Journal of Computational Physics 492 (2023) 112408

Contents lists available at ScienceDirect

Journal of Computational Physics

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

Discrete-time nonlinear feedback linearization via physics-informed machine learning

Hector Vargas Alvarez [a], Gianluca Fabiani [a,d], Nikolaos Kazantzis [b], Constantinos Siettos [c,*], Ioannis G. Kevrekidis [d,e,f,**]

[a] Scuola Superiore Meridionale, Naples, Italy
[b] Department of Chemical Engineering, Worcester Polytechnic Institute, Worcester, USA
[c] Dipartimento di Matematica e Applicazioni "Renato Caccioppoli", Università degli Studi di Napoli "Federico II", Naples, Italy
[d] Department of Chemical & Biomolecular Engineering, Johns Hopkins University, Baltimore, USA
[e] Department of Applied Mathematics and Statistics, Johns Hopkins University, Baltimore, USA
[f] Medical School, Department of Urology, Johns Hopkins University, Baltimore, USA

ARTICLE INFO

Article history:
Received 14 March 2023
Received in revised form 16 July 2023
Accepted 31 July 2023
Available online 5 August 2023

Keywords:
Physics-informed machine learning
Feedback linearization
Nonlinear discrete time systems
Greedy training

ABSTRACT

We present a physics-informed machine learning (PIML) scheme for the feedback linearization of nonlinear discrete-time dynamical systems. The PIML finds the nonlinear transformation law, thus ensuring stability via pole placement, in one step. In order to facilitate convergence in the presence of steep gradients in the nonlinear transformation law, we address a greedy training procedure. We assess the performance of the proposed PIML approach via a benchmark nonlinear discrete map for which the feedback linearization transformation law can be derived analytically; the example is characterized by steep gradients, due to the presence of singularities, in the domain of interest. We show that the proposed PIML outperforms, in terms of numerical approximation accuracy, the traditional numerical implementation, which involves the construction –and the solution in terms of the coefficients of a power-series expansion–of a system of homological equations as well as the implementation of the PIML in the entire domain, thus highlighting the importance of continuation techniques in the training procedure of PIML schemes.

1. Introduction

A fundamental controller synthesis and design approach for nonlinear discrete-time systems relies on the use of feedback to explicitly modify the system dynamics and induce desirable dynamic characteristics that conform to a prespecified set of performance requirements and design objectives [1–6]. In particular, introducing feedback action to explicitly assign desirable dynamic modes to the controlled (closed-loop) system, by placing its poles at specific locations on the complex plane, represents an important and powerful technique in modern nonlinear control theory and practice [1–3,7,8]. Two dominant pole-placing nonlinear feedback control approaches can be discerned in the pertinent literature, with historical roots in geometric control theory [7,8,1,3]. The first, known as the exact input/output feedback linearization approach, uses appropriately derived state feedback control laws to induce linear input/output behavior by forcing the system's output

* Corresponding author at: Dipartimento di Matematica e Applicazioni "Renato Caccioppoli", Università degli Studi di Napoli "Federico II", Naples, Italy.
** Corresponding author at: Department of Chemical & Biomolecular Engineering, Johns Hopkins University, Baltimore, USA.
E-mail addresses: constantinos.siettos@unina.it (C. Siettos), yannisk@jhu.edu (I.G. Kevrekidis).

https://doi.org/10.1016/j.jcp.2023.112408

variable to track a pre-specified "target" linear and stable trajectory. This approach offers a nonlinear analogue to the classic linear pole-placement method, where the closed-loop poles are placed at prespecified values, but remains limited to the special class of minimum-phase systems. However, in broad classes of nonlinear system regulation and/or stabilization problems, the primary objective is not only to force the system output variable to track a prespecified set-point profile, but rather to force *all system states* to return to desirable design steady states (equilibria) in a fast and smooth manner whenever the system experiences inevitable dynamic excursions due to the effect of disturbances [1,3,9]. Within the context of geometric exact feedback linearization, this second approach was first introduced in the seminal and insightful works presented in [10–18] and is realized through a two-step controller synthesis/design procedure. In the first step, a nonlinear coordinate transformation and a state feedback control law are derived capable of transforming the original system into a linear and controllable one, under an external reference input and in an affine state space representation. The natural second step involves the employment of well-established pole-placement methods applied to the transformed linear system. It should be pointed out, however, that the exact feedback linearization approach impinges on a set of rather restrictive conditions that can hardly be met by physical and engineering systems.

In a conceptually different problem reformulation, Guardabassi and Savaresi [19] addressed the feedback linearization problem using the so-called *virtual input direct design* approach, whose principal characteristic is that it reduces the control problem into a standard non-linear mapping approximation problem, without resorting to a preliminary construction of an ODE-based model. It is worth noting that in the formulation of a nonlinear feedback regulation/stabilization problem as described earlier, the presence of an external reference input variable, introduced in the first step of the classic exact feedback linearization approach, becomes irrelevant and redundant [1,3,9]. In light of the above realization, and conceptually inspired by Luenberger's early ideas on a single-step approach to feedback-induced pole-placement in continuous-time linear systems theory [20], Kazantzis [21] developed a nonlinear discrete-time analogue by formulating the problem within the context of nonlinear functional equations theory. This approach allows the assignment of the controlled (closed-loop) system's dynamic modes by meeting *both the feedback linearization and the pole-placement objectives* in a single-step, while effectively overcoming the restrictive conditions associated with the traditional two-step exact feedback linearization approach. Within a similar conceptual and methodological context, further interesting investigations on deriving approximates of the feedback-linearizing and pole-placing control laws [22], rigorously establishing links to key system-theoretic concepts such as immersion and invariance properties [23] as well as implementing the controller synthesis method to partially distributed systems [24] are noteworthy. Between the mid 90s and early 2000s, emerging research activity focused on the development of various feedback-linearization methods with integrated machine learning capabilities [25–30]. For example, Yeşildirek and Lewis [25] addressed the control of a class of single input single output (SISO) nonlinear systems using a multilayer artificial neural network (ANN)-based controller that performs feedback linearization while ensuring Lyapunov stability. He et al. [27], proposed a scheme based on the concept of feedback linearization and ANNs to simultaneously approximate the nonlinear transformation and the controller dynamics itself. Siettos et al. [28] proposed a fuzzy controller for the stabilization of equilibria of fluidized bed dryers and compared its performance with input-output linearization. Ge et al. [29] used multilayer ANNs to reconstruct an implicit feedback linearization scheme for adaptive tracking control purposes. Siettos and Bafas [30] combined fuzzy logic and feedback linearization in order to achieve "semiglobal" stabilization of nonlinear singularly perturbed systems; a fuzzy scheme was used to decompose the full system into fast and slow dynamics, and then feedback linearization was employed under a set of properly derived sufficient conditions for Lyapunov stability. Deng et al. [31] proposed a feedback linearization scheme implemented by ANNs for the adaptive control of non-affine nonlinear discrete-time systems.

More recently, theoretical and technological advances have renewed the interest of the control community towards the development of new schemes, based on machine learning, by revisiting well established methods as well as introducing new ones. For example, reinforcement learning has been used for the feedback linearization of discrete-time nonaffine nonlinear systems with unknown bounded disturbances [32], and uncertain continuous affine models of ODEs [33], the control of PDEs [34], and the data-driven control of trajectories [35]. Umlauft et al. [36] applied feedback linearization on a data-driven model using Gaussian process regression. Wu et al. [37] proposed a machine learning-based predictive control scheme based on recurrent neural networks (RNNs) to approximate nonlinear dynamics and guarantee Lyapunov stability in the presence of model uncertainty; for a comprehensive review on model predictive control schemes via machine learning, the interested reader is referred to the paper of Ren et al. [38]. Tang and Daoutidis [39] proposed a data-driven dissipative-based control scheme based on input–output data, where the learning and the controller design are carried out successively. Physics-informed neural networks [40–42] have been used to solve high-dimensional optimal control problems related to the Hamilton–Jacobi PDE on field-programmable gate arrays (FPGAs) [43]. Recently, Patsatzis et al. [44] proposed an equation/variable free data-driven control approach for agent-based models based on machine/manifold learning which does not require knowledge of the "correct" macroscopic observables nor of any physical insights into the "correct" type of ODEs or PDEs, thus obviating the need to construct explicitly surrogate, reduced-order machine-learning models (such as ANNs, Deep Learning and Gaussian Processes), that *de facto* introduce biases.

Here, we propose a physics-informed machine learning (PIML) scheme for learning feedback linearizing control laws for nonlinear discrete-time systems. In contrast to other previous works that used machine learning and in particular Artificial Neural Networks (ANNs) to first approximate the feedback linearizing transformation, and only then to apply a control law, our approach achieves this objective in a single step, based on a problem reformulation aligned with the methodological framework of simultaneously attaining feedback linearization and pole placement [9,21,45]. The theoretical background

(conceptual, methodological and analytical foundations) of the "traditional" scheme is discussed in [9,21], and its Equation-free version for the control of microscopic simulators is presented in [45] (see also [30,46–48,44] for the Equation-free control approach). In these research studies, the numerical approximation of the feedback-linearizing transformation and control is performed by (i) first approximating the transformation map with a power-series expansion, and (ii) then, by using a symbolic software package, recursively solving a standard Lyapunov matrix equation and a system of linear algebraic equations for the unknown coefficients of the aforementioned power-series expansion. However, such a power-series expansion procedure becomes intractable even for medium-scale dimensions and cannot guarantee the desired numerical approximation accuracy *in the entire domain*, especially in regimes that contain very-steep gradients that resemble singularities. Thus, in order to facilitate the numerical approximation ability of the proposed PIML scheme in the presence of steep-gradients, we follow a step/greedy training procedure (see also [49]): we start learning the nonlinear transformation *on a subset of the entire domain*, where the transformation is sought, and gradually augmenting its size, thus "warm-restarting" the training procedure using as initial guesses for the unknown weights of the ANN, the ones found from the previous step, in the spirit of continuation/homotopy computations.

This proposed PIML scheme can be easily implemented. In fact, for our illustrations, we developed a "home-made" code in Matlab 2022; for training, we wrapped around it the Levenberg-Marquard optimization algorithm as implemented by the `nonlinsq` function. For comparison purposes, an implementation in Python using the Keras API of TensorFlow library was also performed. To demonstrate the efficiency of the proposed continuation/greedy PIML scheme, we used a benchmark two-dimensional discrete-time model whose feedback-linearizing control law can be derived analytically [21]. The particular transformation map (and thus the attendant feedback-linearizing control law) exhibits a singularity at the domain boundary, thus making it difficult to approximate it numerically close to that point, especially through a power-series expansion. For illustration purposes, we also compared the numerical approximation accuracy of the proposed PIML greedy scheme against the standard power-series expansion, as well as Matlab and Python's TensorFlow-based implementations with automatic differentiation that was used to learn the transformation in the entire domain. Furthermore, we considered two different scenarios, namely: (a) one where we assumed that the equations of the model are explicitly known, and (b) one where we assumed that only a black-box simulator is available, i.e., pertinent equations are not available explicitly in closed form.

The paper is organized as follows: in Section 2, we provide a brief review and preliminaries related to the methodological framework associated with the single-step feedback linearization method for nonlinear discrete-time systems, and then we describe the proposed PIML scheme. The benchmark problem is presented in Section 3. Section 4 encompasses the numerical results obtained under the various approaches, as well as a comparative performance assessment. Finally, concluding remarks are offered in Section 5.

2. Methodological framework: using physics-informed machine learning for feedback linearization and pole-placement in a single step

Nonlinear discrete-time input-driven dynamical systems are considered with the following non-affine state-space realization:

$$x(t+1) = f(x(t), u(t)), \tag{1}$$

where $t = 0, 1, \ldots$ is the discrete-time index, $x(t) \in \mathbb{R}^n$ is the vector of state variables, $u(t) \in \mathbb{R}$ is the input variable and $f(x,u)$ is a real analytic vector function defined on $\mathbb{R}^n \times \mathbb{R}$. Without loss of generality, let us assume that the origin $x^0 = 0$ is an equilibrium point of (1) that corresponds to $u^0 = 0$: $f(0,0) = 0$. If a non-zero equilibrium state $(x^0, u^0) \neq (0,0)$ is considered, then a simple transformation of variables: $\hat{x} = x - x^0$, $\hat{u} = u - u^0$ will map it onto the origin in the new coordinates. Moreover, let J be the Jacobian matrix of $f(x,u)$ evaluated at the equilibrium point $(x^0, u^0) = (0,0)$: $J = \dfrac{\partial f}{\partial x}(0,0)$, and G a non-zero vector: $G = \dfrac{\partial f}{\partial u}(0,0) \neq 0$.

We now seek the attainment of the feedback linearization and pole-placement objectives in a single-step. In particular, a transformation map: $z = T(x)$, $T : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ and a state feedback control law: $u = -cz = -cT(x)$, with c an n-dimensional constant row vector are sought, that induce linear dynamics with prescribed modes/poles in the new coordinates:

$$z(t+1) = Az(t), \tag{2}$$

where the matrix A represents a "design adjustable parameter" whose eigenvalues are placed at the desirable set of dynamic modes/poles. The existence of such a nonlinear feedback linearizing control law is guaranteed by the following Theorem [21]:

Theorem 2.1. *Consider the nonlinear discrete-time system* (1) *and the associated system of nonlinear functional equations (NFEs)* (3)

$$\begin{aligned} T(f(x, -cT(x))) &= AT(x) \\ T(0) &= 0 \end{aligned} \tag{3}$$

The following assumptions are made:

Assumption 1. The $(n \times n)$ matrix $\mathcal{C}$:

$$\mathcal{C} = \left[G | JG | ... | J^{n-1}G \right] \tag{4}$$

has rank n: rank $(\mathcal{C}) = n$ (local controllability rank condition).

