Neuro-PINN: A hybrid framework for efficient nonlinear projection equation solutions

Dawen Wu, Abdel Lisser

Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

Neuro-PINN: A Hybrid Framework for Efficient Nonlinear Projection Equation Solutions

Dawen Wu* | Abdel Lisser

Université Paris-Saclay, CNRS, CentraleSupélec, Laboratoire des Signaux et Systèmes, Gif-sur-Yvette, France

Correspondence
*Corresponding author name. Email: dawen.wu@centralesupelec.fr

Summary

Nonlinear projection equations (NPEs) provide a unified framework for solving various constrained nonlinear optimization and engineering problems. This paper presents a deep learning approach for solving NPEs by incorporating neurodynamic optimization and physics-informed neural networks (PINNs). First, we model the NPE as a system of ordinary differential equations (ODEs) using neurodynamic optimization, and the objective becomes solving this ODE system. Second, we use a modified PINN to serve as the solution for the ODE system. Third, the neural network is trained using a dedicated algorithm to optimize both the ODE system and the NPE. Unlike conventional numerical integration methods, the proposed approach predicts the end state without computing all the intermediate states, resulting in a more efficient solution. The effectiveness of the proposed framework is demonstrated on a collection of classical problems, such as variational inequalities and complementarity problems.

KEYWORDS:
Nonlinear projection equation, Physics-informed neural network, Neurodynamic optimization, Ordinary differential equation

1 | INTRODUCTION

Nonlinear optimization problems are prevalent across a wide range of fields, such as engineering, physics, and economics. Solving these problems requires finding an optimal solution that satisfies a defined set of constraints while optimizing the objective function[1]. Nonlinear projection equations (NPEs) are a useful tool for formulating these problems and serve as a unifying framework for treating various nonlinear optimization problems, including nonlinear complementarity problems, variational inequalities, and equilibrium point problems[2,3].

The NPEs are typically addressed by neurodynamic optimization, which models the problem as a system of ordinary differential equations (ODEs)[4,5,6]. The constructed ODE system must be shown to have the global convergence property, i.e., the state solution of the system converges to the optimal solution of the NPE, regardless of the initial point. The NPE problem is then transformed into solving the state solution of the ODE system. However, the ODE system is typically highly nonlinear and has no analytical solutions. Therefore, numerical integration methods such as Runge-Kutta (RK) methods are commonly used to solve the state solution[7].

Motivation. Despite the usefulness of numerical integration methods, they are not efficient enough for solving the NPE problems. This inefficiency is due to the fact that the state solution of the ODE system only provides a solution to the NPE at the end state. To reach the end state, numerical integration methods require the computation of all intermediate states starting

from the initial state, making the process computationally intensive. Therefore, there is a need for a more efficient algorithm to solve the NPE problem.

1.1 | Related Works

Neurodynamic optimization. Over the past few decades, numerous neurodynamic approaches have been developed to solve various constrained optimization problems, including linear and quadratic programming[8,9], general convex programming[10,11,12], biconvex optimization[13], non-smooth optimization[14], and pseudoconvex optimization problems[15]. In particular, a projection neurodynamic model for solving NPEs was introduced and found to have global convergence to the exact solution under mild conditions[4]. The model also showed both asymptotic and exponential stability without the need for a smooth nonlinear mapping. To further improve the performance, a bi-projection neurodynamic model was developed to efficiently solve quadratic optimization problems[5]. In addition, a collaborative approach combining the projection neurodynamic model with particle swarm optimization was introduced for global optimization problems[6].

Physics-informed neural networks. Another avenue of research explored in this paper is the use of deep learning to solve differential equations. In the 1990s, Lagaris et al. used neural networks (NNs) to serve as solutions to both ordinary and partial differential equations (ODEs and PDEs), embedding boundary conditions directly into the network architecture[16,17]. The advent of deep learning has reinvigorated interest in using such methods to tackle high-dimensional, nonlinear PDEs[18,19]. A key contribution in this area is the development of Physics-Informed Neural Networks (PINNs)[20], which integrate differential equations and data errors into the loss function. The versatile architecture and efficient training algorithms of PINNs have enabled numerous successes in a wide range of computational challenges in physics and engineering[21,22,23,24,25]. This expanding research landscape is driven by a combination strategy that adapts PINNs to exploit the structural properties of the target problem. As a result, a variety of PINN variants have emerged to address different problem scenarios[26,27] and improve computational efficiency[28,29,30,31]. Apart from collocation-based PINN approaches, many studies use deep energy methods to solve PDEs, where the energy of the system is used as a loss function to train the NN model[32,33,34]. Many packages have been developed to support the use of deep learning for solving PDEs[35,36].

1.2 | Key Contributions

This paper presents several key contributions:

- We propose a deep learning approach to NPEs that combines neurodynamic optimization with PINNs. Our approach reformulates an NPE problem as an NN training problem, thereby eliminating the need for numerical methods to solve NPEs.
- To improve the performance of our proposed approach, we design a specialized training algorithm that focuses on the end state of the ODE system. In each training iteration, the algorithm uses an evaluation metric based on the NPE error to assess the predictive accuracy of the NN model and retain the optimal result. Therefore, the NN model is not only trained to solve the ODE system, but also to solve the NPE problem.
- In our experiments, we demonstrate the effectiveness of the proposed approach in solving large-scale NPE problems. We compare our approach with the traditional RK method and PINN to validate its performance. In addition, we perform a hyperparameter sensitivity analysis to investigate the impact of different hyperparameter configurations on the performance of our proposed approach.

1.3 | Paper Outline

The remaining sections are organized as follows: Section 2 provides the necessary background, including an introduction to the NPE problem, neurodynamic optimization, and the RK method. In Section 3, we present our NN model for the NPE problem. Section 4 details the design of the loss function and the training algorithm for the proposed NN model. Section 5 presents the experimental results, and we compare our approach with the RK method and PINN. Finally, Section 6 summarizes the main results of the paper and outlines possible directions for future research.

2 | PRELIMINARIES

Section 2.1 introduces the NPEs that this paper aims to solve, and discusses their equivalence to other nonlinear optimization problems. Section 2.2 describes the neurodynamic approach to modeling NPEs as ODE systems. Section 2.3 introduces the RK method, a classical numerical integration method for solving the ODE system.

2.1 | Nonlinear Projection Equation

Definition 1 (Nonlinear projection equation)**.** Consider a nonlinear mapping $G : \mathbb{R}^n \to \mathbb{R}^n$ and a feasible set $\Omega \subset \mathbb{R}^n$. The projection function $P_\Omega : \mathbb{R}^n \to \Omega$ maps a point $z \in \mathbb{R}^n$ onto Ω, such that:

$$P_\Omega(z) = \arg\min_{x\in\Omega} \|z - x\| \tag{1}$$

where $\|\cdot\|$ denotes the Euclidean norm.

The NPE problem, denoted by $NPE(\Omega, G)$, is to find a vector $x^* \in \Omega$ satisfying:

$$P_\Omega(x^* - G(x^*)) = x^* \tag{2}$$

Definition 2 (Nonlinear complementarity problem)**.** Consider a nonlinear mapping $G : \mathbb{R}^n \to \mathbb{R}^n$. The nonlinear complementarity problem, denoted by $NCP(G)$, is to find a vector $x^* \in \mathbb{R}^n$ satisfying:

$$G(x^*) \geq 0, \quad x^* \geq 0, \quad G(x^*)^T x^* = 0. \tag{3}$$

Definition 3 (Variational inequality)**.** Consider a nonlinear mapping $G : \mathbb{R}^n \to \mathbb{R}^n$ and a feasible set $\Omega \subset \mathbb{R}^n$. The variational inequality problem, denoted by $VI(\Omega, G)$, is to find a vector $x^* \in \Omega$ satisfying:

$$(x - x^*)^T G(x^*) \geq 0, \quad x \in \Omega. \tag{4}$$

Proposition 1 (Harker & Pang[2])**.** Let $\Omega \subset \mathbb{R}^n$ be a nonempty closed convex set. Then x^* solves the problem $NCP(G)$ if and only if x^* solves $NPE(\mathbb{R}^n_+, G)$, where $\mathbb{R}^n_+ = \{x \in \mathbb{R}^n | x \geq \mathbf{0}\}$ represents the set of non-negative real vectors.

