$, with $1 \leq i \leq N$, satisfies,

$$\left| \int_{t_0}^{T} ||\boldsymbol{y}(t)|| dt - \sum_{i=1}^{N} w_i ||\boldsymbol{y}(t_i)|| \right| \leq C_{\text{quad}}(||\boldsymbol{y}||_{C^l}) N^{-\alpha} \tag{12}$$

where $\alpha > 0$ and different order quadrature rules can be used.

The estimate of the generalization error for X-TFC is given in the following theorem.

Theorem 1 *Let $\boldsymbol{u} \in C^k([t_0, T])$ be the unique classical solution of the non-linear problem (2) with the dynamics $\boldsymbol{f}$ satisfying Eq. (3). Let $\boldsymbol{u}^* = \boldsymbol{u}_{\boldsymbol{\beta}^*}$ be the solution approximated via X-TFC, corresponding to loss function (10). Then, the generalization error (1) can be estimated as,*

$$\varepsilon_G \leq C_{DE} \left(\varepsilon_T^2 + C_{\text{quad}} N^{-\alpha} \right)^{\frac{1}{2}} \tag{13}$$

where $C_{DE} \in \mathbb{R}$ is given by

$$C_{DE} = \left(\frac{e^{-(1+2C_f)t_0}}{1+2C_f} \left(e^{(1+2C_f)T} - e^{(1+2C_f)t_0} \right) \right)^{\frac{1}{2}} \tag{14}$$

and $C_{\text{quad}} \in \mathbb{R}$ is

$$C_{\text{quad}} = C_{\text{quad}}(||\boldsymbol{\mathcal{R}}||^2_{C^{k-1}})$$

so that it depends on the number of training points, the quadrature scheme used, and the residuals evaluated on the training points.

Proof By the definitions of the residuals (9), and the system (2), we can verify that the error $\hat{\boldsymbol{u}} = \boldsymbol{u}^* - \boldsymbol{u}$ satisfies the following equation,

$$\begin{aligned} &\frac{d\hat{\boldsymbol{u}}}{dt} = \boldsymbol{f}(\boldsymbol{u}^*, t) - \boldsymbol{f}(\boldsymbol{u}, t) + \boldsymbol{\mathcal{R}}, \qquad \forall t \in [t_0, T] \\ &\hat{\boldsymbol{u}}(t_0) = \boldsymbol{0} \end{aligned} \tag{15}$$

Here, we have denoted $\boldsymbol{\mathcal{R}} = \boldsymbol{\mathcal{R}}_{\boldsymbol{\beta}^*}$ for convenience of notation. Multiplying both sides of Eq. (15) by $\hat{\boldsymbol{u}}$ yields,

$$\hat{\boldsymbol{u}} \cdot \frac{d\hat{\boldsymbol{u}}}{dt} = \hat{\boldsymbol{u}} \cdot \left[\boldsymbol{f}(\boldsymbol{u}^*, t) - \boldsymbol{f}(\boldsymbol{u}, t) \right] + \hat{\boldsymbol{u}} \cdot \boldsymbol{\mathcal{R}}$$

where $\cdot$ is the inner product. That is,

$$\frac{1}{2} \frac{d}{dt} ||\hat{\boldsymbol{u}}||^2 = \hat{\boldsymbol{u}} \cdot \left[\boldsymbol{f}(\boldsymbol{u}^*, t) - \boldsymbol{f}(\boldsymbol{u}, t) \right] + \hat{\boldsymbol{u}} \cdot \boldsymbol{\mathcal{R}} \tag{16}$$

By leveraging on the following inequality $\left(\hat{\boldsymbol{u}} - \boldsymbol{\mathcal{R}} \right)^2 \geq 0$, Eq. (16) is bounded by,

$$\frac{1}{2} \frac{d}{dt} ||\hat{\boldsymbol{u}}||^2 \leq ||\hat{\boldsymbol{u}}|| \left| |\boldsymbol{f}(\boldsymbol{u}^*, t) - \boldsymbol{f}(\boldsymbol{u}, t)| \right| + \frac{1}{2} ||\boldsymbol{\mathcal{R}}||^2 + \frac{1}{2} ||\hat{\boldsymbol{u}}||^2$$

By Eq. (3), we have the following,

$$\frac{1}{2} \frac{d}{dt} ||\hat{\boldsymbol{u}}||^2 \leq ||\hat{\boldsymbol{u}}|| C_f \underbrace{||\boldsymbol{u}^* - \boldsymbol{u}||}_{||\hat{\boldsymbol{u}}||} + \frac{1}{2} ||\boldsymbol{\mathcal{R}}||^2 + \frac{1}{2} ||\hat{\boldsymbol{u}}||^2$$

Rearranging we get,

$$\frac{d}{dt}||\hat{\boldsymbol{u}}||^2 \leq \left(1+2C_f\right)||\hat{\boldsymbol{u}}||^2 + ||\boldsymbol{\mathcal{R}}||^2$$

Integrating the above inequality over $[t_0, \bar{T}]$ for any $t_0 \leq \bar{T} \leq T$, we obtain,

$$||\hat{\boldsymbol{u}}(\bar{T})||^2 \leq \left(\int_{t_0}^{\bar{T}} ||\boldsymbol{\mathcal{R}}||^2 dt\right) + \left(\left(1+2C_f\right)\int_{t_0}^{\bar{T}} ||\hat{\boldsymbol{u}}||^2 dt\right)$$

where $||\hat{\boldsymbol{u}}(t_0)||^2 = 0$ because of the CE and,

$$\left(\int_{t_0}^{\bar{T}} ||\boldsymbol{\mathcal{R}}||^2 dt\right) \leq \left(\int_{t_0}^{T} ||\boldsymbol{\mathcal{R}}||^2 dt\right)$$

since $||\boldsymbol{\mathcal{R}}||^2 \geq 0$ and $t_0 \leq \bar{T} \leq T$. Thus,

$$||\hat{\boldsymbol{u}}(\bar{T})||^2 \leq \left(\int_{t_0}^{T} ||\boldsymbol{\mathcal{R}}||^2 dt\right) + \left(\left(1+2C_f\right)\int_{t_0}^{\bar{T}} ||\hat{\boldsymbol{u}}||^2 dt\right)$$

Applying the integral form of the Grönwall's inequality to the above, we get

$$||\hat{\boldsymbol{u}}(\bar{T})||^2 \leq \left(\int_{t_0}^{T} ||\boldsymbol{\mathcal{R}}||^2 dt\right)\left(e^{(1+2C_f)(\bar{T}-t_0)}\right)$$

Integrating over $[t_0, T]$ in $d\bar{T}$,

$$\varepsilon_G^2 = \int_{t_0}^{T} ||\hat{\boldsymbol{u}}(\bar{T})||^2 d\bar{T} \leq \left(\int_{t_0}^{T} ||\boldsymbol{\mathcal{R}}||^2 dt\right) \left(\frac{e^{-(1+2C_f)t_0}}{1+2C_f}\left(e^{(1+2C_f)T} - e^{(1+2C_f)t_0}\right)\right)$$

That is,

$$\varepsilon_G^2 \leq \left(\frac{e^{-(1+2C_f)t_0}}{1+2C_f}\left(e^{(1+2C_f)T} - e^{(1+2C_f)t_0}\right)\right) \left(\sum_{i=1}^{N} w_i ||\boldsymbol{\mathcal{R}}(t_i)||^2 + C_{\text{quad}}(||\boldsymbol{\mathcal{R}}||^2_{C^{k-1}})N^{-\alpha}\right)$$

Thus,

$$\varepsilon_G^2 \leq \left(\frac{e^{-(1+2C_f)t_0}}{1+2C_f}\left(e^{(1+2C_f)T} - e^{(1+2C_f)t_0}\right)\right)\left(\varepsilon_T^2 + C_{\text{quad}}(||\boldsymbol{\mathcal{R}}||^2_{C^{k-1}})N^{-\alpha}\right)$$

Finally,

$$\varepsilon_G \leq \left(\frac{e^{-(1+2C_f)t_0}}{1+2C_f}\left(e^{(1+2C_f)T} - e^{(1+2C_f)t_0}\right)\right)^{\frac{1}{2}}\left(\varepsilon_T^2 + C_{\text{quad}}N^{-\alpha}\right)^{\frac{1}{2}}$$

where,

$$\left(\frac{e^{-(1+2C_f)t_0}}{1+2C_f}\left(e^{(1+2C_f)T} - e^{(1+2C_f)t_0}\right)\right)^{\frac{1}{2}} = C_{\text{DE}} \geq 0, \forall t_0 \leq T \in \mathbb{R}$$

This completes the proof. □

The estimate (13) bounds the generalization error in terms of training and quadrature errors. The training error is computed once the training is completed. As long as X-TFC is well trained, i.e., the training error is small, the bound (13) implies that the generalization error will be small for a large enough number of training points, and if we can control the constant C_{DE}.

2.2 The Case of Higher-Order Initial Value Problems

By following the steps of the previous derivation, we derive an estimate of the generalization error when using X-TFC for learning the solution of a general system of non-linear m^{th} order ODEs.

A general system of n non-linear m^{th} order ODEs is posed as follows,

$$\begin{cases} \boldsymbol{u}_t^{(m)} = \boldsymbol{f}(t, \boldsymbol{u}_t^{(0)}, \boldsymbol{u}_t^{(1)}, \ldots, \boldsymbol{u}_t^{(m-1)}) \in \mathbb{R}^n & \forall t \in [t_0, T], T \geq t_0 \\ \boldsymbol{u}_t^{(0)}(t_0) = \boldsymbol{u}_0 \in \mathbb{R}^n \\ \boldsymbol{u}_t^{(1)}(t_0) = \boldsymbol{u}_0^{(1)} \in \mathbb{R}^n \\ \vdots \\ \boldsymbol{u}_t^{(m-1)}(t_0) = \boldsymbol{u}_0^{(m-1)} \in \mathbb{R}^n \end{cases} \tag{17}$$

where m is the derivative order, $\boldsymbol{u}_t^{(l)} \in \mathbb{R}^n$ with $l = 0, \ldots, m-1$ are the initial conditions (the subscript t means that the derivative is with respect to t), $\boldsymbol{u}_t^{(0)} = \boldsymbol{u} \in \mathbb{R}^n$ defined in $[t_0, T] \subset \mathbb{R}$ is the solution of (17), and $\boldsymbol{f}$ is the non-linear forcing term.

We assume that $\boldsymbol{f}$ is globally Lipschitz i.e., there exists a constant $C_f \geq 0$ (independent of t) such that the condition (3) holds.

By setting the following,

$$\begin{cases} \boldsymbol{x}_1 = \boldsymbol{u}_t^{(0)}, \in \mathbb{R}^n \\ \boldsymbol{x}_2 = \boldsymbol{u}_t^{(1)}, \in \mathbb{R}^n \\ \vdots \\ \boldsymbol{x}_m = \boldsymbol{u}_t^{(m-1)}, \in \mathbb{R}^n \end{cases} \tag{18}$$

we get the following system,

$$\begin{cases} \boldsymbol{x}_{1,t}^{(1)} = \boldsymbol{u}_t^{(1)} = \boldsymbol{x}_2 \in \mathbb{R}^n \\ \boldsymbol{x}_{2,t}^{(1)} = \boldsymbol{x}_3 \in \mathbb{R}^n \\ \vdots \\ \boldsymbol{x}_{m-1,t}^{(1)} = \boldsymbol{x}_m \in \mathbb{R}^n \\ \boldsymbol{x}_{m,t}^{(1)} = \boldsymbol{f}(t, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{m-1}) \in \mathbb{R}^n \end{cases} \quad \text{s.t.} \quad \begin{cases} \boldsymbol{x}_1(t_0) = \boldsymbol{x}_{1,0} = \boldsymbol{u}_0 \in \mathbb{R}^n \\ \boldsymbol{x}_2(t_0) = \boldsymbol{x}_{2,0} = \boldsymbol{u}_0^{(1)} \in \mathbb{R}^n \\ \vdots \\ \boldsymbol{x}_{m-1}(t_0) = \boldsymbol{x}_{m-1,0} = \boldsymbol{u}_0^{(m-2)} \in \mathbb{R}^n \\ \boldsymbol{x}_m(t_0) = \boldsymbol{x}_{m,0} = \boldsymbol{u}_0^{(m-1)} \in \mathbb{R}^n \end{cases}$$

which, in compact form it becomes,

$$\begin{cases} \dfrac{d\boldsymbol{X}}{dt} = \boldsymbol{F}(t, \boldsymbol{X}), \in \mathbb{R}^{(n\cdot m)\times 1} & \forall t \in [t_0, T], T \geq t_0 \\ \boldsymbol{X}(t_0) = \boldsymbol{X}_0 \in \mathbb{R}^{(n\cdot m)\times 1} \end{cases} \tag{19}$$

where $\boldsymbol{X} = [\boldsymbol{x}_1^{\mathrm{T}}, \ldots, \boldsymbol{x}_m^{\mathrm{T}}]^{\mathrm{T}}, \in \mathbb{R}^{(n\cdot m)\times 1}$, which is equivalent to system (2). Notice that in the new notation, the Lipschitz condition (3) holds for $\boldsymbol{F}$.

Now, in order to apply the X-TFC algorithm to learn the solution of (19), we need to specify the training set $\mathcal{S}$ and define the residuals.

Training set The same that was stated for the system of first-order ODEs holds.

Residuals In order to apply the X-TFC algorithm, we need to define the residuals. We approximate the solution of (19) with the following CEs,

$$x_{1,i}(t;\boldsymbol{\beta}_i) = u_i(t;\boldsymbol{\beta}_i) = g(t;\boldsymbol{\beta}_i) + \sum_{j=1}^{m} s_j(t)\eta_{j,i} \tag{20}$$

where $g(t;\boldsymbol{\beta}_i)$ is a single layer NN with random features, $\boldsymbol{s}(t) \in \mathbb{R}^{1\times m}$ is a vector of supporting functions, which can be defined by the user, see [42]. Finally, $\boldsymbol{\eta} \in \mathbb{R}^{m\times n}$ is a matrix of constant coefficients that are found by imposing the initial conditions. In the following we set

$$s_j(t) = \frac{(t-t_0)^{j-1}}{(j-1)!} \tag{21}$$

from which we get

$$\eta_{j,i} = u_{i,0,t}^{(j-1)} - g_t^{(j-1)}(t_0;\boldsymbol{\beta}_i) \tag{22}$$

We have $\boldsymbol{\beta} = [\boldsymbol{\beta}_1,\ldots,\boldsymbol{\beta}_n]$, with $\boldsymbol{\beta}_i \in \mathbb{R}^L$, $\forall i = 1,\ldots,n$, where L is the number of neurons in the hidden layer.

The k^{th} derivatives with respect to t are,

$$\begin{aligned} &x_{k-1,i,t}^{(1)}(t;\boldsymbol{\beta}_i) \in \mathbb{R},\ k = 2,\ldots m \\ &x_{k-1,i,t}^{(1)}(t;\boldsymbol{\beta}_i) = x_{k,i}(t;\boldsymbol{\beta}_i) = u_{i,t}^{(k-1)}(t;\boldsymbol{\beta}_i) = g_t^{(k-1)}(t;\boldsymbol{\beta}_i) + \sum_{j=1}^{m} s_{j,t}^{(k-1)}(t)\,\eta_{j,i} \end{aligned} \tag{23}$$

and,

$$x_{m,i}^{(1)}(t;\boldsymbol{\beta}_i) = u_{i,t}^{(m)}(t;\boldsymbol{\beta}_i) = f_i\left(t, \boldsymbol{x}_1(t;\boldsymbol{\beta}_i), \boldsymbol{x}_2(t;\boldsymbol{\beta}_i), \ldots, \boldsymbol{x}_{m-1}(t;\boldsymbol{\beta}_i)\right) \tag{24}$$

Then, we define the following residuals,

$$\boldsymbol{\mathcal{R}}(t;\boldsymbol{\beta}) = \boldsymbol{\mathcal{R}}_{\boldsymbol{\beta}}(t) = [\boldsymbol{\mathcal{R}}_{1,\boldsymbol{\beta}}^{\mathrm{T}} \ldots, \boldsymbol{\mathcal{R}}_{m,\boldsymbol{\beta}}^{\mathrm{T}}]^{\mathrm{T}} = \frac{d}{dt}\boldsymbol{X}_{\boldsymbol{\beta}}(t) - \boldsymbol{F}(\boldsymbol{X}_{\boldsymbol{\beta}}(t), t) \in \mathbb{R}^{(n\cdot m)\times 1} \tag{25}$$

Loss function To learn the unknown solution, we need to minimize the following vectorial loss function,

$$\boldsymbol{\mathcal{L}}(\boldsymbol{\beta}) = \left\{ \begin{array}{c} \boldsymbol{\mathcal{R}}_1(t_1;\boldsymbol{\beta}) = \boldsymbol{x}_{1,t}^{(1)}(t_1) - \boldsymbol{x}_2(t_1) \\ \vdots \\ \boldsymbol{\mathcal{R}}_1(t_N;\boldsymbol{\beta}) = \boldsymbol{x}_{1,t}^{(1)}(t_N) - \boldsymbol{x}_2(t_N) \\ \vdots \\ \\ \vdots \\ \boldsymbol{\mathcal{R}}_m(t_1;\boldsymbol{\beta}) = \boldsymbol{x}_{m,t}^{(1)}(t_1) - \boldsymbol{f}(t_1, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{m-1}) \\ \vdots \\ \boldsymbol{\mathcal{R}}_m(t_N;\boldsymbol{\beta}) = \boldsymbol{x}_{m,t}^{(1)}(t_N) - \boldsymbol{f}(t_N, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{m-1}) \end{array} \right\} \in \mathbb{R}^{((n\cdot m)\cdot N)\times 1} \tag{26}$$

where $\boldsymbol{\mathcal{R}}_k = \boldsymbol{0} \in \mathbb{R}^n$, for $k = 1,\ldots,m-1$ because of the CE definition.

