# DAE-PINN: A Physics-Informed Neural Network Model for Simulating Differential Algebraic Equations with Application to Power Networks

Christian Moya, *Member, IEEE,* and Guang Lin, *Senior Member, IEEE*

*Abstract*—**Deep learning-based surrogate modeling is becoming a promising approach for learning and simulating dynamical systems. Deep-learning methods, however, find very challenging learning stiff dynamics. In this paper, we develop DAE-PINN, the first effective deep-learning framework for learning and simulating the solution trajectories of *nonlinear differential-algebraic equations* (DAE), which present a form of *infinite stiffness* and describe, for example, the dynamics of power networks. 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 and simulating the solution trajectories of a three-bus power network.**

*Index Terms*—**Deep learning, Data-driven scientific computing, Nonlinear differential-algebraic equations, Implicit Runge-Kutta.**

## I. 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. This transformation, however, pushes the power network to operate under a more diversified set of operating conditions and contingencies that could potentially compromise its security.

To assess the power network's dynamic security [1], operators implement an offline procedure that seeks to predict whether the power network will remain safely operating after facing a single contingency (e.g., the disconnection of a generator) from a set of credible contingencies. Such a procedure is known as the $N-1$ criteria [2] and requires simulating the power network's dynamic response.

Simulating the power network's dynamic response requires integrating a set of nonlinear differential-algebraic equations (DAE) [1]. Solving this set of DAEs is, however, a challenging task. Indeed, the classical explicit integration schemes fail catastrophically on such a task [3]. As a result, most commercial solvers for DAEs use numerically stable schemes

C. Moya and G. Lin are with the Department of Mathematics, Purdue University, West Lafayette, IN, 47907 USA e-mail: {cmoyacal, guanglin}@purdue.edu.

to integrate the dynamic equations and iterative schemes to solve the algebraic equations [4]. However, the computational cost and memory required to integrate DAEs are very high and constitute the main obstacle to deploying dynamic security assessment online [5]. However, with the transformation the power network now faces, soon, it will become imperative for electric utilities to assess security online, which calls for the faster integration/simulation of DAEs.

Motivated by the above power network 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 [6], and promises to also 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. Most conventional DL methods (e.g., convolutional or recurrent neural networks) lack robustness and generalization capabilities in such a small data regime.

In recent years, the field of scientific machine learning [7] has provided us with a series of new transformative works [8], [9], [10], [11] aiming at learning the differential equations describing dynamical systems and, hence providing us with an efficient alternative to traditional costly numerical solvers. Behind most of these transformative works lies the idea of using the physical laws that govern these dynamical systems [8]. Such prior information acts as a regularizing agent, limiting the space of possible solutions and enabling generalizing well even when the amount of data is small. Admittedly, there is still much work needed to scale physics-informed deep learning methods so that they can become *accurate* surrogate models of large-scale dynamical systems. In particular, these accurate surrogate models must (i) predict solution trajectories for a large set of initial conditions and (ii) maintain physical accuracy for long-time horizons.

Despite the success of scientific machine learning for learning the solution trajectories of ordinary differential equations [11], developing a DL-based framework for learning and simulating the solution trajectories of nonlinear differential-algebraic equations remains an open problem. This is because DAEs present a form of infinite stiffness [12] that may produce gradient pathologies [13] 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 stiff differential equations were presented in [14] and [12]. In [14], the authors show that continuous PINN models fail to learn stiff ODEs and propose using quasi-steady-state assumptions to derive a simpler model more suitable for PINNs. In [12], Kim *et al.* modified neural ordinary differential equations [15] so that they can learn the solution trajectories of stiff problems for long-time horizons. Both of the above methods have their merits, but, as presented, they are not suitable for learning the solution trajectories of the DAEs studied in this paper.

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.

We detail our contributions next.

1) We design a deep learning (DL) framework (DAE-PINN - Sections III-A and III-B) 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, our framework effectively extends the method proposed in [8] to DAEs.
2) A *penalty*-based method is then introduced (Section III-C) 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 (Section III-D) 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 (Section IV) using a three-bus power network example described by a set of stiff and nonlinear DAEs.

