2024 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT) | 979-8-3503-1360-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/ISGT59692.2024.10454197

Modeling Power Systems Dynamics with Symbolic Physics-Informed Neural Networks

Huynh T. T. Tran and Hieu T. Nguyen
Department of Electrical & Computer Engineering, North Carolina A&T State University
htran@aggies.ncat.edu, htnguyen1@ncat.edu

***Abstract*—In recent years, scientific machine learning, particularly physic-informed neural networks (PINNs), has introduced new innovative methods to understanding the differential equations that describe power system dynamics, providing a more efficient alternative to traditional methods. However, using a single neural network to capture patterns of all variables requires a large enough size of networks, leading to a long time of training and still high computational costs. In this paper, we utilize the interfacing of PINNs with symbolic techniques to construct multiple single-output neural networks by taking the loss function apart and integrating it over the relevant domain. Also, we reweigh the factors of the components in the loss function to improve the performance of the network for instability systems. Our results show that the symbolic PINNs provide higher accuracy with significantly fewer parameters and faster training time. By using the adaptive weight method, the symbolic PINNs can avoid the vanishing gradient problem and numerical instability.**

***Keywords*—Power system dynamics, physics-informed neural networks, scientific machine learning.**

I. Introduction

Modeling the power system dynamics requires solving a set of complex nonlinear differential equations [1]. This is a non-trivial task due to the large of components of the networks such as generators, loads, and transmission lines. Traditional computers can use numerical discretization methods like Euler, the Runge-Kutta, and the backward differentiation formula to solve these differential equations. However, the computational costs of these methods increase exponentially with system size [2]. In addition, with the growing integration of renewable energy sources, power systems have become more complex, creating new technical challenges [3]. Thus, the demand for modeling methods that enable high-accuracy modeling and lower power system costs has significantly increased [1].

Scientific machine learning techniques, particularly combining machine learning with traditional scientific computing and mechanistic modeling [4], can be considered as one of the alternative solutions for modeling the dynamics of complex systems such as the power grid. One of the key techniques in scientific machine learning is Physics Informed Neural Networks (PINNs), i.e., combining deep neural networks (NN) with physics equations. Specifically, NN as a data-driven method can learn general solutions from a dataset and provides rapid approximation whereas the embedded physics equations help encourage consistency with the known physics of the system [5]. Thanks to the growth of available data and computer resources, PINNs have been applied to tackle several physics and engineering problems, such as modeling power system dynamics [6], heat transfer [7], fluid mechanics [8], and high-speed aerodynamic flows [9].

There have been several structures of PINNs proposed in the literature to model power system dynamics. Ref. [3] used the neural ordinary differential equations (ODEs) where the input variables are mapped to the derivative of the hidden variables and must first be integrated before their use [10]. Although neural ODEs can represent continuously defined time series dynamics with high accuracy, the hidden variables need to be integrated, which means they must be sent through an ODE solver. Thus, the lack of knowledge of hidden variables causes difficulties in modeling the system [11]. Another structure of PINNs had been used in [1], [6], [11], which are originally designed to solve partial differential equations (PDEs), a generalization of ODEs. By using an unsupervised strategy, they do not require labeled data derived from prior simulations or experiments whereas differential equations' solutions can be found by minimizing loss function optimization problems instead of directly solving governing equations [12]. However, this method requires all dependent variables to be defined on the full domain, which causes high computational cost [4]. Also, its performance can suffer from kernel saturation issues if the NN is split into multiple outputs,

To handle these obstacles, we leverage the recent advance of science machine learning techniques provided in the ModelingToolkit (MTK) package [13] to construct a new structure of PINNs. Instead of a single large network, we split the network into multiple single-output NNs, in which each variable is represented by a network with a smaller size. Additionally, we can integrate the loss function over the relevant domain for each portion of the loss function by using a symbolic interface, therefore reducing the number of trainable parameters and the training time. In certain scenarios, the varying scale of the loss function can lead to optimization challenges. To address this, we used the adaptive weight method, as outlined in [14]. This approach involves re-evaluating the factors that contribute to the loss function and adjusting their weights to enhance the performance of the network.