The X-TFC algorithm is now completely specified, and it can be run for learning the solution of the non-linear first-order ODE system of Eq. (19), which we denote as $\boldsymbol{X}^* = \boldsymbol{X}_{\boldsymbol{\beta}^*}$, where $\boldsymbol{\beta}^*$ are the computed optimal output weights.

We are interested in estimating the generalization error for X-TFC, which is defined as

$$\varepsilon_G = \left(\int_{t_0}^{T} ||\boldsymbol{X}(t) - \boldsymbol{X}^*(t)||^2 dt \right)^{1/2} \in \mathbb{R} \tag{27}$$

ε_G, according to [41] and references within, is the error emerging from the approximation of the true solution, $\boldsymbol{u}(t)$, via X-TFC, within in the domain $t_0 \leq t \leq T$. We estimate the generalization error in terms of the *training error* $\varepsilon_T \in \mathbb{R}$ defined as,

$$\varepsilon_T^2 = \sum_{i=1}^{N} w_i ||\boldsymbol{\mathcal{R}}_{\boldsymbol{\beta}^*}(t_i)||^2 = \tag{28}$$

$$= \sum_{i=1}^{N} w_i \left[\left(\underbrace{\sum_{k=2}^{m} \left(\boldsymbol{x}_{k-1}^{*(1)}(t_i) - \boldsymbol{x}_k^*(t_i) \right)^2}_{0 \text{ due to } (23)} \right) + \left(\underbrace{\boldsymbol{x}_m^{*(1)}(t_i)}_{\boldsymbol{u}_t^{*(m)}(t_i)} - \boldsymbol{f}(t_i, \boldsymbol{x}_1^*, \boldsymbol{x}_2^*, \ldots, \boldsymbol{x}_{m-1}^*) \right)^2 \right]$$

which is computed a posteriori once the training is completed.

The considerations of the quadrature error are the same made for the system (2). The estimate of the generalization error for X-TFC is given in the following theorem. It is noticed that theorem 2 can also be considered as a corollary for theorem 1.

Theorem 2 *Let $\boldsymbol{X} \in C^k([t_0, T])$ be the unique classical solution of the non-linear problem (19) with the dynamics $\boldsymbol{F}$ satisfying Eq. (3). Let $\boldsymbol{X}^* = \boldsymbol{X}_{\boldsymbol{\beta}^*}$ be the solution approximated via X-TFC, corresponding to loss function (26). Then, the generalization error (27) can be estimated as,*

$$\varepsilon_G \leq C_{DE} \left(\varepsilon_T^2 + C_{\text{quad}} N^{-\alpha} \right)^{\frac{1}{2}} \tag{29}$$

where ε_T is given by Eq. (28), the constant C_{DE} is given by Eq. (14), and

$$C_{\text{quad}} = C_{\text{quad}}(||\boldsymbol{\mathcal{R}}||_{C^{k-1}}^2)$$

which is a positive constant depending on the number of training points, the quadrature scheme used, and the residuals evaluated on the training points.

Proof The proof of theorem 2 is analogous to the proof of theorem 1. Hence, it is not reported. □

2.3 The Case of Second-Order Order Two-Point Boundary Value Problems

By the same steps, we derive an estimate of the generalization error when using X-TFC for learning the solution of a general system of non-linear second-order BVP.

A general system of n non-linear second-order BVPs is posed as follows,

$$\begin{cases} \dfrac{d^2\boldsymbol{u}}{dt^2} = \boldsymbol{f}\left(t, \boldsymbol{u}, \dfrac{d\boldsymbol{u}}{dt}\right) & \forall t \in [t_0, T], T \geq t_0 \\ \boldsymbol{u}(t_0) = \boldsymbol{u}_0 \\ \boldsymbol{u}(T) = \boldsymbol{u}_T \end{cases} \tag{30}$$

Here, $\boldsymbol{u}_0 \in \mathbb{R}^n$ is the initial conditions and $\boldsymbol{u}_T \in \mathbb{R}^n$ is the final condition, $\boldsymbol{u} \in \mathbb{R}^n$ defined in $\in [t_0, T] \subset \mathbb{R}$ is the solution of (30), and $\boldsymbol{f}$ is the non-linear forcing term.

We assume that $\boldsymbol{f}$ is globally Lipschitz, i.e., there exists a constant $C_f \geq 0$ (independent of t) such that the condition (3) holds.

By set the following,

$$\begin{cases} \boldsymbol{x}_1 = \boldsymbol{u}, \in \mathbb{R}^n \\ \boldsymbol{x}_2 = \dfrac{d\boldsymbol{u}}{dt} \in \mathbb{R}^n \end{cases} \tag{31}$$

we get the following system,

$$\begin{cases} \dfrac{d\boldsymbol{x_1}}{dt} = \dfrac{d\boldsymbol{u}}{dt} = \boldsymbol{x_2} \\ \dfrac{d\boldsymbol{x_2}}{dt} = \dfrac{d^2\boldsymbol{u}}{dt^2} = \boldsymbol{f}(t, \boldsymbol{x_1}, \boldsymbol{x_2}), \end{cases} \quad \text{s.t.} \quad \begin{cases} \boldsymbol{x_1}(t_0) = \boldsymbol{x_{10}} = \boldsymbol{u}_0 \\ \boldsymbol{x_1}(T) = \boldsymbol{x_{1T}} = \boldsymbol{u}_T \end{cases} \tag{32}$$

which in compact form becomes,

$$\begin{cases} \dfrac{d\boldsymbol{X}}{dt} = \boldsymbol{F}(t, \boldsymbol{X}), \in \mathbb{R}^{(n\cdot 2)\times 1} & \forall t \in [t_0, T], T \geq t_0 \\ \boldsymbol{x_1}(t_0) = \boldsymbol{x_{10}} \in \mathbb{R}^{n\times 1} \\ \boldsymbol{x_1}(T) = \boldsymbol{x_{1T}} \in \mathbb{R}^{n\times 1} \end{cases} \tag{33}$$

where $\boldsymbol{X} = [\boldsymbol{x}_1^{\mathrm{T}}, \boldsymbol{x}_2^{\mathrm{T}}]^{\mathrm{T}}, \in \mathbb{R}^{(n\cdot 2)\times 1}$, and $\boldsymbol{F}$ satisfies (3).

Now, in order to apply the X-TFC algorithm to learn the solution of (33), we need to specify the training set $\mathcal{S}$ and define the residuals.

Training set The same that was stated for the system of first-order ODEs holds.

Residuals In order to apply the X-TFC algorithm, we need to define the residuals. We approximate the solution of (30) with the following constrained expressions,

$$x_{1,i}(t; \boldsymbol{\beta}_i) = u_i(t; \beta_i) = g(t; \beta_i) + \sum_{k=1}^{2} s_k(t)\, \eta_{k,i} \in \mathbb{R} \tag{34}$$

where $g(t; \boldsymbol{\beta}_i)$ is a single layer NN with random features, $\boldsymbol{s}(t) \in \mathbb{R}^{1\times m}$ is a vector of supporting functions that can be defined by the user, see [42], and $\boldsymbol{\eta} \in \mathbb{R}^{m\times n}$ is a matrix of constant coefficients which are found by imposing the initial conditions. In the following, we set

$$s_1(t) = \frac{T - t}{T - t_0} \qquad s_2(t) = \frac{t - t_0}{T - t_0} \tag{35}$$

from which we get

$$\eta_{1,i} = u_{0,i} - g(t_0; \boldsymbol{\beta}_i) \qquad \eta_{2,i} = u_{T,i} - g(T; \boldsymbol{\beta}_i) \tag{36}$$

We have $\boldsymbol{\beta} = [\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_n]$, with $\boldsymbol{\beta}_i \in \mathbb{R}^L$, $\forall i = 1, \ldots, n$, where L is the number of neurons in the hidden layer.

Their derivatives with respect to t are,

$$x_{2,i}(t; \boldsymbol{\beta}_i) = \frac{d}{dt} x_{1,i}(t; \boldsymbol{\beta}_i) = \frac{d}{dt} u_i(t; \boldsymbol{\beta}_i) = \frac{d}{dt} g(t; \boldsymbol{\beta}_i) + \sum_{k=1}^{2} \frac{d}{dt} s_k(t)\, \eta_{k,i} \in \mathbb{R} \tag{37}$$

and

$$\frac{d}{dt} x_{2,i}(t; \boldsymbol{\beta}_i) = f_i(t, \boldsymbol{x}_1, \boldsymbol{x}_2) \tag{38}$$

Then, we define the following residuals,

$$\boldsymbol{\mathcal{R}}(t;\boldsymbol{\beta}) = \boldsymbol{\mathcal{R}}_{\boldsymbol{\beta}}(t) = [\boldsymbol{\mathcal{R}}_{1,\boldsymbol{\beta}}^{\mathrm{T}}, \boldsymbol{\mathcal{R}}_{2,\boldsymbol{\beta}}^{\mathrm{T}}]^{\mathrm{T}} = \frac{d}{dt}\boldsymbol{X}_{\boldsymbol{\beta}}(t) - \boldsymbol{F}(\boldsymbol{X}_{\boldsymbol{\beta}}(t), t) \in \mathbb{R}^{(n\cdot 2)\times 1} \tag{39}$$

Loss function To learn the unknown solution, we need to minimize the following vectorial loss function,

$$\boldsymbol{\mathcal{L}}(\boldsymbol{\beta}) = \left\{ \begin{array}{c} \boldsymbol{\mathcal{R}}_1(t_1;\boldsymbol{\beta}) = \dfrac{d}{dt}\boldsymbol{x_1}(t_1) - \boldsymbol{x}_2(t_1) \\ \vdots \\ \boldsymbol{\mathcal{R}}_1(t_N;\boldsymbol{\beta}) = \dfrac{d}{dt}\boldsymbol{x_1}(t_N) - \boldsymbol{x}_2(t_N) \\ \boldsymbol{\mathcal{R}}_2(t_1;\boldsymbol{\beta}) = \dfrac{d}{dt}\boldsymbol{x_2}(t_1) - \boldsymbol{f}(t_1, \boldsymbol{x}_1, \boldsymbol{x}_2) \\ \vdots \\ \boldsymbol{\mathcal{R}}_2(t_N;\boldsymbol{\beta}) = \dfrac{d}{dt}\boldsymbol{x_2}(t_N) - \boldsymbol{f}(t_N, \boldsymbol{x}_1, \boldsymbol{x}_2) \end{array} \right\} \in \mathbb{R}^{((n\cdot 2)\cdot N)\times 1} \tag{40}$$

where $\boldsymbol{\mathcal{R}}_1 = \boldsymbol{0}$ because of the CE definition.

The X-TFC algorithm is now completely specified, and it can be run for learning the solution of the non-linear first-order ODE system of Eq. (33), which we denote as $\boldsymbol{X}^* = \boldsymbol{X}_{\boldsymbol{\beta}^*}$, where $\boldsymbol{\beta}^*$ are the computed optimal output weights.

We are interested in estimating the generalization error for X-TFC, which is defined as

$$\varepsilon_G = \left(\int_{t_0}^{T} ||\boldsymbol{X}(t) - \boldsymbol{X}^*(t)||^2 dt \right)^{1/2} \in \mathbb{R} \tag{41}$$

where ε_G, according to [41] and references within, is the error emerging from the approximation of the true solution, $\boldsymbol{X}(t)$, via X-TFC, within in the domain $t_0 \leq t \leq T$. We estimate the generalization error in terms of the *training error* $\varepsilon_T \in \mathbb{R}$ defined as,

$$\varepsilon_T^2 = \sum_{i=1}^{N} w_i ||\boldsymbol{\mathcal{R}}_{\boldsymbol{\beta}^*}(t_i)||^2 = \tag{42}$$

$$= \sum_{i=1}^{N} w_i \left[\underbrace{\left(\frac{d}{dt}\boldsymbol{x}_1^*(t_i) - \boldsymbol{x}_2^*(t_i) \right)^2}_{0 \text{ due to } (37)} + \left(\underbrace{\frac{d}{dt}\boldsymbol{x}_2^*(t_i)}_{\frac{d^2}{dt^2}\boldsymbol{u}^*(t_i)} - \boldsymbol{f}(t_i, \boldsymbol{x}_1^*, \boldsymbol{x}_2^*) \right)^2 \right] \in \mathbb{R}$$

which is computed a posteriori once the training is completed.

The considerations of the quadrature error are the same made for the system (30). The estimate of the generalization error for X-TFC is given in the following theorem.

Theorem 3 *Let $\boldsymbol{X} \in C^k([t_0, T])$ be the unique classical solution of the non-linear problem (33) with the dynamics $\boldsymbol{F}$ satisfying Eq. (3). Let $\boldsymbol{X}^* = \boldsymbol{X}_{\boldsymbol{\beta}^*}$ be the solution approximated via X-TFC, corresponding to loss function (40). Then, the generalization error (41) can be estimated as,*

$$\varepsilon_G \leq C_{DE} \left(\varepsilon_T^2 + C_{\text{quad}} N^{-\alpha} + ||\hat{\boldsymbol{x}}_2(t_0)||^2 \right)^{\frac{1}{2}} \tag{43}$$

where ε_T is given by Eq. (42), the constant C_{DE} is given by Eq. (14), and

$$\hat{\boldsymbol{x}}_2(t_0) = \boldsymbol{x}_2^*(t_0) - \boldsymbol{x}_2(t_0)$$

which is a function of the residuals $\boldsymbol{\mathcal{R}}$,

$$\hat{\boldsymbol{x}}_2(t_0) = \hat{\boldsymbol{x}}_2(t_0)(||\boldsymbol{\mathcal{R}}||^2_{C^{k-1}})$$

and is estimated as typical in shooting methods. Details can be found in [24]. Also

$$C_{\text{quad}} = C_{\text{quad}}(||\boldsymbol{\mathcal{R}}||^2_{C^{k-1}})$$

which is a positive constant depending on the number of training points, the quadrature scheme used, and the residuals evaluated on the training points.