Proposition 2 (Harker & Pang[2])**.** Let $\Omega \subset \mathbb{R}^n$ be a nonempty closed convex set. Then x^* solves the problem $VI(\Omega, G)$ if and only if x^* solves $NPE(\Omega, G)$.

According to the literature[2], NPEs can be viewed as a unified framework for many nonlinear optimization problems. For example, the Karush-Kuhn-Tucker (KKT) conditions for linear and quadratic programming problems can be represented as linear complementarity problems, and the KKT conditions for convex constraint nonlinear optimization problems can be transformed into nonlinear complementarity problems[37]. Both are recast as NPEs according to Proposition 1. The equilibrium point of a stochastic Nash game is expressed as a variational inequality[38], which is reformulated as an NPE by Proposition 2.

2.2 | Neurodynamic Optimization

Assumption 1.

- The function $G(\cdot)$ is locally Lipschitz continuous.
- The feasible set Ω is a box-constrained set, defined as $\Omega = \{x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n \mid l_i \leq x_i \leq h_i\}$, where l_i and h_i denote the lower and upper bounds of x_i respectively. In this case, the projection function $P_\Omega(\cdot)$ in equation (1) is reduced as follows:

$$P_\Omega(x) = \left(P^1_\Omega(x_1), P^2_\Omega(x_2), \ldots, P^n_\Omega(x_n)\right)^T, \tag{5}$$

where $P^i_\Omega(x_i)$, $i \in \{1, 2, \ldots, n\}$ is defined as:

$$P^i_\Omega(x_i) = \begin{cases} l_i & \text{if } x_i < l_i, \\ x_i & \text{if } l_i \leq x_i \leq h_i, \\ h_i & \text{if } x_i > h_i. \end{cases} \tag{6}$$

Let $y : \mathbb{R} \to \mathbb{R}^n$ be a time-dependent function, where $y(t)$ represents the state at time t. The aim of neurodynamic optimization is to design a first-order ODE system, $\frac{dy}{dt}$, to govern $y(t)$. In this paper, we employ the projection neurodynamic model introduced by[4] to model the NPE, where the ODE system is defined as follows:

$$\frac{dy}{dt} = -G\left(P_\Omega(y)\right) + P_\Omega(y) - y. \tag{7}$$

To simplify the discussion, we define:

$$\Phi(y) = -G\left(P_\Omega(y)\right) + P_\Omega(y) - y. \tag{8}$$

Thus, the ODE system (7) can be written as $\frac{dy}{dt} = \Phi(y)$.

Definition 4 (State solution)**.** Consider an ODE system $\frac{dy}{dt} = \Phi(y)$, where $\Phi : \mathbb{R}^n \to \mathbb{R}^n$ and an initial condition $y(t_0) = y_0$. A vector value function $y : \mathbb{R} \to \mathbb{R}^n$ is the state solution, if it satisfies the ODE system $\frac{dy}{dt} = \Phi(y)$ and the initial condition $y(0) = y_0$.

$y(t)$ is called the state at time t. Given a time range $[t_0, T]$, $y(T)$ is called the end state on the interval.

Theorem 1 (Xia & Feng[4])**.** Consider an NPE problem, $NPE(\Omega, G)$, and let Assumption 1 hold. Given any initial condition, $y(t_0) = y_0$, the state solution of the ODE system of Eq. (7) converges to the optimal solution of $NPE(\Omega, G)$ as time t goes to infinity, i.e,

$$\lim_{t\to\infty} y(t) = x^*, \tag{9}$$

where x^* is a satisfied point of $NPE(\Omega, G)$.

In particular, if $NPE(\Omega, G)$ contains only one satisfied point x^*, then the ODE system is globally asymptotically stable at x^*.

Initial value problem (IVP) construction. In practice, in order to use the neurodynamic approach to solve the NPE, we need to construct an IVP consisting of three components: 1) the ODE system of Eq. (7), 2) an initial condition $y(t_0) = y_0$, and 3) a time range $t \in [t_0, T]$. $y(t)$ for $t \in [t_0, T]$ represents the state solution of this IVP over the time range $[t_0, T]$, where the end state, $y(T)$, is considered to be the predicted solution to the NPE. According to Theorem 1, the larger the time range $[t_0, T]$, the closer $y(T)$ is to the optimal solution x^* of the NPE.

2.3 | Runge-Kutta Method

Runge-Kutta (RK) methods are widely used numerical techniques for approximating the state solution of an ODE system. They are particularly useful when exact analytical solutions are difficult or infeasible to obtain. The methodology involves approximating the state solution over a series of discrete time points. In this paper, we concentrate on the fourth-order RK method, commonly known as the RK method, which is one of the most accurate and popular methods in the family of RK methods.

The RK method takes as input an ODE system $\frac{dy}{dt} = \Phi(y)$, an initial condition $y(t_0) = y_0$, and a time range $[t_0, T]$. The method sets N collocation points that are equally spaced on the time range $[t_0, T]$. The RK method returns an approximate state solution, denoted by $\bar{y}(t)$, where t is a time point within the finite set $\{t_0, t_1, \ldots, t_N, t_T\}$. The RK algorithm is stated as follows:

- Step 1: Initialize the step size $h = \dfrac{(T - t_0)}{N + 1}$, the initial time $t = t_0$, and set the initial state $\bar{y}(t_0) = y_0$.
- Step 2: For $i = 1, 2, \ldots, N + 1$ do Steps 3 and 4.
- Step 3: Set

$$\begin{aligned} K_1 &= h\Phi\left(\bar{y}(t)\right), \\ K_2 &= h\Phi\left(\bar{y}(t) + \frac{K_1}{2}\right), \\ K_3 &= h\Phi\left(\bar{y}(t) + \frac{K_2}{2}\right), \\ K_4 &= h\Phi\left(\bar{y}(t) + K_3\right). \end{aligned} \tag{10}$$

- Step 4: Set

$$\bar{y}(t + h) = \bar{y}(t) + \frac{K_1 + 2K_2 + 2K_3 + K_4}{6}, \tag{11}$$

and $t = t + h$.

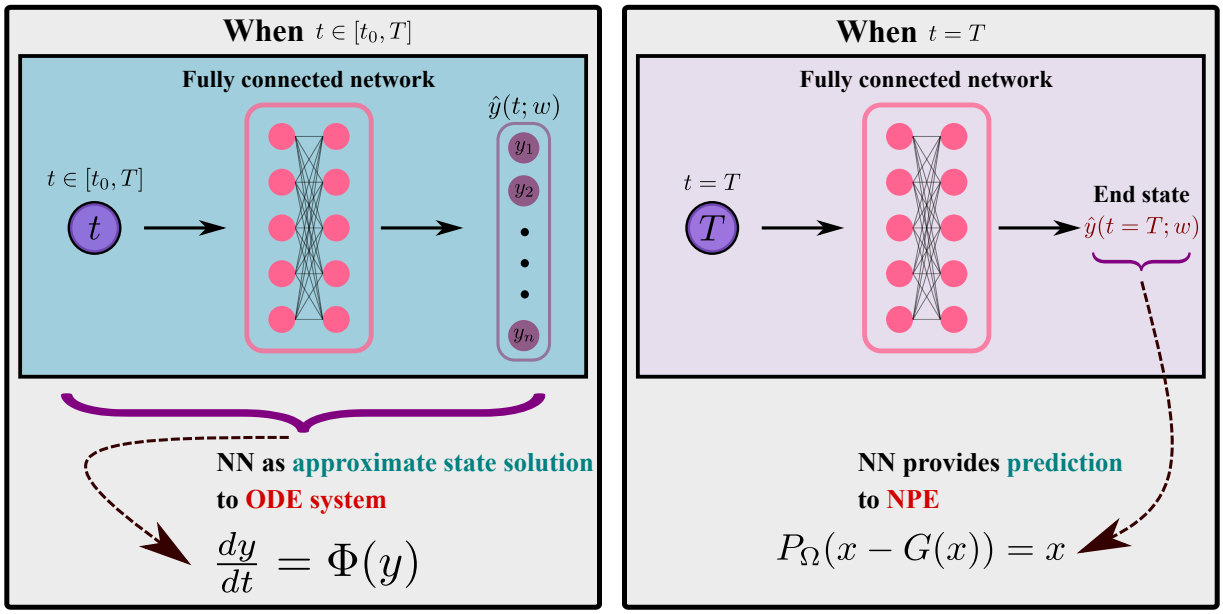

Figure 1 Neural network (NN) solution for the ODE system and the NPE. Left: When $t \in [t_0, T]$, the model $\hat{y}(t; w)$ itself is considered to be an approximate state solution of the ODE system. **Right:** When $t = T$, the end state of the model, $\hat{y}(t = T; w)$, is a predicted solution to the NPE.