We organize this work as follows. In Section II, we introduce the *differential algebraic equations* (DAE) studied in this paper. In Section III, 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 Section III by introducing Algorithm 2 that enables us to use the trained DAE-PINN for simulating DAEs over long-time horizons. In Section IV, 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 Section V and conclude the paper in Section VI.

## II. PROBLEM SETUP

In this paper, we develop DAE-PINN, a deep learning-based framework that employs physics-informed neural networks [8] and implicit Runge-Kutta schemes [3] for learning the solution trajectories of nonlinear *Differential-Algebraic equations* (DAE) [16] 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 $y = y(t) \in \mathbb{R}^n$ are the dynamic states, $z = z(t) \in \mathbb{R}^m$ are the algebraic variables, $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ describes the differential equations, $g : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^m$ the algebraic equations, $t \in [t_0, T]$ the simulation time interval, and $T > t_0$ the time horizon.

*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 [17], which means that 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 [18], 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.* [14] (Stiff-PINNs) correspond to a special case in our problem setup where the algebraic variables $z$ can be solved for explicitly to obtain (2).

*Applications:* DAEs frequently arise in dynamic simulations of power networks [1], mechanical problems, trajectory control, etc. 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 Section III 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 $x \in \mathbb{R}^{n+m}$ 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 dimension corresponding to the rank of $M$. Then, we insert the above into (4) and use $Tu = \left(y^\top, z^\top\right)^\top$ 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 Section III for (1) also applies for problems in descriptor form (4), provided we can decompose the matrix $M$.

## III. PROPOSED METHOD - DAE-PINN

This section describes our DAE-PINN framework, *i.e.,* a physics-informed neural network framework that allows to solve the DAE (1) using the *implicit Runge-Kutta* (IRK) time-stepping scheme with $\nu$ stages.

### A. 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* [3]. We apply the implicit Runge-Kutta scheme with $\nu$ stages [3], [17] to our system of DAEs (1) and obtain

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

$$0 = g(\xi_j,\zeta_j), \quad j = 1,\ldots,\nu \quad (5b)$$

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

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

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 [3] and to let the scheme be of nontrivial order, we impose the following convention for the parameters

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

### B. Discrete Physics-Informed Neural Networks

In classical numerical analysis [3], implicit formulations of Runge-Kutta schemes are usually constrained due to the computational cost of solving (5). And if we increase the number of IRK stages, these constraints become more severe. To overcome these constraints, our DAE-PINN framework employs a *discrete physics-informed neural network* (PINN) model [8] to enable the implicit Runge-Kutta scheme with $\nu$ stages (5).

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

$$[\xi_1^\theta, \ldots, \xi_\nu^\theta, y_{n+1}^\theta] \quad (6a)$$

$$[\zeta_1^\theta, \ldots, \zeta_\nu^\theta, z_{n+1}^\theta]. \quad (6b)$$



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

*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 $y \in \mathbb{R}^n$ and another neural network for the algebraic variables $z \in \mathbb{R}^m$. 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 $z_i \in \mathbb{R}$.

*Remark* 2. *Complexity:* As described in [8], PINN enable us to employ implicit Runge-Kutta schemes with a large number of stages at effectively 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 $\sim O(\nu)$.

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 throughout the domain [9], *i.e.,* the set of training data $\mathcal{T} := \{y_n^1, y_n^2, \ldots, y_n^{|\mathcal{T}|}\}$ of size $|\mathcal{T}|$[1]. To measure the discrepancy between the neural networks and the IRK scheme (5), we use the following loss function:

