Neural Computing and Applications (2023) 35:3789–3804
https://doi.org/10.1007/s00521-022-07886-y

ORIGINAL ARTICLE

DAE-PINN: a physics-informed neural network model for simulating differential algebraic equations with application to power networks

Christian Moya[1] · Guang Lin[1,2]

Received: 23 May 2022 / Accepted: 22 September 2022 / Published online: 15 October 2022

Abstract
Deep learning-based surrogate modeling is becoming a promising approach for learning and simulating dynamical systems. However, deep-learning methods find it very challenging to learn stiff dynamics. In this paper, we develop DAE-PINN, the first effective physics-informed deep-learning framework for learning and simulating the solution trajectories of *nonlinear differential-algebraic equations* (DAE). DAEs are used to model complex engineering systems, e.g., power networks, and present a "form" of infinite stiffness, which makes learning their solution trajectories challenging. Our DAE-PINN bases its effectiveness on the synergy between *implicit Runge–Kutta* time-stepping schemes (designed specifically for solving DAEs) and *physics-informed neural networks* (PINN) (deep neural networks that we train to satisfy the dynamics of the underlying problem). Furthermore, our framework (i) enforces the neural network to satisfy the DAEs as (approximate) hard constraints using a penalty-based method and (ii) enables simulating DAEs for long-time horizons. We showcase the effectiveness and accuracy of DAE-PINN by learning the solution trajectories of a three-bus power network.

Keywords Deep learning · Data-driven scientific computing · Nonlinear differential-algebraic equations · Implicit Runge–Kutta

1 Introduction

In recent years, we have seen the power network incorporate more and more transformative technologies, such as integrating distributed energy resources, enabling a liberalized market, or adopting more complex communication and control algorithms. Such transformation seeks to enhance the reliability and efficiency of the power network operation. However, this transformation pushes the power network to operate under a more diversified set of operating conditions and contingencies that could compromise its security.

To assess the power network's dynamic security [25, 46], utility operators implement an offline procedure that seeks to predict whether the power network will remain stable after facing a single contingency (e.g., the trip of a generator) from a set of credible contingencies. Such a procedure is known as $N-1$ criteria [1] and requires simulating the power network's transient dynamic response. Simulating the transient dynamic response requires integrating a set of nonlinear differential-algebraic equations (DAE) [25]. However, solving these DAEs is a very challenging and expensive task. Indeed, the classical explicit integration schemes (e.g., Euler's method or RK-4) may not be stable enough for such a task [19]. As a result, most commercial solvers implement a partitioned approach or implicit integration [33].

The partitioned approach solves the differential and algebraic equations in sequence. However, freezing the algebraic (resp. dynamic) variables to solve the dynamic (resp. algebraic) variables introduces a delay error that propagates throughout the simulation [48]. On the other hand, the implicit integration method for DAEs simultaneously solves the dynamic and algebraic equations.

✉ Guang Lin
guanglin@purdue.edu

Christian Moya
cmoyacal@purdue.edu

[1] Department of Mathematics, Purdue University, West Lafayette 47906, IN, USA

[2] Department of Mechanical Engineering, Purdue University, West Lafayette 47906, IN, USA

Implicit integration does not introduce delay errors but uses iterative methods (e.g., Newton's method) and matrix inversion [19]. As a result, the computational cost and memory required to integrate DAEs are very high and constitute the main obstacle for deploying dynamic security assessment in real-time [46]. Faster simulation and prediction alternatives for dynamic security assessment have been proposed by many works, which we will review next.

1.1 Previous works

1.1.1 Parallel computing methods

To enable dynamic security assessment in real-time, many works have proposed simulating the transient dynamic response by using distributed, parallel, and high-performance computing [46], which can enhance the assessment's efficiency or simulate numerous contingency cases. For instance, several works [47, 50] have employed parallel computing for power network spatial decomposition. Other methods [2, 13] use parallelization across numerical solvers with possibly different accuracy. Many works [6, 15, 37] have investigated how to implement parallel simulations across the time domain. In particular, the parallel in-time approach [15, 37] has demonstrated computational effectiveness when simulating detailed power network models. While parallel computing methods for simulating DAEs can enhance efficiency, they still require vast computational resources.

1.1.2 Energy function methods

Direct or energy function-based methods [10, 11, 18, 36, 51] constitute an alternate approach to the traditional offline dynamic security assessment. These direct methods provide a dynamic security certification without the expensive integration of the power network's post-disturbance dynamics described using DAEs. In particular, they infer transient stability using a mathematical model of the power network's dynamics and an energy function (i.e., a Lyapunov-like function), which certifies the convergence of the states to the stable operating point. Modern versions of these direct methods [10] have been successfully engineered to the point that they were implemented at the utility level. However, the scalability and conservativeness of the classical energy-function method limit their applicability to even relatively large power networks. Moreover, direct methods do not provide the state trajectories that operators often require for planning.

1.1.3 Deep learning methods

Motivated by the power network's dynamic security assessment application, in this paper, we seek to derive a deep learning (DL) framework that accelerates simulating nonlinear DAEs. Enabled by the exponential growth of computational power and data availability, DL has achieved outstanding performance in computer vision and natural language processing applications [26] and promises to revolutionize the scientific and engineering fields. However, the current application of DL to learn scientific and engineering dynamical systems is, at most, limited since the cost of collecting data is prohibitive. Moreover, most conventional DL methods (e.g., convolutional or recurrent neural networks) lack robustness and generalization capabilities in such a small data regime.

1.1.4 Deep learning methods for solving differential equations

In recent years, the scientific machine learning [3, 22] community has proposed several learning methods for solving ordinary and partial differential equations (e.g., physics-informed neural networks [22, 41]). We roughly classify these methods into (i) identifying the governing differential equations via dictionary learning and (ii) learning to approximate the solution of differential equations. In particular, let us focus our analysis on time-dependent differential equations.

Several works [4, 5, 45] use dictionary learning and enormous time-series datasets to *learn the governing equations* of the underlying dynamical system. For instance, in [4, 5], the authors used a dictionary of smooth functions to learn the governing equations of non-stiff autonomous and non-autonomous systems. Furthermore, the authors of [45] employed sparse approximation schemes and a dictionary of functions to recover the ordinary/partial differential equations of unknown systems. Compared to dictionary learning-based methods, the method proposed in this paper (i) does not require time-series data and (ii) focuses on approximating the dynamic response of more complex systems described by DAEs.

On the other hand, there is growing interest in *learning to predict the future response* of dynamical systems modeled using differential equations. In recent years, the field of scientific machine learning [3, 22] has provided us with many transformative works (e.g., Physics-Informed Neural Networks (PINN) [30, 31, 41, 54]) aimed at learning to predict the future response of dynamical systems and, hence, providing us with efficient alternatives to traditional costly numerical solvers. Several of these works use the idea of encoding the physical laws that govern these systems (i.e., the differential equations) as soft constraints

during training. The information from the differential equations acts as a regularizing agent, limiting the space of possible solutions and enabling generalizing well even when the amount of data inputs is small. Admittedly, much work is still needed to scale physics-informed deep learning methods so they can become accurate surrogate models for large-scale systems (e.g., power networks [55] or vehicular networks [8]). For instance, one could potentially leverage bilayered parallel training architectures for distributed computing environments [7] and develop accurate large-scale surrogate models that can (i) predict solution trajectories for a large set of initial conditions and (ii) maintain good accuracy for long-time horizons.

Other methods adopt data-driven strategies [12, 38–40] for predicting the future response of unknown systems. For instance, in [12], the authors trained a transformer with data from the early stages of the PDE's solution to predict the solution recursively at future stages. Qin et al. [38] used a residual neural network (ResNet) to approximate a mapping from the current state to the next, given the step size. Then, one can predict solution trajectories for long-term horizons via the rollout of the trained ResNet. Most of the above data-driven strategies suffer from compounding errors, require enormous datasets, or generalize poorly outside the training distribution. The proposed method in this paper will also use a recursive prediction strategy for long-term horizons as in [38, 40]. However, compared to these works, DAE-PINN does not require labeled data and inherits the generalization capabilities and robustness from discrete PINNs [22, 41].

Despite the success of scientific machine learning in approximating the solution trajectories of ordinary differential equations [54], developing a DL-based framework for simulating the solution trajectories of nonlinear differential-algebraic equations remains an open problem. This is because DAEs present a "form" of infinite stiffness [23] represented through the algebraic equations. As a result, training deep neural networks to approximate DAEs may produce gradient pathologies [52] and ill-conditioned optimization problems, leading to the failure of the stochastic gradient descent-based training. The first attempts to derive DL frameworks for learning the dynamic response of stiff differential equations were presented in [21] and [23]. In [21], the authors demonstrated that the continuous version of the PINN model fails to learn stiff differential equations and proposed using quasi-steady-state assumptions to derive a simpler model, more suitable for PINN-based training. In [23], Kim et al. modified neural ordinary differential equations [9] so that they could learn the solution trajectories of stiff problems for long-time horizons. These methods for learning stiff ODEs have their merits. However, as presented, they are not suitable for learning the solution trajectories of the DAEs studied in this paper.