When applying the RK method to the NPE problem, the approximate end state $\bar{y}(T)$ is considered as the predicted solution of the NPE problem. Specifically, we have:

$$\bar{y}(T) \approx y(T) \approx x^*, \tag{12}$$

where $\bar{y}(T) \approx y(T)$ indicates that the end state obtained by the RK method is an approximation of the true end state, and $y(T) \approx x^*$ indicates that the true end state is the predicted solution of the NPE problem according to Theorem 1.

3 | NEURAL NETWORK MODEL

Model description. We propose a neural network (NN) model to solve the NPE problem. Our model can be expressed by the following equation:

$$\hat{y}(t; w) = y_0 + \left(1 - e^{-(t-t_0)}\right) \mathrm{N}(t; w), \quad t \in [t_0, T], \tag{13}$$

where $\mathrm{N}(t; w)$ represents a fully connected NN with trainable parameters w, and $[t_0, T]$ is a given time range. We use the Lagaris method to incorporate the initial conditions of the ODE system into the NN model [16]. In particular, the auxiliary function $\left(1 - e^{-(t-t_0)}\right)$ ensures that the NN model always satisfies the initial condition (t_0, y_0), i.e., $\hat{y}(t = t_0; w) = y_0$, regardless of the model parameters w. The exponential form in the auxiliary function has been demonstrated to improve the convergence of the model [39].

Approximate state solution of the ODE. As shown in Fig. 1 (Left), the proposed model itself is an approximate state solution to the ODE system of Eq. (7) on the time range $[t_0, T]$, i.e.,

$$\hat{y}(t; w) \approx y(t), \quad t \in [t_0, T], \tag{14}$$

where $y(t)$ represents the true state solution of the ODE system. Although the input time t of the model $\hat{y}(t; w)$ can be any real number, we only consider $\hat{y}(t; w)$ as the solution of the ODE for the time range $[t_0, T]$. Therefore, we restrict the input to $t \in [t_0, T]$.

Predicted solution of the NPE. The end state of the proposed model, i.e., $\hat{y}(t = T; w)$, is used as a predicted solution to the NPE of Eq. (2), as shown in Fig. 1 (Right). The following equation shows how $\hat{y}(t = T; w)$ approximate the optimal solution x^* of the NPE problem:

$$\hat{y}(t = T; w) \approx y(T) \approx x^*, \tag{15}$$

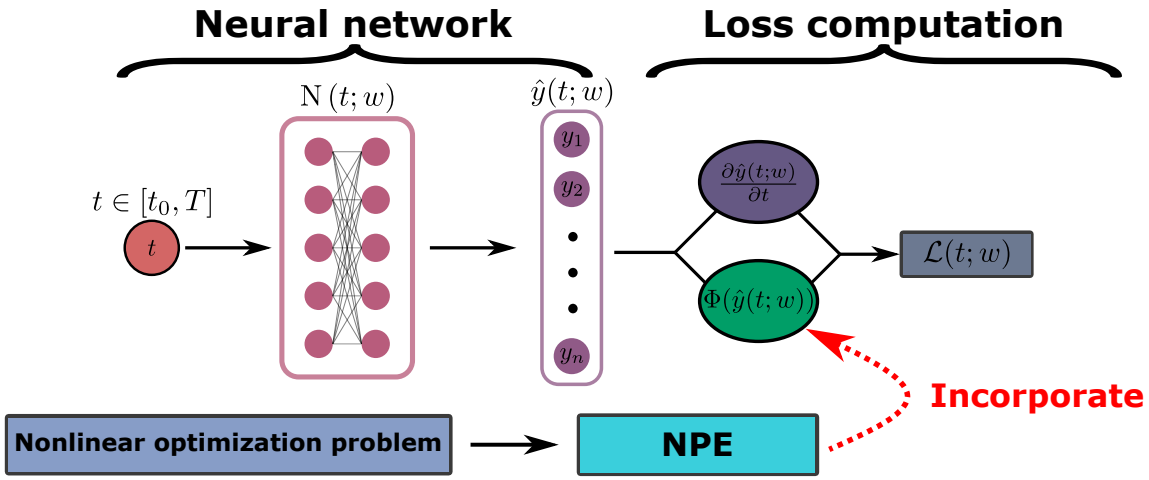

Figure 2 Integrating an NPE into loss computation for NN training.

where $\hat{y}(t = T; w) \approx y(T)$ indicates that the end state of our model approximates the true end state, and $y(T) \approx y^*$ comes from Theorem 1, indicating that $y(T)$ solves the NPE.

Unlike most PINN models, which aim to solve for the entire input space $[t_0, T]$, our NN model focuses on the end state of the ODE system, since it represents the predicted solution to the NPE problem. In the following section, we will show how to train the NN model with an emphasis on improving the prediction accuracy of the end state.

4 | MODEL TRAINING

Section 4.1 provides a definition of the loss function and objective function for the proposed NN model. Section 4.2 presents a training algorithm for the NN model. Section 4.3 compares our proposed NN approach with the RK method.

4.1 | Training Objective

Loss function. The loss function of the proposed NN model is defined as:

$$\mathcal{L}(t, w) = \left\| \frac{\partial \hat{y}(t; w)}{\partial t} - \Phi(\hat{y}(t; w)) \right\|, \tag{16}$$

where $\Phi(\cdot)$ refers to the ODE system, corresponding to the NPE problem to be solved. $\Phi(\hat{y}(t; w))$ represents the expected derivative according to the ODE system, and $\frac{\partial \hat{y}(t;w)}{\partial t}$ represents the actual derivative of the NN model. $\frac{\partial \hat{y}(t;w)}{\partial t}$ can be computed using automatic differentiation tools such as PyTorch or JAX[40,41]. $\mathcal{L}(t, w)$ represents the difference between the two at time t and with network parameters w. As shown in Fig. 2, the NPE is first reformulated as an ODE system via neurodynamic optimization. The ODE system is then incorporated into the loss computational process.

Objective function. The objective function of the NN model is defined as:

$$J(w) = \int_{t_0}^{T} \mathcal{L}(t, w) dt, \tag{17}$$

which is the integral of the loss function over the time range $[t_0, T]$. The loss value $\mathcal{L}(t, w)$ represents the error of the model at time t, while the objective function $J(w)$ represents the total error of the model over the time range $[t_0, T]$.

Batch loss. However, the objective function $J(w)$ is computationally intractable due to its integral part. Therefore, in practice, we cannot directly use $J(w)$ to train the NN model. Instead, we train the model by minimizing the following batch loss:

$$\mathcal{L}(\mathbb{T}, w) = \frac{1}{|\mathbb{T}|} \sum_{t \in \mathbb{T}} \mathcal{L}(t, w), \tag{18}$$

where $\mathbb{T}$ is a set of time points uniformly sampled from the interval $[t_0, T]$, and $|\mathbb{T}|$ denotes the size of the set. The batch loss $\mathcal{L}(\mathbb{T}, w)$ approximates the objective function $J(w)$ by a sum of loss values over a set of sampled time points. By minimizing the batch loss, we can effectively train the model to solve the NPE.

4.2 | Algorithm Design

NPE error. We introduce an evaluation metric, called NPE error, to measure how well a prediction x_{pred} solves the NPE problem. The metric is defined as:

$$\text{NE}(x_{\text{pred}}) = \left\| P_\Omega(x_{\text{pred}} - G(x_{\text{pred}})) - x_{\text{pred}} \right\|_\infty, \tag{19}$$

where $\|\cdot\|_\infty$ represents the infinity norm.