Proof By the definitions of the residuals (39), and the system (33), we can verify that the error $\hat{\boldsymbol{X}} = \boldsymbol{X}^* - \boldsymbol{X}$ satisfies the following equation

$$\begin{aligned} &\frac{d\hat{\boldsymbol{X}}}{dt} = \boldsymbol{F}(\boldsymbol{X}^*, t) - \boldsymbol{F}(\boldsymbol{X}, t) + \boldsymbol{\mathcal{R}}, \qquad \forall t \in [t_0, T] \\ &\hat{\boldsymbol{x}}_1(t_0) = \boldsymbol{0} \\ &\hat{\boldsymbol{x}}_1(T) = \boldsymbol{0} \end{aligned} \tag{44}$$

Here, we have denoted $\boldsymbol{\mathcal{R}} = \boldsymbol{\mathcal{R}}_{\boldsymbol{\beta}^*}$ for convenience of notation. Multiplying both sides of the ODE (44) by $\hat{\boldsymbol{X}}$ yields

$$\hat{\boldsymbol{X}} \cdot \frac{d\hat{\boldsymbol{X}}}{dt} = \hat{\boldsymbol{X}} \cdot \left[\boldsymbol{F}(\boldsymbol{X}^*, t) - \boldsymbol{F}(\boldsymbol{X}, t)\right] + \hat{\boldsymbol{X}} \cdot \boldsymbol{\mathcal{R}}$$

where $\cdot$ is the inner product. That is,

$$\frac{1}{2}\frac{d}{dt}||\hat{\boldsymbol{X}}||^2 = \hat{\boldsymbol{X}} \cdot \left[\boldsymbol{F}(\boldsymbol{X}^*, t) - \boldsymbol{F}(\boldsymbol{X}, t)\right] + \hat{\boldsymbol{X}} \cdot \boldsymbol{\mathcal{R}} \tag{45}$$

By leveraging on the following inequality $\left(\hat{\boldsymbol{X}} - \boldsymbol{\mathcal{R}}\right)^2 \geq 0$, Eq. (45) is bounded by,

$$\frac{1}{2}\frac{d}{dt}||\hat{\boldsymbol{X}}||^2 \leq ||\hat{\boldsymbol{X}}||\ \left|\left|\boldsymbol{F}(\boldsymbol{X}^*, t) - \boldsymbol{F}(\boldsymbol{X}, t)\right|\right| + \frac{1}{2}||\boldsymbol{\mathcal{R}}||^2 + \frac{1}{2}||\hat{\boldsymbol{X}}||^2$$

By (3), we have the following,

$$\frac{1}{2}\frac{d}{dt}||\hat{\boldsymbol{X}}||^2 \leq ||\hat{\boldsymbol{X}}|| C_f \underbrace{||\boldsymbol{X}^* - \boldsymbol{X}||}_{||\hat{\boldsymbol{X}}||} + \frac{1}{2}||\boldsymbol{\mathcal{R}}||^2 + \frac{1}{2}||\hat{\boldsymbol{X}}||^2$$

Rearranging, we get,

$$\frac{d}{dt}||\hat{\boldsymbol{X}}||^2 \leq \left(1 + 2C_f\right)||\hat{\boldsymbol{X}}||^2 + ||\boldsymbol{\mathcal{R}}||^2$$

Integrating the above inequality over $[t_0, \bar{T}]$ for any $t_0 \leq \bar{T} \leq T$, we obtain,

$$||\hat{\boldsymbol{X}}(\bar{T})||^2 - ||\hat{\boldsymbol{X}}(t_0)||^2 \leq \left(\int_{t_0}^{\bar{T}} ||\boldsymbol{\mathcal{R}}||^2 dt\right) + \left(\left(1 + 2C_f\right)\int_{t_0}^{\bar{T}} ||\hat{\boldsymbol{X}}||^2 dt\right)$$

where,

$$||\hat{\boldsymbol{X}}(t_0)||^2 = ||\hat{\boldsymbol{x}}_1(t_0)||^2 + ||\hat{\boldsymbol{x}}_2(t_0)||^2$$

with $||\hat{\boldsymbol{x}}_1(t_0)||^2 = 0$ because of the CE.

Also,

$$\left(\int_{t_0}^{\bar{T}} ||\boldsymbol{\mathcal{R}}||^2 dt\right) \leq \left(\int_{t_0}^{T} ||\boldsymbol{\mathcal{R}}||^2 dt\right)$$

since $||\boldsymbol{\mathcal{R}}||^2 \geq 0$ and $t_0 \leq \bar{T} \leq T$. Thus,

$$||\hat{\boldsymbol{X}}(\bar{T})||^2 \leq \left(\int_{t_0}^{T} ||\boldsymbol{\mathcal{R}}||^2 dt + ||\hat{\boldsymbol{x}}_2(t_0)||^2\right) + \left(\left(1 + 2C_f\right) \int_{t_0}^{\bar{T}} ||\hat{\boldsymbol{X}}||^2 dt\right)$$

Applying the integral form of the Grönwall's inequality to the above, we get

$$||\hat{\boldsymbol{X}}(\bar{T})||^2 \leq \left(\int_{t_0}^{T} ||\boldsymbol{\mathcal{R}}||^2 dt + ||\hat{\boldsymbol{x}}_2(t_0)||^2\right) \left(e^{(1+2C_f)(\bar{T}-t_0)}\right)$$

Integrating over $[t_0, T]$ in $d\bar{T}$,

$$\varepsilon_G^2 = \int_{t_0}^{T} ||\hat{\boldsymbol{X}}(\bar{T})||^2 d\bar{T} \leq \left(\int_{t_0}^{T} ||\boldsymbol{\mathcal{R}}||^2 dt + ||\hat{\boldsymbol{x}}_2(t_0)||^2\right) \left(\frac{e^{-(1+2C_f)t_0}}{1+2C_f}\left(e^{(1+2C_f)T} - e^{(1+2C_f)t_0}\right)\right)$$

That is,

$$\varepsilon_G^2 \leq \left(\frac{e^{-(1+2C_f)t_0}}{1+2C_f}\left(e^{(1+2C_f)T} - e^{(1+2C_f)t_0}\right)\right) \left(\sum_{i=1}^{N} w_i ||\boldsymbol{\mathcal{R}}(t_i)||^2 + C_{\text{quad}}(||\boldsymbol{\mathcal{R}}||^2_{C^{k-1}}) N^{-\alpha} + ||\hat{\boldsymbol{x}}_2(t_0)||^2\right)$$

Thus,

$$\varepsilon_G^2 \leq \left(\frac{e^{-(1+2C_f)t_0}}{1+2C_f}\left(e^{(1+2C_f)T} - e^{(1+2C_f)t_0}\right)\right) \left(\varepsilon_T^2 + C_{\text{quad}}(||\boldsymbol{\mathcal{R}}||^2_{C^{k-1}}) N^{-\alpha} + ||\hat{\boldsymbol{x}}_2(t_0)||^2\right)$$

Finally,

$$\varepsilon_G \leq \left(\frac{e^{-(1+2C_f)t_0}}{1+2C_f}\left(e^{(1+2C_f)T} - e^{(1+2C_f)t_0}\right)\right)^{\frac{1}{2}} \left(\varepsilon_T^2 + C_{\text{quad}} N^{-\alpha} + ||\hat{\boldsymbol{x}}_2(t_0)||^2\right)^{\frac{1}{2}}$$

where,

$$\left(\frac{e^{-(1+2C_f)t_0}}{1+2C_f}\left(e^{(1+2C_f)T} - e^{(1+2C_f)t_0}\right)\right)^{\frac{1}{2}} = C_{\text{DE}} \geq 0, \forall t_0 \leq T \ \in \mathbb{R}$$

This completes the proof. □

3 Numerical Tests

Various problems have been solved using X-TFC: three first-order IVPs, two second-order IVPs (as representative for high-order problems), two second-order two-point boundary value problems, and end with systems of ODEs that are equivalent to these last. For each class, we have selected at least one linear and one nonlinear problem. For all the numerical examples presented in this paper, the support functions used within the X-TFC correspond to the monomial set, that is, $s_k(t) = t^{k-1}$. The selected problems are benchmark problems with known solutions. Having the analytical solutions allows us to compute the generalization error and therefore to numerically test the theoretical findings. All the considered DEs are coded in MATLAB 2022a and run with an Intel Core i7 - 9700 CPU PC with 64 GB of RAM. Moreover, our results were compared with those obtained using MATLAB solvers: $ode45.m$ (explicit nested Runge-Kutta) for the linear IVPs, $ode23t.m$ (implicit trapezoidal) for the nonlinear IVPs and $bvp4c.m$ for the second-order order two-point boundary value problem. MATLAB solvers are used to recover the reference numerical solution and to compare computational costs and execution time. When using the Matlab routines, we pay attention to the structure of the output to evaluate how many points the code uses effectively to produce the result. This needs attention because the routine extrapolates the result when giving the output so that the number of points given in the output is not the one used for the computations. As an example, when using $ode45$, we used the routine in two settings: with the usual tolerance requirements (*default*) and with some very strict ones (*with options*). In the case of non-linear IVPs, we chose to compare the behavior with the trapezoidal method $ode23t$ because for non-linear problems both methods have to solve a non-linear iteration. Finally, for the two-point boundary problems, we found that the comparison with the routine $bvp4c$ can be performed profitably, both being collocation methods. In particular, $bvp4c$ is a finite difference code that implements the three-stage Lobatto IIIa formula. Also for these two routines, the comparison is made with *default* requirements and with strict ones (*with options*). Final comparisons of efficiency and accuracy are dome with the strict requirements, these giving accuracy similar to the one obtained by X-TFC. As can be easily understood, the Matlab reference routines are highly developed to achieve efficiency, while our implementation is far from optimized. However, the comparison gives excellent performance for our code, and only in the stiff problem, Matlab's code overperforms the proposed method.

For each problem, we show the numerical results in four figures and one table. For all figures, $errG_{train}$ corresponds to the generalization error for the training points, $errG_{test}$ corresponds to the generalization error for the test points, $errT$ corresponds to the training error, $bound$ corresponds to the generalization error bound, as computed from the theory, and $elapsed\ time$ is the time required to compute the solutions. Moreover, the plots are organized as follows:

1) First plot with two panels. On the left, analytical solution, MATLAB solver solutions (with two settings), and X-TFC solution (with 100 neurons trained on 100 random points). On the right, the absolute error on training points for the three numerical solvers: X-TFC and MATLAB solver.
2) Second plot with convergence varying the number of neurons (L) for a fixed number of training and test points ($N = 100$, and $N_{\text{test}} = 20$) randomly selected. We plot the generalization error on training points, training error, generalization error bound, generalization error on test points, and elapsed time.
3) Third plot with convergence varying the number of training points (N) for a fixed number of neurons ($L = 100$) and test points ($N_{\text{test}} = 20$); training and test points are randomly

selected. As in the previous case, we plot the generalization error in training points, the training error, the generalization error bound, the generalization error in testing points and training time.

4) Convergence compared to MATLAB solver results, where we report the generalization error at training points and the generalization error at test points (both for the MATLAB solver and X- TFC), training error, generalization error bound, and training time, varying the number of neurons for fixed number test points and training points. When these are available and not too much, we have used as training points for X-TFC the collocation points used by the MATLAB solvers in the strict setting. The test points are selected randomly.

In particular, $errG_{train}$ and $errG_{test}$ are calculated as the root mean square error (RMSE) between the X-TFC solution and the exact solution at the training points and the test points, respectively, and $errT$ is calculated as the RMSE between the residual of the differential equation and zero.

In order to exploit the generalization error bounds shown in (13),(29), and (43), we note that a quadrature of the residual is a good approach only if there are enough training points and it does not oscillate between. In order to obtain a good estimate, in the numerical test presented we compute this bound as

$$bound = C_1 \int_{t_0}^{t_f} f_{bound}^2(t)dt \tag{46}$$

where f_{bound} is a conservative estimate of $\|\mathcal{R}\|$. Here we consider a function such that

$$f_{bound}(t) - \|\mathcal{R}(t^*)\| = L(t - t^*) \tag{47}$$

t^* is the closest training point to t and L is an estimate of the Lipschitz constant of $\mathcal{R}$:

$$L = \max_{t^* \in \mathcal{S}} \| \frac{d}{dt} \mathcal{R}(t^*) \| \tag{48}$$

In this way both small number of training points and oscillating residuals are penalized, resulting in a higher bound.

For all the DEs we used the following hyperparameters for X-TFC: hyperbolic tangent as activation function, input and bias are sampled from $\mathcal{U}(-3; 3)$. The choice for these hyperparameters is completely arbitrary as in Ref. [50] the authors perform a sensitivity analysis showing that X-TFC, at least for this class of problems, is not sensitive to the choice of these hyperparameters.

In the tables, we finally summarize the comparison between Matlab's reference solution and X-TFC in similar settings. All numerical results confirm the theoretical findings.

Before reporting our numerical test, we report in the next subsection 3.1 the structure of the numerical algorithm used later.

3.1 Algorithm Description

In this section, our aim is to present the discrete analogous of the general framework presented before, so to highlight the simple resolution setting, see Section 3.1.1. We explore more specifically the linear case in Section 3.1.2. Being the method implicit, in the nonlinear case we need a solver, and this is detailed in Section 3.1.3, where we also present a pseudo-code. In Figure 1 we give a graphical abstract of the proposed method in the case of the resolution of one of the test problems, the one that we detail in Section 3.4.2.

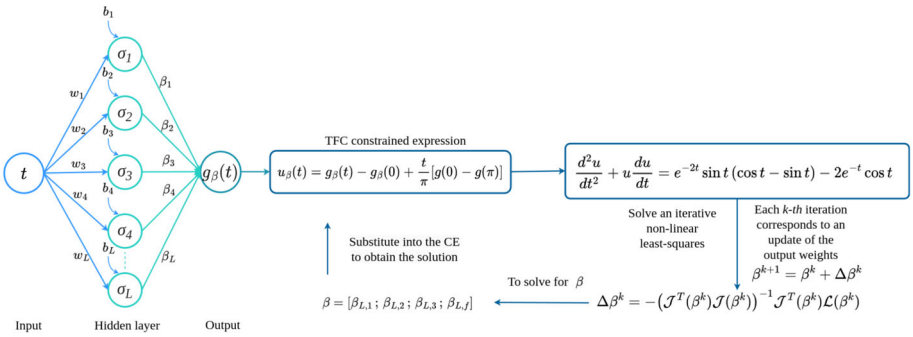

Fig. 1 Diagram for using ELM in a Physics-Informed Machine Learning framework

3.1.1 General Set-up

In the ELM framework we introduce the matrix $\boldsymbol{H} \in \mathbb{R}^{N \times L}$ such that

$$H_{lj} = \sigma(w_j t_l + b_j) \qquad l = 1, \ldots, N \quad j = 1, \ldots, L \tag{49}$$

where each row is the vector of outputs of the neurons in the hidden layer at time t_l, which is an element of the training set. The input weights w_j and the bias of the neurons in the hidden layer b_j are fixed, therefore the optimization is performed only on the output weights in $\boldsymbol{\beta}$. In the following we will consider the matrices $\boldsymbol{H}^{(k)} \in \mathbb{R}^{N \times L}$, whose components are the time derivatives of the components of $\boldsymbol{H}$:

$$H_{lj}^{(k)} = w_j^{k-1} \sigma_t^{(k-1)}(w_j t_l + b_j) \qquad k = 1, \ldots, m+1 \tag{50}$$

Supporting functions: $\boldsymbol{S} \in \mathbb{R}^{N \times m}$

$$S_{lk} = \begin{cases} \frac{(t_l - t_0)^{k-1}}{(k-1)!} & IVP \\ \begin{bmatrix} \frac{T-t_l}{T-t_0} & \frac{t_l - t_0}{T-t_0} \end{bmatrix} & BVP \end{cases} \qquad l = 1, \ldots, N \quad k = 1, \ldots, m \tag{51}$$

We refer to the first $m - k + 1$ columns of $\boldsymbol{S}$ as $\boldsymbol{S}^{(k)}$.

Network's output at the boundary: $\boldsymbol{E} \in \mathbb{R}^{m \times L}$

$$E_{kj} = w_j^{(k-1)} \sigma_t^{(k-1)}(w_j t_0 + b_j) \qquad k = 1, \ldots, m \quad j = 1, \ldots, L \tag{52}$$

We refer to the last $m - k + 1$ rows of $\boldsymbol{E}$ as $\boldsymbol{E}^{(k)}$. Conditions at the boundary: $\boldsymbol{F} \in \mathbb{R}^{m \times n}$

$$F_{ki} = \begin{cases} x_{k,i,0} & k = 1, \ldots, m \quad i = 1, \ldots, n \quad IVP \\ x_{1,i,k} & k \in \{0, T\} \quad i = 1, \ldots, n \quad BVP \end{cases} \tag{53}$$

We refer to the last $m - k + 1$ rows of $\boldsymbol{F}$ as $\boldsymbol{F}^{(k)}$.

We can now compute the function and its derivatives:

$$\boldsymbol{C}^{(k)} = \boldsymbol{H}^{(k)} - \boldsymbol{S}^{(k)} \boldsymbol{E}^{(k)} \qquad \in \mathbb{R}^{N \times L} \tag{54}$$

$$\boldsymbol{D}^{(k)} = \boldsymbol{S}^{(k)} \boldsymbol{F}^{(k)} \qquad \in \mathbb{R}^{N \times n} \tag{55}$$

$$\boldsymbol{X}^{(k)} = \boldsymbol{C}^{(k)} \boldsymbol{\beta} + \boldsymbol{D}^{(k)} \qquad \in \mathbb{R}^{N \times n} \tag{56}$$

And finally, the residual can be computed as

$$\boldsymbol{\mathcal{R}} = \boldsymbol{H}^{(k+1)}\boldsymbol{\beta} - \boldsymbol{F}(\boldsymbol{t}, \boldsymbol{X}^{(1)}, \ldots, \boldsymbol{X}^{(m)}) \quad \in \mathbb{R}^{(n\cdot N)\times 1} \tag{57}$$

to which we associate the following loss function:

$$\mathcal{L} = \frac{1}{2}\|\boldsymbol{\mathcal{R}}\|_2^2 \tag{58}$$

3.1.2 Linear Problem

If the differential problem is linear, every component of $\boldsymbol{F}$ is a linear combination of the components of $\boldsymbol{X}$ and its derivatives. Then there exist coefficients $a_{i,h,k}$ and d_i such that

$$F_i(t, \boldsymbol{X}) = \sum_{h=1}^{n}\sum_{k=1}^{m} a_{i,h,k}(t)x_{k,h}(t; \boldsymbol{\beta}_h) + d_i(t) \quad \in \mathbb{R} \tag{59}$$

It is important to note that the coefficients can also be time-dependent, and in the training phase (in which we consider a total of N points in time) we can associate to them the vectors with the N evaluations:

$$\boldsymbol{F}_i(\boldsymbol{t}, \boldsymbol{X}) = \sum_{h=1}^{n}\sum_{k=1}^{m} \boldsymbol{a}_{i,h,k} \odot \boldsymbol{x}_{k,h} + \boldsymbol{d}_i \quad \in \mathbb{R}^N \tag{60}$$

Where $\odot$ is the Hadamard product of two vectors.