Assumption 2. The eigenspectrum $\sigma(A)$ of matrix A is comprised of eigenvalues: $k_i \in \sigma(A)$, $i = 1, ...n$ that all lie inside the unit disc on the complex plane (Poincaré domain).

Assumption 3. The eigenspectra $\sigma(A), \sigma(J)$ of matrices A and J respectively are disjoint: $\sigma(A) \cap \sigma(J) = \emptyset$.

Assumption 4. The eigenvalues k_i of A are not related to the eigenvalues λ_j of the Jacobian matrix J through any equations of the type:

$$\prod_{i=1}^{n} k_i^{m_i} = \lambda_j \tag{5}$$

$(j = 1, ..., n)$, where all the m_i's are non-negative integers that satisfy the condition:

$$\sum_{i=1}^{n} m_i > 0 \tag{6}$$

Assumption 5. The pair of matrices (c, A) is chosen such that the following matrix O:

$$O = \begin{bmatrix} c \\ cA \\ . \\ . \\ cA^{n-1} \end{bmatrix} \tag{7}$$

has rank n: rank $(O) = n$ (observability rank condition on the (c, A) pair).

Then, the associated system of NFEs (3) with initial condition $T(0) = 0$, admits a unique and locally invertible analytic solution $T(x)$ in a neighborhood of the origin $x = 0$. Furthermore, the simultaneous implementation of the nonlinear coordinate transformation: $z = T(x)$ and the state feedback control law: $u = -cz = -cT(x)$ induces the linear closed-loop dynamics:

$$z(t+1) = Az(t), \tag{8}$$

whose poles coincide with the eigenvalues of the matrix A.

Please notice, that the initial condition $T(0) = 0$ that accompanies the above system of nonlinear functional equations, reflects the fact that under the proposed coordinate transformation, equilibrium properties are preserved. It is worth noting that in the original coordinates, the state feedback law: $u = -cT(x)$ regulates the states of system (1) at their nominal equilibrium values due to the invertibility of the map $T(x)$ and the fact that the entire eigenspectrum of the matrix A lies entirely within the unit disc on the complex domain due to Assumption 2 (thus ensuring local asymptotic stability in the Lyapunov sense). Furthermore, the choice of the eigenspectrum and eigenspace of matrix A induces the desirable dynamic modes and characteristics for the controlled system under the above state feedback law. Finally, due to the fact that matrix A is "adjustable", the set of assumptions of Theorem 2.1 does not introduce any essential restrictions in the implementation of the proposed method. The linearization of (1) around the equilibrium $(0, 0)$ gives:

$$dx(t+1) = \frac{\partial f}{\partial x}(0,0)dx(t) + \frac{\partial f}{\partial u}(0,0)du(t), \tag{9}$$

while the feedback control law in (9) around the equilibrium $(0, 0)$ is given by:

$$du(t) = -c\frac{\partial T}{\partial x}(0)dx(t). \tag{10}$$

Then, (9) can be written as follows:

$$dx(t+1) = \frac{\partial f}{\partial x}(0,0)dx(t) - \frac{\partial f}{\partial u}(0,0)c\frac{\partial T}{\partial x}(0)dx(t) = \Big(\frac{\partial f}{\partial x}(0,0) - \frac{\partial f}{\partial u}(0,0)c\frac{\partial T}{\partial x}(0)\Big)dx(t). \tag{11}$$

Thus, the linearization of the transformed system (2) around the equilibrium reads:

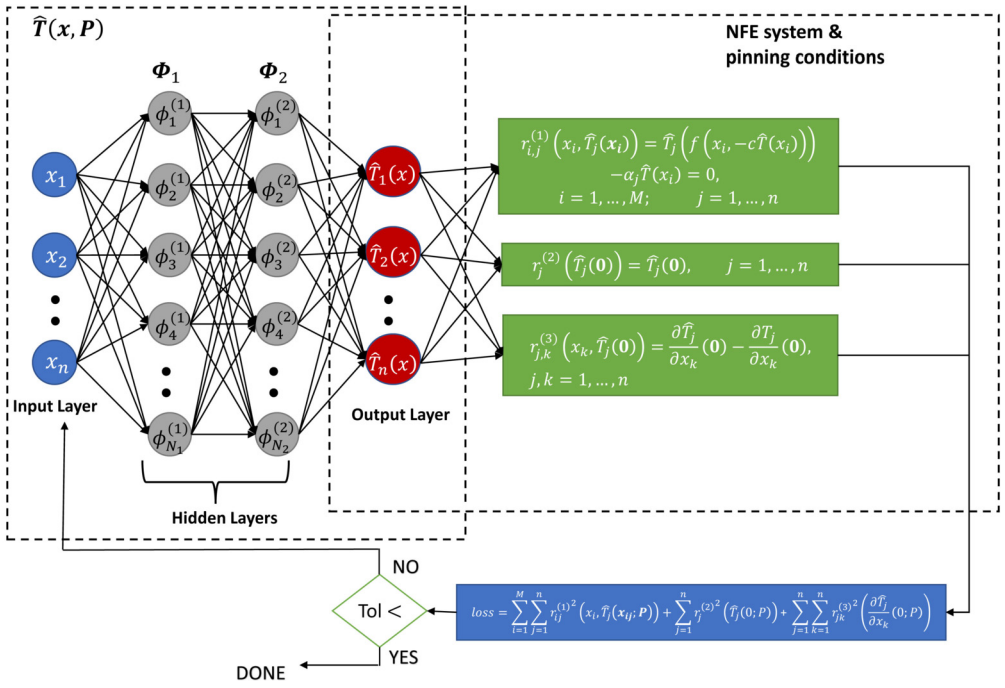

Fig. 1. A schematic of the PIML scheme for the feedback linearization of discrete-time systems.

$$\frac{\partial T}{\partial x}(0)dx(t+1) = A\frac{\partial T}{\partial x}(0)dx(t). \tag{12}$$

Multiplying both sides of Eq. (11) by $\frac{\partial T}{\partial x}(0)$, one obtains:

$$\frac{\partial T}{\partial x}(0)dx(t+1) = \left(\frac{\partial T}{\partial x}(0)\frac{\partial f}{\partial x}(0,0) - \frac{\partial T}{\partial x}(0)\frac{\partial f}{\partial u}(0,0)c\frac{\partial T}{\partial x}(0)\right)dx(t). \tag{13}$$

Hence, from (12), (13), it is inferred that the nonlinear transformation around the equilibrium (x_0, u_0) has to satisfy the following matrix equation which serves as a phase condition:

$$\frac{\partial T}{\partial x}(0)\frac{\partial f}{\partial x}(0,0) - A\frac{\partial T}{\partial x}(0) = \frac{\partial T}{\partial x}(0)\frac{\partial f}{\partial u}(0,0)c\frac{\partial T}{\partial x}(0). \tag{14}$$

If f is explicitly known, the elements of the Jacobian matrix $\frac{\partial T}{\partial x}(0)$ can be calculated analytically.

Here, for learning an approximation of $T(x)$, say $\hat{T}(x)$, we have used an ANN with two hidden layers and N_1, N_2 neurons for the first and second hidden layers respectively, as well as a linear output layer, leading to the following equation:

$$\hat{T}_j(x) = \sum_{l=1}^{N_2} W^o_{jl}\phi^{(2)}_l\left(\sum_{s=1}^{N_1} W^{(2)}_{sl}\phi^{(1)}_s\left(\sum_{k=1}^{n} W^{(1)}_{ks}x_k + \beta^{(1)}_s\right) + \beta^{(2)}_l\right) + \beta^{(o)}_j, \tag{15}$$

where $j = 1, 2, \ldots n$, or equivalently, in matrix form:

$$\hat{T}(x) = W^{(o)^T}\Phi_2(W^{(2)^T}\Phi_1(W^{(1)^T}x + \beta^{(1)}) + \beta^{(2)}) + \beta^{(o)}. \tag{16}$$

$W^{(o)}$ is the $N_2 \times n$ matrix containing the weights $W^{(o)}_{jl}$ connecting the second hidden layer to the linear output layer, $\Phi_1 : \mathbb{R}^n \to \mathbb{R}^{N_1}, \Phi_2 : \mathbb{R}^{N_1} \to \mathbb{R}^{N_2}$ denote multivariate vector-valued functions (maps) with components corresponding to activation functions $\phi^{(1)}_s$ and $\phi^{(2)}_l$ of the first and second hidden layers respectively, $W^{(1)}$ is the $n \times N_1$ matrix containing the weights $W^{(1)}_{ks}$ from the input to the first hidden layer, $W^{(2)}$ is the $N_1 \times N_2$ matrix with the weights $W^{(2)}_{sl}$ connecting the first hidden to the second hidden layer, $\beta^{(1)} \in \mathbb{R}^{N_1}$, $\beta^{(2)} \in \mathbb{R}^{N_2}$ are the column vectors containing the biases $\beta^{(1)}_s$ and $\beta^{(2)}_l$ of the nodes in the first and second layers, respectively, and $\beta^{(o)} \in \mathbb{R}$ is the column vector containing the biases $\beta^{(o)}_j$ of the output nodes. As it has been demonstrated by Chen and Chen [50], such a structure (with sufficient neurons) can approximate, to any accuracy, non-linear laws for the time evolution of dynamical systems).

Here, for learning the transformation $T(x)$ (for a schematic see also Fig. 1, we considered a certain domain $D \subset \mathbb{R}^n$ around the equilibrium point $(0, 0)$. For training purposes, we discretize the domain in a grid of M points $x_i \in D$, with

$i = 1, \ldots, M$, while for testing purposes, after the learning process, we create in D another, denser grid whose points do not coincide with any other point of the grid used for training (more details are given in Section 4: Numerical results). Therefore, finding $T(x)$ reduces to the task of minimizing the loss function:

$$\mathcal{L}(P) = \sum_{i=1}^{M}\sum_{j=1}^{n} {r_{ij}^{(1)}}^2(x_i, \hat{T}(x_i; P)) + \sum_{j=1}^{n} {r_j^{(2)}}^2(\hat{T}_j(0; P)) + \sum_{j=1}^{n}\sum_{k=1}^{n} {r_{jk}^{(3)}}^2(\frac{\partial \hat{T}_j}{\partial x_k}(0; P)), \tag{17}$$

with respect to the unknown parameters $P = (W^{(o)}, W^{(2)}, W^{(1)}, \beta^{(o)}, \beta^{(2)}, \beta^{(1)})$ of the FNN given by (16). In the above:

$$r_{ij}^{(1)}(x_i, \hat{T}(x_i)) = \hat{T}_j\Big(f(x_i, -c\hat{T}(x_i))\Big) - \alpha_j \hat{T}(x_i), \qquad i = 1, \ldots, M, \quad j = 1, 2, \ldots, n, \tag{18}$$

where α_j is the j-th row of the matrix A and $\hat{T}_j$ is j-th output component of $\hat{T}$, and:

$$r_j^{(2)}(\hat{T}_j(0)) = \hat{T}_j(0), \quad r_{jk}^{(3)}(\hat{T}_j(0)) = \frac{\partial \hat{T}_j}{\partial x_k}(0) - \frac{\partial T_j}{\partial x_k}(0), \quad j, k = 1, 2, \ldots, n, \tag{19}$$

where $\frac{\partial T_j}{\partial x_k}(0)$ is the (j, k)-th element of the Jacobian matrix of $T(x)$ computed at the equilibrium, obtained by solving the system of equations in (14). Notice that for illustration purposes, in the loss function, we consider all three terms equally weighted.

Assuming that the objective function in (17) is smooth enough, we may apply an optimization method to solve the least squares problem using (at least) first-order derivatives. To this aim, here we also provide analytically the derivatives with respect to x and the parameters P of the ANN, i.e., $\frac{\partial \hat{T}_j}{\partial x_k}$, $\frac{\partial \hat{T}_j}{\partial W_{jl}^{(o)}}$, $\frac{\partial \hat{T}_j}{\partial W_{sl}^{(2)}}$, $\frac{\partial \hat{T}_j}{\partial W_{ks}^{(1)}}$. Note, that these quantities in TensorFlow but also in Matlab can be computed using automatic differentiation, or numerically, using, e.g., centered finite differences, when only a black-box simulator is available.