The rest of the paper is organized as follows. Section II reviews the basic concept of physics-informed neural networks, introduces a new structure for NNs, and a method to reweigh the loss function components. Section III represents the case studies and implementation. Our numerical results are shown in Section IV and Section V concludes the paper.

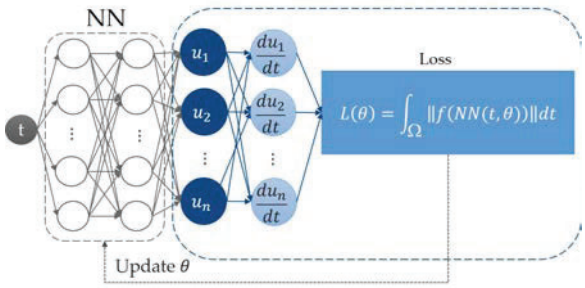


Fig. 1: General structure of conventional PINNs [7].

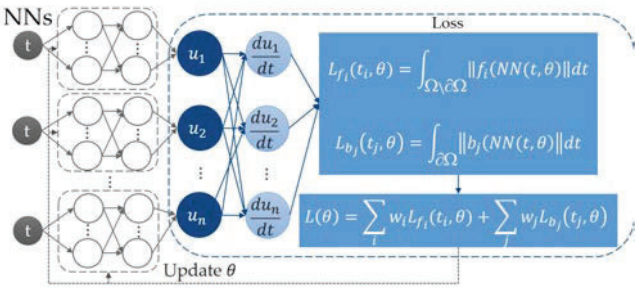


Fig. 2: Schematic of symbolic PINNs.

II. Physics-informed Neural Networks (PINNs)

In this section, we first review the basic concept of conventional PINNs and then provide the structure of symbolic PINNs and the adaptive weight method.

A. Overview of conventional PINNs

The structure of conventional PINNs is shown in Figure 1. In general, physical models for a system of partial differential equations (PDEs) can be defined as [4]:

$$f(u,t) = 0, \quad \forall t \in \Omega, \tag{1}$$

where f is the residual of differential equations containing the nonlinear differential operators acting on $u(t)$, $u(t)$ are state variables, t is the time (independent variable), and Ω is the computational domain. PINNs solve (1) by using the Universal Approximation Theorem, in which if exist any sufficiently regular function $u(t)$, there is a large enough neural network NN with the parameters θ (including weights and biases) such that $\|NN(t,\theta) - u(t)\| < \epsilon$ for all $t \in \Omega$. We can replace unknown $u(t)$ by a NN $NN(t,\theta)$ and find the parameters such that $f(NN(t,\theta)) \approx 0$ for all $t \in \Omega$. Therefore, in PINNs, solving a system of differential equations is converted into an optimization problem with a loss function that is the sum of differences at every point within the domain and can be written as follows:

$$\mathcal{L}(\theta) = \int_\Omega \|f(NN(t,\theta))\|dt, \tag{2}$$

where $\|\cdot\|$ is the norm operator, and $\mathcal{L}(\theta)$ is the difference from the exact solution, and the goal is minimizing $\mathcal{L}(\theta)$. If $\mathcal{L}(\theta) = 0$, then by definition, the outputs from the NN are the solution to the differential equations.

B. Sympolic PINNs

Suppose we leverage traditional PINNs to solve a set of m equations with n variables. The advantage of this method is that it enables larger Basic Linear Algebra Subprograms (BLAS) operations. However, if we want to split the equations into multiple single-output NNs, it can lead to performance issues with kernel saturation. In addition, when employing automatic differentiation (AD), both forward and reverse mode automatic differentiation scale with the number of outputs $\mathcal{O}(mn)$ and $\mathcal{O}(m+n)$, respectively. This means that if a differentiation term is only necessary for one variable, it can result in a significantly higher computational cost, as it will require computing the derivative with respect to all variables. Furthermore, this approach requires that all dependent variables be defined on the full domain, which is not always the case in PDEs [4].