1.1.5 Deep learning for power networks

Many works have proposed to address the power network's dynamic security assessment problem using machine and deep learning strategies. The main idea of most of these works [14, 17, 20, 59] is to learn from data a binary indicator that maps the initial post-disturbance conditions of the power network to a certificate for transient stability. For example, in [20], the authors designed a convolutional neural network that observes PMU measurements from disturbances and outputs a transient stability certificate. Similarly, the authors of [59] developed a convolutional neural network that maps the PMU measurements from disturbances to a transient stability certificate and an estimate of the stability margin.

The above machine/deep learning tools for dynamic security assessment are fast. However, the information they present may be insufficient for operators and planners. For instance, operators often require trajectory information after the disturbance to predict voltage and frequency violations that may trigger load shedding. To predict whole post-disturbance trajectories, the authors of this paper have developed several deep learning-based alternatives [28, 35, 57]. For instance, in [28], we developed a deep learning framework that employs the long-term memory network (LSTM) to predict the transient dynamic response of a generator. However, this work requires vast amounts of supervision and training data. To alleviate such requirements, we formulated in [35] the post-disturbance transient trajectory prediction problem as an infinite-dimensional operator regression problem. The results illustrated that a deep operator network (DeepONet) effectively learns solution trajectories from small datasets obtained by simulating DAEs. However, we cannot guarantee that the proposed methods can predict the solution trajectories of power networks for a given distribution feasible operating conditions.

1.2 Our work

In this paper, we develop DAE-PINN, the first deep learning-based framework for learning and simulating the solution trajectories of semi-explicit *differential-algebraic equations* (DAE) of index-1. In particular, our objectives in this paper are:

1. *Forward problem*: deriving a framework that learns to map a given distribution of initial conditions to the solution trajectories (within a short-time interval) of a dynamical system described by DAEs.

2. *Long-time simulation of DAEs*: designing an algorithm that uses the trained framework to simulate DAEs over long-time horizons.

Compared to more traditional methods to simulate the dynamic response of power networks [33], we train our DAE-PINN to solve the differential and algebraic equations simultaneously. As a result, the proposed DAE-PINN framework does not (i) introduce delay error or (ii) require matrix inversion during simulation.

We detail our contributions next.

1. We design a deep learning (DL) framework (DAE-PINN—Sects. 3.1 and 3.2) that tackles the *forward problem* by enabling the synergistic combination of a discrete *physics-informed neural network* model with an *implicit Runge–Kutta* scheme designed specifically for solving DAEs. Thus, it effectively extends the method proposed in [41] to DAEs.
2. A *penalty*-based method is then introduced (Sect. 3.3) to facilitate the training of DAE-PINN. The penalty method aims to enforce DAE-PINN to satisfy the DAEs as (approximate) hard constraints.
3. For the *long-time simulation of DAEs*, we propose an algorithm (Sect. 3.4) that iteratively evaluates the trained DAE-PINN. Following a Markov-like procedure, the proposed algorithm uses the DAE-PINN prediction of the previous evaluation step as the initial condition for the next step.
4. We illustrate the training protocols for DAE-PINN and evaluate its effectiveness (Sect. 4) using a three-bus power network example described by a set of stiff and nonlinear DAEs.

We organize this work as follows. In Sect. 2, we introduce the *differential algebraic equations* (DAE) studied in this paper. In Sect. 3, after describing the implicit Runge–Kutta (IRK) time-stepping scheme, we describe DAE-PINN, i.e., the discrete *physics-informed neural network* that allows us to use the IRK scheme (with an arbitrary number of stages) for solving DAEs. We then describe the penalty method that enforces DAE-PINN to satisfy the DAEs as approximate hard constraints. We conclude Sect. 3 by introducing Algorithm 2 that enables us to use the trained DAE-PINN for simulating DAEs over long-time horizons. In Sect. 4, we verify the effectiveness of the proposed framework using a three-bus power network example. We provide a discussion of our results and future work in Sect. 5 and conclude the paper in Sect. 6.

2 Problem setup

In this paper, we develop DAE-PINN, a deep learning-based framework that employs physics-informed neural networks [41] and implicit Runge–Kutta schemes [19] for learning the solution trajectories of nonlinear *Differential-Algebraic equations* (DAE) [53] given in the semi-explicit form

$$\dot{y} = f(y,z), \qquad y(t_0) = y_0 \tag{1a}$$

$$0 = g(y,z), \qquad z(t_0) = z_0, \tag{1b}$$

where $y = y(t) \in \mathbb{R}^{n_y}$ are the dynamic states, $z = z(t) \in \mathbb{R}^{n_z}$ are the algebraic variables, $f : \mathbb{R}^{n_y} \times \mathbb{R}^{n_z} \to \mathbb{R}^{n_y}$ describes the differential equations, $g : \mathbb{R}^{n_y} \times \mathbb{R}^{n_z} \to \mathbb{R}^{n_z}$ the algebraic equations, $t \in [t_0, T]$ the simulation time interval, and $T > t_0$ the time horizon.

2.1 Assumptions

Let us assume that f and g are sufficiently often differentiable and the initial conditions satisfy $g(y_0, z_0) = 0$. We also assume that the DAEs (1) are of index 1 [42], which means the inverse of the Jacobian $g_z = \partial g / \partial z$ exists and is bounded in a neighborhood of the exact solution. This implies that, by the implicit function theorem [44], the algebraic equations (1b) have locally a unique solution $z = G(y)$. Hence, the DAE (1) is equivalent to the following system of ordinary differential equations

$$\dot{y} = f(y, G(y)), \tag{2}$$

with initial conditions $(y(t_0), z(t_0)) = (y_0, G(y_0))$. Notice that the examples studied in Ji et al. [21] (Stiff-PINNs) correspond to a special case in our problem setup where the algebraic variables z can be solved explicitly to obtain (2).

2.2 Applications

DAEs frequently arise in dynamic simulations of power networks [25], mechanical problems, or trajectory control. DAEs also originate from singular perturbation problems (SPP) of the form

$$\dot{y} = f(y,z) \tag{3a}$$

$$\epsilon \dot{z} = g(y,z), \tag{3b}$$

by letting the parameter $\epsilon > 0$ approach zero. SPPs have been used to study (i) nonlinear oscillations with large parameters, (ii) structure-preserving power networks with frequency-dependent dynamic loads, and (iii) chemical kinetics with slow and fast reactions.

We conclude this section with the following remark. The DAE-PINN that we will develop in Sect. 3 could also be used for solving problems described in descriptor form

$$M\dot{x} = \varphi(x), \qquad x(t_0) = x_0, \tag{4}$$

where $x \in \mathbb{R}^{n_y+n_z}$ and M is a singular matrix. To that end, we show next that (4) is mathematically equivalent to the DAE (1). First, we decompose M (e.g., via Gaussian elimination with total pivoting) as

$$M = S\begin{pmatrix} I & 0 \\ 0 & 0 \end{pmatrix}T$$

where S and T are invertible matrices and I is the identity matrix with a dimension corresponding to the rank of M. Then, we insert the above into (4) and use $Tx = (y^\top, z^\top)^\top$ to obtain

$$\begin{pmatrix} \dot{y} \\ 0 \end{pmatrix} = S^{-1}\varphi\left(T^{-1}\begin{pmatrix} y \\ z \end{pmatrix}\right) =: \begin{pmatrix} f(y,z) \\ g(y,z) \end{pmatrix},$$

i.e., the semi-explicit DAE (1). Thus, the deep learning framework that we will derive in Sect. 3 for (1) also applies for problems in descriptor form (4), provided we can decompose the matrix M.

3 Proposed method: DAE-PINN

This section describes our DAE-PINN framework, i.e., a physics-informed neural network framework that allows solving the DAE (1) using the *implicit Runge–Kutta* (IRK) time-stepping scheme with $v \in \mathbb{Z}_+$ stages. We selected IRK due to its stability guarantees for stiff problems [19]. However, we would like to remark that one can design DAE-PINN using any stable scheme that can simultaneously solve differential and algebraic equations.

3.1 Implicit Runge–Kutta scheme

Let us start by assuming that the integration of (1) has been carried out up to (t_n, y_n, z_n) and we seek to advance it to $(t_{n+1}, y_{n+1}, z_{n+1})$, where $t_{n+1} = t_n + h$ and $h > 0$ is the *time step* [19]. We apply the implicit Runge–Kutta scheme with v stages [19, 42] to our system of DAEs (1) and obtain

$$\xi_j = y_n + h\sum_{i=1}^{v} a_{j,i} f(\xi_i, \zeta_i), \quad j = 1, \ldots, v \tag{5a}$$

$$0 = g(\xi_j, \zeta_j), \qquad j = 1, \ldots, v \tag{5b}$$

$$y_{n+1} = y_n + h\sum_{j=1}^{v} b_j f(\xi_j, \zeta_j) \tag{5c}$$

$$0 = g(y_{n+1}, z_{n+1}). \tag{5d}$$

Here $\xi_j = y(t_n + c_j h)$, $\zeta_j = z(t_n + c_j h)$, and $\{a_{j,i}, b_j, c_i\}$ are the *known* parameters of the IRK scheme. Following [19] and to let the scheme be of nontrivial order, we impose the following convention for the parameters

$$\sum_{i=1}^{v} a_{j,i} = c_j.$$

3.2 Discrete physics-informed neural networks

In classical numerical analysis [19], implicit formulations of Runge–Kutta schemes are usually constrained due to the computational cost of solving (5). Moreover, these constraints become more severe if we increase the number of IRK stages. To overcome these constraints, our DAE-PINN framework employs a *discrete physics-informed neural network* (PINN) model [41] to enable the implicit Runge–Kutta scheme (5) with an arbitrary number of stages v.