Algorithm 1 Deep learning solver for NPE based on neurodynamic optimization

Input: $NPE(\Omega, G)$ as defined in Eq. (2); Time range $[t_0, T]$; Initial condition (t_0, y_0).
Output: x_{best}, the NN prediction to $NPE(\Omega, G)$.

function SOLVER($NPE(\Omega, G)$, $[t_0, T]$, y_0)
  Derive the ODE system, $\Phi(\cdot)$, according to Eq. (7).
  Initialize a NN model $\hat{y}(t; w)$.
  $\text{NE}_{\text{best}} = \text{NE}(\hat{y}(t = T; w))$ ▷ Initialize the best NPE error.
  while iteration ≤ maximum iteration **do**
    $\mathbb{T} \sim U(t_0, T)$ ▷ Sample $\mathbb{T}$ by the uniform distribution on the interval $[t_0, T]$.
    $\mathcal{L}(\mathbb{T}, w)$ ▷ Compute the batch loss through forward propagation.
    $w \leftarrow \nabla_w \mathcal{L}(\mathbb{T}, w)$ ▷ Update w by $\nabla_w \mathcal{L}(\mathbb{T}, w)$ through backward propagation.
    $x_{\text{curr}} = \hat{y}(t = T; w)$ ▷ Obtain the prediction from the NN model.
    $x_{\text{curr}} = P_\Omega(x_{\text{curr}})$ ▷ Project x_{curr} onto the feasible set Ω by Eq. (1).
    $\text{NE}_{\text{curr}} = \text{NE}(x_{\text{curr}})$ ▷ Calculate the NPE error of x_{curr}.
    if $\text{NE}_{\text{curr}} < \text{NE}_{\text{best}}$ **then**
      $\text{NE}_{\text{best}} = \text{NE}_{\text{curr}}$ ▷ Update NE_{best}.
      $x_{\text{best}} = x_{\text{curr}}$ ▷ Update the best prediction.
    end if
  end while
  return x_{best}
end function

Pipeline. Alg. 1 summarizes how to use our proposed approach to solve an NPE problem. First, we need to specify an initial condition (t_0, y_0) and a time range $[t_0, T]$ to construct an IVP for the NPE problem. Then, we initialize the proposed NN model (13), which serves as an approximate state solution for this IVP. The model is trained by performing gradient descent on the batch loss of Eq. (18) to improve the approximation. Note that our solver is completely based on the deep learning infrastructure and does not require any standard optimization solver or numerical integration solver.

Optimal result retention (ORR) mechanism. A key to Alg. 1 is that we use an ORR mechanism based on the evaluation metric of Eq. (19). Specifically, in each iteration, the algorithm compares the NPE error of the current iteration, denoted as NE_{curr}, with the lowest NPE error found so far, denoted as NE_{best}. Correspondingly, x_{curr} and x_{best} represent the current prediction and the best predictions found so far, respectively. If NE_{curr} is less than NE_{best}, it means that the model found a better prediction in the current iteration. The algorithm then updates NE_{best} to equal NE_{curr} and stores the best prediction as $x_{\text{best}} = x_{\text{curr}}$. This mechanism ensures that the best prediction obtained by the model is maintained throughout the training process, improving the overall performance of the algorithm.

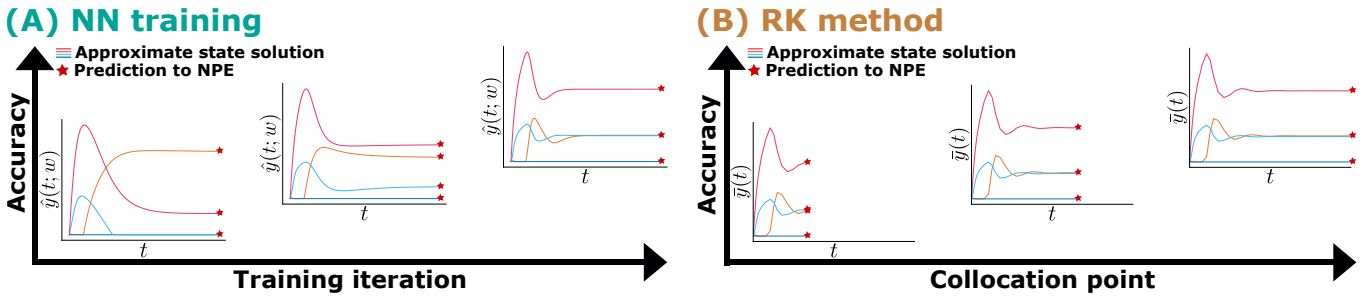

Figure 3 Comparison of the solution procedures between the NN approach and RK method.

4.3 | Comparison with the RK Method

Fig. 3 compares the solution procedures between our proposed NN approach and the RK method, both of which solve the NPE problem by solving the IVP. Both approaches use the end state as the predicted solution for the NPE problem. However, they differ in how they enhance the accuracy of the end state.

NN solution procedure. The NN approach employs gradient descent on the batch loss of Eq. (18) to improve the NN prediction at each iteration. The evolution of the NN model is represented by $\hat{y}(t; w_1), \hat{y}(t; w_2), \ldots, \hat{y}(t; w_M)$, where w_i and $\hat{y}(t; w_i)$ denote the model parameters and the approximate state solution at the i-th iteration, respectively. The predicted end states are $\hat{y}(t = T; w_1), \hat{y}(t = T; w_2), \ldots, \hat{y}(t = T; w_M)$, where $\hat{y}(t = T; w_i)$ represents the NPE prediction at the i-th iteration.

RK solution procedure. In contrast, the RK method computes discrete collocation points iteratively. The method progresses by solving Eq. (10) and Eq. (11) to obtain $\bar{y}_j$ and t_j, which represent the solved state values and collocation point at the j-th iteration, respectively. The state value $\bar{y}_j$ incorporates all previously solved state values. At the end of the j-th iteration, $\bar{y}_j$ is used as the prediction for the NPE.

5 | EXPERIMENTS

Section 5.1 delineates the application of the proposed NN approach for solving various types of NPE problems. Section 5.2 contrasts our approach with the PINN. Section 5.3 investigates the performance of our NN approach over different network architectures and hyperparameter configurations. Section 5.4 demonstrates the effectiveness of our NN approach in solving large-scale NPE problems. Finally, Section 5.5 discusses the distinctive features and limitations of our proposed NN approach, while also outlining possible avenues for future research.

5.1 | Three Examples

Experimental setup of our NN approach. We used PyTorch 1.12.1[40] to implement the proposed NN model and JAX 0.4.1[41] to implement the ODE system. The NN model consists of a single fully connected layer with 100 neurons, and the activation function is Tanh. For training, we used the Adam optimizer with a learning rate of 0.001, a batch size of 128.

Experimental setup of the RK method. We used the RK method[42] for comparison and called it via the Scipy library[43]. We set the number of collocation points to 50,000, evenly distributed over the time range.

5.1.1 | Linear Complementarity Problem

Example 1: Consider the following linear complementarity problem:

$$x^T(Mx + q) = 0, \quad x \geq \mathbf{0}, \quad Mx + q \geq \mathbf{0}, \tag{20}$$

where

$$M = \begin{bmatrix} 50 & -8 & -6 & -9 & 12 \\ -8 & 33 & -1 & -25 & 3 \\ -6 & -1 & 38 & 10 & -4 \\ -9 & -25 & 10 & 55 & -24 \\ 12 & 3 & -4 & -24 & 20 \end{bmatrix}, \quad q = \begin{bmatrix} -2 \\ -20 \\ -16 \\ -12 \\ -14 \end{bmatrix}. \tag{21}$$

The goal is to find an optimal solution $x^* \in \mathbb{R}^5$ that solves Eq. (20). Example 1 is reformulated as $NPE(\mathbb{R}^+, G)$ by Proposition 1, where $G(x) = Mx + q$. Then, $NPE(\mathbb{R}^+, G)$ is modeled by the ODE system of Eq. (7). We establish an IVP by specifying the initial condition as $y(0) = \mathbf{0}$ and the time range as $[0, 10]$. We use the NN model proposed in Eq. (13), denoted as $\hat{y}(t; w)$, to serve as an approximate state solution for this IVP. The end state, $\hat{y}(t = 10; w)$, serves as the predicted solution for both $NPE(\mathbb{R}^+, G)$ and Example 1. The NN model is trained using Alg. 1 to improve accuracy.