Using the symbolic technique allows us to represent the loss function in an abstract form while preserving its mathematical structure [4]. Hence, we can divide the loss function into components and integrate only the relevant domains for each portion. In most cases, f can consist of boundary conditions which are needed to be satisfied on (some subset) $\partial\Omega$. Therefore, f can be separated into its component functions, and we obtain the following losses:

$$\mathcal{L}_{f_i}(t_i,\theta) = \int_{\Omega\setminus\partial\Omega} \|f_i(NN(t,\theta))\|dt \tag{3}$$

$$\mathcal{L}_{b_i}(t_i,\theta) = \int_{\partial\Omega} \|b_i(NN(t,\theta))\|dt, \tag{4}$$

where f_i is each equation in the system of PDEs, (3) represents the PDE loss term, b_i are the boundary conditions and (4) represents the boundary condition loss term. The PDE loss term represents the residual obtained when substituting the outputs from the NN into the given PDEs, and the boundary condition loss term represents the difference between the outputs and the boundary conditions [7]. To evaluate the PDE loss term, we need to find the set of points t_i. A simple way to approximate the integral is to select a grid of t_i and use the Trapezoidal method [4]:

$$\int_{\Omega\setminus\partial\Omega} \|f_i(NN(t,\theta))\|dt \approx \sum_i \Delta t \|f(t_i)\|. \tag{5}$$

Another method is to take t_i at fixed numbers of random points and integrate them using Monte-Carlo methods [4]:

$$\int_{\Omega\setminus\partial\Omega} \|f_i(NN(t,\theta))\|dt \approx \alpha \sum_i \|f(t_i)\|, \tag{6}$$

where α is the arbitrary constant. In the end, each loss function of (3) and (4) is multiplied by some factors and added together to form the final loss function:

$$\mathcal{L}(\theta) = \sum_i w_i \mathcal{L}_{f_i}(t_i,\theta) + \sum_j w_j \mathcal{L}_{b_j}(t_j,\theta), \tag{7}$$

where w_i and w_j are the factors associated with the PDE and boundary condition loss terms, respectively. By calculating

979-8-3503-1360-4/24/$31.00 ©2024 IEEE

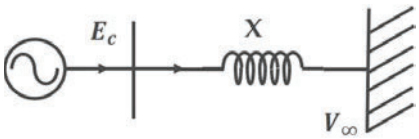


Fig. 3: Single machine infinite bus (SMIB) system [1].

at multiple points in the domain, symbolic PINNs can avoid kernel saturation issues. The structure of symbolic PINNs is shown in Figure 2.

C. Adaptive weight of loss functions

From (4), we can notice that the boundary conditions only computationally evaluate when $t \in \partial\Omega$ and would be 0 for all $t \in \Omega \backslash \partial\Omega$, causing dimensional of the boundary condition loss term is lower than the PDE loss term. Commonly, NNs have low derivative frequency scale amplification in the component loss functions because the initial distribution of loss functions is fairly linear. If the solution $u(t)$ has a high frequency in certain regions of the domain, the differences in scale of the component loss functions can become more pronounced and cause changes in relation to one another. This is one of several ways the component loss functions have different scales, leading to many optimization difficulties during training the PINNs. To address this obstacle, we need to adjust the gradients of the boundary condition loss term to the same scale as the gradients of the PDE loss term. One method had been introduced in [14], in which an adaptive weight $\hat{w}_i$ can be defined as:

$$\hat{w_i} = \frac{\max_\theta\{|\nabla_\theta \mathcal{L}_{f_i}(\theta_n)|\}}{\text{mean}_\theta\{|\nabla_\theta \mathcal{L}_{b_j}(\theta_n)|\}}, \quad n \in S, \tag{8}$$

where $\max_\theta\{|\nabla_\theta \mathcal{L}_{f_i}(\theta_n)|\}$ is the maximum of the absolute values of the gradients of the PDE loss terms, $mean_\theta\{|\nabla_\theta \mathcal{L}_{b_j}(\theta_n)|\}$ is the mean of the absolute values of the gradients of the boundary condition loss terms, and S is the set of certain iteration in which the adaptive weigh will be calculated (e.g., every iteration or every ten optimization iteration). Then, the component factor associated with each loss term can be updated as follows:

$$w_i = (1-\gamma)w_i + \gamma\hat{w}_i, \tag{9}$$

where γ is an additional hyperparameter. The recommended value of γ is 0.9 [14].