In the discrete PINN model, the first step is to construct multi-output neural networks with parameters θ (see Fig. 1a and 1b) as a surrogate for the solution of the IRK scheme (5), which takes the input y_n and outputs

$$[\xi_1^\theta, \ldots, \xi_v^\theta, y_{n+1}^\theta] \tag{6a}$$

$$[\zeta_1^\theta, \ldots, \zeta_v^\theta, z_{n+1}^\theta]. \tag{6b}$$

Remark 1 *Architecture:* In this paper, we mainly adopt the *unstacked* architecture depicted in Fig. 1b, which assigns one neural network for the dynamic state variables $y \in \mathbb{R}^{n_y}$ and another neural network for the algebraic variables $z \in \mathbb{R}^{n_z}$. We remark, however, that one can also adopt a *stacked* architecture, which assigns a single neural network for each of the dynamic variables $y_i \in \mathbb{R}$ and each of the algebraic variables $z_i \in \mathbb{R}$. Empirically, we have observed that the unstacked architecture provides improved training results.

Remark 2 *Complexity:* As described in [41], PINN enables us to employ implicit Runge–Kutta schemes with a large number of stages at very little extra cost. More specifically, only the number of neurons of the last layer of the neural networks grows linearly with the total number of stages, i.e., with cost $\sim O(v)$.

In the second step for the discrete PINN model, we restrict the neural networks to satisfy the differential and algebraic equations described by the IRK scheme (5). In practice, we restrict the neural networks on some set of randomly distributed initial conditions scattered/sampled

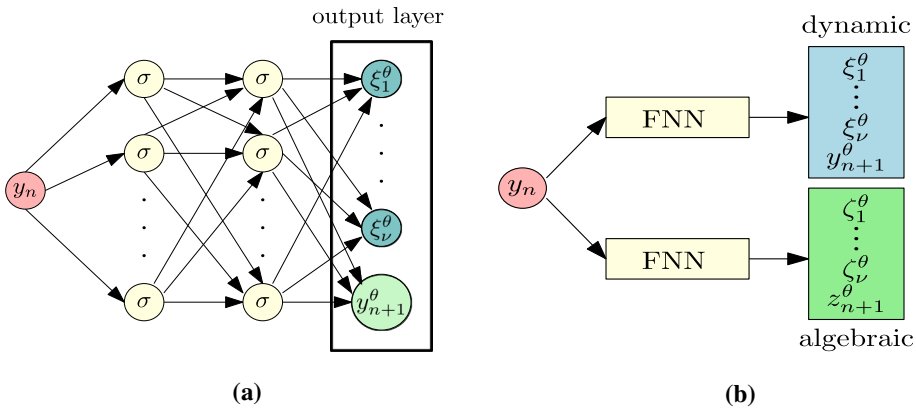

Fig. 1 **a** The multi-output fully connected neural network (FNN) for the dynamic states y. **b** Unstacked architecture—DAE-PINN framework for solving the DAEs (1) using IRK (5)

throughout the domain [30], i.e., the set of training data $\mathcal{T} := \{y_{n,1}, y_{n,2}, \ldots, y_{n,N_\mathcal{T}}\}$ of size $N_\mathcal{T}$[1]. To measure the discrepancy between the neural networks and the IRK scheme (5), we use the following loss function:

$$\mathcal{L}(\theta; \mathcal{T}) = w_f \mathcal{L}_f(\theta; \mathcal{T}) + w_g \mathcal{L}_g(\theta; \mathcal{T}). \tag{7}$$

In the above, w_g and w_f are the weights,

$$\mathcal{L}_f(\theta; \mathcal{T}) = \frac{1}{N_\mathcal{T}(v+1)} \sum_{k=1}^{N_\mathcal{T}} \sum_{j=1}^{v+1} \|y_{n,k} - y^j_{n,k}(\theta)\|_2^2,$$

where

$$y^j_{n,k}(\theta) := \xi^\theta_{j,k} - h \sum_{i=1}^{v} a_{j,i} f(\xi^\theta_{i,k}, \zeta^\theta_{i,k}), \quad j = 1, \ldots, v$$

$$y^{v+1}_{n,k}(\theta) := y^\theta_{n+1,k} - h \sum_{j=1}^{v} b_j f(\xi^\theta_{j,k}, \zeta^\theta_{j,k}),$$

and

$$\mathcal{L}_g(\theta; \mathcal{T}) = \frac{1}{N_\mathcal{T}(v+1)} \sum_{k=1}^{N_\mathcal{T}} \left(\sum_{j=1}^{v} \|g(\xi^\theta_{j,k}, \zeta^\theta_{j,k})\|_2^2 + \|g(y^\theta_{n+1,k}, z^\theta_{n+1,k})\|_2^2 \right).$$

In the last step for the discrete PINN model, we train the neural network parameters by minimizing the loss function using gradient-based optimizers, e.g., the Adam optimizer [24]:

$$\theta^* = \arg\min_\theta \mathcal{L}(\theta; \mathcal{T}). \tag{8}$$

We use the weight coefficients w_f and w_g in (7) to balance the residual loss terms for the dynamic variables $\mathcal{L}_f$ and the algebraic variables $\mathcal{L}_g$. In this paper, we use a *penalty*-based method [31] to update the value of the weight coefficients w_f and w_g.

[1] Observe that our proposed framework does not require supervision, i.e., it does not require to know target values of the solution trajectory.

3.3 Enforcing DAEs as approximate hard constraints

In the DAE problem (1), the solution trajectories must always satisfy the dynamic equations (1a) and lie in the manifold described by the algebraic equations (1b), i.e.,

$$\{(y, z) : g(y, z) = 0\},$$

which, for power networks, represents satisfying the power flow equations [25]. However, it may be difficult to satisfy the dynamic and algebraic equations exactly by using the soft constraints approach for the loss function (7). This can be seen as follows. Suppose the weight coefficients w_f and w_g are selected too large, which severely penalizes the violation of the DAEs. In that case, the optimization problem may become ill-conditioned; hence, it may be difficult to converge to a minimum. On the other hand, if the values selected for w_f and w_g are too small, the solution will not satisfy the dynamic equations or will not lie in the manifold described by the algebraic equations.

To impose the DAEs as approximate hard constraints, we implement the *penalty-based method* introduced in [31] and summarized in Algorithm 1. The main idea behind this method is to replace the optimization problem with equality constraints (i.e., the differential and algebraic equations) with a *sequence* of unconstrained problems with varying penalty coefficients w_f^k and w_f^k. More specifically, during the kth "outer" iteration, we solve the following unconstrained optimization problem

$$\min_\theta \mathcal{L}(\theta; \mathcal{T}) = w_f^k \mathcal{L}_f + w_g^k \mathcal{L}_g,$$

where w_f^k and w_g^k are the penalty coefficients for the kth iteration. Furthermore, at the beginning of each iteration, we increase the penalty coefficients by a constant factor $\beta > 1$:

$$w_g^{k+1} = \beta w_g^k = (\beta)^k w_g^0,$$
$$w_f^{k+1} = \beta w_f^k = (\beta)^k w_f^0.$$

As $k \to \infty$, and given that the neural networks are well trained, the solution of the sequence of unconstrained optimization problems will converge to the solution that satisfies the DAEs approximately as hard constraints [31, 32]. In practice, however, if we fail to carefully select the hyper-parameters w_f^0, w_g^0, and β, the optimization problem may become ill-conditioned or experience slow convergence.

Algorithm 1 Training using the penalty method [31]

Require: initial penalty coefficients w_f^0 and w_g^0, factor β, and number of iterations K.

1: $k \longleftarrow 0$
2: $\theta^0 \longleftarrow \text{argmin}_\theta \ \mathcal{L}^0(\theta;\ \mathcal{T})$: train the neural network (7) from random initialization, until the training loss has converged, *i.e.*, $\mathcal{L}^0(\theta;\ \mathcal{T}) \leq$ 1e-5.
3: **while** $k \leq K$ **do**
4: $\quad k \longleftarrow k+1$
5: $\quad w_g^k \longleftarrow \beta w_g^{k-1}$
6: $\quad w_f^k \longleftarrow \beta w_f^{k-1}$
7: $\quad \theta^k \longleftarrow \text{argmin}_\theta \ \mathcal{L}^k(\theta;\ \mathcal{T})$: train the networks (7) from the initialization θ^{k-1}, until the training loss has converged, *i.e.*, $\mathcal{L}^k(\theta;\ \mathcal{T}) \leq$ 1e-5.
8: **end while**

3.4 Simulating DAEs for long-time horizons

Until now, we have described how the DAE-PINN framework enables integrating DAEs (1) from (t_n, y_n, z_n) to $(t_n + h, y_{n+1}, z_{n+1})$. Such a framework can use a large number ν of stages to take a large time step h. However, when simulating stiff and nonlinear DAEs (e.g., the power network dynamics) for long-time horizons, it may be necessary to take multiple time steps. Thus, this subsection briefly describes how we can use our trained DAE-PINN framework to simulate DAEs for long-time horizons. We provide a detailed description of the proposed iterative strategy in Algorithm 2 and an illustrative example in Fig. 2. The main idea behind our strategy is to update recurrently (in a Markov-like fashion) the input to DAE-PINN y_n using the predicted dynamic states $y_{n+1}^{\theta^*}$ from the previous evaluation step.