In particular, the analytical derivative w.r.t. the k-th component of $x = (x_1, x_2, \ldots, x_k, \ldots, x_n)$ of the first layer of activation function is given by:

$$\frac{\partial}{\partial x_k}\phi_s^{(1)}\Big(\sum_{k=1}^{n} W_{ks}^{(1)} x_k + \beta_s\Big) = W_{ks}^{(1)}\phi_s^{(1)'}\Big(\sum_{k=1}^{n} W_{ks}^{(1)} x_k + \beta_s\Big), \tag{20}$$

where $\phi_s^{(1)'}$ is the first derivative of the activation function. Thus, Eq. (20) in matrix form reads

$$\frac{\partial \Phi_1(W^{(1)T}x + \beta^{(1)})}{\partial x_k} = {W_k}^{(1)T} \odot \Phi_1'(W^{(1)T}x + \beta^{(1)}), \tag{21}$$

where Φ_1' is the column vector with the values of the corresponding derivative of the activation functions $\phi_s^{(1)'}$ of the first layer and $W_k^{(1)}$ is the k-th row of $W^{(1)}$. Then, the derivative of the composition of two consecutive layers activation functions is given by:

$$\begin{aligned}&\frac{\partial}{\partial x_k}\phi_l^{(2)}\Big(\sum_{s=1}^{N_1} W_{sl}^{(2)}\phi_s^{(1)}\Big(\sum_{h=1}^{n} W_{ks}^{(1)} x_k + \beta_s^{(1)}\Big) + \beta_l^{(2)}\Big) = \\ &\phi_l^{(2)'}\Big(\sum_{s=1}^{N_1} W_{sl}^{(2)}\phi_s^{(1)}\Big(\sum_{k=1}^{n} W_{ks}^{(1)} x_k + \beta_s^{(1)}\Big) + \beta_l^{(2)}\Big)\Big(\sum_{s=1}^{N_1} W_{sl}^{(2)} W_{ks}^{(1)}\phi_s^{(1)'}\Big(\sum_{k=1}^{n} W_{ks}^{(1)} x_k + \beta_s^{(1)}\Big)\Big),\end{aligned} \tag{22}$$

where $\phi_l^{(2)'}$ denotes the first derivative of activation functions of the second hidden layer. Thus, Eq. (22) in matrix form reads:

$$\begin{aligned}&\frac{\partial \Phi_2(W^{(2)T}\Phi_1(W^{(1)T}x + \beta^{(1)}) + \beta^{(2)})}{\partial x_k} = \\ &= \Phi_2'(W^{(2)T}\Phi_1(W^{(1)T}x + \beta^{(1)}) + \beta^{(2)}) \odot \Big(W^{(2)T} \cdot \big({W_k^{(1)}}^T \odot \Phi_1'(W^{(1)T}x + \beta^{(1)})\big)\Big),\end{aligned} \tag{23}$$

where Φ_2' is the column vector with the values of the corresponding derivative of the activation functions $\phi_l^{(2)'}$ of the second hidden layer, $W_k^{(1)}$ is the k-th row of $W^{(1)}$ and the symbol $\odot$ denotes the Hadamard product (element-wise product). Finally, in order to compute the Jacobian matrix $\frac{\partial \tilde{T}}{\partial x}$, let's consider the j-th component, say $\hat{T}_j$, of the transformation $\tilde{T} = (\hat{T}_1, \hat{T}_2, \ldots, \hat{T}_j, \ldots, \hat{T}_n)$, so that the element (j, k) of the Jacobian matrix is given by:

$$\frac{\partial}{\partial x_k}\hat{T}_j(x) = \sum_{l=1}^{N_2} W^o_{lj}\phi_l^{(2)'}\left(\sum_{s=1}^{N_1} W^{(2)}_{sl}\phi_s^{(1)}\left(\sum_{k=1}^{n} W^{(1)}_{ks}x_k + \beta_s^{(1)}\right) + \beta_l^{(2)}\right)\left(\sum_{s=1}^{N_1} W^{(2)}_{sl}W^{(1)}_{ks}\phi_s^{(1)'}\left(\sum_{k=1}^{n} W^{(1)}_{ks}x_k + \beta_s^{(1)}\right)\right), \tag{24}$$

and equivalently, in matrix form is expressed as follows:

$$\frac{\partial \hat{T}_j}{\partial x_k} = W^{j(o)T}\cdot\left(\Phi_2' W^{(2)T}\Phi_1\left(W^{(1)T}x + \beta^{(1)}\right) + \beta^{(2)}\right)\odot\left(W^{(2)T}\cdot\left(W_k^{(1)T}\odot\Phi_1'\left(W^{(1)T}x + \beta^{(1)}\right)\right)\right), \tag{25}$$

where $W^{j(o)}$ is the j-th column of the matrix $W^{(o)}$.

The derivative of the loss function w.r.t. an unknown parameter, say, $p \in P$ is calculated as follows:

$$\frac{\partial \mathcal{L}(P)}{\partial p} = \sum_{i=1}^{M}\sum_{j=1}^{n} r_{ij}^{(1)}\frac{\partial r_{ij}^{(1)}}{\partial p} + \sum_{j=1}^{n} r_j^{(2)}\frac{\partial r_j^{(2)}}{\partial p} + \sum_{j=1}^{n}\sum_{k=1}^{n} r_{jk}^{(3)}\frac{\partial r_{jk}^{(3)}}{\partial p}. \tag{26}$$

Hence, for the three residuals $r^{(i)}, i = 1, 2, 3$, we have:

$$\frac{\partial r_{ij}^{(1)}}{\partial p} = \frac{\partial \hat{T}_j\left(f(x_i, -c^T\hat{T}(x_i))\right)}{\partial p}\frac{\partial f(x_i, -c^T\hat{T}(x_i))}{\partial u}\cdot\left(-c^T\frac{\partial \hat{T}(x_i)}{\partial p}\right) - {\alpha_j}^T\frac{\partial \hat{T}(x_i)}{\partial p} \tag{27}$$

$$\frac{\partial r_j^{(2)}}{\partial p} = \frac{\partial \hat{T}_j(x_0)}{\partial p}, \tag{28}$$

$$\frac{\partial r_{jk}^{(3)}(x_k, \hat{T}_j(x_0))}{\partial p} = \frac{\partial^2 \hat{T}_j}{\partial p \partial x_k}(x_0) \tag{29}$$

$$i = 1, \ldots, M, \quad j, k = 1, \ldots, n, \quad p \in P \tag{30}$$

In what follows, we compute the derivatives w.r.t. the weights and biases of the ANN. In particular, one obtains:

- for $p = W_{lq}^{(o)}$:

$$\frac{\partial \hat{T}_j(x)}{\partial W_{lq}^{(o)}} = \left\{\begin{array}{ccc} \phi_l^{(2)}\left(\sum_{s=1}^{N_1} W_{sl}^{(2)}\phi_s^{(1)}\left(\sum_{k=1}^{n} W_{ks}^{(1)}x_k + \beta_s^{(1)}\right) + \beta_l^{(2)}\right) & if & q = j \\ 0 & if & q \neq j \end{array}\right\} \tag{31}$$

- for $p = W_{sl}^{(2)}$

$$\frac{\partial \hat{T}_j(x)}{\partial W_{sl}^{(2)}} = W^o_{lj}\phi_l^{(2)'}\left(\sum_{s=1}^{N_1} W_{sl}^{(2)}\phi_s^{(1)}\left(\sum_{k=1}^{n} W_{ks}^{(1)}x_k + \beta_s^{(1)}\right) + \beta_l^{(2)}\right)\phi_s^{(1)}\left(\sum_{k=1}^{n} W_{ks}^{(1)}x_k + \beta_s^{(1)}\right) \tag{32}$$

- for $p = W_{ks}^{(1)}$

$$\frac{\partial \hat{T}_j(x)}{\partial W_{ks}^{(1)}} = \sum_{l=1}^{N_2} W^o_{lj}\phi_l^{(2)'}\left(\sum_{s=1}^{N_1} W_{sl}^{(2)}\phi_s^{(1)}\left(\sum_{k=1}^{n} W_{ks}^{(1)}x_k + \beta_s^{(1)}\right) + \beta_l^{(2)}\right)\left(W_{sl}^{(2)}x_k\phi_s^{(1)'}\left(\sum_{k=1}^{n} W_{ks}^{(1)}x_k + \beta_s^{(1)}\right)\right) \tag{33}$$

- for $p = \beta_h^{(o)}$

$$\frac{\partial \hat{T}_j(x)}{\partial \beta_h^{(o)}} = \left\{\begin{array}{ccc} 1 & if & h = j \\ 0 & if & h \neq j \end{array}\right\} \tag{34}$$

- for $p = \beta_l^{(2)}$

$$\frac{\partial \hat{T}_j(x)}{\partial \beta_l^{(2)}} = W^o_{lj}\phi_l^{(2)}\left(\sum_{s=1}^{N_1} W_{sl}^{(2)}\phi_s^{(1)}\left(\sum_{k=1}^{n} W_{ks}^{(1)}x_k + \beta_s^{(1)}\right) + \beta_l^{(2)}\right) \tag{35}$$

- for $p = \beta_s^{(1)}$

$$\frac{\partial \hat{T}_j(x)}{\partial \beta_s^{(1)}} = \sum_{l=1}^{N_2} W^o_{lj}\phi_l^{(2)'}\left(\sum_{s=1}^{N_1} W_{sl}^{(2)}\phi_s^{(1)}\left(\sum_{k=1}^{n} W_{ks}^{(1)}x_k + \beta_s^{(1)}\right) + \beta_l^{(2)}\right)\left(W_{sl}^{(2)}\phi_s^{(1)}\left(\sum_{k=1}^{n} W_{ks}^{(1)}x_k + \beta_s^{(1)}\right) + \beta_l^{(2)}\right). \tag{36}$$

2.1. The greedy procedure for the learnable parameters

An important problem is the potential existence of sharp gradients that resemble singularities within the domain D of interest. In such cases, to ensure efficient convergence and an adequate numerical approximation during the training process, the utilization of a continuation (homotopy) method becomes necessary for the PIML learning procedure. Here, to deal with this problem, we solve with PIML the system of Eq. (3) progressively using a *nested family* of say, m subdomains:

$$D_1 \subset D_2 \subset \cdots \subset D_k \subset \cdots \subset D_m = D; \tag{37}$$

the sizes of the subdomains are set so that adequate accuracy is achieved. The first subdomain D_1 is realized in a neighborhood around the equilibrium. Then, upon convergence of the PIML scheme in D_1, the attained weights P_1 are used as the initial guess $P_2^{(0)}$ for the solution of Eq. (3) in the subsequent subdomain D_2 and so on, thus setting $P_k^{(0)} = P_{k-1}$. This procedure corresponds to a zero-order continuation method for the learnable parameters. A comprehensive presentation of the specific details of the implementation of the greedy procedure employed for our benchmark problem can be found within Section 4. For a first-order continuation method for RPNNs used to solve stiff differential equations, please see [51].

3. The benchmark problem

To evaluate the performance of the methods proposed in this work, we have considered the following nonlinear discrete-time system [21]:

$$\begin{aligned} x_1(t+1) &= exp(0.3x_2(t))\sqrt{(1+x_1(t)+x_2(t))} - 1 - 0.4x_2(t) + 0.5u(t) \\ x_2(t+1) &= 0.5\ln(1+x_1(t)+x_2(t)) + 0.4x_2(t) \end{aligned} \tag{38}$$

The Jacobian matrix of the above system at the equilibrium $(0,0)$ is $\frac{\partial f}{\partial x}(0,0) = \begin{bmatrix} 0.5 & 0.4 \\ 0.5 & 0.9 \end{bmatrix}$, and its eigenvalues $\lambda_1 = 0.2101$ and $\lambda_2 = 1.1899$. The matrix A is chosen to be $A = \begin{bmatrix} 0.5 & 0.3 \\ 0.5 & 0.4 \end{bmatrix}$, with eigenvalues $k_1 = 0.8405$ and $k_2 = 0.0595$. Due to the choice of matrix A, its eigenvalues are not related to the eigenvalues of the Jacobian matrix $\frac{\partial f}{\partial x}(0,0)$ through any equations of the type (5), (6). Moreover, the following row vector c was chosen:

$$c = \begin{bmatrix} 1 & 0 \end{bmatrix}. \tag{39}$$

Please notice, that all conditions of Theorem 2.1 are met by the system (38), and therefore the associated system of NFEs (3):

$$\begin{aligned} &T_1(exp(0.3x_2(t))\sqrt{(1+x_1(t)+x_2(t))} - 1 - 0.4x_2(t) - 0.5T_1, \\ &0.5\ln(1+x_1(t)+x_2(t)) + 0.4x_2(t)) = 0.5T_1 + 0.3T_2 \\ &T_2(exp(0.3x_2(t))\sqrt{(1+x_1(t)+x_2(t))} - 1 - 0.4x_2(t) - 0.5T_1, \\ &0.5\ln(1+x_1(t)+x_2(t)) + 0.4x_2(t)) = 0.5T_1 + 0.4T_2 \\ &T_1(0,0) = 0 \\ &T_2(0,0) = 0, \end{aligned} \tag{40}$$

admits a unique locally analytic and invertible solution around the equilibrium point $(x_1, x_2) = (0,0)$. Indeed, please notice that

$$\frac{\partial T}{\partial x}(0,0) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \tag{41}$$

with a $det[T] \neq 0$, results in a locally invertible around the equilibrium point $(x_1, x_2) = (0,0)$ solution, that can be also calculated analytically in closed-form [21]:

$$T_1(x_1, x_2) = \ln(1 + x_1 + x_2), \quad T_2(x_1, x_2) = x_2. \tag{42}$$

In agreement with Theorem 2.1, the proposed feedback-linearizing and pole-placing nonlinear feedback control law can be explicitly written as follows:

$$u = -cT(x) = -\ln(1 + x_1 + x_2). \tag{43}$$

Our benchmark problem encompasses a set of singularities of the nonlinear transformation when $x_1 + x_2 = -1$. Here, we sought to learn the feedback-linearizing control law in the domain $[x_L, 0] \times [x_L, 0] = [-0.495, 0] \times [-0.495, 0]$.

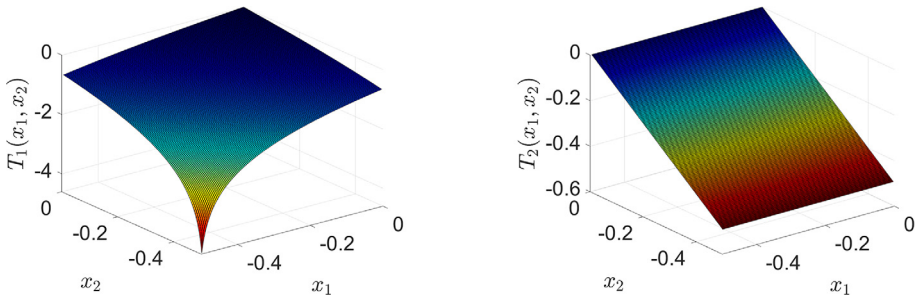

Fig. 2. Analytical solution of the NFEs (40) in $[-0.495, 0] \times [-0.495, 0]$. (a) $T_1(x_1, x_2) = \ln(1 + x_1 + x_2)$. A steep-gradient at $(-0.495, -0.495)$ is due to the presence of a singular point at $(x_1, x_2) = (-0.5, -0.5)$. (b) $T_2(x_1, x_2) = x_2$.