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

---

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

In the above, $w_g$ and $w_f$ are the weights,

$$\mathcal{L}_f(\theta; \mathcal{T}) = \frac{1}{|\mathcal{T}|(\nu+1)} \sum_{y_n \in \mathcal{T}} \sum_{j=1}^{\nu+1} ||y_n - y_j^n(\theta)||_2^2,$$

where

$$y_j^n(\theta) := \xi_k^\theta - h \sum_{i=1}^\nu a_{j,i} f(\xi_i^\theta, \zeta_i^\theta), \quad j = 1, \ldots, \nu$$

$$y_{\nu+1}^n(\theta) := y_{n+1}^\theta - h \sum_{j=1}^s b_j f(\xi_j^\theta, \zeta_j^\theta),$$

and

$$\mathcal{L}_g(\theta; \mathcal{T}) = \frac{1}{(\nu+1)} \left( \sum_{j=1}^\nu ||g(\xi_j^\theta, \zeta_j^\theta)||_2^2 + ||g(y_{n+1}^\theta, z_{n+1}^\theta)||_2^2 \right).$$

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 [19]:

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

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 [10] to update the value of the weight coefficients $w_f$ and $w_g$.

### C. 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 [1]. However, by using the soft constraints approach for the loss function (7), it may be difficult to satisfy the dynamic and algebraic equations exactly. This can be seen as follows. If the weight coefficients $w_f$ and $w_g$ are selected too large, which severely penalizes the violation of the DAEs, the optimization problem may become ill-conditioned, and, 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 [10] 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 $k$th "outer" iteration, we solve the following unconstrained optimization problem

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

---

**Algorithm 1:** Training using the penalty method [10]

**Hyperparameters:** initial penalty coefficients $w_f^0$ and $w_g^0$, factor $\beta$, and number of iterations $K$

$k \longleftarrow 0$

$\theta^0 \longleftarrow \arg\min_\theta \mathcal{L}^0(\theta; \mathcal{T})$: 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

**while** $k \leq K$ **do**

    $k \longleftarrow k + 1$
    $w_g^k \longleftarrow \beta w_g^{k-1}$
    $w_f^k \longleftarrow \beta w_f^{k-1}$
    $\theta^k \longleftarrow \arg\min_\theta \mathcal{L}^k(\theta; \mathcal{T})$: train the networks (7) from the initialization $\theta^{k-1}$, until the training loss has converged, *i.e.*, $\mathcal{L}^k(\theta; \mathcal{T}) \leq$ 1e-5;

---

where $w_f^k$ and $w_g^k$ are the penalty coefficients for the $k$th 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 [20], [10]. In practice, however, if we fail to carefully select the hyperparameters $w_f^0$, $w_g^0$, and $\beta$, the optimization problem may become ill-conditioned or experience slow convergence.

### D. 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 $\nu$ of stages to take a very 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, in this subsection, we briefly describe 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 Markov-like fashion) the input to DAE-PINN $y_n$ using the predicted dynamic states $y_{n+1}^{\theta^*}$ from the previous evaluation step.

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).

## IV. 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.

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

---

**Algorithm 2:** Simulating DAEs for long-time horizons

Given is the number of time steps $N$, and the DAE-PINN with *trained* parameters $\theta^*$ and time step $h$. Let the initial condition of (1a), $y_0$, be the input to the DAE-PINN, *i.e.,* $y_n = y_0$.

**for** $k = 1,\ldots,N$ **do**

(1) compute the forward pass using the proposed framework, *i.e.,*

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

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

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

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 $\{Z_k^{\theta^*}\}_{k=1}^N$

---



Fig. 3: Three-bus-two-generators power network [21]

### A. Three-bus power network

We consider the three-bus (a slack bus, a generator bus, and a load bus) power network depicted in Fig. 3 and described

by the following set of nonlinear and stiff DAEs [21]

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

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

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

We fix the parameters of the power network to the following values $M_1 = .52$, $M_2 = .0531$, $D = .05$, $D_l = .005$, $V_1 = 1.02$, $V_2 = 0.05$, $B_{12}, B_{13}, B_{23} = 10$, $P_g = -2.0$, $P_l = 3.0$, and $Q_l = .1$.