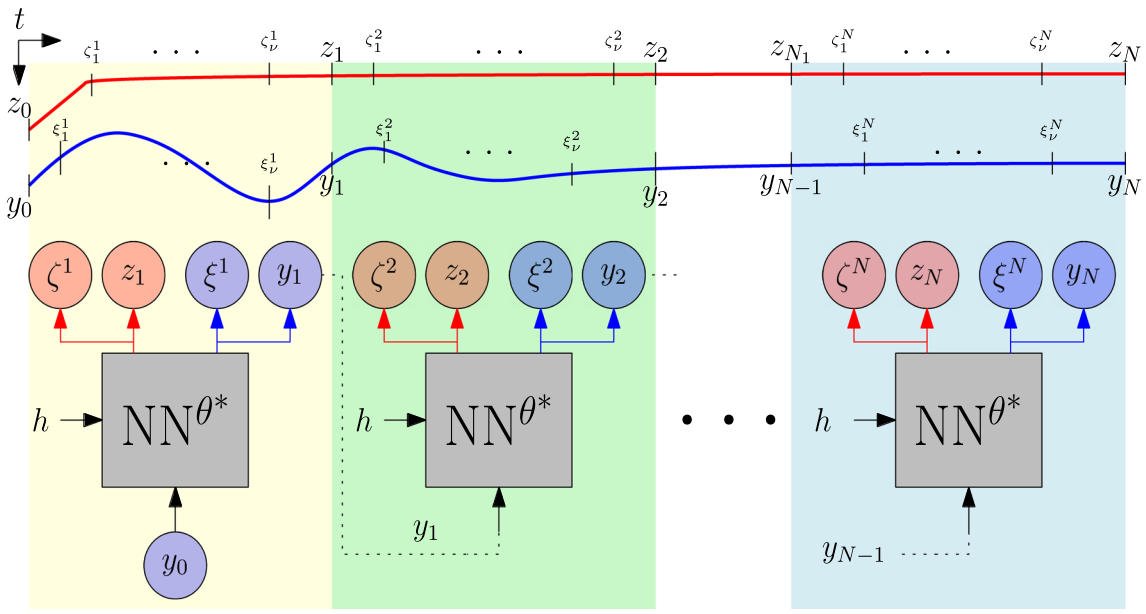

Fig. 2 Illustration of how to use the proposed trained DAE-PINN (NN^{θ^*}) to simulate index-1 DAEs (1) for long-time horizons (i.e., for N time steps of size h)

Thus, Algorithm 2 enables us to simulate the solution trajectories of DAEs (1), $y(t)$ and $z(t)$, within the time interval $t \in [0, h \cdot N]$, using a single trained DAE-PINN with time step h. We remark that one can easily extend Algorithm 2 to work with multiple trained discrete PINNs with possibly different time steps h. Such a strategy can be applied, for example, to problems with multiple time scales (e.g., transients and steady-state).

Algorithm 2 Simulating DAEs for long-time horizons

Require: The number of time steps N, and the DAE-PINN with *trained* parameters θ^* and time step h.

1: Let the initial condition of (1a), y_0, be the input to the DAE-PINN, *i.e.*, $y_n = y_0$.
2: **for** k = 1,...,N **do**
3: compute the forward pass using the proposed framework, *i.e.*,

$$y_n \mapsto [\xi_1^{\theta^*}, \ldots, \xi_\nu^{\theta^*}, y_{n+1}^{\theta^*}] =: Y_k^{\theta^*}$$
$$y_n \mapsto [\zeta_1^{\theta^*}, \ldots, \zeta_\nu^{\theta^*}, z_{n+1}^{\theta^*}] =: Z_k^{\theta^*}$$

4: update the input using the predicted value $y_{n+1}^{\theta^*}$, *i.e.*,

$$y_n \longleftarrow y_{n+1}^{\theta^*}$$

5: **end for**
6: The Algorithm computes the solution trajectory in the time interval $[0, h \cdot N]$. Such a solution trajectory is obtained by concatenating the outputs from all forward passes, *i.e.*, $\{Y_k^{\theta^*}\}_{k=1}^N$ and $\{Z_k^{\theta^*}\}_{k=1}^N$

4 Numerical experiments

This section contains a systematic study on a three-bus power (Fig. 3) network that aims to demonstrate the performance of our DAE-PINN framework. We also refer the interested reader to Appendix A for an illustrative example that uses the Van der Pol system [16, 49] to demonstrate how a dynamical system transitions from being non-stiff to stiff and, finally, to a system of differential-algebraic equations.

4.1 Three-bus power network

We consider the following three-bus system (a slack bus, a generator bus, and a load bus), which has been used for energy function analysis [56], cascading failure prediction [43], and reduced-order modeling [27]. The three-bus system is depicted in Fig. 3 and described by the following set of nonlinear DAEs [56]

$$\dot{\omega}_1 = (1/M_1)(-D\omega_1 + f_1 + f_2) \tag{9a}$$
$$\dot{\omega}_2 = (1/M_2)(-D\omega_2 - f_1) \tag{9b}$$
$$\dot{\delta}_2 = \omega_2 - \omega_1 \tag{9c}$$
$$\dot{\delta}_3 = -(\omega_1 - f_2/D_l) \tag{9d}$$
$$0 = -(1/V_3)(g_1), \tag{9e}$$

where $y = (\omega_1, \omega_2, \delta_2, \delta_3)^\top$ are the dynamic states, $z = V_3$ the only unknown algebraic state, and

$$f_1 = B_{12}V_1V_2\sin(\delta_2) + B_{23}V_2V_3\sin(\delta_2 - \delta_3) + P_g,$$
$$f_2 = B_{13}V_1V_3\sin(\delta_3) + B_{23}V_2V_3\sin(\delta_3 - \delta_2) + P_l,$$
$$g_1 = (B_{13} + B_{23})V_3^2 - B_{13}V_1V_3\cos(\delta_3) - B_{23}V_2V_3\cos(\delta_3 - \delta_2) + Q_l.$$

We fix the parameters of the power network to the following values $M_1 = 0.52$, $M_2 = 0.0531$, $D = 0.05$,

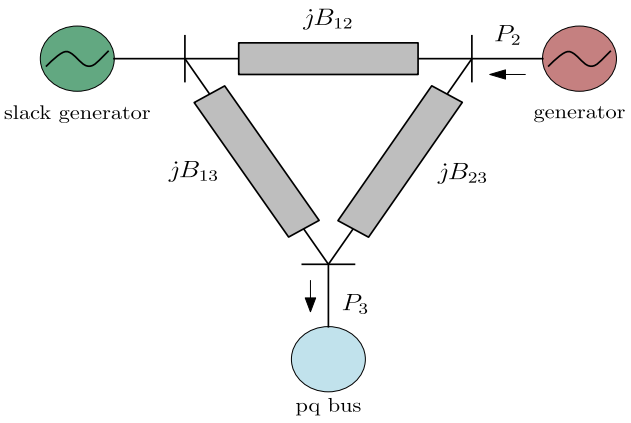

Fig. 3 Three-bus two-generator power network [56]

$D_l = 0.005$, $V_1 = 1.02$, $V_2 = 1.05$, $B_{12}, B_{13}, B_{23} = 10$, $P_g = -2.0$, $P_l = 3.0$, and $Q_l = 0.1$.

4.2 Neural networks, hyper-parameters, and learning protocols

We implemented DAE-PINN using PyTorch and published all the codes on GitHub. All the experiments presented in this section were trained by minimizing the loss function $\mathcal{L}(\theta\mathcal{T})$ (7) using the Adam [24] optimizer with default hyper-parameters and initial learning rate $\eta = 10^{-3}$. We reduced the learning rate whenever the value of the loss function $\mathcal{L}$ reached a plateau or started to increase. The training and test datasets consist of initial conditions sampled uniformly at random and as follows: $\omega_1(0), \omega_2(0) \sim \mathcal{U}(-\pi, \pi)$ and $\delta_2(0), \delta_3(0) \sim \mathcal{U}(-0.1, 0.1)$.

The neural network that approximates the mapping $y_n \mapsto (\xi_1, \ldots, \xi_v, y_{n+1})$ (i.e., dynamic equations) and the neural network that approximates the mapping $y_n \mapsto (\zeta_1, \ldots, \zeta_v, z_{n+1})$ (i.e., algebraic equations) were implemented using the improved fully-connected architecture proposed in [52], which has the following forward pass:

$$
\begin{aligned}
&U = \phi(XW^1 + b^1), V = \phi(XW^2 + b^2)\\
&H^{(1)} = \phi(XW^{z,1} + b^{z,1})\\
&Z^{(k)} = \phi(H^{(k)}W^{z,k} + b^{z,k}), \quad k = 1, \ldots, d\\
&H^{(k+1)} = (1 - Z^{(k)}) \odot U + Z^{(k)} \odot V, \quad k = 1, \ldots, d\\
&f_\theta(x) = H^{(d+1)}W + b,
\end{aligned}
$$

Here, X is the input tensor to the neural network, d is the number of hidden layers (i.e., the network's depth), $\odot$ is the Hadamard or element-wise product, and ϕ, in this paper, is a point-wise sinusoidal activation function. We also assume that each hidden layer has a width w. The trainable parameters of this novel network architecture, which we initialize using the Glorot normal algorithm, are collected in the following set:

$$\theta = \{W^1, b^1, W^2, b^2, \{W^{z,l}, b^{z,l}\}_{l=1}^{d}, W, b\}.$$

Our experiments show that this novel architecture outperforms the conventional fully connected architecture. This is because it explicitly accounts for the multiplicative interactions between different inputs and enhances hidden-state representation with residual connections [52]. Let us conclude this subsection with the following remark.