Fig. 2 depicts the analytical solutions $T_1(x_1, x_2), T_2(x_1, x_2)$, of the associated system of NFEs. We note that it encompasses the solution domain, which has been deliberately chosen to be $x_1, x_2 \in [-0.495, 0]$ since $T_1(x_1, x_2)$ exhibits a singular point in the particular domain at $(x_1, x_2) = (-0.5, -0.5)$. The first reasonable step is, therefore, to verify by comparing with the analytical solution, the numerical approximation accuracy of the proposed PIML scheme in this region, and assess its impact on the performance profile of the resulting feedback-linearizing control law.

4. Numerical results

We present the numerical results in two subsections. In subsection 4.1, we provide numerical results for the case when the discrete nonlinear model (1) is assumed to be explicitly available. In this case, one can compute the necessary derivatives required in the optimization algorithm analytically/symbolically as above, or by exploiting the automatic differentiation toolkit. Within the greedy PIML framework, we implemented two schemes: (a) the Keras API of TensorFlow (TF) library running in Python using the BFGS optimization algorithm; this scheme uses by default automatic differentiation to compute the necessary derivatives, and, (b) a home-made code in Matlab, using the Levenberg-Marquardt algorithm for training, where in order to be consistent with the Keras API TF implementation, the necessary derivatives are also computed via AD using the function *dlgradient* of Matlab. To assess the performance of the greedy approach, we also show the results obtained with Matlab and TensorFlow, when the PIML was trained in the entire domain at once. In subsection 4.2, we present the results assuming that the discrete nonlinear model (1) is not explicitly available, but we have access to a black-box simulator. In this case, the necessary derivatives are estimated numerically using centered finite differences.

Training set For training purposes using the greedy approach, we considered 20 equispaced distributed collocation points for each one of the two inputs x_1 and x_2, i.e. we used a grid of 20×20 points equispaced distributed for each step of the greedy approach. Different (denser) sizes of the grid (e.g. using a grid of 100×100 points) with more neurons for each layer did not affect qualitatively the results and corresponding performances, as we also show. Thus, for the greedy-approach, we started by considering a grid in $[-0.2, 0] \times [-0.2, 0]$. Then with a step of -0.05, we used as initial guesses of the unknown weights and biases of the PIML the ones obtained from the training procedure from the previous grid, and repeated training until the interval $[-0.45, 0] \times [-0.45, 0]$ was reached. From this interval and on, we optimized iteratively the PIML using progressively bigger intervals with a step of -0.01, for each of the two inputs x_1 and x_2, up to the interval $[-0.49, 0] \times [-0.49, 0]$. After this interval we augmented progressively the grid with steps of -0.001 until the entire domain of interest, $[-0.495, 0] \times [-0.495, 0]$, was reached. As mentioned before, additionally, we have also used the TensorFlow library to learn the transformation law both with the greedy approach and in the entire domain $[-0.495, 0] \times [-0.495, 0]$. The performances of the PIML schemes were also compared with a $6th$ order power-series expansion of $T_1(x_1, x_2), T_2(x_1, x_2)$, thus resulting in equivalent, to the PIML, number of unknowns.

Test set For testing purposes, we used the roots of the Chebyshev polynomial of the first kind of degree k. For our illustrations, we selected a grid of 50×50 Chebyshev points for each of the intervals in $[-0.495, 0] \times [-0.495, 0]$. In particular, in the interval $[a, b]$ with $a, b \in \mathbb{R}$, the grid was created using the following Chebyshev collocation points:

$$x_n = \frac{1}{2}(a+b) - \frac{1}{2}(a-b)cos\left(\frac{2n-1}{2n}\pi\right), \quad n = 1, \cdots, k \tag{44}$$

with $a = -0.495$ and $b = 0$, and therefore equation (44) can now be expressed as follows:

$$x_n = \frac{1}{2}(-0.495) - \frac{1}{2}(-0.495)cos\left(\frac{2n-1}{2n}\pi\right), \quad n = 1, \cdots, k \tag{45}$$

Table 1
Model explicitly available. Training sets (grids of 20×20 equispaced distributed points). Error norms (L_1, L_2 and L_∞) between the analytical and computed solution of $T_1(x_1, x_2)$ and $T_2(x_1, x_2)$ using the various schemes trained both greedy and in the entire domain $[-0.495, 0] \times [-0.495, 0]$.

Transformation $T(x)$	Error norm	power-series $6th$ order	PIML (TF) Entire domain	PIML (Matlab) Greedy	PIML (TF) Greedy
$T_1(x_1, x_2)$	$\|\cdot\|_1$	6.76E + 01	6.28E + 00	2.03E − 03	3.65E − 02
	$\|\cdot\|_2$	3.62E + 00	3.73E + 00	1.12E − 03	3.26E − 02
	$\|\cdot\|_\infty$	1.21E + 00	2.81E + 00	1.05E − 03	3.10E − 03
$T_2(x_1, x_2)$	$\|\cdot\|_1$	0	1.40E + 00	6.33E − 03	6.61E − 02
	$\|\cdot\|_2$	0	1.00E + 00	1.40E − 03	3.68E − 02
	$\|\cdot\|_\infty$	0	5.94E − 01	6.73E − 04	1.00E − 02

4.1. The case of the explicitly available model

Power-series solution First, we have expanded both the right-hand-side of the model (1) and the nonlinear transformation $T(x_1, x_2)$ to a $6th$ order power-series, and equated same order terms up to the $6th$ order of $T(f(x, -cT(x)))$ and $AT(x)$ on both sides of the associated NFEs (see Appendix A), thus building a system of algebraic equations that can be solved for the coefficients of the $6th$ order polynomial approximation. For this task, we used Matlab's symbolic toolbox.

PIML-based solution For our PIML scheme, we constructed an ANN with two hidden layers, fully connected, with five neurons in each layer. More specifically, for both the home-made Matlab code and the TensorFlow (TF) implementation, the activation function was chosen to be the $sigmoid(x)$ function. For the training, in the Matlab implementation, we used the function *lsqnonlin*, which implements the Levenberg-Marquart (LM) algorithm. In the LM scheme, we have set as stopping criterion a threshold of FuncTol $= 10^{-12}$. Moreover, we have set a maximum number of 100,000 iterations and a maximum number of function iterations equal to 12,000. Finally, to initialize the weights and biases, we have used uniformly distributed random numbers in the interval $[0, 1]$ by employing the *rand* function of Matlab.

For our computations, we have also used the Keras API of TensorFlow. Thus, we have used automatic differentiation (AD) [52] to find the required derivatives for the optimization process. With the TensorFlow, we have tried different optimizers; namely the Stochastic gradient descent (SGD), the ADAM optimizer and the BFGS optimizer. For the SGD and ADAM optimizers, we have used the default values, except for the learning rate which has been selected using a piecewise constant decay varying from 10^{-2} to 10^{-4} as the number of epochs increased. We have chosen 100,000 epochs since using more epochs did not seem to considerably improve the results derived, whereas for the BFGS we have used the library *tensorflow probability*. Furthermore, we have used a maximum number of iterations equal to 100,000, and a stopping condition of FuncTol $= 10^{-16}$. The best results obtained with the TF were derived with the aid of BFGS, and these are the results presented in our comparative assessment, shown below.

Fig. 3 shows the numerical approximation accuracy (difference between computed and analytical solutions) obtained by the various schemes for the training set. Figs. 3(a),(b) show the results obtained with a $6th$ order power-series expansion of the nonlinear transformation and the right-hand-side of the discrete model. As expected, the power-series expansion results in zero error for the $T_2(x_1, x_2)$ component, and in a good approximation accuracy for the $T_1(x_1, x_2)$ component, but only in the region close to the linearization-relevant point, which in this case is $[0, 0]$, while away from it, the numerical approximation is poor (of the order of 10^0 at the edge of the grid) close to the singular point. Figs. 3(c),(d) depict the results obtained with the PIML implemented in Tensorflow using the BFGS optimizer for learning the non-linear transformation law in the entire domain. The approximation accuracy of the scheme is rather poor, especially near the singularity (of the order of 10^0). Figs. 3(e),(f) depict the results obtained from the PIML implemented with the Tensorflow and the BFGS optimizer using the greedy training procedure. Figs. 3(g),(h) depict the results obtained with the PIML implemented in Matlab using the LM optimizer and the greedy training procedure. It is evident that the greedy training procedure results in significantly enhanced numerical approximation accuracy compared to the PIML schemes trained in the entire domain at once. In particular, the PIML scheme implemented in TensorFlow resulted in a numerical approximation error of the order of 10^{-2} and the home-made Matlab resulted in a numerical approximation error of the order of 10^{-3}, in the region close to the edge point $(-0.495, -0.495)$. In addition, Fig. 4 depicts the performance of the schemes tested on a Chebyshev grid. The results are similar for the training and the test set. Table 1 and Table 2 detail the numerical approximation accuracy of the various schemes for the training and the test sets, respectively.

Taking denser grids and more neurons in each hidden layer, did not change qualitatively the numerical approximation accuracy. Indicatively, in Fig. 5, we depict the numerical approximation accuracy obtained with the PIML implemented in TensorFlow (TD) trained with BFGS in the entire domain $[-0.495, 0] \times [-0.495, 0]$ using 100×100 equispaced points and different number of neurons in each hidden layer. In particular, Figs. 5(a),(b) show the numerical approximation accuracy in the training set using two hidden layers with five neurons in each layer, Figs. 5(c),(d) with ten neurons, and Figs. 5(e),(f) with fifteen neurons in each layer. Finally, Fig. 6, reports the corresponding numerical approximation accuracy plots for the test set (a grid of Chebyshev-distributed points in $[-0.495, 0] \times [-0.495, 0]$).

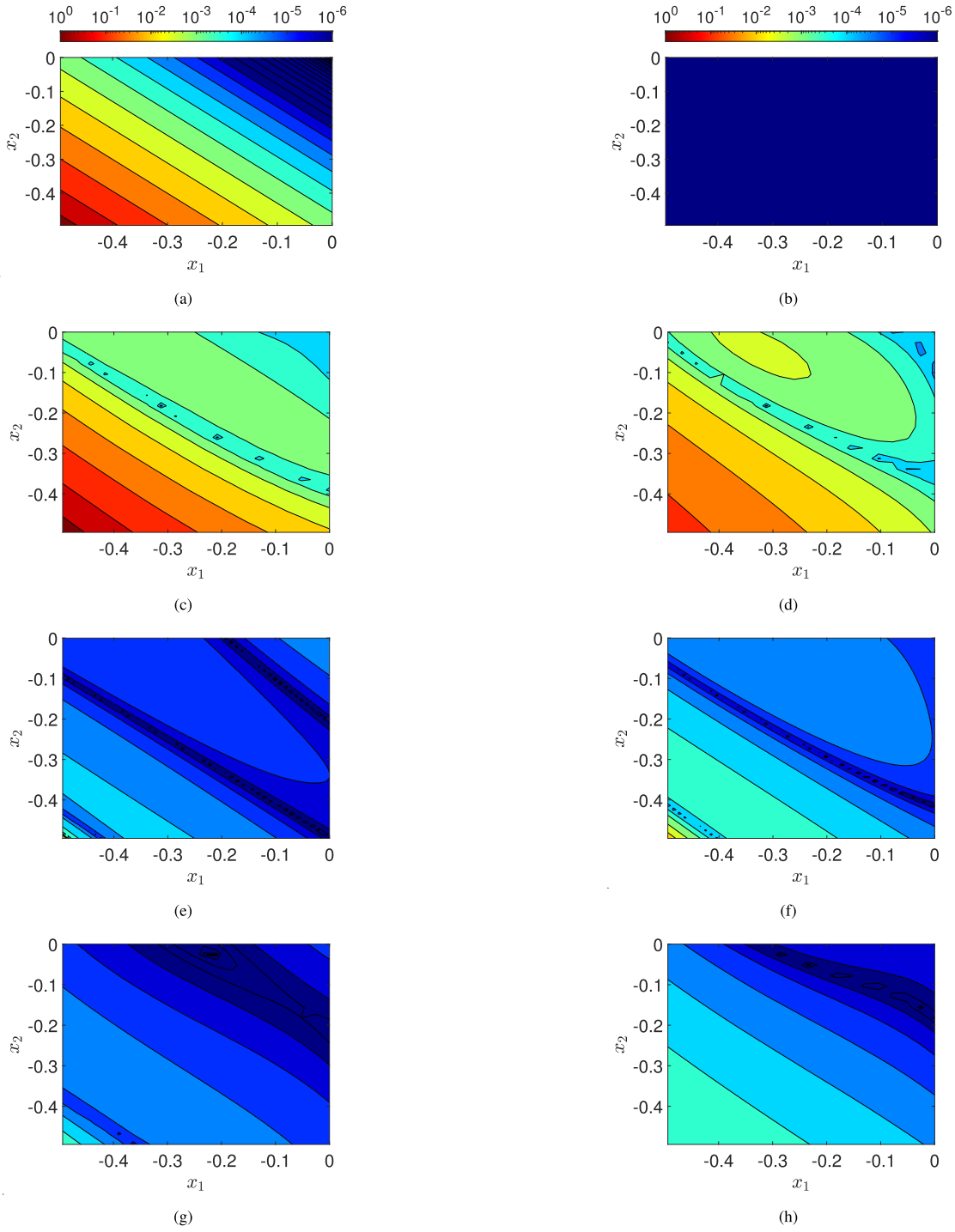

Fig. 3. Model explicitly available. Training sets (grids of 20×20 equispaced distributed points). Numerical approximation accuracy (difference between the computed and analytical solution) of $T_1(x_1, x_2)$ (left column) and $T_2(x_1, x_2)$ (right column) using the various schemes. (a),(b) 6*th* order power-series expansion of $T_1(x_1, x_2)$ and $T_2(x_1, x_2)$ and the right-hand side of the model (38) in $[-0.495, 0] \times [-0.495, 0]$. (c),(d) PIML in Tensorflow trained in the entire domain $[-0.495, 0] \times [-0.495, 0]$. (e),(f) PIML in Tensorflow trained via the greedy approach. (g),(h) PIML in Matlab trained via the greedy approach. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