Substituting (56) in (60) we get

$$\begin{aligned}\boldsymbol{F}_i(\boldsymbol{t}, \boldsymbol{X}_1, \ldots, \boldsymbol{X}_k) &= \sum_{h=1}^{n}\sum_{k=1}^{m} \boldsymbol{a}_{i,h,k} \odot \left(\boldsymbol{C}^{(k)}\boldsymbol{\beta}_h + \boldsymbol{D}_h^{(k)}\right) + \boldsymbol{d}_i \\ &= \sum_{h=1}^{n}\left(\sum_{k=1}^{m} \boldsymbol{a}_{i,h,k} * \boldsymbol{C}^{(k)}\right)\boldsymbol{\beta}_h + \sum_{h=1}^{n}\sum_{k=1}^{m} \boldsymbol{a}_{i,h,k} \odot \boldsymbol{D}_h^{(k)} + \boldsymbol{d}_i\end{aligned}$$

Where $*$ is a Khatri-Rao product and in this case means that each element of $\boldsymbol{a}_{i,h,k}$ multiplies the corresponding row of $\boldsymbol{C}^{(k)}$.

Combining this with (56) and (57) we get a set of $n \cdot N$ linear equations where the unknowns are the $n \cdot L$ elements of $\boldsymbol{\beta}$.

3.1.3 Nonlinear Problem

In the nonlinear case the solution can be approximated by means of an Iterative Least-Squares procedure.

This routines usually involve the numerical estimation of the jacobian $\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{\beta}}$, but if an analytical expression of the jacobian $\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}_k}$ (the derivatives of $\boldsymbol{F}$ with respect to the original function $\boldsymbol{u}$ and its time derivatives) is available, we can exploit it in order to compute $\frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{\beta}}$ in a more efficient way.

Starting from (57) and applying the chain rule and the total time derivative we get

$$J_{lijh} = \frac{\partial \mathcal{R}_{li}}{\partial \beta_{jh}} = H_{lj}^{(m+1)}\delta_{ih} - \sum_{k=1}^{m} \frac{\partial F_{il}}{\partial x_{k,h}} C_{lj}^{(k)} \tag{61}$$

which in matrix form can be written as

$$\boldsymbol{J} = \frac{\partial \boldsymbol{\mathcal{R}}}{\partial \boldsymbol{\beta}} = \boldsymbol{I}_n \otimes \boldsymbol{H}^{(k+1)} - \sum_{k=1}^{m} \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}_k} * \boldsymbol{C}^{(k)} \quad \in \mathbb{R}^{(n\cdot N)\times(n\cdot L)} \tag{62}$$

where in this case $\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}_k} * \boldsymbol{C}^{(k)}$ means that each element of $\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}_k}$ multiplies the corresponding row of $C^{(k)}$.

In this work we use a Gauss-Newton method and (62) in order to get a more efficient computation of the Jacobian matrix of the residuals vector. In order to make clear all the steps of the resolution, we provide a pseudo-code in Algorithm 1.

Algorithm 1: Solution of a nonlinear ODE of order m in n dimensions, the procedure is different for IVP's and BVP's only in the way lines 6 and 8 are executed.

1: Require $t_0, T, N, L, \boldsymbol{X}_0, \boldsymbol{X}_T, \boldsymbol{F}(t, \boldsymbol{X}), \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}_k}(t, \boldsymbol{X})$
2: set $M \in \mathbb{R}$
3: get $\boldsymbol{w} \in \mathbb{R}^L$: $w_j \in \mathcal{U}(-M, M) \quad j = 1, \ldots, L$
4: get $\boldsymbol{b} \in \mathbb{R}^L$: $b_j \in \mathcal{U}(-M, M) \quad j = 1, \ldots, L$
5: compute $\boldsymbol{H}^{(k)} \in \mathbb{R}^{N\times L} \quad k = 1, \ldots, m+1$ as in (50)
6: compute $\boldsymbol{S} \in \mathbb{R}^{N\times m}$ as in (51)
7: compute $\boldsymbol{E} \in \mathbb{R}^{m\times L}$ as in (52)
8: compute $\boldsymbol{F} \in \mathbb{R}^{m\times n}$ as in (53)
9: compute $\boldsymbol{C}^{(k)} \in \mathbb{R}^{N\times L} \quad k = 1, \ldots, m$ as in (54)
10: compute $\boldsymbol{D}^{(k)} \in \mathbb{R}^{N\times n} \quad k = 1, \ldots, m$ as in (55)
11: initialize
$\boldsymbol{\beta} = \boldsymbol{0}, \quad \hat{\boldsymbol{X}}^{(k)} = \boldsymbol{D}^{(k)}, \quad \boldsymbol{R} = -\boldsymbol{F}(\boldsymbol{t}, \hat{\boldsymbol{X}}^{(1)}, \ldots, \hat{\boldsymbol{X}}^{(m)}), \quad \mathcal{L}_{new} = \|\boldsymbol{R}\|_2^2, \quad \mathcal{L}_{old} = \infty, \quad counter = 0$
12: **while** $(\mathcal{L}_{new} > tol1)$ & $(|\mathcal{L}_{new} - \mathcal{L}_{old}| > tol2)$ & $(counter < maxIterations)$ **do**
13: compute $\boldsymbol{J} \in \mathbb{R}^{(n\cdot N)\times(n\cdot L)}$ as in (62)
14: compute $\boldsymbol{\Delta\beta} = \boldsymbol{J}^{\dagger}\boldsymbol{R}$
15: update $\boldsymbol{\beta} = \boldsymbol{\beta} + \boldsymbol{\Delta\beta}$
16: compute $\hat{\boldsymbol{X}}^{(k)} = \boldsymbol{C}^{(k)}\boldsymbol{\beta} + \boldsymbol{D}^{(k)}, \quad k = 1, \ldots, m$
17: compute $\boldsymbol{R} = \boldsymbol{H}^{(m+1)}\boldsymbol{\beta} - \boldsymbol{F}(\boldsymbol{t}, \hat{\boldsymbol{X}}^{(1)}, \ldots, \hat{\boldsymbol{X}}^{(m)})$
18: set $\mathcal{L}_{old} = \mathcal{L}_{new}$
19: compute $\mathcal{L}_{new} = \|\boldsymbol{R}\|_2^2$
20: increase counter
21: **end while**
22: **return** $\boldsymbol{w}, \boldsymbol{b}, \boldsymbol{\beta}$

In all the examples shown in the following, the parameters $tol1, tol2$ and $maxIterations$ are set to 10^{-10}, 10^{-10} and 100, respectively.

The way of selecting the input weights and the biases of the neurons in the hidden layer is also arbitrary, and the uniform sampling in a given interval is the most common approach. In the following, we always set $M = 3$.

3.2 First Order Initial Value Problems (IVPs)

3.2.1 First Order Linear IVP

The first order IVP in this example is given by an ODE and its initial condition is as follows:

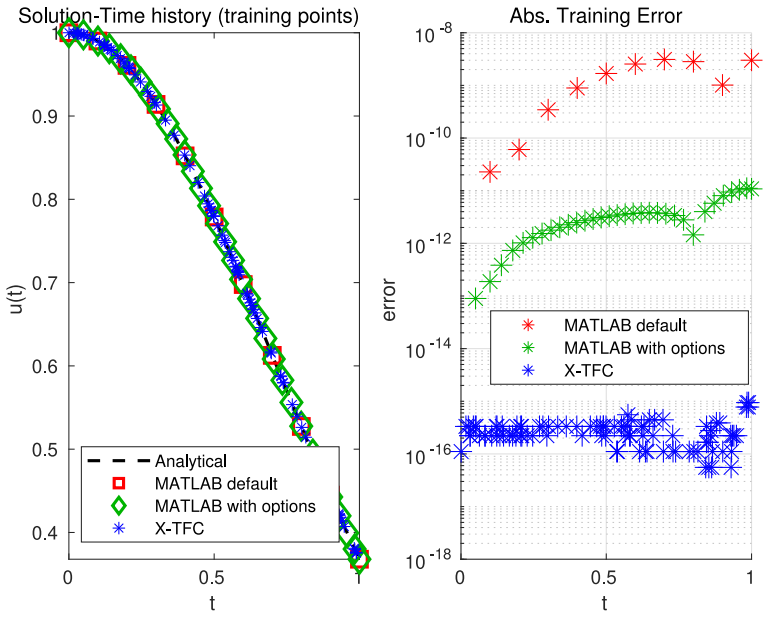

Fig. 2 Analytical, ode45, and X-TFC solutions (left) and absolute errors (right) for training points, in the case of the first order linear IVP (63). XTFC is trained on 100 random points, with 100 hidden neurons, ode45 uses 11 points (10 + initial condition) with default options and 33 points (32 + initial condition) with bound 10^{-10} on absolute and relative error

$$\begin{cases} \dfrac{du}{dt} = -2tu & \forall t \in [0,1] \\ u(0) = 1 \end{cases} . \tag{63}$$

where the exact solution is given by,

$$u(t) = e^{-t^2} .$$

Figures 2, 3, 4 and 5 follow what is presented in Section 3, which we refer to for the description of these figures. In Table 1 accuracy and computational costs -measures in execution time- are reported for rapid comparison.

Here, we only point out some additional remarks:

- $ode45$ has been selected as a reference method because it is explicit and, in the case of solving linear problems, also XTFC is explicit, so the comparison can be made fairly.

- In Figure 3 we can see how performance varies by varying the number of neurons L. In particular, the generalization errors at the training points and test points, and the training error decrease from 0 to 100 neurons, until they reach the value of about 10^{-15} for a greater number of neurons. The generalization error bound and the computational time are invariant when varying the number of neurons (in the order of 10^{-1} and 10^{-4} seconds, respectively).

- In Figure 4 we can see how performance varies by varying the number of training points (N). We chose to start from $N = 100$ and increase by 100. The number of hidden neurons L is fixed to 20. In this plot, we can see that the procedure is stable when increasing the training set and the computational cost, measured through time of execution, is slowly increasing.

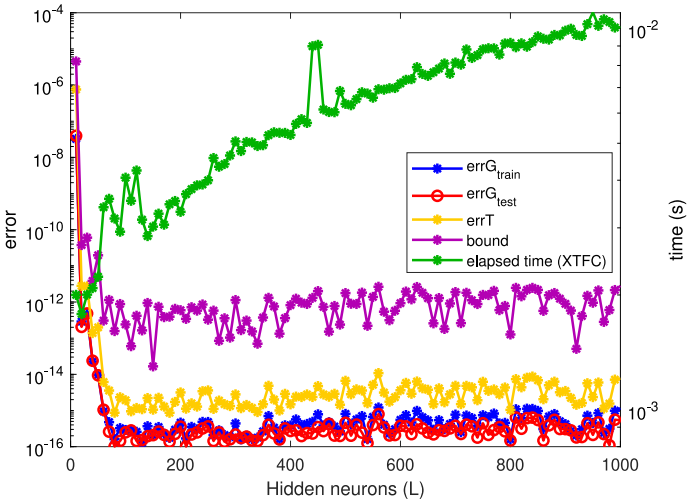

Fig. 3 Computed errors for the first order linear IVP (63) calculated varying the number of neurons with fixed training and test points. We report X-TFC generalization error on training points, generalization error on test points, training error, generalization error bound, and training time. Notice that elapsed time is plotted on a dedicated scale

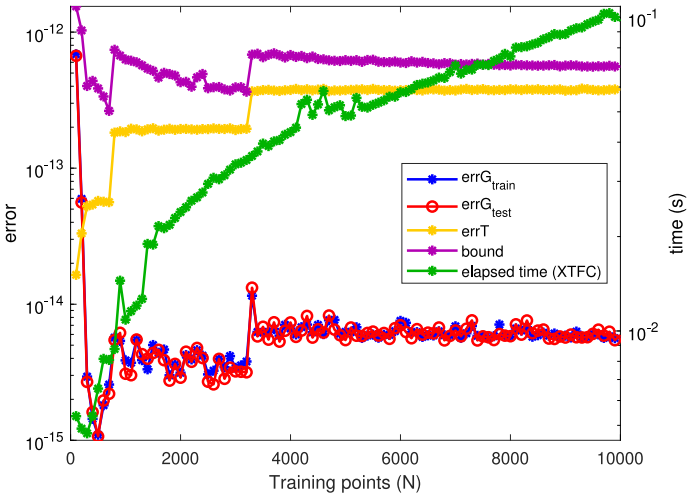

Fig. 4 Computed errors for the first order linear IVP (63) calculated varying the number of training points with fixed neurons and test points. We report X-TFC generalization error on training points, generalization error on test points, training error, generalization error bound, and training time. Notice that elapsed time is plotted on a dedicated scale

- In Figure 5 the comparison with the Matlab solver shows that our code, even if not optimized, compares well with the reference method in this case. This can be noticed in terms of both computational cost, measured through the execution time, and overall accuracy, measured through the different computed errors, see also Table 1.

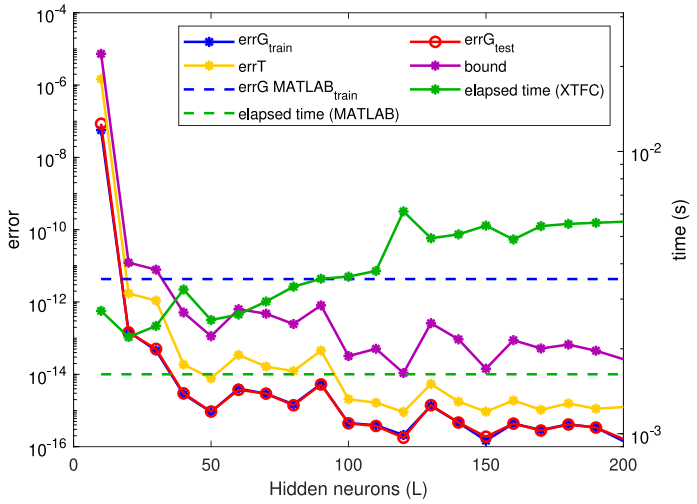

Fig. 5 Convergence in comparison with MATLAB solver results for the problem in eq. (63). Plot reports generalization error on training points and generalization error on test points (both for MATLAB solver and X- TFC), training error, generalization error bound, and training time, varying the number of neurons for fixed number of test points and training points. XTFC is trained on the 33 points used by ode45, for an increasing number of hidden neurons

Table 1 Summary of the comparison between ode45 and XTFC for the resolution of the first order linear IVP (63)

	XTFC, L=30	XTFC, L=60	ode45
errT	$1.10 \cdot 10^{-12}$	$3.44 \cdot 10^{-14}$	$4.32 \cdot 10^{-12}$
errG	$5.06 \cdot 10^{-14}$	$3.73 \cdot 10^{-15}$	$4.32 \cdot 10^{-12}$
average time (ms)	2.4	2.6	1.6

3.2.2 First Order Nonlinear IVP

The second example is given by a nolinear ODE and its initial condition is as follows:

$$\begin{cases} \dfrac{du}{dt} = (1-2t)u^2 & \forall t \in [0,1] \\ u(0) = 1 \end{cases} . \tag{64}$$

Where the exact solution is given by

$$u(t) = \frac{1}{t(t-1)+1} .$$

Figures 6, 7, 8 and 9 follow what is presented in Section 3, which we refer to for the description of these figures. In Table 2 accuracy and computational costs -measures in execution time- are reported for rapid comparison.

Here, we only point out some additional remarks:

- $ode23t$ has been selected as a reference method because it is implicit and, in the case of solving nonlinear problems, also XTFC is a nonlinear method, so the comparison can be made fairly.