Remark 3 *Output feature layer for the algebraic equation.* The term $(1/V_3)$ in (9e) may lead to the loss for the algebraic variables $\mathcal{L}_g$ being a few orders of magnitude larger than the loss for the dynamic variables $\mathcal{L}_f$. We have observed empirically that such an imbalance compromises gradient descent optimization. To mitigate such an issue, we added the following output feature layer for the neural network associated with the algebraic equations:

$$(\zeta_1, \ldots, \zeta_v, z_{n+1}) \mapsto \text{softplus}(\zeta_1, \ldots, \zeta_v, z_{n+1}).$$

Note that the above feature layer constraints the bus voltage to be nonnegative, i.e., $V_3 > 0$.

4.3 Convergence experiments

In this subsection, we investigate how the network architecture, the network size, and the training dataset affect the training convergence of DAE-PINN. To this end, we train DAE-PINN with a time step $h = 0.1$ for 50,000 epochs.

4.3.1 Network architecture

Our first convergence experiment evaluates which architecture, stacked or unstacked, provides better convergence results for the training of DAE-PINN. The stacked architecture uses a neural network (width $w = 25$ and depth $d = 4$ layers) for each dynamic and algebraic state. On the other hand, the unstacked architecture uses one neural network (width $w = 100$ and depth $d = 4$ layers) for all the dynamic states and another neural network (width $w = 25$ and depth $d = 4$ layers) for the algebraic state. Figure 4a shows the results of running this experiment 10 times. We observe that the unstacked architecture provides us with the best training performance.

4.3.2 Network size

This experiment evaluates the effect of the size of the networks during training. More specifically, we use an unstacked architecture (to eliminate the network architecture effect) to verify the training convergence while varying the width and depth of the neural networks. Figure 4b illustrates the results when we vary the width (depth fixed to $d = 2$ layers). We note that increasing the width from 10 to 200 decreases the train and test errors, but the errors increase instead when we further increase the width. On the other hand, Fig. 4c shows the results when we vary the depth (width fixed to $w = 100$). We observe that the train and test errors reach a minimum when we set the depth to $d = 4$ or $d = 8$ hidden layers.

4.3.3 Training dataset

Our last experiment investigates how the size of the training dataset, i.e., the number of training examples $N_{\mathcal{T}} \equiv |\mathcal{T}|$, affects the training convergence of DAE-PINN. To eliminate the effect of the architecture and size of the neural networks, we choose unstacked architectures with a

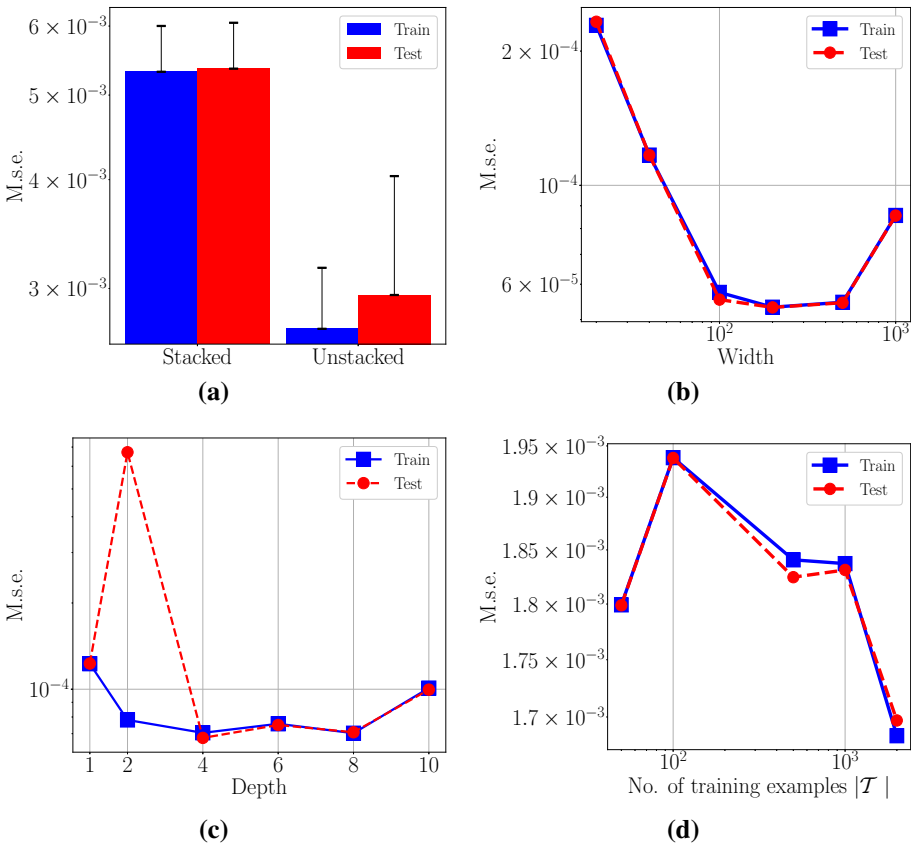

Fig. 4 Convergence experiments. **a** stacked versus unstacked architectures. **b** Network width. **c** Network depth. **d** Number of training examples

depth of $d = 4$ for the neural networks of the dynamic and algebraic states. Further, we select a width of $w = 100$ (resp. $w = 40$) for the neural network of the dynamic (resp. algebraic) states. The results illustrate that (see Fig. 4d) including more training examples (initial conditions), in general, leads to smaller train and test errors. We conclude this section by describing the characteristics of our best DAE-PINN model. This best model trains with $N_{\mathcal{T}} = 2000$ initial condition points sampled from the state-space, evaluates the performance of DAE-PINN every 1000 epochs using a test dataset with 1500 initial conditions not included in the training dataset and uses unstacked neural network architectures with $d = 4$ hidden layers. Moreover, for this best DAE-PINN model, the neural network representing the dynamic (resp. algebraic) states has a width of $w = 100$ (resp. $w = 40$).

4.4 Results for the best DAE-PINN model

In this subsection, we verify the effectiveness of DAE-PINN in performing a long-time simulation of DAEs using Algorithm 2. To this end, we train the best DAE-PINN model with time-step $h = 0.1$ using the penalty-method described in Algorithm 1 with hyper-parameters $w_f^0 = w_g^0 = 1$ and $\beta = 2$.

Figure 5 presents a simulated DAE trajectory for $N = 80$ time steps, corresponding to a representative initial condition selected uniformly at random from the test dataset. We note excellent agreement between the simulated trajectory and the true trajectory (obtained by integrating (9) using conventional numerical methods [53]). To better understand the long-time simulation accuracy of the DAE-PINN framework, we sample 100 initial conditions from the test dataset and compute the mean and standard deviation of each state variable's L^2 relative error. Table 1 reports the L^2 relative errors of each state variable. From the reported results, we conclude that DAE-PINN can simulate DAEs for long-time horizons with excellent accuracy.

4.5 Comparison with other numerical integration schemes

In this subsection, we design a brief comparison study to illustrate that having more IRK stages ν enhances the robustness against error accumulation. Let us remark first that using Algorithm 2 to simulate DAEs for long-time horizons may accumulate errors, jeopardizing accuracy. Thus, our goal in this section is to compare the prediction accuracy of the proposed best DAE-PINN model (enabling the IRK scheme with $\nu = 100$ stages) with two other DAE-

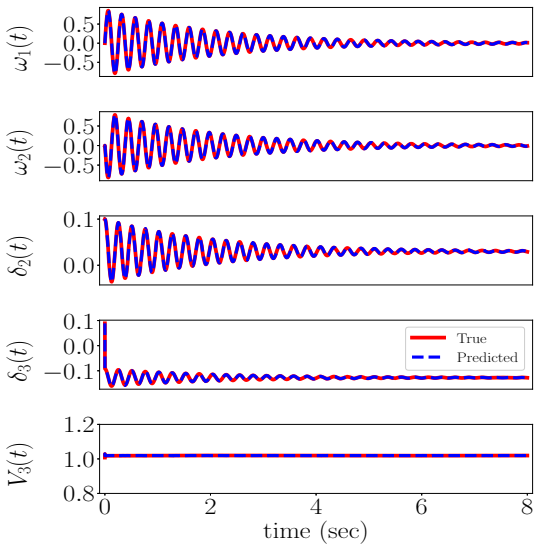

Fig. 5 Predicted and true solution trajectories of the DAEs describing the three-bus power network dynamics (9) within the simulation time interval $[0, N \cdot h] = [0, 8]$ seconds for a initial condition sampled from the test dataset

PINN models. The first model is a DAE-PINN that encodes the classical backward Euler integration method [19] into its loss function. The second model is a DAE-PINN that enables the Gauss–Legendre IRK with $v = 3$ stages, which is probably the largest IRK numerical scheme that is consistent, stable, and with reasonable implementation costs [19].