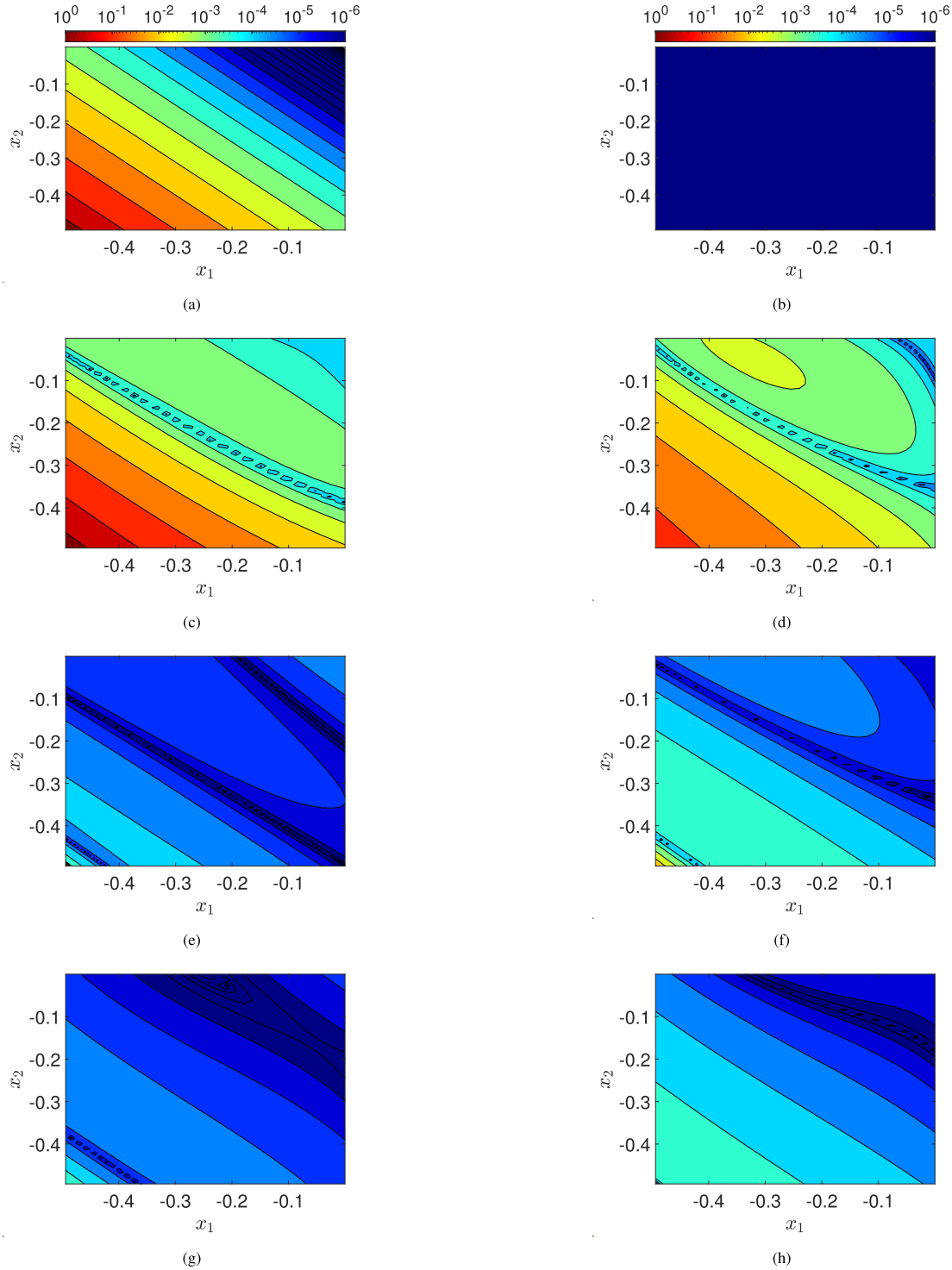

Fig. 4. Model explicitly available. Test sets (grids of 50×50 Chebyshev-distributed points). Numerical approximation accuracy (difference between the computed and analytical solution) of $T_1(x_1, x_2)$ (left column) and $T_2(x_1, x_2)$ (right column) using the various schemes. (a),(b) $6th$ order power-series expansion of $T_1(x_1, x_2)$ and $T_2(x_1, x_2)$ and the right-hand side of the model (38) in $[-0.495, 0] \times [-0.495, 0]$. (c),(d) PIML in Tensorflow trained in the entire domain $[-0.495, 0] \times [-0.495, 0]$. (e),(f) PIML in Tensorflow trained via the greedy approach. (g),(h) PIML in Matlab trained via the greedy approach.

Table 2
Model explicitly available. Test sets (grids of 50×50 Chebyshev-distributed points). Error norms (L_1, L_2 and L_∞) between the analytical and computed solution of $T_1(x_1, x_2)$ and $T_2(x_1, x_2)$ using the various schemes trained both greedy and in the entire domain $[-0.495, 0] \times [-0.495, 0]$.

Transformation $T(x)$	Error norm	power-series 6*th* order	PIML (TF) Entire domain	PIML (Matlab) Greedy	PIML (TF) Greedy
$T_1(x_1, x_2)$	$\|\cdot\|_1$	6.89E + 01	2.17E + 01	3.40E − 02	4.77E − 02
	$\|\cdot\|_2$	4.55E + 00	1.44E + 01	2.63E − 03	4.70E − 02
	$\|\cdot\|_\infty$	2.88E + 00	1.00E + 01	1.41E − 03	2.60E − 02
$T_2(x_1, x_2)$	$\|\cdot\|_1$	0	2.87E + 00	1.45E − 02	1.89E − 01
	$\|\cdot\|_2$	0	1.97E + 00	1.55E − 03	1.84E − 01
	$\|\cdot\|_\infty$	0	1.23E + 00	7.62E − 04	1.28E − 01

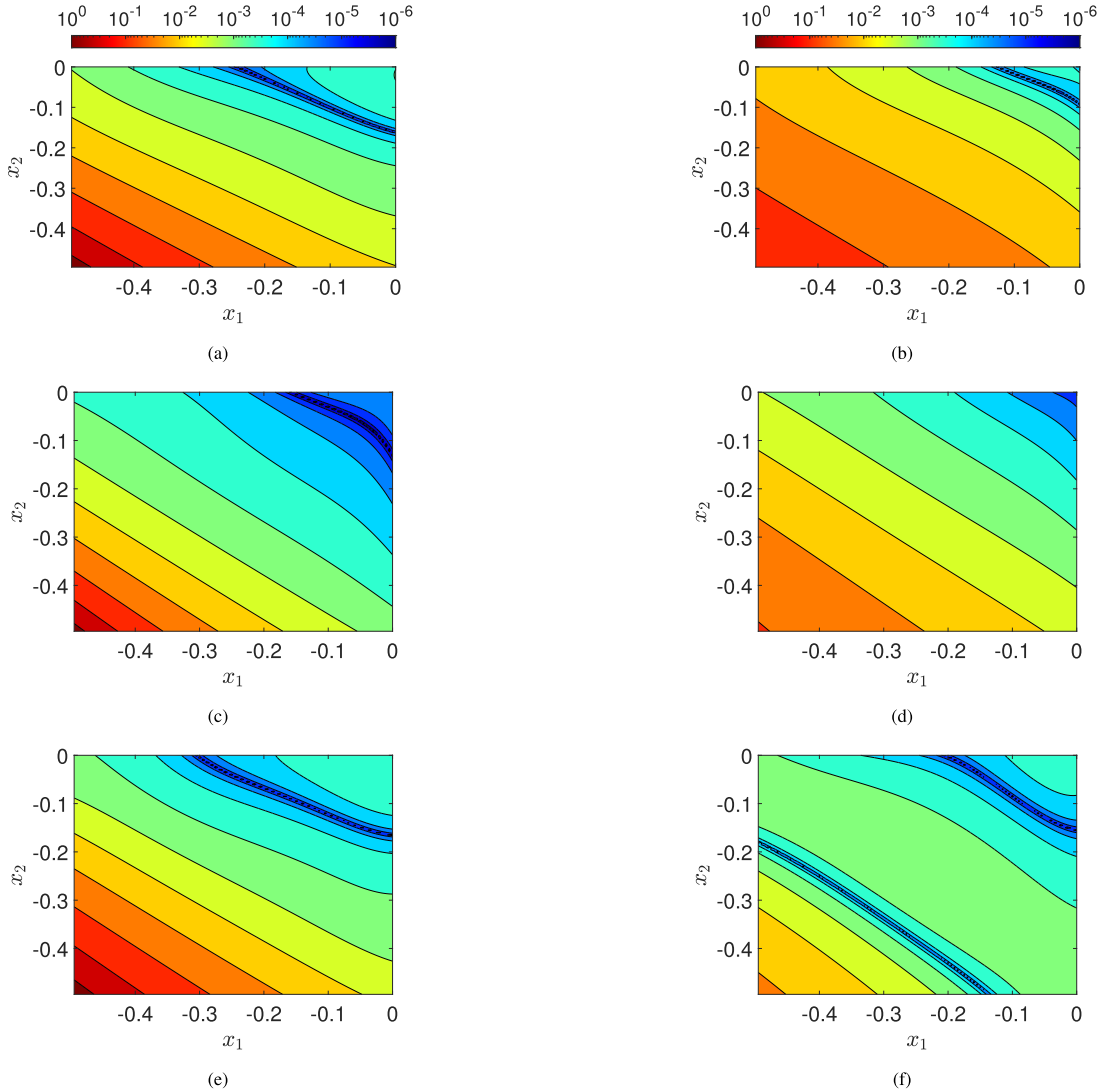

Fig. 5. Model explicitly available. Training set (grid of 100×100 equispaced distributed points in $[-0.495, 0] \times [-0.495, 0]$). Numerical approximation accuracy (difference between the computed and analytical solution) of $T_1(x_1, x_2)$ (left column) and $T_2(x_1, x_2)$ (right column) using the Keras API in TensorFlow; training was performed in the entire domain. (a),(b) PIML, two hidden layers, five neurons in each layer. (c),(d) PIML, two hidden layers, ten neurons in each layer. (e),(f) PIML, two hidden layers, fifteen neurons in each layer.

Finally, in Fig. 7, we depict the numerical approximation error of the transformation map $T_1(x_1, x_2)$ in terms of the L_2 norm with respect to the size of the domain for the four different schemes, namely: (a) a PIML implemented in TensorFlow, trained in the entire domain (blue line), (b) a PIML implemented in Matlab trained with the Levenberg-Marquardt in the

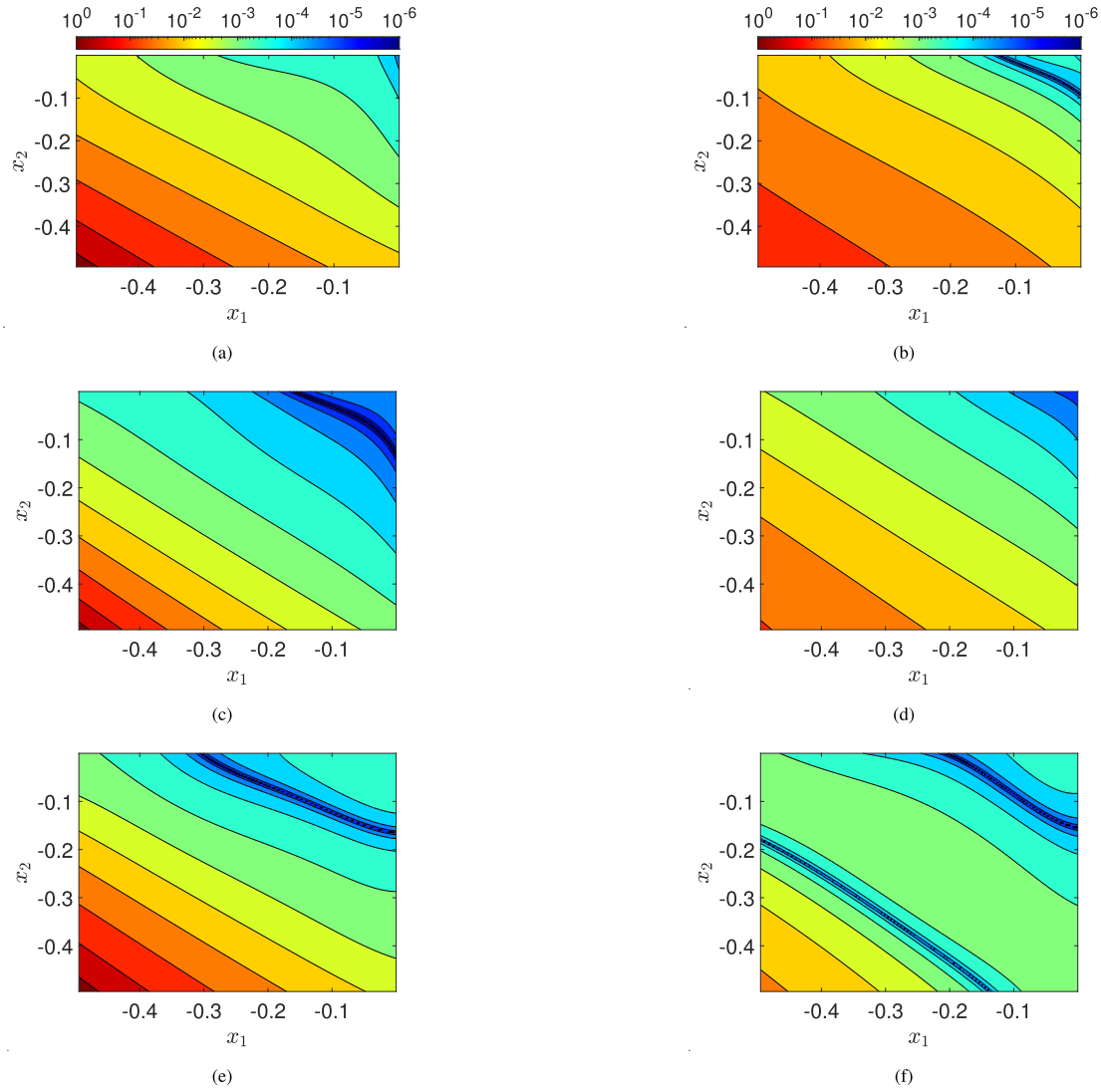

Fig. 6. Model explicitly available. Test set (grid of 150×150 Chebyshev-distributed points in $[-0.495, 0] \times [-0.495, 0]$). Numerical approximation accuracy (difference between the computed and analytical solution) of $T_1(x_1, x_2)$ (left column) and $T_2(x_1, x_2)$ (right column) using the Keras API in TensorFlow; training was performed in the entire domain. (a),(b) PIML, two hidden layers, five neurons in each layer. (c),(d) PIML, two hidden layers, ten neurons in each layer. (e),(f) PIML, two hidden layers, fifteen neurons in each layer.

entire domain (orange line), (c) a PIML implemented in Matlab trained with the Levenberg-Marquardt using the greedy procedure (yellow line), and, (d) a PIML implemented in TensorFlow using the greedy procedure (purple line). For the construction of the diagram, we have used a grid of 20×20 equispaced distributed collocation points. Starting with a -0.2×-0.2 grid and using a step of -0.05 we performed the training process each time until the interval -0.45×-0.45 was reached. Then, for the interval between -0.45×-0.45 and -0.49×-0.49, the step chosen for augmenting the grid were of size -0.01 and finally from this last interval to -0.495×-0.495 the step chosen was of size -0.001.