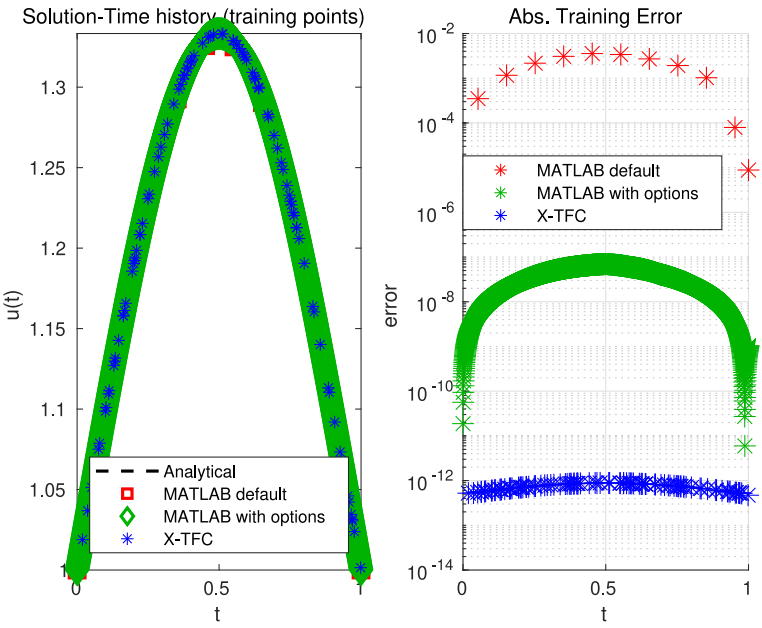

Fig. 6 Analytical, ode23t, and X-TFC solutions (left) and absolute errors (right) for training points, in the case of the first order nonlinear IVP (64). XTFC is trained on 100 random points, with 100 hidden neurons, ode23t uses 12 points (11 + initial condition) with default options and 2242 points (2241 + initial condition) with bound 10^{-10} on absolute and relative error.

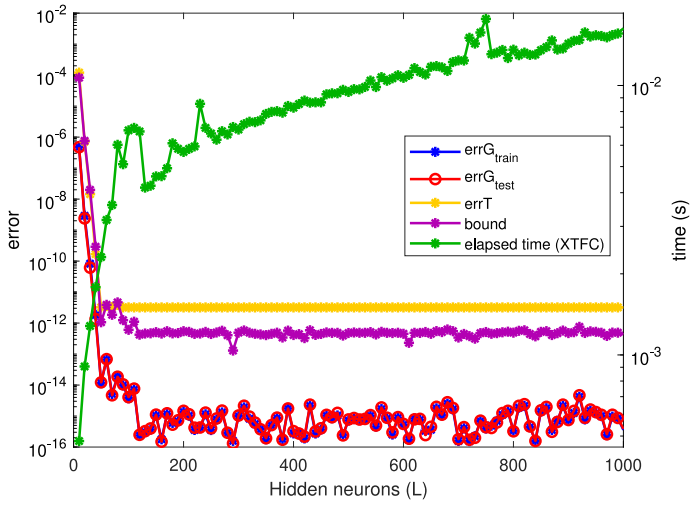

Fig. 7 Computed errors for the first order nonlinear IVP (64) calculated varying the number of neurons with fixed training and test points. We report X-TFC generalization error on training points, generalization error on test points, training error, generalization error bound, and training time. Notice that elapsed time is plotted on a dedicated scale

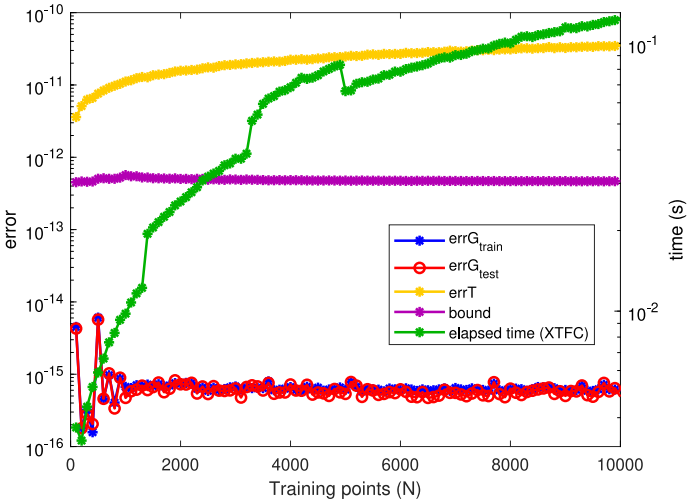

Fig. 8 Computed errors for the first order nonlinear IVP (64) calculated varying the number of training points with fixed neurons and test points. We report X-TFC generalization error on training points, generalization error on test points, training error, generalization error bound, and training time. Notice that elapsed time is plotted on a dedicated scale

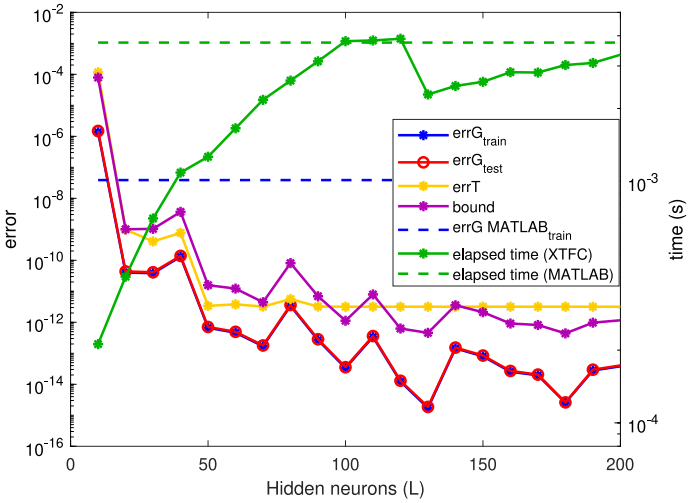

Fig. 9 Convergence in comparison with MATLAB solver results for the problem in eq. (64). Plot reports generalization error on training points and generalization error on test points (both for MATLAB solver and X- TFC), training error, generalization error bound, and training time, varying the number of neurons for fixed number of test points and training points. XTFC is trained on 100 points, and the results are compared to ode23t, which uses 2242 points

Table 2 Summary of the comparison between ode23t and XTFC for the resolution of the first order nonlinear IVP (64)

	XTFC, L=50	XTFC, L=100	ode23t
errT	$3.39 \cdot 10^{-12}$	$3.32 \cdot 10^{-12}$	$3.87 \cdot 10^{-8}$
errG	$7.08 \cdot 10^{-13}$	$3.56 \cdot 10^{-14}$	$3.87 \cdot 10^{-8}$
average time (ms)	1.1	3.8	3.7

- In Figure 7 and 8 we can see what already highlighted in the linear case. Our code has not to be changed and presents very good performance.

- The execution time grows a little bit fast, but this is related to the solution of nonlinear problems, and overall the computational cost is also favorable in this case with respect to the execution time of the reference method, as can be seen in Figure 9 and Table 2.

3.2.3 Stiff First Order Nonlinear IVP

We consider a nonlinear ODE with steeper solutions as follows:

$$\begin{cases} \dfrac{du}{dt} = (1-2t)u^2 & \forall t \in [0,1] \\ u(0) = \frac{1}{\alpha} \end{cases} . \tag{65}$$

Where the exact solution is given by

$$u(t) = \frac{1}{t(t-1)+\alpha} .$$

When $\alpha = 0.25$ the solution is singular and as it approaches this value from above the ODE becomes stiff, we consider values $\alpha = 0.33, \alpha = 0.27$ (See Figs. 10, 11, 12, 13, 14, 15, 16, 17)

3.3 Second-Order IVPs

3.3.1 Second-Order Linear IVP

The third example is given by a second-order ODE subject to two constraints at the initial point as follows:

$$\begin{cases} t^2 \dfrac{d^2u}{dt^2} - t(t+2)\dfrac{du}{dt} + (t+2)\,u = 0 & \forall t \in [1,2] \\ u(1) = 1,\ \dfrac{du}{dt}|_1 = 0 \end{cases} . \tag{66}$$

Where the exact solution is given by

$$u(t) = \left(2 - e^{t-1}\right) t .$$

Figures 18, 19, 20 and 21 follow what is presented in Section 3, which we refer to for the description of these figures. In Table 3 accuracy and computational costs -measures in execution time- are reported for rapid comparison.

Results seem very similar to what was obtained in the first-order case in Section 3.2.1, thus we omit to repeat them and refer to the remarks reported there.

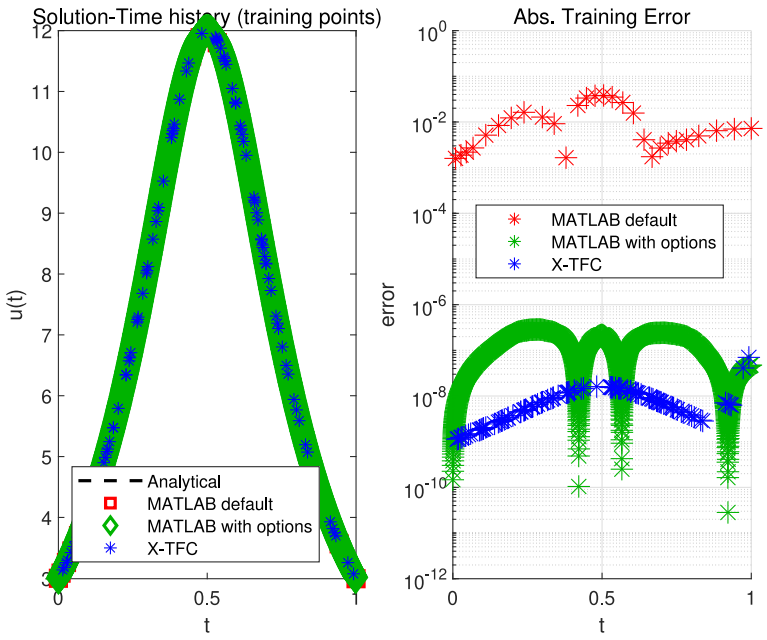

Fig. 10 Analytical, ode23t, and X-TFC solutions (left) and absolute errors (right) for training points, in the case of the first order nonlinear IVP (64). XTFC is trained on 100 random points, with 100 hidden neurons, ode23t uses 29 points (28 + initial condition) with default options and 5437 points (5436 + initial condition) with bound 10^{-10} on absolute and relative error.

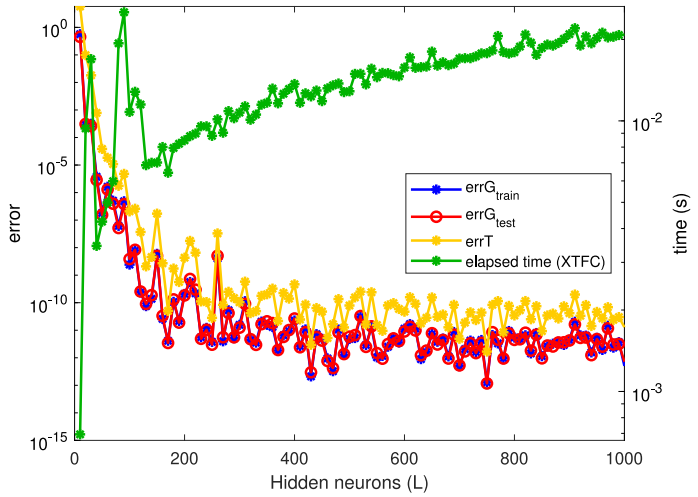

Fig. 11 Computed errors for the first order nonlinear IVP (64) calculated varying the number of neurons with fixed training and test points. We report X-TFC generalization error on training points, generalization error on test points, training error, generalization error bound, and training time. Notice that elapsed time is plotted on a dedicated scale

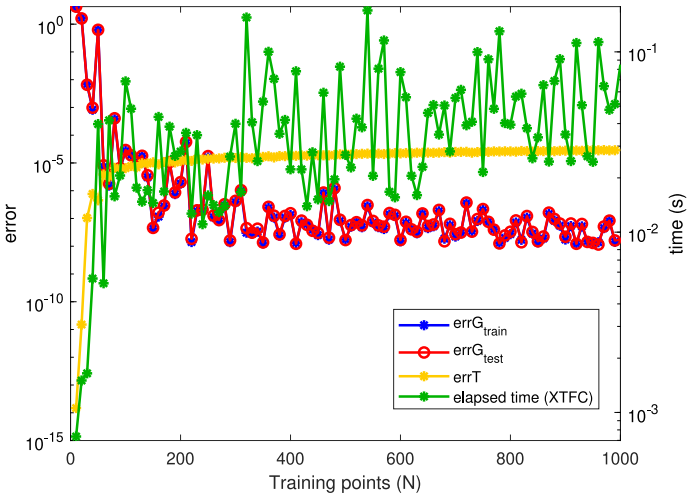

Fig. 12 Computed errors for the first order nonlinear IVP (64) calculated varying the number of training points with fixed neurons and test points. We report X-TFC generalization error on training points, generalization error on test points, training error, generalization error bound, and training time. Notice that elapsed time is plotted on a dedicated scale

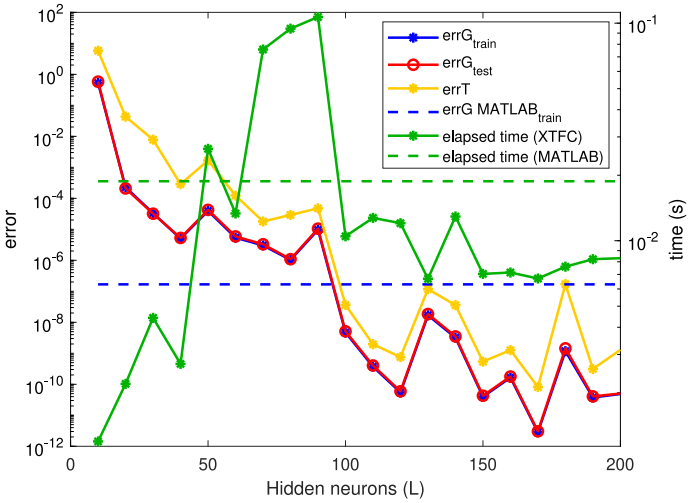

Fig. 13 Convergence in comparison with MATLAB solver results for the problem in eq. (64). Plot reports generalization error on training points and generalization error on test points (both for MATLAB solver and X- TFC), training error, generalization error bound, and training time, varying the number of neurons for fixed number of test points and training points. XTFC is trained on 100 points, and the results are compared to ode23t, which uses 5437 points

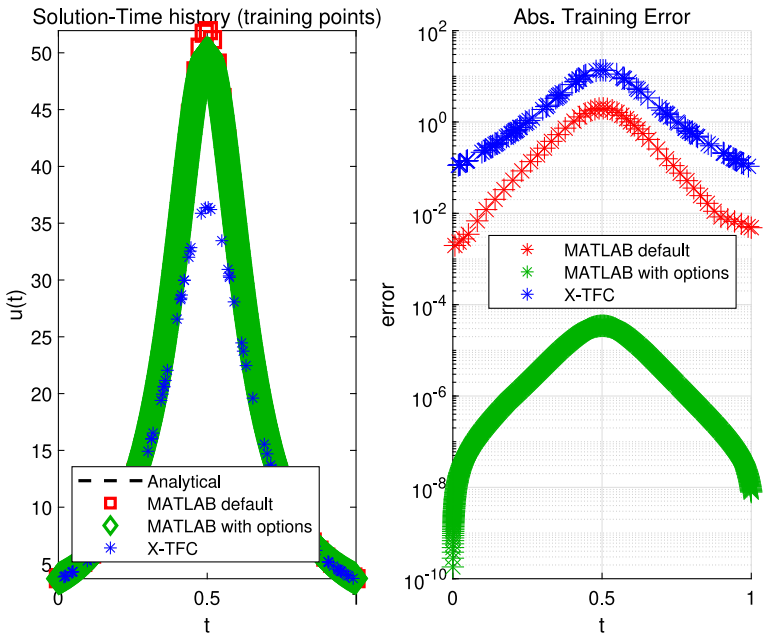

Fig. 14 Analytical, ode23t, and X-TFC solutions (left) and absolute errors (right) for training points, in the case of the first order nonlinear IVP (64). XTFC is trained on 100 random points, with 100 hidden neurons, ode23t uses 49 points (48 + initial condition) with default options and 9572 points (9571 + initial condition) with bound 10^{-10} on absolute and relative error.