III. Case studies and Implementation

A. Physics model of power systems dynamics

In this paper, we consider a specific model of power system dynamics, which is a single-machine infinite bus (SMIB) system, shown in Figure 3, represented as follows [15]:

$$\begin{cases} \dfrac{d\delta}{dt} = \omega_t & \text{(10a)} \\ \dfrac{d\omega_t}{dt} = K_1 - K_2\sin(\delta) - K_3\omega_t, & \text{(10b)} \end{cases}$$

where $K_1 = \dfrac{\omega_s}{2H}T_m, K_2 = \dfrac{\omega_s}{2H}\dfrac{E_c V_\infty}{X}, K_3 = \dfrac{\omega_s}{2H}D,$

with δ is the rotor angle behind the transient reactance, $\omega_t = \omega - \omega_s$ is the transient speed (ω is the generator's angular speed). The parameters used in (10) including H, the generator's inertia constant, D, the damping coefficient, T_m, the mechanical torque constant, ω_s, the referenced angular speed, E_c corresponding to the internal generator voltage, V_∞ corresponding to the infinite bus voltage, and X representing the sum of the generator's internal reactance and the reactance of the losses line.

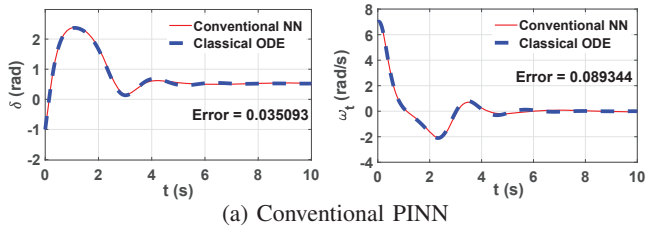
B. Implementation

We perform the case studies in Julia Programming Language version 1.9.2 on a desktop - Intel core i9-10900. To investigate the performance of symbolic PINNs, we consider the SMIB system in normal operation with $k_1 = 5$, $K_2 = 10$, and $K_3 = 1.7$ (the parameters can be found in [15] chapter 5.8), and the simulation duration is $[0, 10s]$. The initial rotor angle, $\delta(0) = -1rad$, and the initial transient speed, $\omega_t = 7rad/s$. The implementation is the following steps:

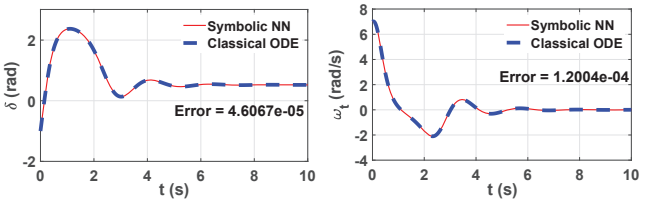
- We implement the equation (10) and the initial conditions in Julia via the ModelingToolkit (MTK) package.
- We use the Lux package [16] to create an NN that solves the given equations. The NN architecture is built using the *Chain* function, which has a fully connected layer defined by the *Dense* function. The NN consists of two sub-networks that correspond to two state variables, δ and ω_t. Each sub-network has an input dimension of 1, four hidden layers with ten neurons each, and an output dimension of 1. The hidden layers use hyperbolic tangent as an activation function.
- We leverage *PhysicsInformedNN* function to present the NN as a trial solution. The outputs from the NN are substituted to the implemented equation and using *GridTraining* function to estimate the loss function follow the method in (5) with a time steps of 0.01s.
- By using a *symbolic_discretize* interface, we can take the loss function apart and then reassemble it before converting it into the optimization problem by the *OptimizationProblem* function via the Optimization package and optimized by the Broyden-Fletcher-Goldfarb-Shanno (BFGS) optimizer [17].