4.2. The black-box simulator case

In the black-box simulator scheme, the necessary derivatives for the optimization process were estimated using central finite differences with a perturbation step of $eps = 10^{-4}$. Here, for illustration purposes, we have implemented the PIML only on Matlab in a "fully numerical way".

Fig. 8 shows the results for the training sets. As Figs. 8(a),(b) show, in the case of the black-box simulator, the power-series expansion attained a lower/worse numerical approximation accuracy level compared to the one when the model is assumed to be explicitly known (see in Fig. 3(a),(b)). The approximation error is rather poor (of the order of 10^0) in the region close to the singularity, i.e., the point $[-0.495, -0.495]$. Figs. 8(c),(d) depict the PIML scheme as implemented on Matlab trained in the entire domain. Here the approximation error is of the order of 10^{-1} in the region close to the singular

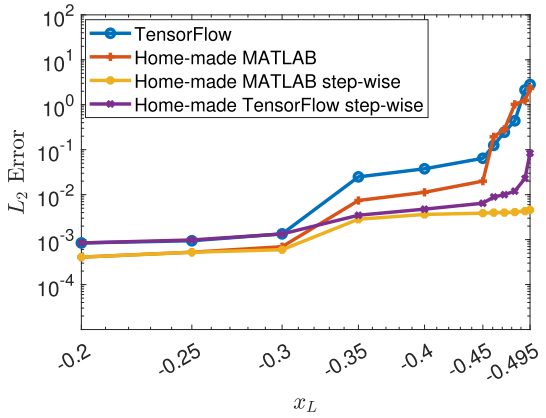

Fig. 7. Model explicitly available. L_2 norm of the numerical approximation accuracy between the analytical transformation law $T_1(x_1, x_2)$ (see Eq. (42)) and the various PIML implementations trained both with the greedy approach and in the entire domain, for various sizes of the domain, $[x_L, 0] \times [x_L, 0]$. For our illustrations, we have used grids of 20×20 equispaced distributed collocation points.

Table 3
Black-box simulator. Training sets (grids of 20×20 equispaced distributed points). Error norms (L_1, L_2 and L_∞) between the analytical and computed solution of $T_1(x_1, x_2)$ and $T_2(x_1, x_2)$ using the various schemes trained both greedy and in the entire domain $[-0.495, 0] \times [-0.495, 0]$.

Transformation $T(x)$	Error norm	power-series 6*th* order	PIML (Matlab) Entire domain	PIML (Matlab) Greedy
$T_1(x_1, x_2)$	$\|\cdot\|_1$	1.50E + 01	1.21E + 00	2.86E − 03
	$\|\cdot\|_2$	9.73E + 00	7.84E − 01	2.78E − 03
	$\|\cdot\|_\infty$	1.27E + 00	1.35E − 01	1.74E − 03
$T_2(x_1, x_2)$	$\|\cdot\|_1$	4.95E − 01	2.36E − 01	7.03E − 03
	$\|\cdot\|_2$	4.09E − 01	2.44E − 01	6.54E − 03
	$\|\cdot\|_\infty$	3.07E − 01	1.10E − 01	4.82E − 03

Table 4
Black-box simulator. Test sets (grids of 50×50 Chebyshev-distributed points). Error norms (L_1, L_2 and L_∞) between the analytical and computed solution of $T_1(x_1, x_2)$ and $T_2(x_1, x_2)$ using the various schemes trained both greedy and in the entire domain $[-0.495, 0] \times [-0.495, 0]$.

Transformation $T(x)$	Error norm	power-series 6*th* order	PIML (Matlab) Entire domain	PIML (Matlab) Greedy
$T_1(x_1, x_2)$	$\|\cdot\|_1$	2.17E + 01	3.54E + 00	7.35E − 03
	$\|\cdot\|_2$	1.84E + 01	2.65E + 00	7.18E − 03
	$\|\cdot\|_\infty$	4.89E + 00	1.81E + 00	4.36E − 03
$T_2(x_1, x_2)$	$\|\cdot\|_1$	1.19E + 00	7.87E − 01	1.77E − 02
	$\|\cdot\|_2$	1.05E + 00	4.84E − 01	1.64E − 02
	$\|\cdot\|_\infty$	9.03E − 01	2.98E − 01	9.18E − 03

point. Figs. 8(e),(f) depict the PIML scheme implemented on Matlab trained with the greedy procedure. The numerical approximation error in the domain where the step gradient appears is of the order of 10^{-3}, thus outperforming the power-series expansion approximation of the transformation law, as well as the PIML trained in the entire domain at once. It should be noted that the numerical approximation error obtained in the region close to the singular point $[-0.495, -0.495]$ is of the same order as that obtained with the PIML implemented when the model is assumed to be explicitly known. Finally, Fig. 9 depicts the numerical approximation errors in the test set. Table 3 and Table 4 detail the numerical approximation accuracy of the various schemes, for the training and test sets, respectively. The results are similar both for the training and test sets.

To this end, it should be pointed out that in the present study, conducted in the discrete time domain, a numerical simulation-based approach is proposed through which the attainment of sufficiently small and bounded testing errors and closed-loop stability can be verified.

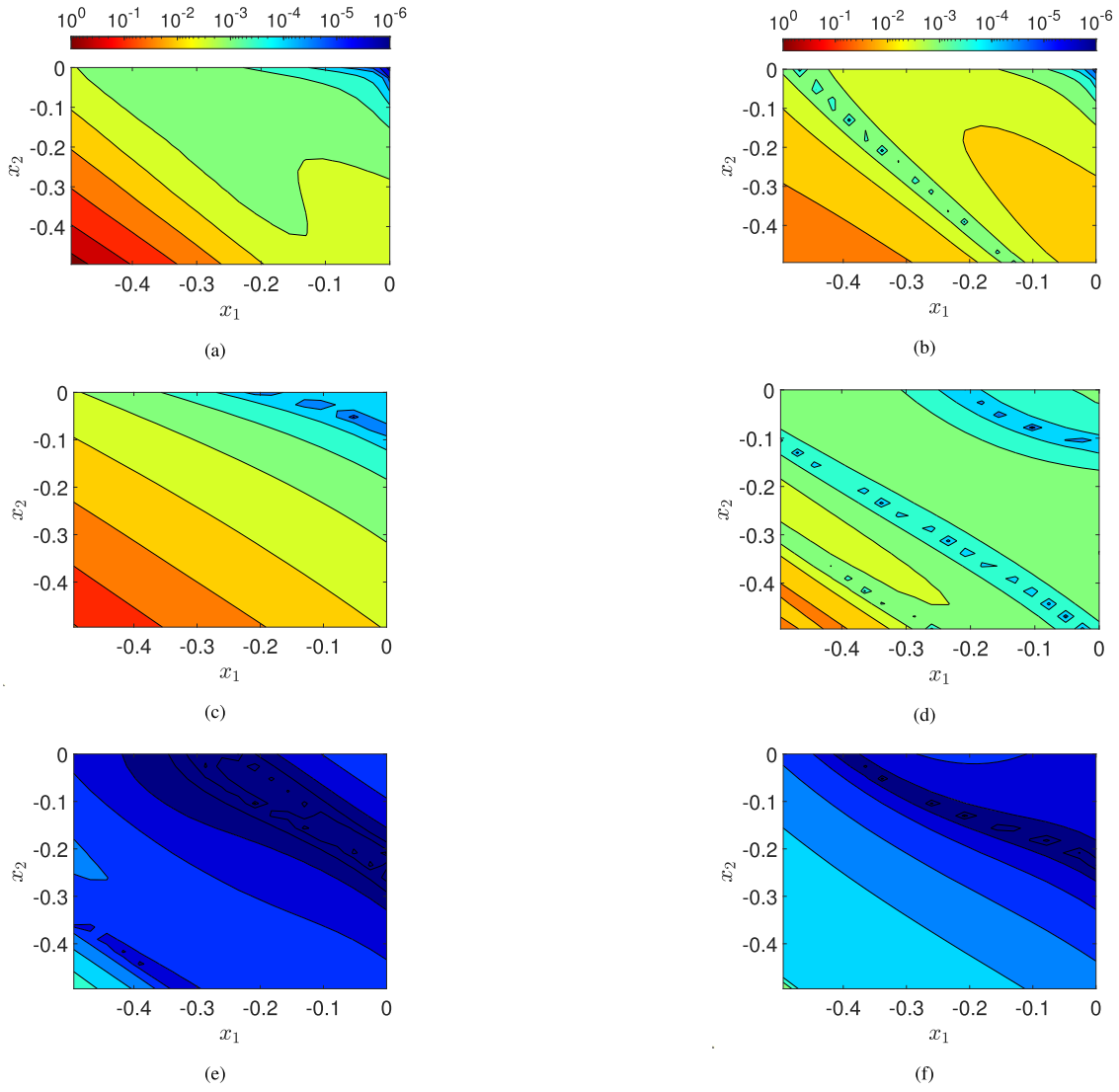

Fig. 8. Black-box simulator. Training sets (grids of 20×20 equispaced distributed points). Numerical approximation accuracy (difference between the computed and analytical solution) of $T_1(x_1, x_2)$ (left column) and $T_2(x_1, x_2)$ (right column) using the various schemes. (a),(b) $6th$ order power-series expansion of $T_1(x_1, x_2)$ and $T_2(x_1, x_2)$ in $[-0.495, 0] \times [-0.495, 0]$. (c),(d) PIML in Matlab trained in the entire domain $[-0.495, 0] \times [-0.495, 0]$. (e),(f) PIML in Matlab trained via the greedy approach.

5. Conclusions

We proposed a PIML-based scheme for the single-step feedback linearization with pole placement for nonlinear discrete-time systems. Within the context of the present study, we considered a system for which the linearizing transformation map and state feedback control law exhibit a singular point, and thus very steep transformation gradients near it. As the underlying optimization problem may lead to a poor solution (for example due to the effect of random initialization of the weights of the PIML), we have chosen to implement a greedy approach, thus tessellating the "hard" (in the entire domain) training/optimization problem into a sequence of simpler ones; this has also been suggested in other studies (see e.g. [49,51]). Such a greedy training strategy, used to initialize weights in a region near a good local minimum, facilitates the optimization algorithm by implicitly acting as a regularization technique and thus resulting in a better generalization [49]. The existence of a singularity, on and beyond which the feedback linearization fails, is a hallmark of many problems that seek useful transformation by formulating and solving functional differential equations [53]. The same type of issue will, for example, arise in trying to compute flow-box transformations [54], or transformations to linearity (in the Koopman operator context [55,56]). Understanding how to test for such singularities, adaptively re-mesh in their neighborhood, estimate the associated singularity exponents, and even considering possible analytic continuations beyond them, is an important issue [53]. In fact, here we have implemented a simple zero-th order continuation in order to provide better initial guesses for