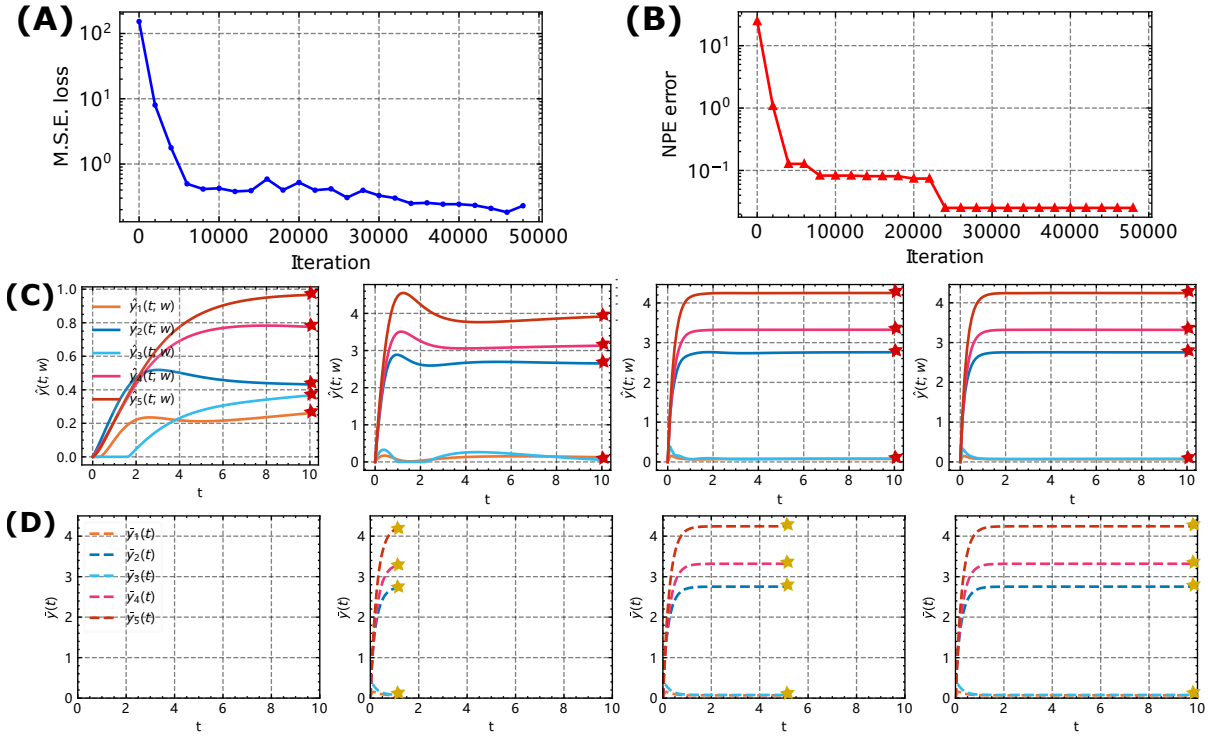

Figure 4 Solving Example 1: **(A)** Mean square error (MSE) loss versus training iterations. **(B)** NPE error versus training iterations, where the NPE error is defined in Eq. (2). **(C)** Evolution of the NN solution, $\hat{y}(t; w)$. **(D)** Evolution of the RK solution, $\bar{y}(t)$.

Fig. 4(A) and Fig. 4(B) show the decreasing loss and NPE error, respectively, where the loss drops from 152.59 to 0.23 and the NPE error drops from 24.69 to 0.03. Fig. 4(C) shows the NN model $\hat{y}(t; w)$ at the 0th, 1,000th, 10,000th, and 50,000th training iterations, from left to right, and the predicted solutions for Example 1 are marked with red stars. Fig. 4(D) shows the results of the RK method after accumulating 0, 5,000, 25,000, and 50,000 collocation points, and the predicted solutions are marked with yellow stars.

Table 1 shows the predictions of our NN approach and the RK method for Example 1 at different iterations. The results suggest that our approach is comparably accurate to the RK method given the same initial condition and time range. After 10,000 iterations, the NPE error of our prediction is reduced to less than 0.1. The final solution from our NN approach is $[0.07, 2.75, 0.08, 3.32, 4.24]$, with an NPE error of 0.03.

Table 1 Comparison of predicted solutions for Example 1 between our approach and the RK method

Our NN approach			The RK method		
Iteration	Prediction	NPE error	Collocation point	Prediction	NPE error
0	[0.26, 0.43, 0.37, 0.77, 0.97]	**24.67**	0	[0.00, 0.00, 0.00, 0.00, 0.00]	20.00
100	[0.17, 3.07, 0.00, 3.64, 4.50]	**2.32**	100	[0.05, 0.34, 0.22, 0.24, 0.28]	14.60
1000	[0.13, 2.65, 0.06, 3.13, 3.92]	**1.62**	1000	[0.15, 1.70, 0.26, 1.82, 2.20]	7.86
5000	[0.08, 2.76, 0.13, 3.32, 4.25]	**0.13**	5000	[0.07, 2.71, 0.09, 3.25, 4.15]	0.37
10000	[0.07, 2.76, 0.08, 3.33, 4.25]	**0.08**	10000	[0.07, 2.75, 0.08, 3.32, 4.24]	0.03
30000	[0.07, 2.75, 0.08, 3.32, 4.24]	**0.03**	30000	[0.07, 2.75, 0.08, 3.32, 4.24]	0.03
50000	[0.07, 2.75, 0.08, 3.32, 4.24]	**0.03**	50000	[0.07, 2.75, 0.08, 3.32, 4.24]	0.03

5.1.2 | Nonlinear Complementarity Problem

Example 2: Consider the following nonlinear complementarity problem:

$$x^T F(x) = 0, \quad x \geq \mathbf{0}, \quad F(x) \geq \mathbf{0}, \tag{22}$$

where

$$F(x) = \begin{pmatrix} x_1 e^{(x_1^2+(x_2-1)^2)} + x_2^2 + x_3 - 10 \\ (x_2-1)e^{x_1^2+(x_2-1)^2} + 4x_1 + x_2x_3 + 2x_3^2 + e^{x_4-2} \\ x_1 + 8x_2 + 3x_3 - 3 \\ x_4 - 4 \end{pmatrix}. \tag{23}$$

We reformulate Example 2 as $NPE(\mathbb{R}^+, F)$ by Proposition 1. Then, $NPE(\mathbb{R}^+, F)$ is modeled by the ODE system of Eq. (7). We establish an IVP by specifying the initial point as $y(0) = \mathbf{0}$ and the time range as $[0, 10]$. We use the NN model, $\hat{y}(t; w)$, to serve as an approximate state solution for this IVP, where the end state $\hat{y}(t = 10; w)$ is the predicted solution for Example 2.

Table 2 Comparison of predicted solutions for Example 2 between our approach and the RK method

Our NN approach			The RK method		
Iteration	Prediction	NPE error	Collocation point	Prediction	NPE error
0	[0.86, 0.00, 0.00, 0.00]	**5.04**	0	[0.00, 0.00, 0.00, 0.00]	10.00
100	[1.30, 0.94, 0.00, 2.01]	**2.00**	100	[0.19, 0.04, 0.05, 0.08]	9.44
1000	[1.08, 0.00, 0.59, 3.99]	**0.14**	1000	[1.16, 0.19, 0.20, 0.73]	3.27
5000	[1.08, 0.00, 0.64, 3.99]	**0.01**	5000	[1.22, 0.22, 0.00, 2.53]	1.47
10000	[1.08, 0.00, 0.64, 4.00]	**0.00**	10000	[1.10, 0.01, 0.40, 3.46]	0.62
30000	[1.08, 0.00, 0.64, 4.00]	**0.00**	30000	[1.08, 0.00, 0.64, 3.99]	0.08
50000	[1.08, 0.00, 0.64, 4.00]	**0.00**	50000	[1.08, 0.00, 0.64, 4.00]	0.09

Fig. 5(A) and Fig. 5(B) illustrate the decrease in loss and NPE error, respectively. Specifically, the loss decreases from 17.78 to 0.05, while the NPE error decreases from 5.04 to a value close to zero. Notably, the most substantial reduction in both loss and NPE error occurs within the first 10,000 iterations. In addition, the implementation of the ORR mechanism, as described in Alg. 1, ensures that the NPE error decreases consistently, despite occasional small increases in the loss values.