We perform the study over 50 times, with the maximum iteration of each is 50,000. To validate the results, we calculate the error between them with the results from the classical method by root means square method. For the classical method, the equation (10) is solved by *Tsit5* solver (similarly with *ode45* solver in MATLAB) with a time step of 0.01s by using DifferentialEquations package [18]. After training, we evaluate the effectiveness of the transfer model of the previous case study with new initial conditions in two cases: *case 1*: initial angle $\delta(0) = 1$rad, and initial transient speed $\omega_t = -5$rad/s, *case 2*: initial angle $\delta(0) = 0$rad, and initial transient speed $\omega_t = 2$rad/s. Finally, to investigate the performance of the adaptive weight method, we consider the SMIB system in a faulty condition, namely pole slipping. Pole slipping happens when the electromagnetic torque used to produce the power output is lower than the mechanical torque generated by a prime mover [19]. Thus, we change a parameter K_3 to

979-8-3503-1360-4/24/$31.00 ©2024 IEEE



(a) Conventional PINN



(b) Symbolic PINN

Fig. 4: Results of SMIB of conventional and symbolic PINNs.

1.6 while keeping the remaining parameters unchanged. The adaptive weight is calculated every 10 iterations.

IV. Numerical Results

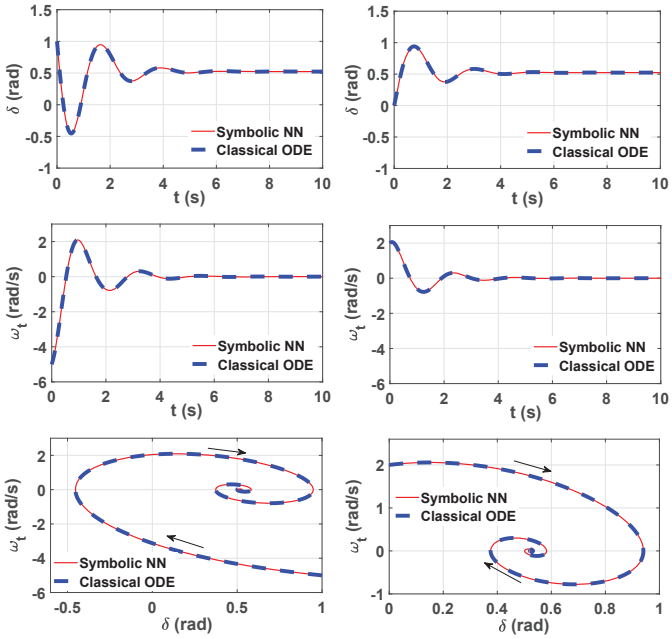
A. Performance of symbolic PINNs

Figure 4 shows the results of the SMIB system (rotor angle, δ, and generator's transient speed, ω_t). The dashed line represents the values from the classical method, the solid line represents the values from the PINNs, where Figure 4a illustrates the values of conventional PINN and Figure 4b illustrates the value of symbolic PINN. From Figure 4a, we can see that after training, the conventional PINN can model the power system dynamics with an accuracy upper 90%. However, the total parameters used in the conventional PINN are 1342 for 5 layers (including weights and biases), and the average time to train the network is 828s. For the symbolic PINN in Figure 4b, there is a total of 502 parameters for two sub-networks with 4 layers in each, the average training time is 332s, and the errors are very close to zero. Based on the results, it is evident that the symbolic PINN provides much higher accuracy while using significantly fewer parameters and has a training time that is around 2.5 times faster than the conventional PINN. The similar results can be found in [15].

B. Symbolic PINNs with transfer model

The training time of networks depends on the initial conditions. For the networks trained from scratch (without transfer parameters), we recorded the following results[1]: *case 1*) the average training time is 363s and the average error is $2.85 \cdot 10^{-3}$, *case 2*) the average training time is 295s and the average error is $8.15 \cdot 10^{-4}$. For the network with the transfer model: *case 1*) the average training time is 176s, and the error is $9.18 \cdot 10^{-5}$, *case 2*) the average training time is 161s, and the error is $6.6 \cdot 10^{-5}$. The results show that the transfer model reduces the training time by around two times and improves the networks' accuracy. Additionally, the downside of the BFGS algorithm is that it may stop at the local optimal instead of the global optimal [4]. With the transfer model, we can avoid getting stuck at the local optimal and reduce the total training time.