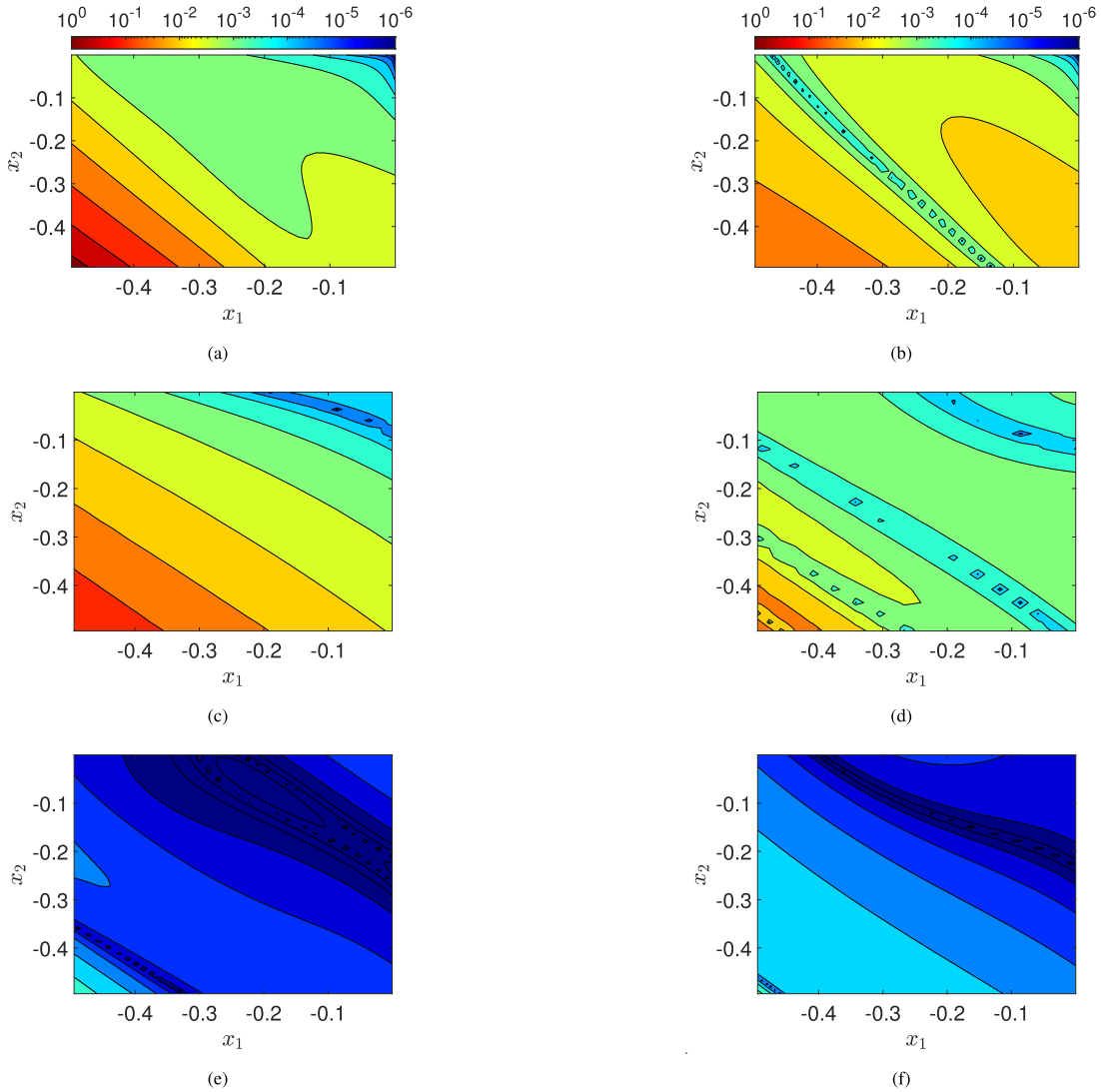

Fig. 9. Black-box simulator. Test sets (grids of 50×50 Chebyshev-distributed points). Numerical approximation accuracy (difference between the computed and analytical solution) of $T_1(x_1, x_2)$ (left column) and $T_2(x_1, x_2)$ (right column) using the various schemes. (a),(b) 6*th* order power-series expansion of $T_1(x_1, x_2)$ and $T_2(x_1, x_2)$ in $[-0.495, 0] \times [-0.495, 0]$. (c),(d) PIML in Matlab trained in the entire domain $[-0.495, 0] \times [-0.495, 0]$. (e),(f) PIML in Matlab trained via the greedy approach.

the unknown weights of the PIML scheme to regions close to the singularity. In a future work, we will aim at exploiting more advanced continuation techniques, such as the natural continuation technique proposed in Fabiani et al. [51] for providing analytically initial guesses for the unknown weights of random projection networks for the solution of stiff ODEs and index-1 DAEs containing steep gradients in their solution profiles, or arc-length continuation for tracing branches of solutions and approximation of manifolds up to or even beyond critical/singular points (see for example [57,58]). Thus bridging ML programming techniques with concepts from continuation techniques, have the potential to significantly facilitate computational experimentation and learning, and thus assist in the study of such singularities, possibly suggesting approaches to their mitigation.

Finally, it should be pointed out that a theoretically rigorous establishment of closed-loop stability under feedback action, considering the approximation errors introduced by machine learning algorithms, remains a challenging research endeavor. The latter requires a creative integration of conceptual and methodological tools derived from machine learning theory, control theory and uncertainty quantification, as suggested in [59–62,44]. Indeed, in the aforementioned studies, closed-loop stability is guaranteed under a certain set of assumptions/conditions relying on: i) sufficiently small and bounded errors induced by the underlying neural network architecture [59–62], or the uncertainty quantification of the error bounds from the solution of the pre-image problem when manifold learning techniques are used [44], ii) more sophisticated generalization error bounds using statistical machine learning theory, as well as iii) stochastic formulations of the stability problem

under consideration. For a general review of the problem of uncertainty quantification in scientific machine learning, the interested reader is referred to a recent study presented in [63].

CRediT authorship contribution statement

Hector Vargas Alvarez: Formal analysis, Investigation, Software, Validation, Visualization, Writing – original draft. **Gianluca Fabiani:** Formal analysis, Investigation, Software, Validation, Visualization, Writing – original draft. **Nikolaos Kazantzis:** Investigation, Methodology, Supervision, Validation, Writing – original draft, Writing – review & editing. **Constantinos Siettos:** Conceptualization, Formal analysis, Investigation, Methodology, Supervision, Validation, Writing – original draft, Writing – review & editing. **Ioannis G. Kevrekidis:** Conceptualization, Investigation, Methodology, Supervision, Writing – review & editing.

Declaration of competing interest

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

Data availability

No data was used for the research described in the article.

Appendix A. Appendix: multivariate power-series expansion approach

In order to perform a comparative assessment of the performance of the proposed computational approach, a practical solution scheme for the associated system of NFE's (3) is needed. Since $f(x,u)$ as well as the solution $T(x)$ are all locally analytic around the origin, it is possible to calculate the solution in the form of a multivariate power-series. The proposed solution method involves the expansion of $f(x,u)$ as well as the unknown solution $T(x)$ in a power-series followed by equating the power coefficients of the same order of both sides of the NFE's (3). Such a procedure leads to a hierarchy of recursion formulas, through which one can calculate the N-th order power coefficients of $T(x)$, given the power coefficients of $T(x)$ up to the order $N-1$ (evaluated in previous recursive steps) [21].

In the derivation of the associated recursion formulas, it is quite convenient to employ the following tensorial notation:

a) The entries of a constant matrix A are represented as a_i^j, where the subscript i refers to the corresponding row and the superscript j to the corresponding column of the matrix.

b) The partial derivatives of the μ-th component $f_\mu(x,u)$ of the vector function $f(x,u)$ with respect to the state variables x evaluated at $(x,u)=(0,0)$ are denoted as follows:

$$\begin{aligned}\hat{f}_\mu^i &= \frac{\partial f_\mu}{\partial x_i}(0,0)\\ \hat{f}_\mu^{ij} &= \frac{\partial^2 f_\mu}{\partial x_i \partial x_j}(0,0)\\ \hat{f}_\mu^{ijk} &= \frac{\partial^3 f_\mu}{\partial x_i \partial x_j \partial x_k}(0,0)\end{aligned} \tag{A.1}$$

etc., where $i,j,k,.. = 1,...,n$.

c) The partial derivatives of the μ-th component $f_\mu(x,u)$ of the vector function $f(x,u)$ with respect to the input variable u evaluated at $(x,u)=(0,0)$ are denoted as follows:

$$g_\mu^i = \frac{\partial^i f_\mu}{\partial u^i}(0,0) \tag{A.2}$$

etc.

d) The standard summation convention where repeated upper and lower tensorial indices are summed up.

Under the above notation the l-th component $T_l(x)$ of the unknown solution $T(x)$ can be expanded in a multivariate power series as follows:

$$\begin{aligned}T_l(x) = &\frac{1}{1!}T_l^{i_1}x_{i_1} + \frac{1}{2!}T_l^{i_1 i_2}x_{i_1}x_{i_2} + ... +\\ &+ \frac{1}{N!}T_l^{i_1 i_2 ... i_N}x_{i_1}x_{i_2}...x_{i_N} + ...\end{aligned} \tag{A.3}$$

As mentioned earlier, the proposed procedure is initiated by considering the expansion of the components of the vector function $f(x,u)$ in multivariate power-series. Substituting the power-series expansions of $T(x)$, $f(x,u)$ into (3) and matching the power coefficients of the same order, the following recursive relations are obtained:

$$T_l^{\mu}(\hat{f}_{\mu}^{i_1} - g_{\mu}^{i} c^{k} T_k^{i_1}) = T_l^{\mu}\hat{f}_{\mu}^{i_1} - T_l^{\mu} g_{\mu}^{i} c^{k} T_k^{i_1} = a_l^{\mu} T_{\mu}^{i_1} \tag{A.4}$$

with: $i_1 = 1, \ldots n$ and $l = 1, .. n$. Note that under the matrix notation and the summation convention introduced above, the set of algebraic equations (A.4) can be recast into the following matrix equation:

$$\bar{T}\,J - A\bar{T} = \bar{T} G c \bar{T} \tag{A.5}$$

where the unknown matrix $\bar{T}$ in (A.5) is the Jacobian of the map $T(x)$ evaluated at the origin. Under the assumptions of Theorem 2.1, the unique invertible solution of the above quadratic matrix equation is given by: $\bar{T} = W^{-1}$, where W is the unique and invertible solution of the Lyapunov matrix equation shown below [21]:

$$JW - WA = Gc \tag{A.6}$$

Since Lyapunov equations of the above type can be solved using a software package such as Matlab/Maple, the calculation of the solution of the first-order algebraic equations (A.4) does not pose any challenges.

N-th order terms; $N \geq 2$

$$\sum_{L=1}^{N} \sum_{\substack{0 \leq m_1 \leq m_2 \leq \ldots \leq m_L \\ m_1 + m_2 + \ldots + m_L = N}} T_l^{j_1 \ldots j_L} (\hat{f}_{j_1}^{m_1} .. \hat{f}_{j_L}^{m_L} - \pi_{j_1}^{m_1} \ldots \pi_{j_L}^{m_L}) = a_l^{\mu} T_{\mu}^{i_1 \ldots i_N} \tag{A.7}$$

where:

$$\pi_{j_l}^{m_L} = \sum_{P=1}^{L} \sum_{\substack{0 \leq n_1 \leq n_2 \leq \ldots \leq n_P \\ n_1 + n_2 + \ldots + n_P = m_L}} g_{j_l}^{n_1} c^{k} T_k^{n_2 \ldots n_P} \tag{A.8}$$

with $i_1, \ldots, i_N = 1, \ldots, n$ and $l = 1, \ldots, n$. Notice that the second summation symbol in (A.7) indicates summing up the relevant quantities over the $\dfrac{N!}{m_1! \ldots m_L!}$ possible combinations to assign the N indices $(i_1, \ldots, i_N)$ as upper indices to the L positions: $\{\hat{f}_{j_1}, \ldots \hat{f}_{j_L}\}$ and $\{\pi_{j_1}, \ldots \pi_{j_L}\}$, with m_1 of them being put in the first position, m_2 of them in the second position, etc. $(\sum_{i=1}^{L} m_i = N)$. Similar rules apply to equation (A.8). Please notice that equations (A.7), (A.8) represent a set of linear algebraic equations in the unknown coefficients $T_{\mu}^{i_1, \ldots, i_N}$ for $N \geq 2$. Furthermore, it should be pointed out, that the above series solution method for the system of NFEs (3) may be accomplished in an automatic fashion by exploiting the computational capabilities of a symbolic software package such as Wolfram Mathematica and MAPLE.

Appendix B. Appendix: learning the feedback linearization operator from black-box simulators

One of the main differences between this method and the previous one, is that in this method we are not expanding both sides of the NFEs (40), but just $T(x)$ in order to calculate the unknown coefficients of its series expansion through let's say nonlinear least squares. The information regarding the system is derived from or given by the output of the black-box simulator. Therefore, the problem under consideration can be stated as one whose target is finding the values of the vector h such that the sum of squared errors on the discretization mesh is minimized in some norm for instance the 2-norm, i.e.

$$\min_{h} \sum_{i=1}^{N} \| R_i(h) \|_2^2 \tag{B.1}$$

where the vector function $R_i(h)$ is defined as follows:

$$R_i(h) = \hat{T}(f(x_i, -c\hat{T}(x_i); h) - A\hat{T}(x_i; h), \quad \forall x_i \tag{B.2}$$

As mentioned earlier, this nonlinear optimization problem can be solved using a Gauss-Newton method or the Levenberg Marquard method in an iterative fashion, by enforcing equality (B.2) at every point of the discretized mesh. For example, for the power-series expansion the algorithm for computing the transformation law using a black-box simulator reads as follows:

- Choose a subdomain of the solution state space of the nonlinear system, i.e. $D \subseteq \mathbb{R}^n$ in a mesh of $N \times N$ points. In such a domain, the solution of the NFEs system (on the basis of which the feedback controller itself is synthesized) will be learned as well.
- Expand the transformation map $T(x)$ in a power-series up to order p around the equilibrium x_o, meaning that $T(x)$ must be expressed as a function of the vector x and also the power-series coefficients $h \in \mathbb{R}^m$, i.e., $\hat{T}(x, h)$, i.e.

$$\begin{aligned}
\hat{T}_1(x_{i=1,\cdots,n}; h_{j,k=1,\cdots,p}) &= h_{1,1}x_1 + h_{1,2}x_2 + \tfrac{1}{2!}h_{1,3}x_1^2 + \tfrac{1}{2!}h_{1,4}x_2^2 + h_{1,5}x_1x_2 + \cdots + O_1(p+1)\\
\hat{T}_2(x_{i=1,\cdots,n}; h_{j,k=1,\cdots,p}) &= h_{2,1}x_1 + h_{2,2}x_2 + \tfrac{1}{2!}h_{2,3}x_1^2 + \tfrac{1}{2!}h_{2,4}x_2^2 + h_{2,5}x_1x_2 + \cdots + O_2(p+1)\\
&\vdots\\
\hat{T}_n(x_{i=1,\cdots,n}; h_{j,k=1,\cdots,p}) &= h_{n,1}x_1 + h_{n,2}x_2 + \tfrac{1}{2!}h_{n,3}x_1^2 + \tfrac{1}{2!}h_{n,4}x_2^2 + h_{n,5}x_1x_2 + \cdots + O_n(p+1)
\end{aligned} \tag{B.3}$$