4.5.1 DAE-PINN enabling backward Euler

We built a DAE-PINN that encodes the backward Euler integration method into its loss function. To this end, we used the same neural network architecture and input/hidden layers' structure as for the best DAE-PINN model (see Sects. 4.3 and 4.4). Note that the output layers for this DAE-PINN are $(y^{\theta}_{n+1}, z^{\theta}_{n+1})$. We trained this DAE-PINN using the learning protocols described in Sect. 4.2 and the following loss function that enables the backward Euler [19] method:

$$\mathcal{L}(\theta; \mathcal{T}) = w_f \mathcal{L}_f(\theta; \mathcal{T}) + w_g \mathcal{L}_g(\theta; \mathcal{T}).$$

In the above, w_f and w_g are the weights,

$$\mathcal{L}_f(\theta; \mathcal{T}) = \frac{1}{N_{\mathcal{T}}} \sum_{k=1}^{N_{\mathcal{T}}} \|y_{k,k} - y_{n,k}(\theta)\|_2^2,$$

where

$$y_{n,k}(\theta) := y^{\theta}_{n+1,k} - hf(y^{\theta}_{n+1,k}, z^{\theta}_{n+1,k}),$$

and

$$\mathcal{L}_g(\theta; \mathcal{T}) = \frac{1}{N_{\mathcal{T}}} \sum_{k=1}^{N_{\mathcal{T}}} \|g(y^{\theta}_{n+1,k}, z^{\theta}_{n+1,k})\|_2^2.$$

4.5.2 DAE-PINN enabling Gauss–Legendre IRK

We built and trained a DAE-PINN for enabling the Gauss–Legendre IRK [19], i.e., an IRK with only $v = 3$ stages. Note that the only differences between this DAE-PINN and the best DAE-PINN model of Sect. 4.4 are the (i) structure of the output layers, which depends on the number of stages $v = 3$ and (ii) *known* parameters $\{a_{j,i}, b_j, c_i\}$. We refer the interested reader to [19] for the exact values of the aforementioned parameters.

4.5.3 Comparison results

We used the three trained DAE-PINN models to simulate (see Algorithm 2) the power network DAEs. In particular, we simulated the solution trajectories for $N = 80$ time steps (with time-step $h = 0.1$) and starting from an initial condition $y(0) \in \{y : \omega_1, \omega_2 \in [-\pi, \pi]$ and $\delta_1, \delta_2 \in [-0.1, 0.1]\}$ that does not belong to the training dataset.

Figure 6 illustrates the predicted solution trajectories from the three DAE-PINNs. As described in the previous section, the best DAE-PINN model (with IRK of $v = 100$ stages) generalizes well and effectively simulates the solution trajectories, i.e., without accumulating errors. For the *dynamic* variables y, the backward Euler DAE-PINN fails to track the oscillatory response of y. At the same time, the Gauss–Legendre DAE-PINN follows the oscillations except for the angle state δ_3. On the other hand, for the *algebraic* variable $z = V_3$, the backward Euler DAE-PINN tracks the instantaneous response. The Gauss–Legendre DAE-PINN, however, accumulates errors significantly.

We also illustrate (see Fig. 7) the L^2 relative error as a function of N, i.e., the number of time steps, for the slack machine speed ω_1 and the load bus angle δ_3. In both cases, the best DAE-PINN model is robust and does not accumulate errors. The backward Euler DAE-PINN always suffers from error accumulation, and the Gauss–Legendre DAE-PINN accumulates errors significantly for the load bus angle δ_3. In conclusion, a DAE-PINN that enables IRK

Table 1 Mean and standard deviation of the L^2 relative error of the long-time simulation of 100 initial conditions sampled from the test dataset

	ω_1	ω_2	δ_2	δ_3	V_3
Mean	0.0382	0.0381	0.0093	0.0011	0.0002
St. dev.	1.01*e*-2	1.07*e*-2	2.44 *e*-3	2.96*e*-4	2.98*e*-07

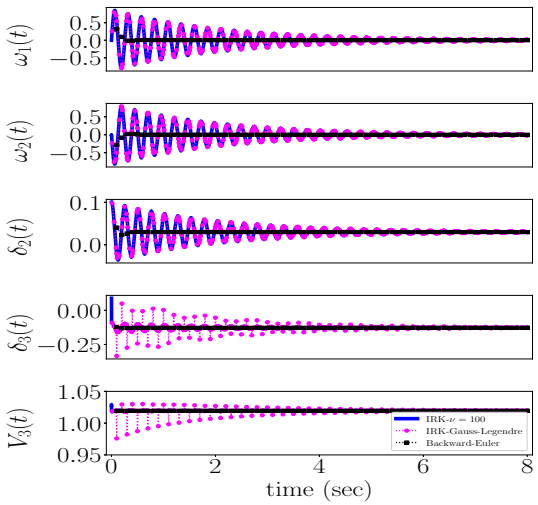

Fig. 6 Comparing the long-time simulation accuracy of DAE-PINNs enabling (i) IRK scheme with $\nu = 100$ stages, (ii) IRK Gauss–Legendre scheme, and (iii) backward Euler method

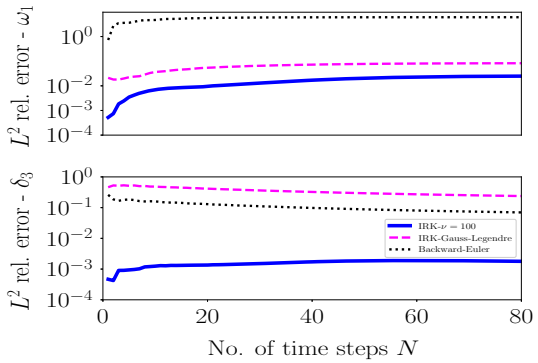

Fig. 7 L^2 relative error for the slack generator speed ω_1 and load bus angle δ_3 as a function of the number of time steps N

methods with a large number of stages ν provides robustness when integrating DAEs using Algorithm 2. This is because having a large number of stages controls the errors propagated to the next integration/simulation step.

5 Discussion

5.1 On extending our framework to large-scale power networks

Developing deep learning methods for simulating large-scale scientific and engineering systems remains an open problem. Thus, the straightforward application of DAE-PINN for simulating large-scale power networks is not feasible. We, however, believe that, similar to the author's previous work [28], our proposed framework can be used in a plug-and-play fashion and replace the numerical solvers for the DAEs describing the individual components of the network (e.g., generators). To showcase such plug-and-play ability, in our future work, we plan to construct a surrogate model that can predict the response of a medium-size power network whose components are pre-trained using our framework. To this end, our method must generalize to unseen events and even predict unstable behaviors. The fully connected neural networks used in this paper may not be powerful enough for such a challenge. Thus, it is also part of our future work to enhance our method using more sophisticated architectures, which we briefly describe next.

5.2 On using more sophisticated Neural Network architectures

In this paper, to simulate DAEs over a long-time horizon, we employed a modified version of the conventional fully connected neural network architecture. However, we realize that other architectures may increase our ability to simulate long-time dependencies [58]. Thus, in our future work, we plan to implement DAE-PINN using neural networks that can generalize well to unseen events. In particular, we plan to employ the state-of-the-art deep operator neural network (DeepONet) [29], a neural network that approximates nonlinear operators (a mapping from functions to functions), which has shown great potential to reduce the generalization error significantly. We can apply DeepONets to our framework by noting that integration is an operation of the form: $T_h : x(\cdot) \mapsto x(\cdot + h)$ where the time-step h is a parameter.

5.3 On the inverse problem

We remark that extending the proposed framework to learn unknown but identifiable parameters of DAEs is straightforward (see [41] for more details). Furthermore, in [34], the authors already used a physics-informed continuous deep learning model to learn unknown power network parameters. It is, however, unclear whether the authors' framework learns the stiff nonlinear DAEs or a non-stiff ODE-based approximation of the power network dynamics. As reported in [21] (and also our experience with physics-informed continuous models), the learning process of stiff ODEs and DAEs using physics-informed continuous models is extremely unstable. Thus, it requires a problem-dependent solution to avoid the failure of gradient-based training.

5.4 On the stochastic setting

With the increasing penetration of renewable resources, the operating conditions for power networks are becoming

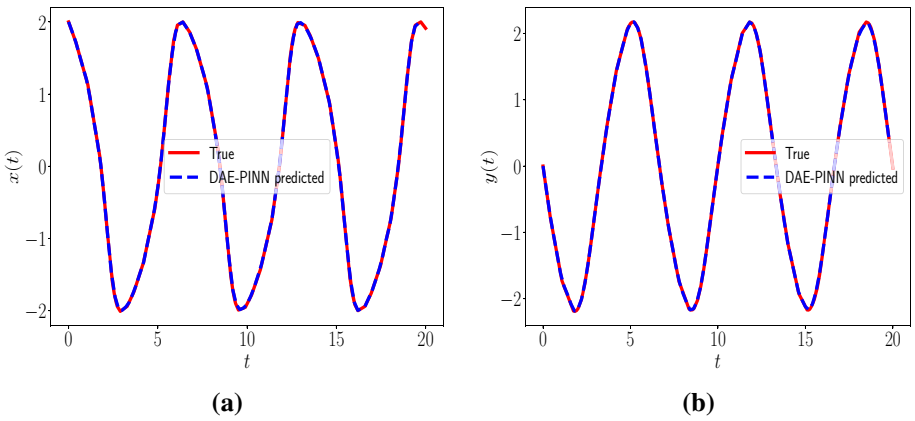

Fig. 8 Comparing the long-time simulation accuracy of DAE-PINN for the non-stiff Van der Pol system (parameter $\mu = 1$ and time-step $h = 0.2$). **a** $x(t)$ trajectory. **b** $y(t)$ trajectory