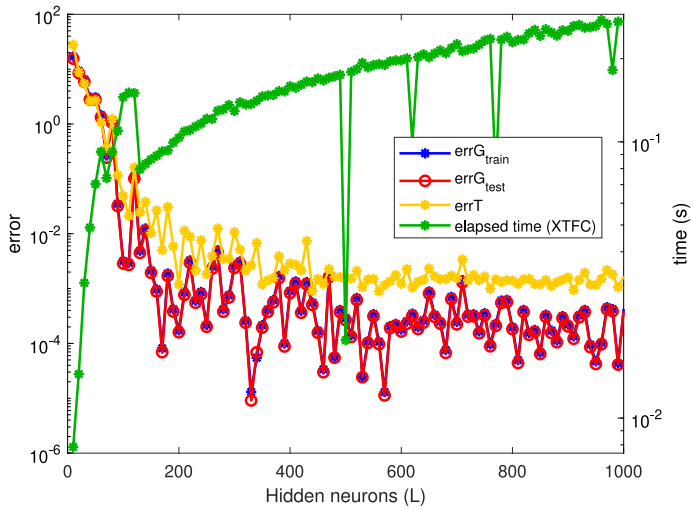

Fig. 15 Computed errors for the first order nonlinear IVP (64) calculated varying the number of neurons with fixed training and test points. We report X-TFC generalization error on training points, generalization error on test points, training error, generalization error bound, and training time. Notice that elapsed time is plotted on a dedicated scale

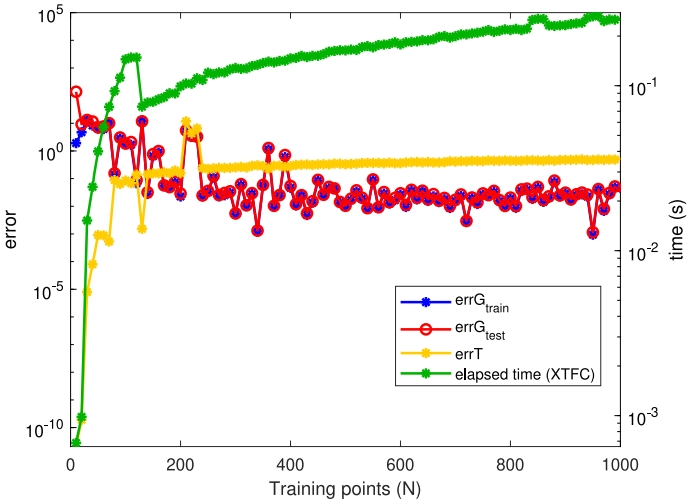

Fig. 16 Computed errors for the first order nonlinear IVP (64) calculated varying the number of training points with fixed neurons and test points. We report X-TFC generalization error on training points, generalization error on test points, training error, generalization error bound, and training time. Notice that elapsed time is plotted on a dedicated scale

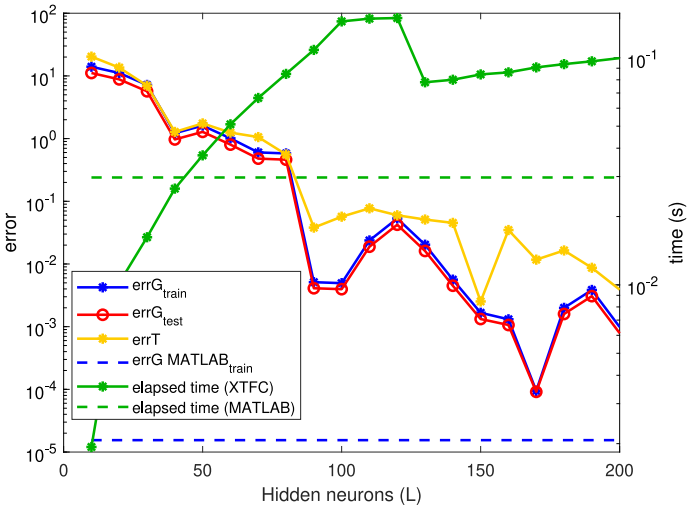

Fig. 17 Convergence in comparison with MATLAB solver results for the problem in eq. (64). Plot reports generalization error on training points and generalization error on test points (both for MATLAB solver and X- TFC), training error, generalization error bound, and training time, varying the number of neurons for fixed number of test points and training points. XTFC is trained on 100 points, and the results are compared to ode23t, which uses 9572 points

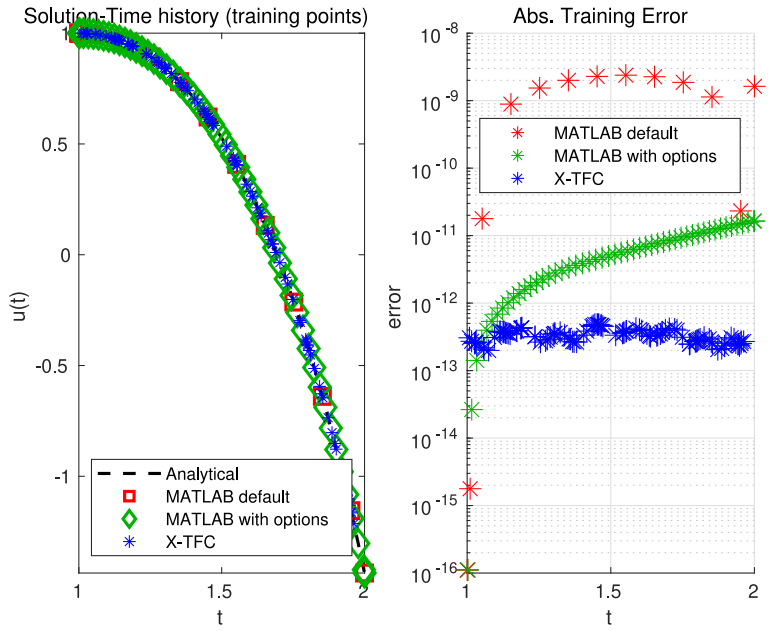

Fig. 18 Analytical, ode45, and X-TFC solutions (left) and absolute errors (right) for training points, in the case of the second order linear IVP (66). XTFC is trained on 100 random points, with 100 hidden neurons, ode45 uses 16 points (15 + initial condition) with default options and 52 points (51 + initial condition) with bound 10^{-10} on absolute and relative error.

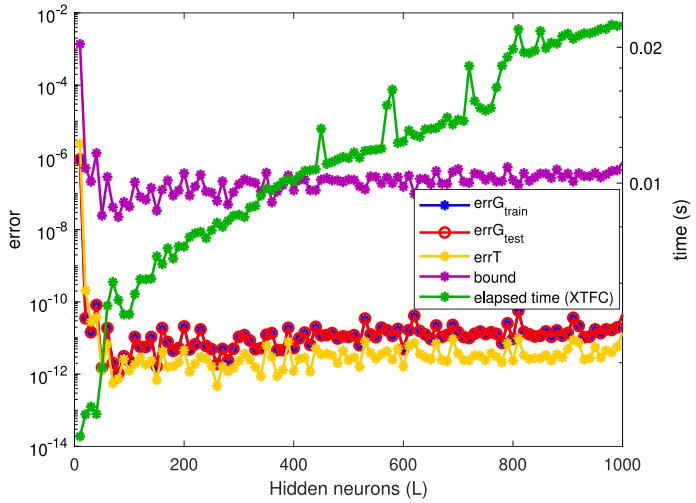

Fig. 19 Computed errors for the second order linear IVP (66) calculated varying the number of neurons with fixed training and test points. We report X-TFC generalization error on training points, generalization error on test points, training error, generalization error bound, and training time. Notice that elapsed time is plotted on a dedicated scale

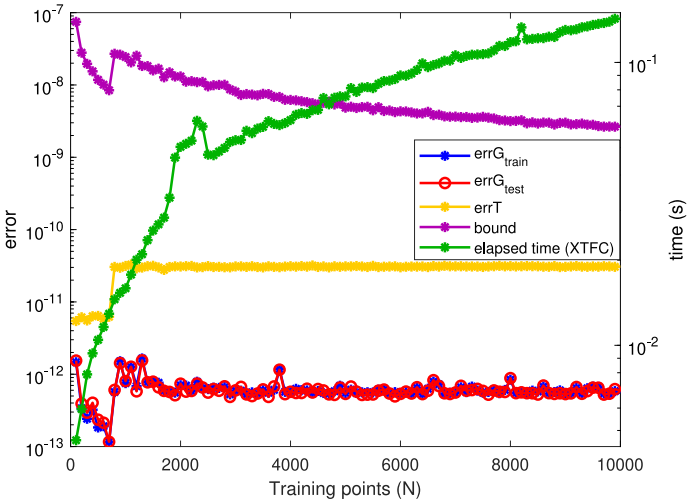

Fig. 20 Computed errors for the second order linear IVP (66) calculated varying the number of training points with fixed neurons and test points. We report X-TFC generalization error on training points, generalization error on test points, training error, generalization error bound, and training time. Notice that elapsed time is plotted on a dedicated scale

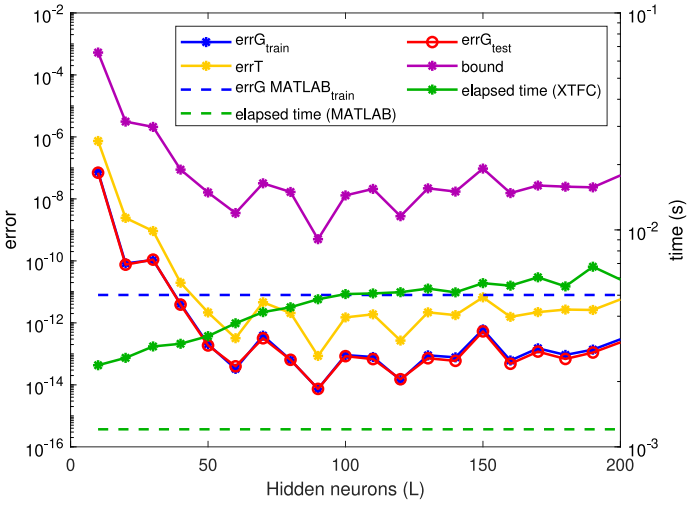

Fig. 21 Convergence in comparison with MATLAB solver results for the problem in eq. (66). Plot reports generalization error on training points and generalization error on test points (both for MATLAB solver and X- TFC), training error, generalization error bound, and training time, varying the number of neurons for fixed number of test points and training points. XTFC is trained on the 52 points used by ode45

Table 3 Summary of the comparison between ode45 and XTFC for the resolution of the second order linear IVP (66)

	XTFC, L=50	XTFC, L=100	ode45
errT	$2.17 \cdot 10^{-12}$	$1.50 \cdot 10^{-12}$	$7.95 \cdot 10^{-12}$
errG	$1.86 \cdot 10^{-13}$	$8.44 \cdot 10^{-14}$	$7.95 \cdot 10^{-12}$
average time (ms)	3.2	5.1	1.2

Table 4 Summary of the comparison between ode23t and XTFC for the resolution of the second order nonlinear IVP (67)

	XTFC, L=50	XTFC, L=100	ode23t
errT	$1.49 \cdot 10^{-11}$	$3.11 \cdot 10^{-11}$	$2.54 \cdot 10^{-8}$
errG	$2.07 \cdot 10^{-10}$	$8.69 \cdot 10^{-9}$	$2.54 \cdot 10^{-8}$
average time (ms)	5.4	14	8.2

3.3.2 Second-Order Nonlinear IVP

In this example, the problem is a second-order nonlinear ODE, subject to two constraints at the initial point as follows:

$$\begin{cases} \dfrac{d^2u}{dt^2} + u\dfrac{du}{dt} = e^{-2t}\sin t\,(\cos t - \sin t) - 2e^{-t}\cos t & \forall t \in [0, \pi] \\ u(0) = 0,\ \dfrac{du}{dt}|_0 = 1 \end{cases} . \tag{67}$$

Where the exact solution is

$$u(t) = e^{-t}\sin t\,.$$

Figures 22, 23, 24 and 25 follow what is presented in Section 3, which we refer to for the description of these figures. In Table 4 accuracy and computational costs -measures in execution time- are reported for rapid comparison.

Results seem very similar to what was obtained in the first-order case in Section 3.2.2, thus we omit to repeat them and refer to the remarks reported there.

3.4 Second-Order Order Two-Points Boundary Value Problems (BVP)

The boundary value problems considered in this paper are treated as a specific case of ODEs with the match of special conditions. For this reason, all that has been commented on before applys and we report figures and tables as before, comments are not repeated.

3.4.1 Second-Order Order Linear BVP

In this example, we aim to solve a two points BVP modeled by the following linear ODE and constraints (See Figs. 26, 27, 28, 29 and Table 5)

$$\begin{cases} \dfrac{d^2u}{dt^2} - \dfrac{du}{dt} = 0 & \forall t \in [0, 1] \\ u(0) = 1,\ u(1) = 0 \end{cases} . \tag{68}$$

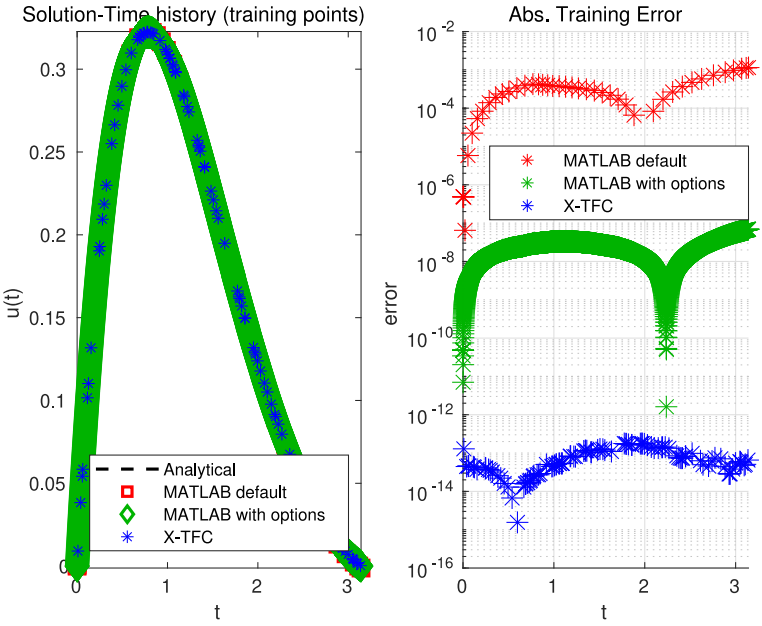

Fig. 22 Analytical, ode23t, and X-TFC solutions (left) and absolute errors (right) for training points, in the case of the second order nonlinear IVP (67). XTFC is trained on 100 random points, with 100 hidden neurons, ode23t uses 45 points (44 + initial condition) with default options and 4079 points (4078 + initial condition) with bound 10^{-10} on absolute and relative error.

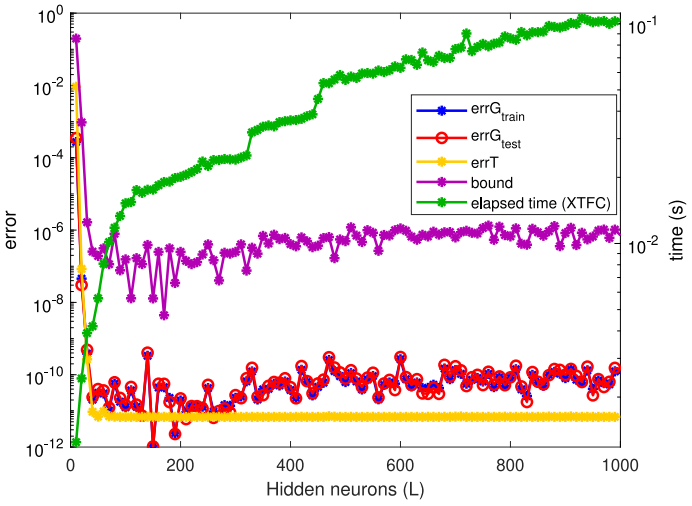

Fig. 23 Computed errors for the second order nonlinear IVP (67) calculated varying the number of neurons with fixed training and test points. We report X-TFC generalization error on training points, generalization error on test points, training error, generalization error bound, and training time. Notice that elapsed time is plotted on a dedicated scale

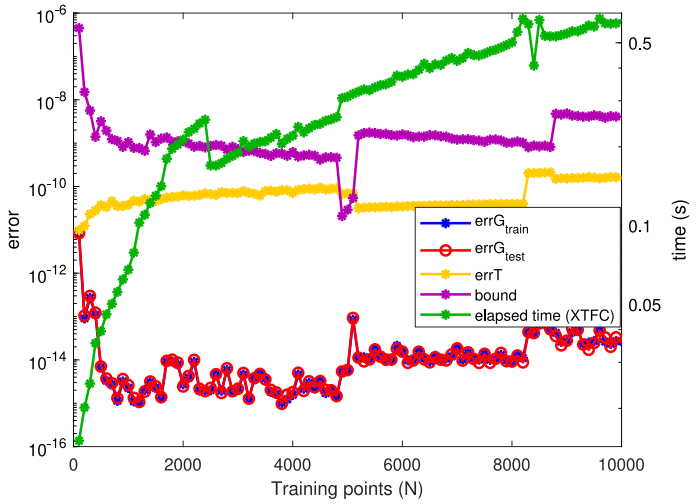

Fig. 24 Computed errors for the second order nonlinear IVP (67) calculated varying the number of training points with fixed neurons and test points. We report X-TFC generalization error on training points, generalization error on test points, training error, generalization error bound, and training time. Notice that elapsed time is plotted on a dedicated scale