  Then, write the feedback control law as $u = -c\hat{T}(x_i)$.
- Obtain the information of the system by calling the output of the black-box simulator using the series expansion up to order p of $T(x)$ as u.
- Construct both sides of the NFEs system, and get a residual as indicated in (B.2).
- Add to the residual

$$R_i(h) = \hat{T}(f(x_i, -c\hat{T}(x_i); h) - A\hat{T}(x_i; h) = 0 \tag{B.4}$$

  the following conditions:
  - Initial condition

$$\hat{T}_j(0) = 0, \quad j, k = 1, 2, \ldots, n \tag{B.5}$$

  - Derivative of $\hat{T}(x_i)$ evaluated the equilibrium $x_o = 0$

$$\frac{\partial \hat{T}_j}{\partial x_k}(0) - \frac{\partial T_j}{\partial x_k}(0) = 0, \quad j, k = 1, 2, \ldots, n \tag{B.6}$$

    where, as mentioned before, $\frac{\partial T_j}{\partial x_k}(0)$ is the (j,k)-th element of the Jacobian matrix of $T(x)$ computed at the equilibrium and obtained by solving equation (14). The latter serves as a pinning condition for the optimization problem to find the best coefficients for the series expansion of the transformation map $T(x)$ that satisfy the residual $R_i(h) = 0$ equality. Indeed, without this pinning condition, it is also probable that the optimization process finds the trivial solution that also satisfies these properties, but of course does not represent the feedback controller since the trivial solution maps all the states to the kernel of the linearized space. Finally, the derivative of $T(x)$ can be computed, for instance using finite differences.
- Compute the unknown coefficients of $T(x_i; h)$ using a nonlinear optimization algorithm, such as the Levenberg–Marquardt, Gauss-Newton or perhaps using an unconstrained optimization algorithm, such as the Broyden, Fletcher, Goldfarb, Shanno (BFGS) method.

A similar procedure can be used for the Physics Informed Machine-Learning (PIML) scheme.

References

[1] A. Isidori, Nonlinear Control Systems, Communications and Control Engineering, Springer, London, 1995.
[2] M. Krstic, P.V. Kokotovic, I. Kanellakopoulos, Nonlinear and Adaptive Control Design, John Wiley & Sons, Inc., 1995.
[3] A. Sepulchre, M. Jankovic, Constructive Nonlinear Control, Communications and Control Engineering, Springer, London, 2011.
[4] C.T. Chen, Linear System Theory and Design, Oxford University Press, NY, 1998.
[5] K.J. Åström, R.M. Murray, Feedback Systems: an Introduction for Scientists and Engineers, Princeton University Press, 2021.
[6] M. di Bernardo, Controlling Collective Behavior in Complex Systems, Springer London, London, 2020, pp. 1–10.
[7] C. Kravaris, J.C. Kantor, Geometric methods for nonlinear process control. 1. Background, Ind. Eng. Chem. Res. 29 (12) (1990) 2295–2310.
[8] C. Kravaris, J.C. Kantor, Geometric methods for nonlinear process control. 2. Controller synthesis, Ind. Eng. Chem. Res. 29 (12) (1990) 2310–2323.
[9] N. Kazantzis, C. Kravaris, Synthesis of state feedback regulators for nonlinear processes, Chem. Eng. Sci. 55 (17) (2000) 3437–3449.
[10] S. Monaco, D. Normand-Cyrot, The immersion under feedback of a multidimensional discrete-time non-linear system into a linear system, Int. J. Control 38 (1983) 245–261.
[11] H.-G. Lee, S.I. Marcus, Approximate and local linearizability of non-linear discrete-time systems, Int. J. Control 44 (1986) 1103–1124.
[12] J.W. Grizzle, Feedback linearization of discrete-time systems, in: A. Bensoussan, J.L. Lions (Eds.), Analysis and Optimization of Systems, Springer, Berlin, Heidelberg, 1986, pp. 273–281.
[13] B. Jakubczyk, Feedback linearization of discrete-time systems, Syst. Control Lett. 9 (5) (1987) 411–416.
[14] N. Kwnaghee, Linearization of discrete-time nonlinear systems and a canonical structure, IEEE Trans. Autom. Control 34 (1) (1989) 119–122.
[15] W. Lin, C.I. Byrnes, Remarks on linearization of discrete-time autonomous systems and nonlinear observer design, Syst. Control Lett. 25 (1) (1995) 31–40.
[16] E. Aranda-Bricaire, Ü. Kotta, C.H. Moog, Linearization of discrete-time systems, SIAM J. Control Optim. 34 (1996) 1999–2023.
[17] A. Kumar, P. Daoutidis, State-space realizations of linear differential-algebraic-equation systems with control-dependent state space, IEEE Trans. Autom. Control 41 (2) (1996) 269–274.

[18] A.J. Krener, Feedback Linearization, Springer New York, New York, NY, 1999, pp. 66–98, Ch. 1.
[19] G.O. Guardabassi, S.M. Savaresi, Approximate feedback linearization of discrete-time non-linear systems using virtual input direct design, Syst. Control Lett. 32 (2) (1997) 63–74.
[20] D.G. Luenberger, Observing the state of a linear system, IEEE Trans. Mil. Electron. 8 (2) (1963) 74–80.
[21] N. Kazantzis, A functional equations approach to nonlinear discrete-time feedback stabilization through pole-placement, Syst. Control Lett. 43 (5) (2001) 361–369.
[22] J. Deutscher, C. Schmidt, A state space embedding approach to approximate feedback linearization of single input nonlinear control systems, Int. J. Robust Nonlinear Control 16 (9) (2006) 421–440.
[23] D. Karagiannis, A. Astolfi, R. Ortega, Nonlinear stabilization via system immersion and manifold invariance: survey and new results, Multiscale Model. Simul. 3 (4) (2005) 801–817.
[24] Q. Xu, I. Aksikas, S. Dubljevic, Single-step full-state feedback control design for nonlinear hyperbolic PDEs, Int. J. Control 92 (11) (2019) 2484–2498.
[25] A. Yeşildirek, F. Lewis, Feedback linearization using neural networks, Automatica 31 (11) (1995) 1659–1664.
[26] A. Taprantzis, C. Siettos, G. Bafas, Fuzzy control of a fluidized bed dryer, Dry. Technol. 15 (2) (1997) 511–537.
[27] S. He, K. Relf, R. Unbehauen, A neural approach for control of nonlinear systems with feedback linearization, IEEE Trans. Neural Netw. 9 (6) (1998) 1409–1421.
[28] C. Siettos, C. Kiranoudis, G. Bafas, Advanced control strategies for fluidized bed dryers, Dry. Technol. 17 (10) (1999) 2271–2291.
[29] S. Ge, C. Hang, T. Zhang, Nonlinear adaptive control using neural networks and its application to cstr systems, J. Process Control 9 (4) (1999) 313–323.
[30] C. Siettos, G. Bafas, Semiglobal stabilization of nonlinear systems using fuzzy control and singular perturbation methods, Fuzzy Sets Syst. 129 (3) (2002) 275–294.
[31] H. Deng, H.-X. Li, Y.-H. Wu, Feedback-linearization-based neural adaptive control for unknown nonaffine nonlinear discrete-time systems, IEEE Trans. Neural Netw. 19 (9) (2008) 1615–1625.
[32] X. Yang, D. Liu, D. Wang, Q. Wei, Discrete-time online learning control for a class of unknown nonaffine nonlinear systems using reinforcement learning, Neural Netw. 55 (2014) 30–41.
[33] T. Westenbroek, D. Fridovich-Keil, E. Mazumdar, S. Arora, V. Prabhu, S.S. Sastry, C.J. Tomlin, Feedback linearization for uncertain systems via reinforcement learning, in: 2020 IEEE International Conference on Robotics and Automation (ICRA), 2020, pp. 1364–1371.
[34] H. Yu, S. Park, A. Bayen, S. Moura, M. Krstic, Reinforcement learning versus pde backstepping and pi control for congested freeway traffic, IEEE Trans. Control Syst. Technol. 30 (4) (2021) 1595–1611.
[35] M. Lombardi, D. Liuzza, M. di Bernardo, Using learning to control artificial avatars in human motor coordination tasks, IEEE Trans. Robot. 37 (6) (2021) 2067–2082.
[36] J. Umlauft, T. Beckers, M. Kimmel, S. Hirche, Feedback linearization using Gaussian processes, in: 2017 IEEE 56th Annual Conference on Decision and Control (CDC), 2017, pp. 5249–5255.
[37] Z. Wu, D. Rincon, P.D. Christofides, Real-time adaptive machine-learning-based predictive control of nonlinear processes, Ind. Eng. Chem. Res. 59 (6) (2019) 2275–2290.
[38] Y.M. Ren, M.S. Alhajeri, J. Luo, S. Chen, F. Abdullah, Z. Wu, P.D. Christofides, A tutorial review of neural network modeling approaches for model predictive control, Comput. Chem. Eng. (2022) 107956.
[39] W. Tang, P. Daoutidis, Dissipativity learning control (dlc): a framework of input–output data-driven control, Comput. Chem. Eng. 130 (2019) 106576.
[40] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[41] G.E. Karniadakis, I.G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, Nat. Rev. Phys. 3 (6) (2021) 422–440.
[42] S. Cai, Z. Mao, Z. Wang, M. Yin, G.E. Karniadakis, Physics-informed neural networks (pinns) for fluid mechanics: a review, Acta Mech. Sin. 37 (12) (2021) 1727–1738.
[43] J. Darbon, P.M. Dower, T. Meng, Neural network architectures using min-plus algebra for solving certain high-dimensional optimal control problems and Hamilton–Jacobi PDEs, Math. Control Signals Syst. 35 (1) (2023) 1–44.
[44] D.G. Patsatzis, L. Russo, I.G. Kevrekidis, C. Siettos, Data-driven control of agent-based models: an equation/variable-free machine learning approach, J. Comput. Phys. (2023) 111953.
[45] C.I. Siettos, I.G. Kevrekidis, N. Kazantzis, An equation-free approach to nonlinear control: coarse feedback linearization with pole-placement, Int. J. Bifurc. Chaos 16 (07) (2006) 2029–2041.
[46] C.I. Siettos, I.G. Kevrekidis, D. Maroudas, Coarse bifurcation diagrams via microscopic simulators: a state-feedback control-based approach, Int. J. Bifurc. Chaos 14 (01) (2004) 207–220.
[47] A. Armaou, C.I. Siettos, I.G. Kevrekidis, Time-steppers and 'coarse' control of distributed microscopic processes, Int. J. Robust Nonlinear Control 14 (2) (2004) 89–111.
[48] C. Siettos, C. Gear, I. Kevrekidis, An equation-free approach to agent-based computation: bifurcation analysis and control of stationary states, Europhys. Lett. 99 (4) (2012) 48007.
[49] H. Larochelle, Y. Bengio, J. Louradour, P. Lamblin, Exploring strategies for training deep neural networks, J. Mach. Learn. Res. 10 (2009) 1–40.
[50] T. Chen, H. Chen, Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems, IEEE Trans. Neural Netw. 6 (4) (1995) 911–917.
[51] G. Fabiani, E. Galaris, L. Russo, C. Siettos, Parsimonious physics-informed random projection neural networks for initial-value problems of odes and index-1 daes, arXiv preprint, arXiv:2203.05337.
[52] L.B. Rall, Automatic differentiation: techniques and applications, in: Lecture Notes in Computer Science, 1981.
[53] P.G. Kevrekidis, C.I. Siettos, Y.G. Kevrekidis, To infinity and some glimpses of beyond, Nat. Commun. 8 (1) (2017) 1562.
[54] M.E. Henderson, Computing invariant manifolds by integrating fat trajectories, SIAM J. Appl. Dyn. Syst. 4 (4) (2005) 832–882.
[55] M. Budišić, R. Mohr, I. Mezić, Applied koopmanism, Chaos 22 (4) (2012) 047510.
[56] E.M. Bollt, Q. Li, F. Dietrich, I. Kevrekidis, On matching, and even rectifying, dynamical systems through Koopman operator eigenfunctions, SIAM J. Appl. Dyn. Syst. 17 (2) (2018) 1925–1960.
[57] G. Fabiani, F. Calabrò, L. Russo, C. Siettos, Numerical solution and bifurcation analysis of nonlinear partial differential equations with extreme learning machines, J. Sci. Comput. 89 (2021) 1–35.
[58] E. Galaris, G. Fabiani, I. Gallos, I. Kevrekidis, C. Siettos, Numerical bifurcation analysis of PDEs from lattice Boltzmann model simulations: a parsimonious machine learning approach, J. Sci. Comput. 92 (2) (2022) 34.
[59] Z. Wu, A. Tran, D. Rincon, P.D. Christofides, Machine learning-based predictive control of nonlinear processes. Part I: Theory, AIChE J. 65 (2019) 1–14.
[60] Z. Wu, D. Rincon, P.D. Christofides, Real-time adaptive machine-learning-based predictive control of nonlinear processes, Ind. Eng. Chem. Res. 59 (6) (2020) 2275–2290.
[61] Z. Wu, D. Rincon, Q. Gu, P.D. Christofides, Statistical machine learning in model predictive control of nonlinear processes, Mathematics 9 (2021) 1912.
[62] Z. Wu, A. Alnajd, Q. Gu, P.D. Christofides, Statistical machine-learning-based predictive control of uncertain nonlinear processes, AIChE J. 68 (2022) 17642.
[63] A.F. Psaros, X. Meng, Z. Zou, L. Guo, G.E. Karniadakis, Uncertainty quantification in scientific machine learning: methods, metrics, and comparisons, J. Comput. Phys. 477 (2023) 111902.