Fig. 5(C) shows the NN model $\hat{y}(t; w)$ at the 0th, 1,000th, 10,000th, and 50,000th training iterations, from left to right, and the predicted solution for Example 2 are marked with red stars. As shown in the figure, the NN model at the 1,000th iteration is already very close to the final result at the 50,000th iteration. Fig. 5(D) shows the results of the RK method after accumulating 0, 5,000, 25,000, and 50,000 collocation points, and the predicted solutions are marked with yellow stars.

Table 2 shows the predictions and their NPE errors of our NN approach and the RK method for Example 2 at different iterations. Thanks to the adoption of the projection function in Alg. 1, our NN approach has a lower initial NPE error compared

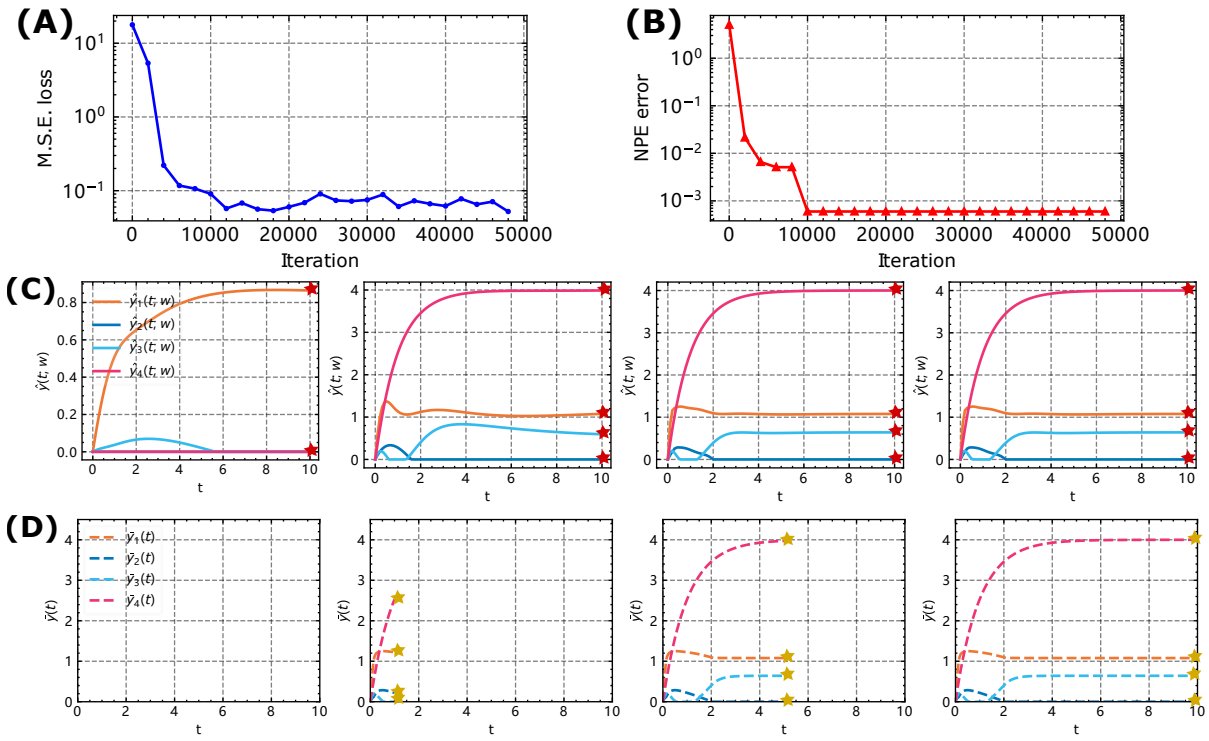

Figure 5 Solving Example 2: **(A)** MSE loss versus training iterations. **(B)** NPE error versus training iterations. **(C)** Evolution of the NN solution, $\hat{y}(t; w)$. **(D)** Evolution of the RK solution, $\bar{y}(t)$.

to the RK method. By the 1,000th iteration, the NPE error associated with our NN approach has been reduced to 0.14. By the 5,000th iteration, the NPE error has further diminished to 0.01. Upon completion of 10,000 iterations, our NN approach yields an optimal solution of $[1.08, 0.00, 0.64, 4.00]$, which accurately resolves Example 2.

5.1.3 | Variational Inequality

Example 3: Consider the following variational inequality:

$$(x - x^*)^T G(x^*) \geq 0, \quad x \in \Omega, \tag{24}$$

where

$$G(x) = \begin{bmatrix} x_1 - \frac{2}{x_1+0.8} + 5x_2 - 13 \\ 1.2x_1 + 7x_2 \\ 3x_3 + 8x_4 \\ 1x_3 + 2x_4 - \frac{4}{x_4+2} - 12 \end{bmatrix}, \quad \begin{aligned} \Omega = \{x \in \mathbb{R}^4 \,|\, & 1 \leq x_1 \leq 100, -3 \leq x_2 \leq 100, \\ & -10 \leq x_3 \leq 100, 1 \leq x_4 \leq 100\}. \end{aligned} \tag{25}$$

We reformulate Example 3 as $NPE(\Omega, G)$ by Proposition 2. Then, $NPE(\Omega, G)$ is modeled by the ODE system of Eq. (7). We establish an IVP by specifying initial point as $y(0) = \mathbf{0}$ and the time interval as $[0, 10]$. We use the NN model, $\hat{y}(t; w)$, to serve as an approximate state solution for this IVP, where the end state $\hat{y}(t = 10; w)$ is the predicted solution for Example 3.

Fig. 6(A) and Fig. 6(B) illustrate the decrease in loss and NPE error, respectively. Specifically, the loss decreases from 126.99 to 0.10, while the NPE error decreases from 15.42 to 0.00. Note that there are small fluctuations in the loss value around the 18,000th iteration, but the NPE error remains unaffected because the training algorithm retains the best prediction from its training history.

Fig. 6(C) shows the NN model $\hat{y}(t; w)$ at the 0th, 1,000th, 10,000th, and 50,000th training iterations, from left to right, and the predicted solution for Example 3 are marked with red stars. In particular, at the 1,000th iteration, i.e., the first subplot on the left in Fig. 6(C), the NN model is already very close to the final result of the 50,000th iteration. Fig. 6(D) shows the results of the RK method after accumulating 0, 5,000, 25,000, and 50,000 collocation points, and the predicted solutions are marked with yellow stars.