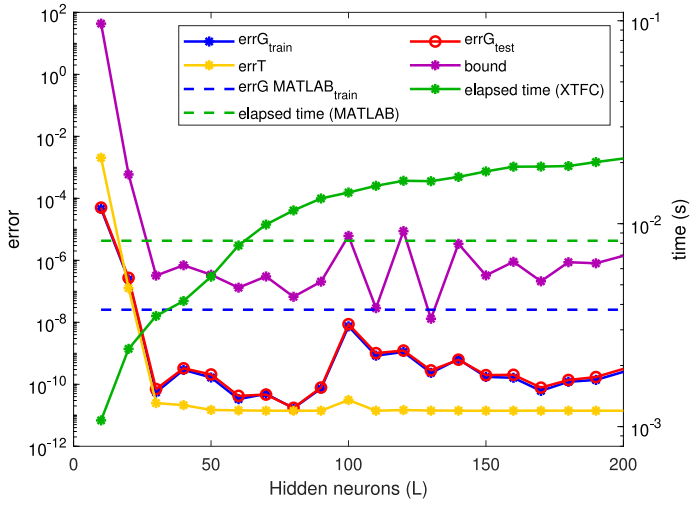

Fig. 25 Convergence in comparison with MATLAB solver results for the problem in eq. (67). Plot reports generalization error on training points and generalization error on test points (both for MATLAB solver and X- TFC), training error, generalization error bound, and training time, varying the number of neurons for fixed number of test points and training points. XTFC is trained on 100 points and the results are compared to ode23t, which uses 4079 points

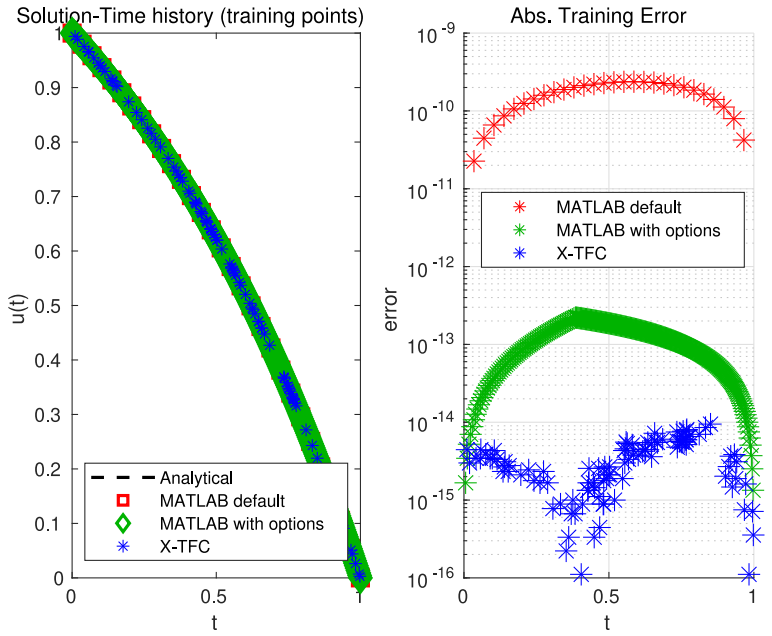

Fig. 26 Analytical, bvp4c, and X-TFC solutions (left) and absolute errors (right) for training points, in the case of the second order linear BVP (68). XTFC is trained on 100 random points, with 100 hidden neurons, bvp4c uses a mesh with 10 points with default options and 303 points with bound 10^{-10} on absolute and relative error

with exact solution as follows:

$$u(t) = \frac{1 - e^{(t-1)}}{1 - e^{-1}} .$$

3.4.2 Second-Order Order Nonlinear BVP

In this last example, we aim to solve a two points BVP modeled by the following nonlinear (Burgers-type) ODE and constraints

$$\begin{cases} \dfrac{d^2u}{dt^2} + u\dfrac{du}{dt} = e^{-2t}\sin t\,(\cos t - \sin t) - 2e^{-t}\cos t & \forall t \in [0, \pi] \\ u(0) = 0,\ u(\pi) = 0 \end{cases} . \tag{69}$$

with exact solution as follows: (See Figs. 30, 31, 32, 33 and Table 6)

$$u(t) = e^{-t}\sin t .$$

3.4.3 System of First-Order Order Nonlinear BVPs

In this last example, we aim to solve the same two points nonlinear BVP but modeled as a system of nonlinear ODEs

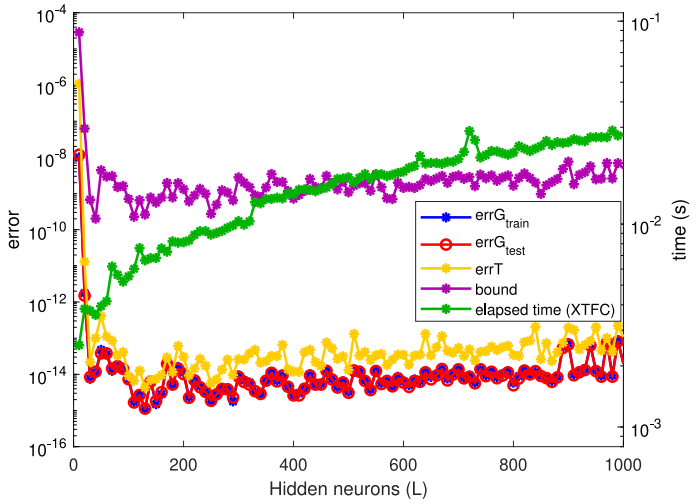

Fig. 27 Computed errors for the second order linear BVP (68) calculated varying the number of neurons with fixed training and test points. We report X-TFC generalization error on training points, generalization error on test points, training error, generalization error bound, and training time. Notice that elapsed time is plotted on a dedicated scale

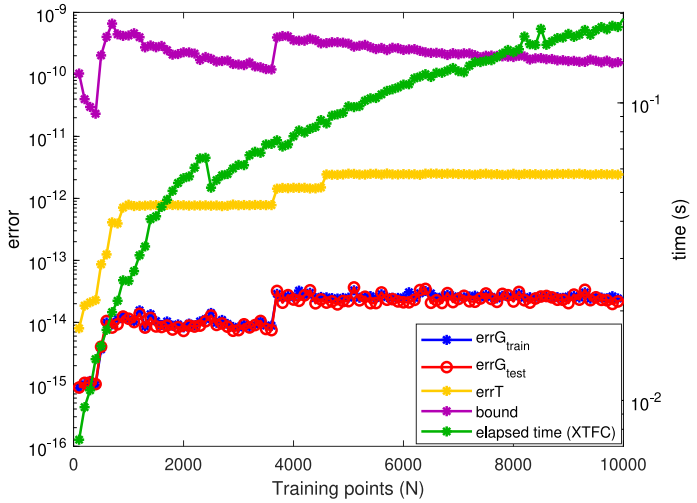

Fig. 28 Computed errors for the second order linear BVP (68) calculated varying the number of training points with fixed neurons and test points. We report X-TFC generalization error on training points, generalization error on test points, training error, generalization error bound, and training time. Notice that elapsed time is plotted on a dedicated scale

$$\begin{cases} \dfrac{dv}{dt} + uv = e^{-2t}\sin t\,(\cos t - \sin t) - 2e^{-t}\cos t & \forall t \in [0, \pi] \\ \frac{du}{dt} = v \\ u(0) = 0,\ u(\pi) = 0 \end{cases} . \tag{70}$$

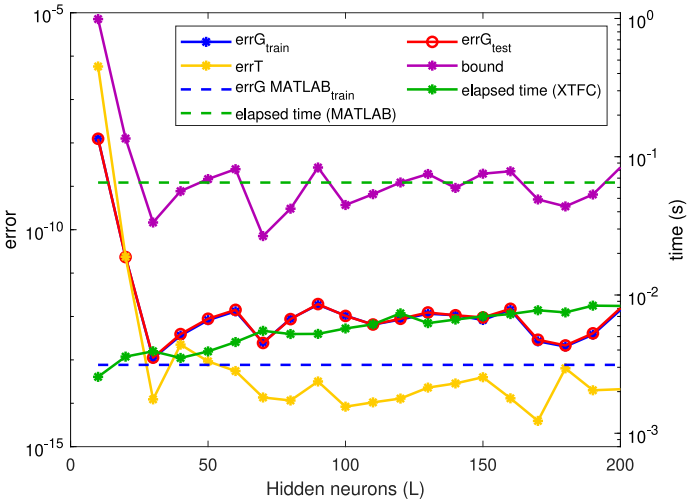

Fig. 29 Convergence in comparison with MATLAB solver results for the problem in eq. (68). Plot reports generalization error on training points and generalization error on test points (both for MATLAB solver and X- TFC), training error, generalization error bound, and training time, varying the number of neurons for fixed number of test points and training points. XTFC is trained on 100 points, and the results are compared to MATLAB's bvp4c, which uses 282 points

Table 5 Summary of the comparison between bvp4c and XTFC for the resolution of the second order linear BVP (68)

	XTFC, L=30	XTFC, L=60	bvp4c
errT	$1.23 \cdot 10^{-14}$	$5.53 \cdot 10^{-14}$	$7.67 \cdot 10^{-14}$
errG	$1.12 \cdot 10^{-13}$	$1.43 \cdot 10^{-12}$	$7.67 \cdot 10^{-14}$
average time (ms)	3.9	4.6	65

Table 6 Summary of the comparison between bvp4c and XTFC for the resolution of the second order nonlinear BVP (69)

	XTFC, L=50	XTFC, L=100	bvp4c
errT	$4.66 \cdot 10^{-12}$	$1.66 \cdot 10^{-13}$	$4.26 \cdot 10^{-13}$
errG	$2.00 \cdot 10^{-11}$	$1.76 \cdot 10^{-12}$	$4.26 \cdot 10^{-13}$
average time (ms)	6.9	18	77

with exact solution as follows: (See Figs. 34, 35, 36 and 37)

$$u(t) = e^{-t} \sin t \, .$$

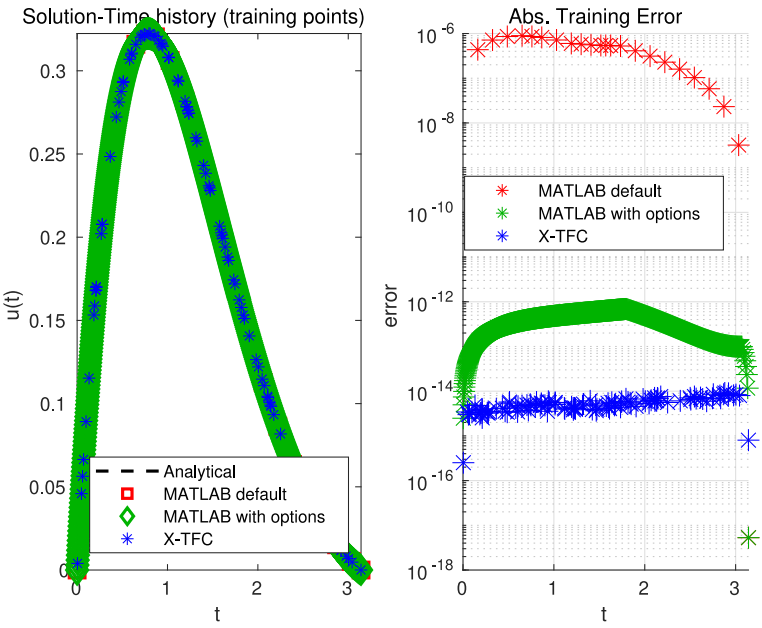

Fig. 30 Analytical, bvp4c, and X-TFC solutions (left) and absolute errors (right) for training points, in the case of the second order nonlinear BVP (69). XTFC is trained on 100 random points, with 100 hidden neurons, bvp4c uses a mesh with 24 points with default options and 817 points with bound 10^{-10} on absolute and relative error.

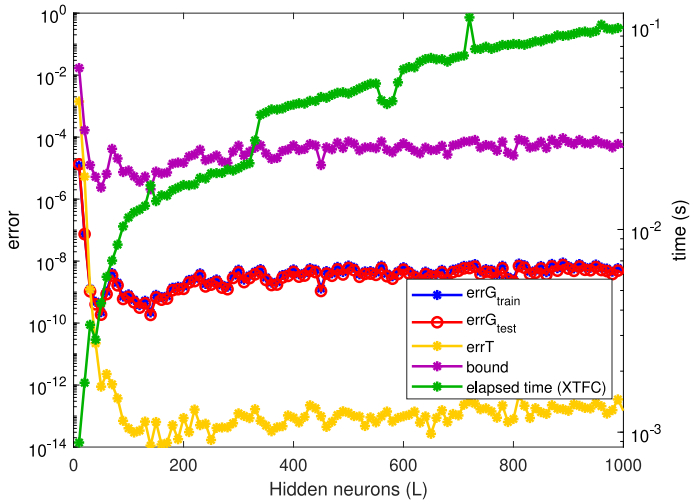

Fig. 31 Computed errors for the second order nonlinear BVP (69) calculated varying the number of neurons with fixed training and test points. We report X-TFC generalization error on training points, generalization error on test points, training error, generalization error bound, and training time. Notice that elapsed time is plotted on a dedicated scale

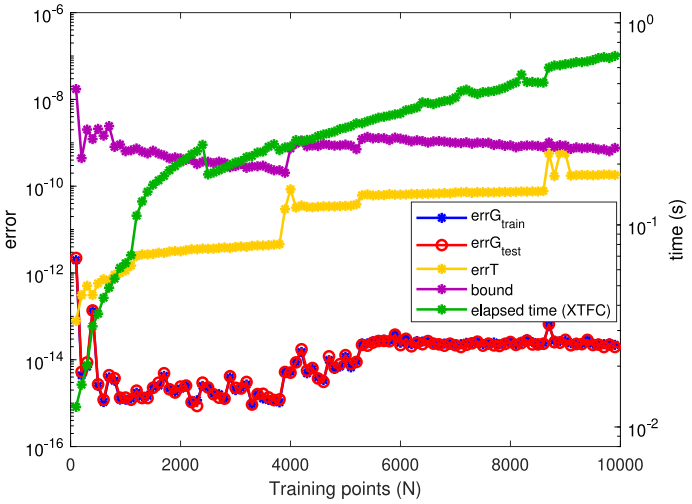

Fig. 32 Computed errors for the second order nonlinear BVP (69) calculated varying the number of training points with fixed neurons and test points. We report X-TFC generalization error on training points, generalization error on test points, training error, generalization error bound, and training time. Notice that elapsed time is plotted on a dedicated scale

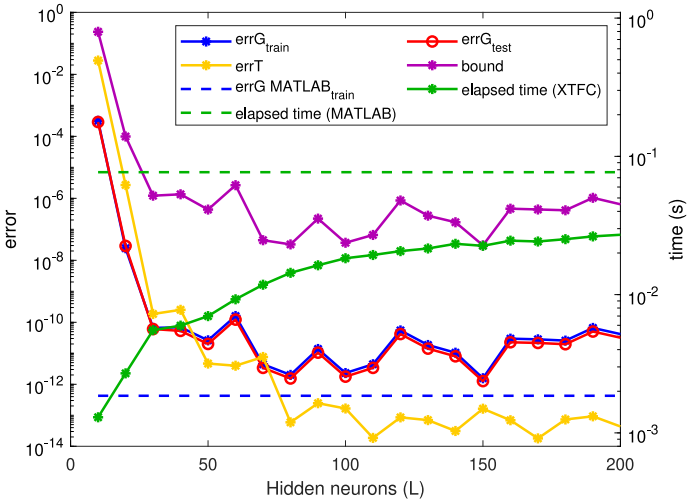

Fig. 33 Convergence in comparison with MATLAB solver results for the problem in eq. (69). Plot reports generalization error on training points and generalization error on test points (both for MATLAB solver and X- TFC), training error, generalization error bound, and training time, varying the number of neurons for fixed number of test points and training points. XTFC is trained on 100 points, and the results are compared to MATLAB's bvp4c, which uses 934 points

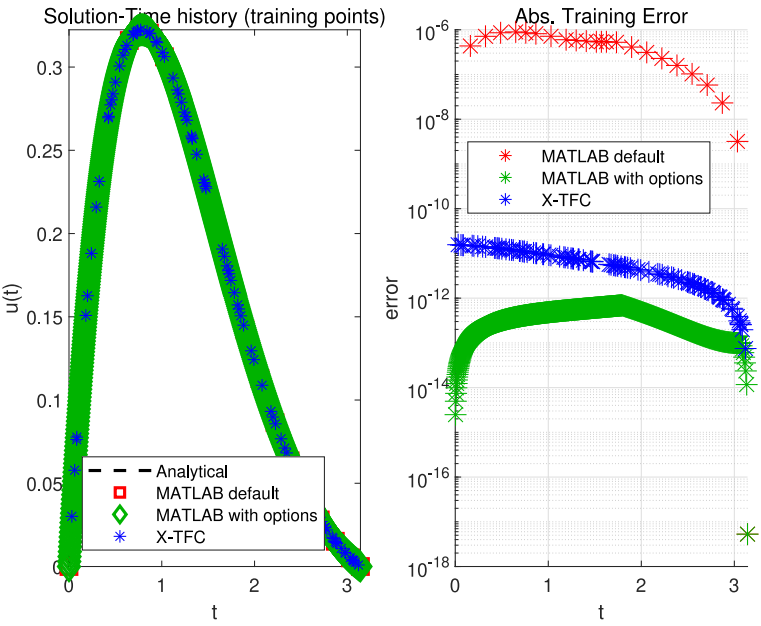

Fig. 34 Analytical, bvp4c, and X-TFC solutions (left) and absolute errors (right) for training points, in the case of the second order nonlinear BVP (69). XTFC is trained on 100 random points, with 100 hidden neurons, bvp4c uses a mesh with 24 points with default options and 817 points with bound 10^{-10} on absolute and relative error. .