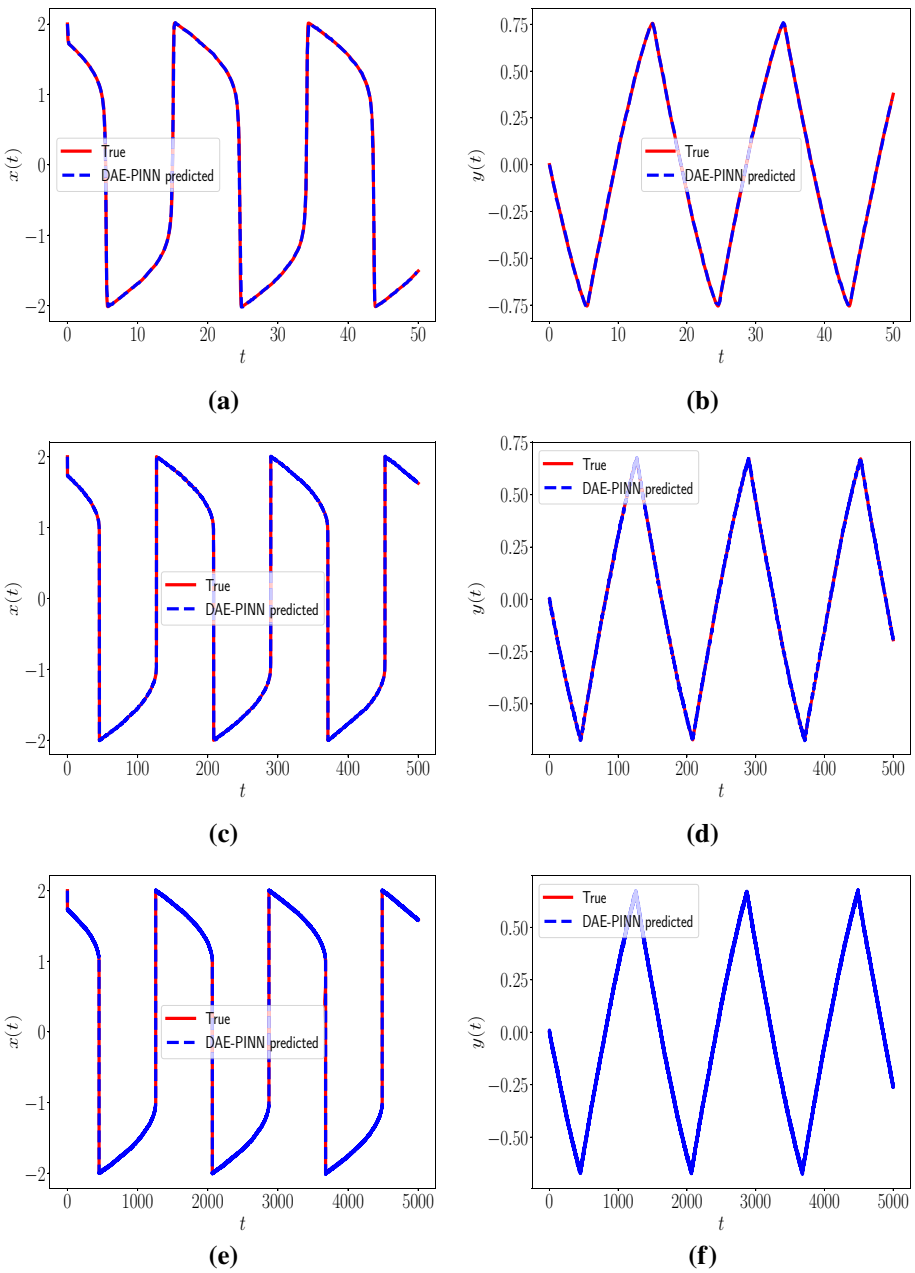

Fig. 9 Comparing the long-time simulation accuracy of DAE-PINN for the stiff Van der Pol system ($\mu \in \{10, 100, 1000\}$ and time-step $h = 0.2$). **a** $x(t)$ trajectory for $\mu = 10$. **b** $y(t)$ trajectory for $\mu = 10$. **c** $x(t)$ trajectory for $\mu = 100$. **d** $y(t)$ trajectory for $\mu = 100$. **e** $x(t)$ trajectory for $\mu = 1000$. **e** $y(t)$ trajectory for $\mu = 1000$

more uncertain. Thus, developing an online dynamic security assessment tool that considers such a stochastic environment is necessary. To this end, in our future work, we will develop a deep learning framework that learns and simulates the stochastic differential-algebraic equations describing power network dynamics for a given distribution of initial conditions and a set of uncertain parameters.

6 Conclusion

We developed DAE-PINN, a deep learning framework for learning and simulating the set *differential-algebraic equations* (DAE) that describes power networks. DAE-PINN consists of a discrete *physics-informed neural network* model that enables employing arbitrarily accurate implicit Runge–Kutta schemes with a large number of stages. Moreover, we implemented a penalty-based that enforces DAE-PINN to satisfy the DAEs as approximate hard constraints. We then proposed Algorithm 2, which uses the trained DAE-PINN to simulate DAEs over long-time horizons. Finally, we demonstrated the effectiveness of our proposed framework using a three-bus power network.

Appendix A: The Van der Pol system

In this appendix, we consider the classical Van der Pol (VDP) system [16, 49]:

$$\ddot{x} + \mu(x^2 - 1) + x = 0,$$

where $(x, \dot{x})$ is the state vector and μ is the system's parameter. To facilitate our analysis, let us transform the VDP system as follows. Consider the following identity:

$$\ddot{x} + \mu\dot{x}(x^2 - 1) = \frac{d}{dt}\left(\dot{x} + \mu\left[\frac{1}{3}x^3 - x\right]\right).$$

If we let $\varphi(x) := \frac{1}{3}x^3 - x$ and $z = \dot{x} + \mu\varphi(x)$, the above identity and the VDP system imply

$$\dot{z} = \ddot{x} + \mu\dot{x}(x^2 - 1) = -x.$$

If we also let $y := \frac{z}{\mu}$, then we can write the VDP system as follows:

$$\dot{x} = \mu[y - \varphi(x)] \tag{A1}$$

$$\dot{y} = -\frac{x}{\mu}. \tag{A2}$$

Now suppose the initial condition (x_0, y_0) is not too close to the line described by the function $y = \varphi(x)$, i.e., assume $y - \varphi(x) \sim O(1)$. Then Eq. (A1) implies that $|\dot{x}| \sim O(\mu)$ and Eq. (A2) implies that $|\dot{y}| \sim O(\mu^{-1})$.

A.1 Non-stiff Van Der Pol system

It is well known [49] that whenever $\mu = 1$, the Van Der Pol system is non-stiff, i.e., $|\dot{x}|, |\dot{y}| \sim O(1)$, the time-scales are similar. Figure 8 presents the non-stiff solution trajectories for the VDP system with initial condition $(x_0, y_0) = (2, 0)$. Figure 8 also presents the predicted trajectory using a modified version of DAE-PINN. Note that this version of DAE-PINN is trained to satisfy Eqs. (5a) and (5c) for the VDP system. The results illustrate that the proposed DAE-PINN can easily simulate the dynamic response of the non-stiff VDP system.

A.2 Increasing stiffness

To increase the stiffness of the VDP system, one must increase the value of the parameter μ. Clearly, $|\dot{x}| \sim O(\mu) \gg 1$ and $|\dot{y}| \sim O(\mu^{-1}) \ll 1$ when μ increases and is much bigger than one. In particular, the velocity in the horizontal direction is considerable, while the velocity in the vertical direction is minimal [49]. These two widely separated time scales are a distinctive property of stiff systems. Figure 9 illustrates the solution trajectories for the VDP system with initial condition $(x_0, y_0) = (2.0, 0.0)$ and parameter $\mu \in \{10, 100, 1000\}$. Note that we can observe both time scales in the graph of the $x(t)$ trajectory. Figure 9 also illustrates the prediction of a modified DAE-PINN. This DAE-PINN can effectively predict/simulate the trajectory even for the very stiff VDP system, i.e., when $\mu = 1000$.

A.3 Differential algebraic equations and infinite stiffness

If we let the parameter increase to infinity, i.e., if we let $\mu \to \infty$, then the stiffness increases to infinity. Then, the differential equations of the VDP system become the following differential-algebraic equations:

$$0 = y - \varphi(x)$$

$$\dot{y} = -\frac{1}{\mu}x$$

Note that the claim that DAEs present a "form" of infinite stiffness [23] can be easily inferred from the previous analysis.

Author contributions CM: Conceptualization, Methodology, Investigation, Formal Analysis, Writing—Original Draft. GL: Conceptualization, Supervision, Writing—Review, Editing, Funding Acquisition.

Funding This work was supported by the National Science Foundation (DMS-1555072, DMS-1736364, DMS-2053746, and DMS-2134209), and Brookhaven National Laboratory Subcontract 382247,

and U.S. Department of Energy (DOE) Office of Science Advanced Scientific Computing Research program DE-SC0021142.

Availability of data and materials The data used for training the proposed methods of this article can be generated using the code available in Github (https://github.com/cmoyacal/DAE-PINNs).

Code availability The code for this article is available in GitHub (https://github.com/cmoyacal/DAE-PINNs).