### B. Neural Networks, hyper-parameters and learning protocols

We implemented DAE-PINN using PyTorch and published all the codes in GitHub. All the experiments presented in this section were trained by minimizing the loss function $\mathcal{L}(\theta; \mathcal{T})$ (7) using the Adam [19] 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 collected 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 $y_n \mapsto (\xi_1, \ldots, \xi_\nu, y_{n+1})$ (*i.e.,* dynamic equations) and the neural network that approximates the mapping $y_n \mapsto (\zeta_1, \ldots, \zeta_\nu, z_{n+1})$ (*i.e.,* algebraic equations) were implemented using the improved fully-connected architecture proposed in [13], which has the following forward pass:

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

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 $\phi$, in this paper, is a point-wise sinusoidal activation function. We also assume that each hidden-layer has 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 [13]. 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:

$$(\zeta_1, \ldots, \zeta_\nu, z_{n+1}) \mapsto \mathrm{softplus}(\zeta_1, \ldots, \zeta_\nu, z_{n+1}).$$

Note that the above feature layer constraints the bus voltage to be non-negative, *i.e.,* $V_3 > 0$.

### C. 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 50000 epochs.

*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. Fig. 4a shows the results of running this experiment 10 times. We observe that the unstacked architecture provides us with the best training performance.

*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. Fig. 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.

*Training dataset:* Our last experiment investigates how the size of the training dataset $|\mathcal{T}|$ 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 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 $|\mathcal{T}| = 2000$ initial condition points sampled from the state-space, tests the performance of DAE-PINN every 1000 epochs using a test dataset witn 1500 initial conditions not included in the train 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. $w = 40$).

### D. Results for the best DAE-PINN model

In this subsection, we verify the effectiveness of DAE-PINN to perform 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 $\beta = 2$.

Fig. 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 [16]). 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 the $L^2$ relative error of each state variable. Table I 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.

| | $\omega_1$ | $\omega_2$ | $\delta_2$ | $\delta_3$ | $V_3$ |
|---|---|---|---|---|---|
| **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 |

TABLE I: Mean and standard deviation of the $L^2$ relative error of the long-time simulation of 100 initial conditions sampled from the test dataset.

### E. Comparison with other numerical integration schemes

In this subsection, we compare the proposed DAE-PINN framework that enables the IRK scheme with $\nu = 100$ stages with other discrete PINN models enabling the following DAE numerical integration schemes [3]: (i) Backward-Euler method and (ii) the Gauss-Legendre IRK with $\nu = 3$, which is probably the largest IRK scheme that is consistent, stable, and with reasonable implementation costs [3]. We train and implement all the previously mentioned discrete PINN models using the same protocols and with time-step $h = 0.1$. We then test their capability of simulating DAEs for $N = 80$ time steps. Fig. 6 compares the three discrete PINN models for simulating the DAEs for the representative initial condition selected from the test dataset. The results clearly illustrate that our DAE-PINN, which enables the IRK scheme with $\nu = 100$ stages, significantly outperforms all other discrete PINN models. We also illustrate (see Fig. 7) the $L^2$ relative error as a function of the number of time steps $N$ for the slack machine speed $\omega_1$ and the load bus angle $\delta_3$. One should observe that the discrete PINN model for the Gauss-Legendre IRK method effectively simulates the speed of the slack generator but fails to simulate

(a)



(b)



(c)



(d)

Fig. 4: Convergence experiments. (a) stacked vs unstacked architectures. (b) Network width. (c) Network depth. (d) Number of training examples.



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.

the load bus angle. On the other hand, the discrete PINN model for the Backward-Euler method fails to simulate the machine speed and also the load bus angle dynamics.

## V. DISCUSSION

*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 [22], 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 and 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.

*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

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 $\omega_1$ and load bus angle $\delta_3$ as a function of the number of time steps $N$.

connected neural network architecture. However, we realize that other architectures may increase our ability to simulate long-time dependencies [23]. 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) [24], 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.

*On the inverse problem:* We remark that extending the proposed framework to learn unknown but identifiable parameters of DAEs is straightforward (see [8] for more details). Furthermore, in [25], the authors already used a physics-informed continuous deep learning model to learn unknown parameters of the power network. 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 [14] (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.

*On the stochastic setting:* With the increasing penetration of renewable resources, the operating conditions for power networks are becoming 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 networks dynamics for a given distribution of initial conditions and a set of uncertain parameters.