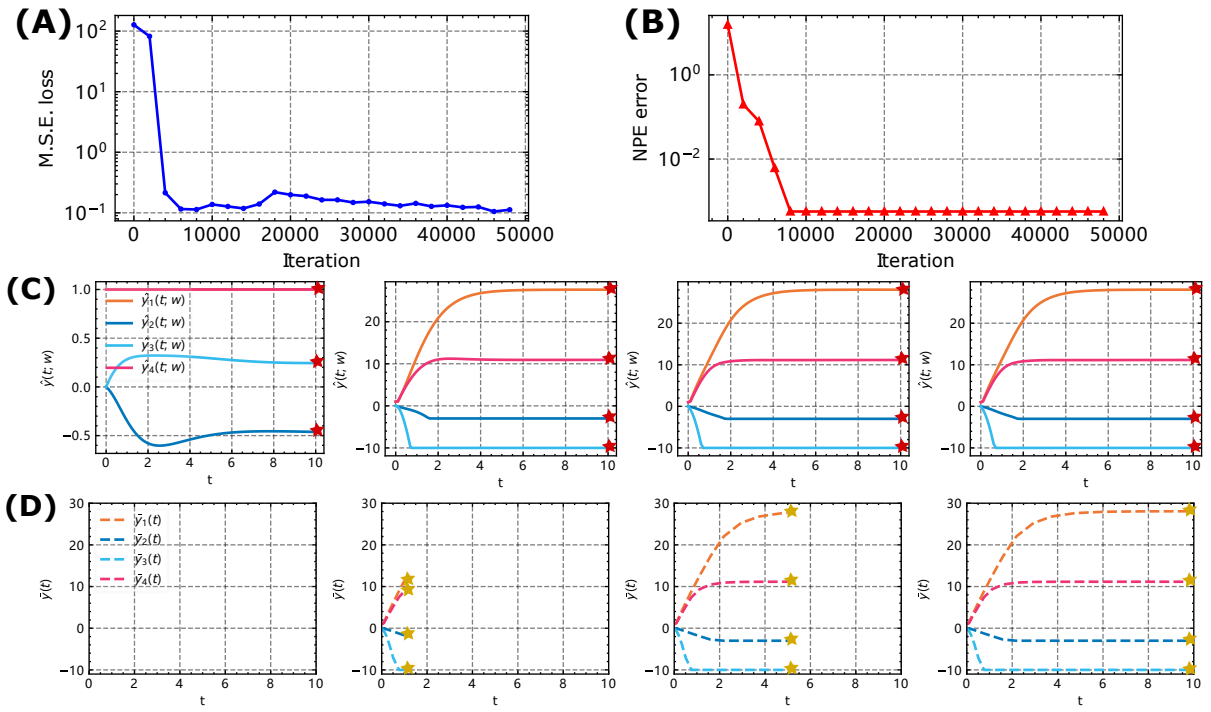

Figure 6 Solving Example 3: **(A)** MSE loss versus training iterations. **(B)** NPE error versus training iterations. **(C)** Evolution of the NN solution, $\hat{y}(t; w)$. **(D)** Evolution of the the RK solution, $\bar{y}(t)$.

Table 3 Comparison of predicted solutions for Example 3 between our approach and the RK method

Our NN approach			The RK method		
Iteration	Prediction	NPE error	Collocation point	Prediction	NPE error
0	[1.00, -0.46, 0.24, 1.00]	**15.42**	0	[1.00, 0.00, 0.00, 1.00]	13.11
100	[1.77, 0.12, -1.57, 1.66]	**11.43**	100	[1.00, -0.02, -0.16, 1.00]	13.22
1000	[27.62, -3.00, -10.00, 10.92]	**0.46**	1000	[2.61, -0.22, -1.75, 2.28]	12.09
5000	[28.15, -3.00, -10.00, 11.18]	**0.08**	5000	[11.38, -1.69, -10.00, 8.88]	10.26
10000	[28.07, -3.00, -10.00, 11.15]	**0.00**	10000	[20.78, -3.00, -10.00, 10.87]	7.32
30000	[28.07, -3.00, -10.00, 11.15]	**0.00**	30000	[27.96, -3.00, -10.00, 11.15]	0.11
49999	[28.07, -3.00, -10.00, 11.15]	**0.00**	49999	[28.07, -3.00, -10.00, 11.15]	0.00

Table 3 shows the predictions for Example 3 provided by our NN approach and the RK method. Remarkably, after only 1,000 iterations, our NN approach yields an acceptable prediction with an NPE error of 0.46. After 5,000 iterations, our NN approach refines this prediction further to an NPE error of less than 0.1. After 10,000 iterations, our approach converges to the solution of $[28.07, -3.00, -10.00, 11.15]$, which is an optimal solution of Example 3.

5.2 | Comparision with PINN

In this subsection, we compare our approach with PINNs in terms of accuracy. Specifically, we have chosen the basic version of PINN[20] for comparison. Despite the existence of more advanced models such as CPINNs[44] or XPINNs[45], however, we find that the basic version of PINN is sufficiently effective for the NPE problem, obviating the need for more complex variants. Regarding the setting for the PINN, we use the same network architecture and hyperparameters as in Section 5.1.

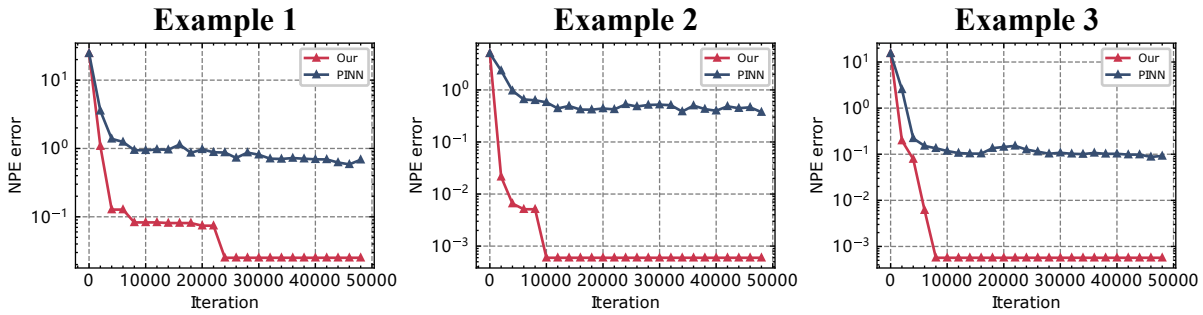

Figure 7 Comparison of NPE errors between our proposed approach and the PINN for the three examples given in Section 5.1.

Our approach can be viewed as a modification of the PINN, specifically designed for the NPE problem to improve computational performance. The core distinction between our approach and the PINN lies in our focus on the end state of the NN model. Building on this, we employ the ORR mechanism in Alg. 1, which continuously monitors the NPE error of the end state throughout the training process. Therefore, the NN model optimizes simultaneously for both the ODE system and the NPE problem at hand, consistently maintaining the best result throughout the solution process. This unique focus results in improved performance, as demonstrated below.

Fig. 7 compares our approach with the PINN for solving the three examples given in Section 5.1. As shown in the figure, both approaches start with similar initial errors, but our approach is significantly outperforms the PINN as training progresses. Specifically, in Example 1, the NPE error converges to 0.03 with our approach, while the PINN converges to 0.7. In Example 2, our approach converges to less than 0.001, while the PINN could only converge to 0.4. In Example 3, our approach again converges to less than 0.001, while the PINN only converges to 0.1. These experimental results show that our approach provides superior solutions for solving these NPE problems. Moreover, this validates the effectiveness of the key designs in Alg. 1, such as the use of the ORR mechanism and the projection function.

5.3 | Hyperparameter Study

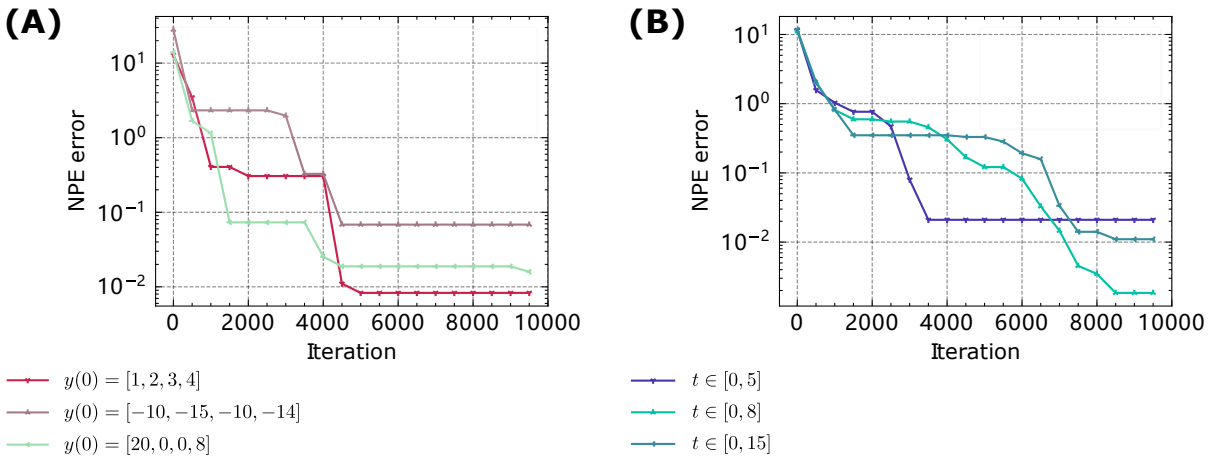

Figure 8 **(A)** NPE errors versus training iterations for three different initial points. **(B)** NPE errors versus training iterations for three different time ranges.

In this subsection, we explore the influence of various hyperparameters on the computational performance of our NN model. Specifically, we focus on Example 3, as given in Section 5.1.3. The hyperparameters under investigation include the initial point, the time range, the number of hidden layers, the number of neurons, and the activation function used.