[1]Note: for the networks with transfer model, the optimal values are the same due to identity initial weights and biases, but have different training times because of the rate of the computer.



(a) case 1 (b) case 2

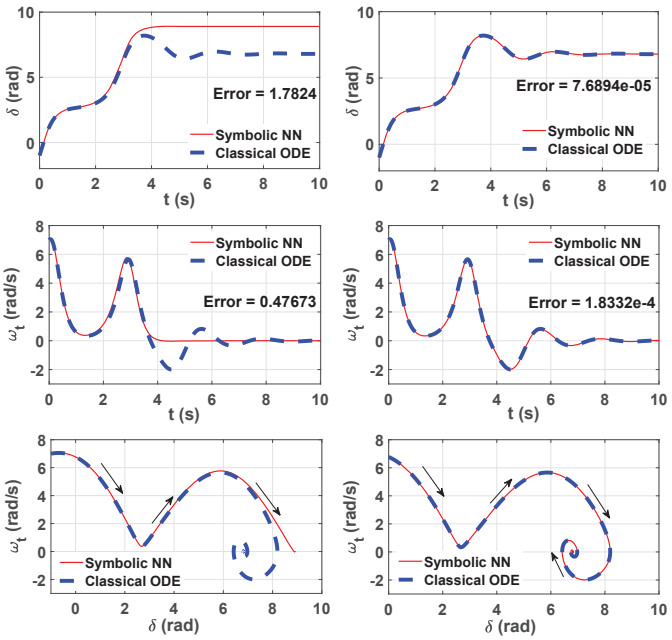
Fig. 5: Results of SMIB when changing initial conditions

Figure 5 shows the results from the symbolic PINNs using the transfer model after training in IV-A, where the first row represents the rotor angle, the second row represents the generator's transient speed, and the third row represents the phase portrait. Phase portrait is the trajectories' generic representation of power system dynamics in the phase plane, which illustrates the behavior of the system by state variables. The values of symbolic PINNs are represented in the solid line, and the values of the classical method are represented in the dashed line. We can see that for different initial conditions, the trajectories of δ and ω_t are different, but the time to converge is nearly the same. Furthermore, while the trajectories in phase portrait converged in different directions, they have the same stable point at (0.523, 0).

C. Symbolic PINNs with adaptive weight

Results of the SMIB system in pole slipping are shown in Figure 6, where Figure 6a illustrates the values of symbolic PINNs without using adaptive weight, and Figure 6b illustrates the values of symbolic PINNs with adaptive weight every 10 iterations. The results shown in Figure 6a is the best one, getting over 50 times. The network can model the system in the first several seconds and then converge to the stable point, which is not the true equilibrium point. The evidence for that is the rotor angle and transient speed reach a new stable point (first and second row of Figure 6a) but in the phase portrait (third row of Figure 6a) the trajectory of the values of networks did not converge to the same point as the

979-8-3503-1360-4/24/$31.00 ©2024 IEEE



(a) Without adaptive weight (b) Having adaptive weight

Fig. 6: Results of SMIB in pole slipping

classical one. This is because the network got the vanishing gradient at the first time the transient speed ω_t lower than zero. As a result, the errors of rotor angle δ, and transient speed ω_t are 1.7824 and 0.47672, respectively. After applying adaptive weight, the vanishing gradient issue was solved. By reweighing the factors between loss function components, the system can be modeled well. The accuracy of networks is nearly 100% due to the errors of rotor angle and transient speed being very small ($7.6894 \cdot 10^{-5}$, and $1.8332 \cdot 10^{-4}$).