## VI. 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.

## ACKNOWLEDGMENT

## REFERENCES

[1] P. Kundur, "Power system stability," *Power system stability and control*, pp. 7–1, 2007.
[2] F. Alvarado and S. Oren, "Transmission system operation and interconnection," *National transmission grid study–Issue papers*, pp. A1–A35, 2002.
[3] A. Iserles, *A first course in the numerical analysis of differential equations*. Cambridge university press, 2009, no. 44.
[4] B. Stott, "Power system dynamic response calculations," *Proceedings of the IEEE*, vol. 67, no. 2, pp. 219–241, 1979.
[5] R. Schainker, P. Miller, W. Dubbelday, P. Hirsch, and G. Zhang, "Real-time dynamic security assessment: fast simulation and modeling applied to emergency outage security of the electric grid," *IEEE Power and Energy magazine*, vol. 4, no. 2, pp. 51–58, 2006.

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[7] N. Baker, F. Alexander, T. Bremer, A. Hagberg, Y. Kevrekidis, H. Najm, M. Parashar, A. Patra, J. Sethian, S. Wild *et al.*, "Workshop report on basic research needs for scientific machine learning: Core technologies for artificial intelligence," USDOE Office of Science (SC), Washington, DC (United States), Tech. Rep., 2019.

[8] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[9] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, "Deepxde: A deep learning library for solving differential equations," *SIAM Review*, vol. 63, no. 1, pp. 208–228, 2021.

[10] L. Lu, R. Pestourie, W. Yao, Z. Wang, F. Verdugo, and S. G. Johnson, "Physics-informed neural networks with hard constraints for inverse design," *arXiv preprint arXiv:2102.04626*, 2021.

[11] A. Yazdani, L. Lu, M. Raissi, and G. E. Karniadakis, "Systems biology informed deep learning for inferring parameters and hidden dynamics," *PLoS computational biology*, vol. 16, no. 11, p. e1007575, 2020.

[12] S. Kim, W. Ji, S. Deng, and C. Rackauckas, "Stiff neural ordinary differential equations," *arXiv preprint arXiv:2103.15341*, 2021.

[13] S. Wang, Y. Teng, and P. Perdikaris, "Understanding and mitigating gradient pathologies in physics-informed neural networks," *arXiv preprint arXiv:2001.04536*, 2020.

[14] W. Ji, W. Qiu, Z. Shi, S. Pan, and S. Deng, "Stiff-pinn: Physics-informed neural network for stiff chemical kinetics," *arXiv preprint arXiv:2011.04520*, 2020.

[15] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," *arXiv preprint arXiv:1806.07366*, 2018.

[16] G. Wanner, *Solving ordinary differential equations II*, vol. 375.

[17] M. Roche, "Implicit runge–kutta methods for differential algebraic equations," *SIAM journal on numerical analysis*, vol. 26, no. 4, pp. 963–975, 1989.

[18] W. Rudin *et al.*, *Principles of mathematical analysis*. McGraw-hill New York, 1976, vol. 3.

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[20] D. G. Luenberger, *Introduction to linear and nonlinear programming*. Addison-wesley Reading, MA, 1973, vol. 28.

[21] H. Zheng and C. L. DeMarco, "A bi-stable branch model for energy-based cascading failure analysis in power systems," in *North American Power Symposium 2010*. IEEE, 2010, pp. 1–7.

[22] J. Li, M. Yue, Y. Zhao, and G. Lin, "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, 2020, pp. 1–6.

[23] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, "Informer: Beyond efficient transformer for long sequence time-series forecasting," *arXiv preprint arXiv:2012.07436*, 2020.

[24] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, "Learning nonlinear operators via deeponet based on the universal approximation theorem of operators," *Nature Machine Intelligence*, vol. 3, no. 3, pp. 218–229, 2021.

[25] G. S. Misyris, A. Venzke, and S. Chatzivasileiadis, "Physics-informed neural networks for power systems," in *2020 IEEE Power & Energy Society General Meeting (PESGM)*. IEEE, 2020, pp. 1–5.