Declarations

Ethical approval Not applicable.

Consent to participate Not applicable.

Consent for publication Not applicable.

Conflict of interest The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Alvarado F, Oren S (2002) Transmission system operation and interconnection. National transmission grid study–Issue papers, pp A1–A35
2. Aristidou P, Fabozzi D, Van Cutsem T (2013) Dynamic simulation of large-scale power systems using a parallel schur-complement-based decomposition method. IEEE Trans Parallel Distrib Syst 25(10):2561–2570
3. Baker N, Alexander F, Bremer T et al (2019) Workshop report on basic research needs for scientific machine learning: Core technologies for artificial intelligence. Tech. rep, USDOE Office of Science (SC), Washington, DC (United States)
4. Brunton SL, Proctor JL, Kutz JN (2016) Discovering governing equations from data by sparse identification of nonlinear dynamical systems. Proc Natl Acad Sci 113(15):3932–3937
5. Brunton SL, Proctor JL, Kutz JN (2016) Sparse identification of nonlinear dynamics with control (sindyc). IFAC-PapersOnLine 49(18):710–715
6. Chao H (2002) Implementation of parallel-in-time Newton method for transient stability analysis on a message passing multicomputer. In: Proceedings of international conference on power system technology. IEEE, pp 1239–1243
7. Chen J, Li K, Bilal K et al (2018) A bi-layered parallel training architecture for large-scale convolutional neural networks. IEEE Trans Parallel Distrib Syst 30(5):965–976
8. Chen J, Li K, Philip SY (2021) Privacy-preserving deep learning model for decentralized vanets using fully homomorphic encryption and blockchain. IEEE Trans Intell Transp Syst
9. Chen RT, Rubanova Y, Bettencourt J et al (2018) Neural ordinary differential equations. arXiv preprint arXiv:1806.07366
10. Chiang HD (2011) Direct methods for stability analysis of electric power systems: theoretical foundation, BCU methodologies, and applications. Wiley, New York
11. Chiang HD, Wu FF, Varaiya PP (1994) A bcu method for direct analysis of power system transient stability. IEEE Trans Power Syst 9(3):1194–1208
12. Chung E, Leung WT, Pun SM et al (2021) A multi-stage deep learning based algorithm for multiscale model reduction. J Comput Appl Math 394(113):506
13. Fabozzi D, Chieh AS, Haut B et al (2013) Accelerated and localized newton schemes for faster dynamic simulation of large power systems. IEEE Trans Power Syst 28(4):4936–4947
14. Gupta A, Gurrala G, Sastry P (2018) An online power system stability monitoring system using convolutional neural networks. IEEE Trans Power Syst 34(2):864–872
15. Gurrala G, Dimitrovski A, Pannala S et al (2015) Parareal in time for fast power system dynamic simulations. IEEE Trans Power Syst 31(3):1820–1830
16. Hairer E, Lubich C, Roche M (2006) The numerical solution of differential-algebraic systems by Runge–Kutta methods, vol 1409. Springer, Berlin
17. He M, Zhang J, Vittal V (2013) Robust online dynamic security assessment using adaptive ensemble decision-tree learning. IEEE Trans Power Syst 28(4):4089–4098
18. Hiskens IA, Hill DJ (1989) Energy functions, transient stability and voltage behaviour in power systems with nonlinear loads. IEEE Trans Power Syst 4(4):1525–1533
19. Iserles A (2009) A first course in the numerical analysis of differential equations, vol 44. Cambridge University Press, New York
20. James J, Hill DJ, Lam AY et al (2017) Intelligent time-adaptive transient stability assessment system. IEEE Trans Power Syst 33(1):1049–1058
21. Ji W, Qiu W, Shi Z et al (2020) Stiff-pinn: physics-informed neural network for stiff chemical kinetics. arXiv preprint arXiv:2011.04520
22. Karniadakis GE, Kevrekidis IG, Lu L et al (2021) Physics-informed machine learning. Nat Rev Phys 3(6):422–440
23. Kim S, Ji W, Deng S et al (2021) Stiff neural ordinary differential equations. arXiv preprint arXiv:2103.15341
24. Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980
25. Kundur P (2007) Power system stability. Power system stability and control pp 7–1
26. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444
27. Li J, Stinis P (2019) Model reduction for a power grid model. arXiv preprint arXiv:1912.12163
28. Li J, Yue M, Zhao Y et al (2020) Machine-learning-based online transient analysis via iterative computation of generator dynamics. In: 2020 IEEE international conference on communications, control, and computing technologies for smart grids (SmartGridComm). IEEE, pp 1–6
29. Lu L, Jin P, Pang G et al (2021) Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. Nat Mach Intell 3(3):218–229
30. Lu L, Meng X, Mao Z et al (2021) Deepxde: a deep learning library for solving differential equations. SIAM Rev 63(1):208–228
31. Lu L, Pestourie R, Yao W et al (2021c) Physics-informed neural networks with hard constraints for inverse design. arXiv preprint arXiv:2102.04626
32. Luenberger DG (1973) Introduction to linear and nonlinear programming, vol 28. Addison-Wesley, Reading
33. Milano F (2010) Power system modelling and scripting. Springer, Berlin
34. Misyris GS, Venzke A, Chatzivasileiadis S (2020) Physics-informed neural networks for power systems. In: 2020 IEEE power & energy society general meeting (PESGM). IEEE, pp 1–5
35. Moya C, Zhang S, Yue M et al (2022) Deeponet-grid-uq: A trustworthy deep operator framework for predicting the power grid's post-fault trajectories. arXiv preprint arXiv:2202.07176
36. Pai M, Padiyar K, Radhakrishna C (1981) Transient stability analysis of multi-machine ac/dc power systems via energy-function method. IEEE Trans Power Appar Syst 12:5027–5035

37. Park B, Sun K, Dimitrovski A et al (2021) Examination of semi-analytical solution methods in the coarse operator of parareal algorithm for power system simulation. IEEE Trans Power Syst 36(6):5068–5080
38. Qin T, Wu K, Xiu D (2019) Data driven governing equations approximation using deep neural networks. J Comput Phys 395:620–635
39. Qin T, Chen Z, Jakeman JD et al (2021) Data-driven learning of nonautonomous systems. SIAM J Sci Comput 43(3):A1607–A1624
40. Raissi M, Perdikaris P, Karniadakis GE (2018) Multistep neural networks for data-driven discovery of nonlinear dynamical systems. arXiv preprint arXiv:1801.01236
41. Raissi M, Perdikaris P, Karniadakis GE (2019) Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J Comput Phys 378:686–707
42. Roche M (1989) Implicit Runge–Kutta methods for differential algebraic equations. SIAM J Numer Anal 26(4):963–975
43. Roth J, Barajas-Solano DA, Stinis P et al (2021) A kinetic Monte Carlo approach for simulating cascading transmission line failure. SIAM J Multiscale Model Simul 19(1)
44. Rudin W et al (1976) Principles of mathematical analysis, vol 3. McGraw-Hill, New York
45. Schaeffer H (2017) Learning partial differential equations via data discovery and sparse optimization. Proc R Soc A Math Phys Eng Sci 473(2197):20160446
46. Schainker R, Miller P, Dubbelday W et al (2006) Real-time dynamic security assessment: fast simulation and modeling applied to emergency outage security of the electric grid. IEEE Power Energ Mag 4(2):51–58
47. Shu J, Xue W, Zheng W (2005) A parallel transient stability simulation for power systems. IEEE Trans Power Syst 20(4):1709–1717
48. Stott B (1979) Power system dynamic response calculations. Proc IEEE 67(2):219–241
49. Strogatz SH (2018) Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering. CRC Press, Boca Raton
50. Tomim MA, Marti JR, Wang L (2009) Parallel solution of large power system networks using the multi-area thévenin equivalents (mate) algorithm. Int J Electr Power Energy Syst 31(9):497–503
51. Varaiya P, Wu FF, Chen RL (1985) Direct methods for transient stability analysis of power systems: Recent results. Proc IEEE 73(12):1703–1715
52. Wang S, Teng Y, Perdikaris P (2020) Understanding and mitigating gradient pathologies in physics-informed neural networks. arXiv preprint arXiv:2001.04536
53. Wanner G, Hairer E (1996) Solving ordinary differential equations II, vol 375. Springer Berlin Heidelberg, New York
54. Yazdani A, Lu L, Raissi M et al (2020) Systems biology informed deep learning for inferring parameters and hidden dynamics. PLoS Comput Biol 16(11):e1007575
55. Zhao T, Yue M, Wang J (2022) Structure-informed graph learning of networked dependencies for online prediction of power system transient dynamics. IEEE Trans Power Syst
56. Zheng H, DeMarco CL (2010) A bi-stable branch model for energy-based cascading failure analysis in power systems. In: North American power symposium 2010. IEEE, pp 1–7
57. Zheng Y, Hu C, Lin G et al (2022) Glassoformer: a query-sparse transformer for post-fault power grid voltage prediction. In: ICASSP 2022–2022 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, pp 3968–3972
58. Zhou H, Zhang S, Peng J et al (2020) Informer: beyond efficient transformer for long sequence time-series forecasting. arXiv preprint arXiv:2012.07436
59. Zhu L, Hill DJ, Lu C (2019) Hierarchical deep learning machine for power system online transient stability prediction. IEEE Trans Power Syst 35(3):2399–2411