V. Conclusion

This paper represents a symbolic PINN model to simulate the power system dynamics and the adaptive weight method used to capture numerical instability. We review the conventional PINNs and the physics model of power system dynamics, particularly the single-machine infinite buses (SMIB) system. The model of the SMIB system is implemented into Julia by using the ModelingToolkit package, then using them as regularities in the PINNs. After using the Lux package to create a network, the equation is discretized and trained using the PhysicsInformedNN function. The loss function can be taken apart and reassembled by utilizing the symbolic_discretize function in the NeuralPDE package, then embedded into the optimization problem by the Optimization package. This loss function is minimized by the BFGS optimizer to find the optimized parameters. Our results show that the symbolic PINNs can model the SMIB system with higher accuracy when they require fewer parameters and the training time faster than 2.5 times. The transfer model helps us to reduce the time needed to train in the new initial conditions and avoid stuck at the local optimal when using the BFGS optimizer, with lower errors. Finally, with the adaptive weight method, the networks would capture the optimization difficulties, such as vanishing gradient. In the scope of the paper, we just focus on the SMIB system with two state variables, in the future, we will enhance the networks for the more complex systems.

VI. Acknowledgement

This research is supported by the Alfred P. Sloan Foundation Grant #10358 and the NCAT's Intel Foundation Gift.

References

[1] M. Mohammadian, K. Baker, and F. Fioretto, "Gradient-enhanced physics-informed neural networks for power systems operational support," *Electric Power Systems Research*, vol. 223, p. 109551, 2023.

[2] O. Kyriienko, A. E. Paine, and V. E. Elfving, "Solving nonlinear differential equations with differentiable quantum circuits," *Physical Review A*, vol. 103, no. 5, p. 052416, 2021.

[3] T. Xiao, Y. Chen, S. Huang, T. He, and H. Guan, "Feasibility study of neural ode and dae modules for power system dynamic component modeling," *IEEE Transactions on Power Systems*, 2022.

[4] K. Zubov, Z. McCarthy, Y. Ma, F. Calisto, V. Pagliarino, S. Azeglio, L. Bottero, E. Luján, V. Sulzer, A. Bharambe *et al.*, "Neuralpde: Automating physics-informed neural networks (pinns) with error approximations," 2021.

[5] 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.

[6] 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.

[7] S. Cai, Z. Mao, Z. Wang, M. Yin, and G. E. Karniadakis, "Physics-informed neural networks (pinns) for fluid mechanics: A review," *Acta Mechanica Sinica*, vol. 37, no. 12, pp. 1727–1738, 2021.

[8] ——, "Physics-informed neural networks (pinns) for fluid mechanics: A review," *Acta Mechanica Sinica*, vol. 37, no. 12, pp. 1727–1738, 2021.

[9] Z. Mao, A. D. Jagtap, and G. E. Karniadakis, "Physics-informed neural networks for high-speed flows," *Computer Methods in Applied Mechanics and Engineering*, vol. 360, p. 112789, 2020.

[10] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," *Advances in neural information processing systems*, vol. 31, 2018.

[11] X. Kong, K. Yamashita, B. Foggo, and N. Yu, "Dynamic parameter estimation with physics-based neural ordinary differential equations," in *2022 IEEE Power & Energy Society General Meeting (PESGM)*. IEEE, 2022, pp. 1–5.

[12] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, "Scientific machine learning through physics–informed neural networks: Where we are and what's next," *Journal of Scientific Computing*, vol. 92, no. 3, p. 88, 2022.

[13] Y. Ma, S. Gowda, R. Anantharaman, C. Laughman, V. Shah, and C. Rackauckas, "Modelingtoolkit: A composable graph transformation system for equation-based modeling," 2021.

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

[15] P. W. Sauer, M. A. Pai, and J. H. Chow, *Power system dynamics and stability: with synchrophasor measurement and power system toolbox*. John Wiley & Sons, 2017.

[16] A. Pal, "Lux: Explicit Parameterization of Deep Neural Networks in Julia," Apr. 2023, if you use this software, please cite it as below. [Online]. Available: https://doi.org/10.5281/zenodo.7808904

[17] S. J. Wright, *Numerical optimization*, 2006.

[18] C. Rackauckas and Q. Nie, "Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia," *Journal of Open Research Software*, vol. 5, no. 1, p. 15, 2017.

[19] M. Redfern and M. Checksfield, "A study into a new solution for the problems experienced with pole slipping protection [of synchronous generators]," *IEEE Transactions on Power Delivery*, vol. 13, no. 2, pp. 394–404, 1998.