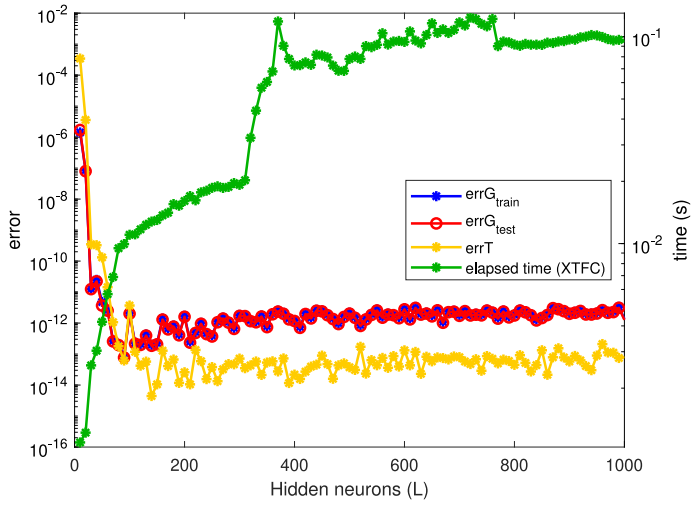

Fig. 35 Computed errors for the second order nonlinear BVP (69) calculated varying the number of neurons with fixed training and test points. We report X-TFC generalization error on training points, generalization error on test points, training error, generalization error bound, and training time. Notice that elapsed time is plotted on a dedicated scale

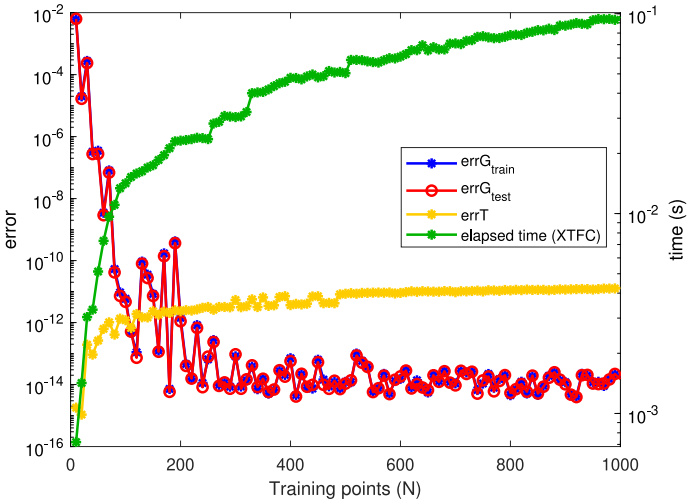

Fig. 36 Computed errors for the second order nonlinear BVP (69) calculated varying the number of training points with fixed neurons and test points. We report X-TFC generalization error on training points, generalization error on test points, training error, generalization error bound, and training time. Notice that elapsed time is plotted on a dedicated scale

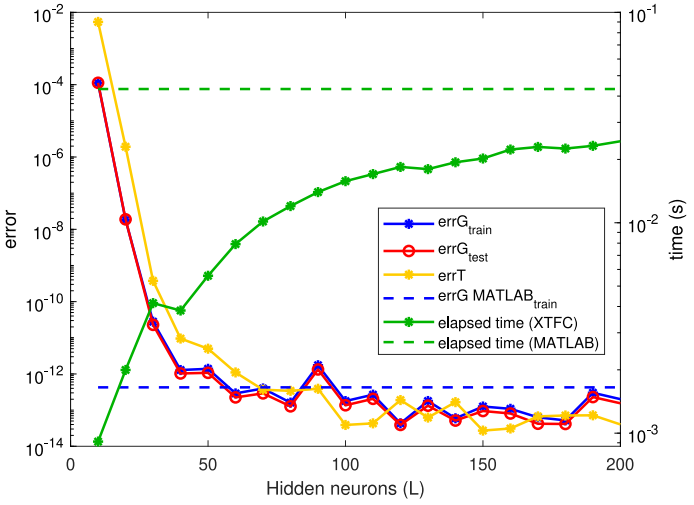

Fig. 37 Convergence in comparison with MATLAB solver results for the problem in eq. (69). Plot reports generalization error on training points and generalization error on test points (both for MATLAB solver and X- TFC), training error, generalization error bound, and training time, varying the number of neurons for fixed number of test points and training points. XTFC is trained on 100 points, and the results are compared to MATLAB's bvp4c, which uses 817 points

4 Conclusions

We have derived upper bounds on the generalization error, in terms of the training error and number and choice of training points, of X-TFC in learning the solutions of univariate DEs. We focused on first-order initial value problems (IVPs), higher-order IVPs, and second-order order two-point boundary value problems. The theoretical results are validated with numerical experiments on some benchmark linear and nonlinear ODEs. Furthermore, the results are compared against those generated with MATLAB solvers: ode45 (for the linear IVPs), ode23t (for the nonlinear IVPs), and bvp4 (for the TPBVPs). X-TFC outperforms the MATLAB solvers both in terms of accuracy and computational effort. For all the cases, X-TFC reaches the machine-level accuracy with a lower computational time. Furthermore, it has to be considered that X-TFC is not (at all) optimized as the MATLAB solvers are. For instance, for a more complex problem, where domain decomposition may be required to achieve better performances, MATLAB solvers will most likely achieve better results than X-TFC. This has been noticed in the stiff problem, where convergence is slower and MATLAB's solver ode23t overperforms the proposed method. Therefore, leveraging on this paper's results (in particular on the superiority of X-TFC on simple benchmark problems such as those considered in this manuscript), work is in progress to optimize X-TFC. In particular, the authors are working on adaptive domain-decomposition techniques for both IVPs and TPBVPs. Moreover, in future works, the authors will focus on the consequences of these theoretical findings on the solution of optimal control problems tackled with indirect methods (calculus of variation and Pontryagin Minimum Principle), which usually belong to one of the categories of problems treated in this paper (TPBVPs).

Acknowledgements The authors thank Prof. Furfaro and Dott. D'Ambrosio of the University of Arizona for the fruitful discussions on the topic.

Funding Open access funding provided by Università degli Studi di Napoli Federico II within the CRUI-CARE Agreement. FC and DDF are members of the INdAM research group GNCS and were partially supported by GNCS-INdAM; this support is gratefully acknowledged. FC was partially supported by PRIN 2022 PNRR P2022WC2ZZ "A multidisciplinary approach to evaluate ecosystems resilience under climate change" and by PRIN 2022 2022N3ZNAX "Numerical Optimization with Adaptive Accuracy and Applications to Machine Learning". Open Access funding enabled and organized by Italy Transformative Agreement.

Data Availability The paper contains all the information to simply generate the datasets supporting the conclusions of this article.

Declarations

Conflict of Interest The authors declare that they have no conflict of interest.

References

1. Ahmadi Daryakenari, N., De Florio, M., Shukla, K., Karniadakis, G.E.: Ai-aristotle: a physics-informed framework for systems biology gray-box identification. PLoS Comput. Biol. **20**(3), 1–33 (2024)
2. Bellman, R.: Dynamic programming. Princeton University Press, Princeton, NJ, USA **1**, 3–25 (1958)
3. Butcher, J.C.: Numerical Methods For Ordinary Differential Equations. John Wiley & Sons, (2016)
4. Calabrò, F., Fabiani, G., Siettos, C.: Extreme learning machine collocation for the numerical solution of elliptic PDEs with sharp gradients. Comput. Methods Appl. Mech. Eng. **387**, 114188 (2021)
5. Canuto, C., Hussaini, M.Y., Quarteroni, A., Zang, T.A.: Spectral methods: fundamentals in single domains. Springer Science & Business Media, (2007)
6. Cottrell, J.A., Hughes, T.J.R., Bazilevs, Y.: Isogeometric analysis: toward integration of CAD and FEA. John Wiley & Sons, (2009)
7. D'Ambrosio, A., Schiassi, E., Curti, F., Furfarom, R.: Physics-informed neural networks applied to a series of constrained space guidance problems. In 31st AAS/AIAA Space flight MEchanics Meeting, (2021)
8. D'Ambrosio, A., Schiassi, E., Curti, F., Furfaro, R.: Pontryagin neural networks with functional interpolation for optimal intercept problems. Mathematics **9**(9), 996 (2021)
9. D'Ambrosio, A., Schiassi, E., Johnston, H., Curti, F., Mortari, D., Furfaro, R.: Time-energy optimal landing on planetary bodies via theory of functional connections. Adv. Space Res. **69**(12), 4198–4220 (2022)
10. Falco, D.E.D., Schiassi, E., Calabrò, F.: Least squares with equality constraints extreme learning machines for the resolution of pdes. J. Comput. Phys. **547**, 114553 (2026)
11. De Florio, M., Kevrekidis, I.G., Karniadakis, G.E.: AI-Lorenz: a physics-data-driven framework for black-box and gray-box identification of chaotic systems with symbolic regression. arXiv:2312.14237, (2023)
12. De Florio, M., Schiassi, E., Calabrò, F., Furfaro, R.: Physics-informed neural networks for 2nd order odes with sharp gradients. J. Comput. Appl. Math. **436**, 115396 (2024)
13. De Florio, M., Schiassi, E., D'Ambrosio, A., Mortari, D., Furfaro, R.: Theory of functional connections applied to linear odes subject to integral constraints and linear ordinary integro-differential equations. Math Comput App **26**(3), 65 (2021)
14. De Florio, M., Schiassi, E., Furfaro, R.: Physics-informed neural networks and functional interpolation for stiff chemical kinetics. Chaos: An Interdisciplinary J Nonlinear Sci **32**(6), 063107 (2022)
15. 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. Transfer **259**, 107384 (2021)
16. De Florio, M., Schiassi, E., Ganapol, B.D., Furfaro, R.: Physics-informed neural networks for rarefied-gas dynamics: thermal creep flow in the bhatnagar-gross-krook approximation. Phys. Fluids **33**(4), 047110 (2021)
17. De Florio, M., Schiassi, E., Ganapol, B.D., Furfaro, R.: Physics-informed neural networks for rarefied-gas dynamics: poiseuille flow in the bgk approximation. Z. Angew. Math. Phys. **73**(3), 1–18 (2022)
18. Ding, S., Zhao, H., Zhang, Y., Xu, X., Nie, R.: Extreme learning machine: algorithm, theory and applications. Artif. Intell. Rev. **44**, 103–115 (2015)
19. Dissanayake, M., Phan-Thien, N.: Neural-network-based approximations for solving partial differential equations. Commun. Numer. Methods Eng. **10**(3), 195–201 (1994)
20. Dong, S., Li, Z.: Local extreme learning machines and domain decomposition for solving linear and nonlinear partial differential equations. Comput. Methods Appl. Mech. Eng. **387**, 114129 (2021)
21. Dwivedi, V., Srinivasan, B.: Physics informed extreme learning machine (PIELM) a rapid method for the numerical solution of partial differential equations. Neurocomputing **391**, 96–118 (2020)
22. Eymard, R., Gallouët, T., Herbin, R.: Finite volume methods. Handb. Numer. Anal. **7**, 713–1018 (2000)
23. Fabiani, G., Calabrò, F., Russo, L., Siettos, C.: Numerical solution and bifurcation analysis of nonlinear partial differential equations with extreme learning machines. J. Sci. Comput. **89**(2), 1–35 (2021)
24. Filipov, S.M., Gospodinov, I.D., Faragó, I.: Shooting-projection method for two-point boundary value problems. Appl. Math. Lett. **72**, 10–15 (2017)
25. Hairer, E., Norsett, S.P., Wanner, G.: Solving Ordinary Differential Equations. Springer, (1987)
26. Huang, G., Huang, G.-B., Song, S., You, K.: Trends in extreme learning machines: a review. Neural Netw. **61**, 32–48 (2015)
27. Huang, G.-B., Chen, L., Siew, C.-K.: Universal approximation using incremental constructive feedforward networks with random hidden nodes. IEEE Trans. Neural Networks **17**(4), 879–892 (2006)
28. Huang, G.-B., Zhu, Q.-Y., Siew, C.-K.: Extreme learning machine: theory and applications. Neurocomputing **70**(2006), 489–501 (2006)

29. Hughes, T.J.R.: The finite element method: linear static and dynamic finite element analysis. Courier Corporation, (2012)
30. Johnston, H.: The theory of functional connections: a journey from theory to application. arXiv:2105.08034, (2021)
31. Johnston, H., Mortari, D.: Linear Differential Equations Subject to Multivalued, Relative and/or Integral Constraints with Comparisons to Chebfun. SIAM Journal of Numerical Analysis, 2018. Submitted
32. 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. **67**(4), 1521–1552 (2020)
33. Jovanović, B.S., Süli, E.: Analysis of finite difference schemes: for linear partial differential equations with generalized solutions, volume 46. Springer Science & Business Media, (2013)
34. Karniadakis, G.E., Kevrekidis, I.G., Lu, L., Perdikaris, P., Wang, S., Yang, L.: Physics-informed machine learning. Nat. Rev. Phys. **3**(6), 422–440 (2021)
35. Lagaris, I.E., Likas, A., Fotiadis, D.I.: Artificial neural networks for solving ordinary and partial differential equations. IEEE Trans. Neural Networks **9**(5), 987–1000 (1998)
36. Laghi, L., Schiassi, E., De Florio, M., Furfaro, R., Mostacci, D.: Physics-informed neural networks for 1-d steady-state diffusion-advection-reaction equations. Nucl. Sci. Eng. **197**(9), 2373–2403 (2023)
37. Leake, C.: The multivariate theory of functional connections: an n-dimensional constraint embedding technique applied to partial differential equations. PhD thesis, Texas A&M University, (2021)
38. Leake, C., Johnston, H., Mortari, D.: The multivariate theory of functional connections: theory, proofs, and application in partial differential equations. Mathematics **8**(8), 1303 (2020)
39. Leake, C., Johnston, H., Mortari, D.: The theory of functional connections: a functional interpolation framework with applications. Lulu, (2022)
40. Leake, C., Mortari, D.: Deep theory of functional connections: a new method for estimating the solutions of partial differential equations. Mach. Learn. Knowl. Extr. **2**(1), 37–55 (2020)
41. Mishra, S., Molinaro, R.: Estimates on the generalization error of physics-informed neural networks for approximating PDEs. IMA J. Numer. Anal. **43**(1), 1–43 (2023)
42. Mortari, D.: The theory of connections: connecting points. MDPI Mathematics, 5(57), (2017)
43. Moukalled, F., Mangani, L., Darwish, M.: The finite volume method. In The finite volume method in computational fluid dynamics, pp. 103–135. Springer, (2016)
44. 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. **378**, 686–707 (2019)
45. Reddy, J.N.: Introduction to the finite element method. McGraw-Hill Education, (2019)
46. Schiassi, E., D'Ambrosio, A., Drozd, K., Curti, F., Furfaro, R.: Physics-informed neural networks for optimal planar orbit transfers. Journal of Spacecraft and Rockets, pp. 1–16, (2022)
47. Schiassi, E., D'Ambrosio, A., Scorsoglio, A., Furfaro, R., Curti, F.: Class of optimal space guidance problems solved via indirect methods and physics-informed neural networks. In 31st AAS/AIAA Space flight MEchanics Meeting, (2021)
48. 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 **9**(17), 2069 (2021)
49. Schiassi, E., De Florio, M., Ganapol, B.D., Picca, P., Furfaro, R.: Physics-informed neural networks for the point kinetics equations for nuclear reactor dynamics. Ann. Nucl. Energy **167**, 108833 (2022)
50. 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. Neurocomput (2021)
51. Wang, J., Lu, S., Wang, S.-H., Zhang, Y.-D.: A review on extreme learning machine. Multi Tools App **81**(29), 41611–41660 (2022)
52. Wang, S., Teng, Y., Perdikaris, P.: Understanding and mitigating gradient flow pathologies in physics-informed neural networks. SIAM J. Sci. Comput. **43**(5), A3055–A3081 (2021)
53. Zienkiewicz, O.C., Taylor, R.L., Zhu, J.Z.: The finite element method: its basis and fundamentals. Elsevier, (2005)