Table 4 Predictions and their NPE errors at different training iterations for three different initial points

	Initial point: $y(0) = [1, 2, 3, 4]$		Initial point: $y(0) = [-10, -15, -10, -14]$		Initial point: $y(0) = [20, 0, 0, 8]$	
Iteration	Prediction	NPE error	Prediction	NPE error	Prediction	NPE error
0	[1.00, 2.79, 2.86, 3.18]	12.86	[1.00, -3.00, -10.00, 1.00]	28.11	[20.10, 1.38, 0.23, 7.94]	13.89
100	[4.37, -0.06, -0.55, 5.96]	9.45	[1.00, -3.00, -10.00, 1.00]	28.11	[17.92, -2.25, -3.62, 6.54]	6.43
300	[23.89, -3.00, -10.00, 13.22]	4.19	[11.23, 5.10, 0.28, 6.14]	10.28	[26.40, -3.00, -10.00, 11.98]	1.68
500	[25.77, -3.00, -10.00, 12.87]	3.48	[14.21, -0.67, -10.00, 10.50]	2.33	[26.40, -3.00, -10.00, 11.98]	1.68
1000	[28.47, -3.00, -10.00, 11.28]	0.41	[14.21, -0.67, -10.00, 10.50]	2.33	[27.88, -3.00, -10.00, 11.72]	1.15
3000	[28.37, -3.00, -10.00, 11.30]	0.31	[30.04, -3.00, -10.00, 12.00]	1.97	[28.14, -3.00, -10.00, 11.14]	0.07
5000	[28.06, -3.00, -10.00, 11.16]	0.01	[28.01, -3.00, -10.00, 11.19]	0.07	[28.09, -3.00, -10.00, 11.15]	0.02
10000	[28.06, -3.00, -10.00, 11.16]	0.01	[28.01, -3.00, -10.00, 11.19]	0.07	[28.06, -3.00, -10.00, 11.16]	0.01

Table 5 Prediction and their NPE errors at different training iterations for three different time ranges.

	Time range: $t \in [0, 5]$		Time range: $t \in [0, 8]$		Time range: $t \in [0, 15]$	
Iteration	Predition	NPE error	Predition	NPE error	Predition	NPE error
0	[1.00, 1.23, -0.22, 1.00]	11.56	[1.00, 0.63, 0.15, 1.00]	11.183	[1.00, 0.90, -0.40, 1.00]	11.74
100	[5.47, -0.52, -6.10, 5.25]	10.44	[5.53, -0.56, -5.13, 4.76]	10.59	[4.30 -0.236 -4.644 4.08]	10.27
300	[22.82, -3.00, -10.00, 14.27]	6.28	[23.81, -3.00, -10.00, 14.69]	7.14	[22.12, -3.00, -10.00, 14.18]	6.12
500	[27.04, -3.00, -10.00, 11.93]	1.57	[27.73, -3.00, -10.00, 12.17]	2.06	[27.14, -3.00, -10.00, 12.14]	2.00
1000	[27.04, -3.00, -10.00, 11.66]	1.03	[27.26, -3.00, -10.00, 11.55]	0.81	[27.25, -3.00, -10.00, 11.55]	0.82
3000	[28.08, -3.00, -10.00, 11.11]	0.08	[28.62, -3.00, -10.00, 10.91]	0.55	[27.77, -3.00, -10.00, 10.98]	0.35
5000	[28.05, -3.00, -10.00, 11.16]	0.02	[28.18, -3.00, -10.00, 11.21]	0.12	[28.40, -3.00, -10.00, 11.32]	0.33
9999	[28.05, -3.00, -10.00, 11.16]	0.02	[28.07, -3.00, -10.00, 11.15]	0.00	[28.08, -3.00, -10.00, 11.15]	0.01

Initial point. Fig. 8(A) and Table 4 show the results of different initial point configurations with the same time range of $[0, 10]$. The results suggest the following:

- All initial points converge to the same optimal solution, as supported by Theorem 1.
- The convergence rates of different initial points vary, with initial points closer to the optimal solution converging faster.
- The initial point $y(0) = [1, 2, 3, 4]$ is closest to the optimal solution and achieves the fastest convergence with the smallest initial NPE error.
- The initial point $y(0) = [-10, -15, -10, -14]$ is the furthest away from the optimal solution and still converges, but has a higher initial error and slower convergence rate.

Time range. Fig. 8(B) and Table 5 show the results of different time range configurations with the same initial points of $y(0) = [0, 0, 0, 0]$. The results suggest the following:

- Shorter time ranges lead to faster convergence but may result in less accurate predictions. As shown in the table, the shortest time range of $[0, 5]$ converges very fast, but its NPE error does not decrease much after 3,000 iterations.
- Longer time ranges provide better predictions, but require more training iterations. The longest range of $[0, 15]$ converges slowly, but with more training it can give better results than the other two ranges.
- The choice of time ranges represents a trade-off. Longer ranges may enhance accuracy but require more training, whereas shorter ranges are easier to train but may yield less satisfactory predictions. As shown in Table 5, considering a fixed maximum number of iterations at 10,000, the time range of $[0, 8]$ achieves the optimal performance.

Table 6 Comparison of NPE errors for NNs with different model sizes. The top half of the table presents the results for single-layer networks with different numbers of neurons, while the bottom half presents the results for multi-layer networks with 500 neurons per layer.

Iteration	1 layer, 100 neurons	1 layer, 500 neurons	1 layer, 1000 neurons	1 layer, 1500 neurons	1 layer, 2000 neurons
100	11.43	0.33	0.23	0.78	2.32
300	7.08	0.33	0.23	0.75	0.71
500	2.07	0.33	0.23	0.52	0.71
1000	0.46	0.33	0.23	0.16	0.29
3000	0.10	0.00	0.03	0.02	0.02
5000	0.08	0.00	0.00	0.01	0.01
10000	0.00	0.00	0.00	0.00	0.01

Iteration	1 layer, each 500 neurons	2 layer, each 500 neurons	3 layer, each 500 neurons	4 layer, each 500 neurons	5 layer, each 500 neurons
100	0.33	1.73	2.22	6.43	6.88
300	0.33	0.10	0.76	1.82	6.70
500	0.33	0.07	0.27	0.57	0.52
1000	0.33	0.01	0.03	0.23	0.34
3000	0.00	0.01	0.01	0.03	0.03
5000	0.00	0.00	0.00	0.03	0.03
10000	0.00	0.00	0.00	0.02	0.03

Number of layers and neurons. Table 6 shows the NPE errors of different model sizes, specifically different numbers of hidden layers and neurons. As shown in the table, all models of different sizes converge to solutions with NPE errors below 0.03 by the 10,000th iteration. However, the model size has a significant impact on the speed of convergence. Models that are too small or too large perform worse than the others. For example, the 100-neuron single-layer model has higher NPE errors in the first 1000 iterations. The 5-layer model with 500 neurons per layer has similar results.

Therefore, selecting an appropriately-sized model is importance to achieve optimal performance. Small model sizes have limited capacity, while large model sizes require a lot of training to converge, which may not be necessary. Among the model sizes considered in Table 6, the two-layer model with 500 neurons each achieves the best performance. It converges to an NPE error of 0.1 within 300 iterations and further reduces to 0.01 within 1000 iterations, outperforming other model architectures.

Table 7 Comparison of NPE errors for different activation functions. The considered NN model is a two-layer network, with 500 neurons in each layer.

Iteration	2 layers, each 500 neurons				
	tanh	sinx	sigmoid	relu	leaky relu
100	1.73	2.18	3.01	2.62	2.32
300	0.10	0.73	0.28	1.13	0.71
500	0.07	0.15	0.05	1.13	0.71
1000	0.01	0.03	0.01	1.13	0.29
3000	0.01	0.03	0.01	0.01	0.01
5000	0.00	0.03	0.01	0.01	0.01
10000	0.00	0.03	0.00	0.01	0.00

Activation Function. Table 7 shows the NPE errors for different activation functions on the same NN model. The data show that regardless of the activation function chosen, all NN models converge to solutions with NPE errors less than 0.03. Notably, the influence of the activation function on model performance is relatively minor when compared to the effects of model size. Among the tested activation functions, ReLU and Leaky ReLU were found to be slightly less effective than the others. Specifically, the tanh activation function stands out as the most efficient, which corroborates its widespread adoption in research related to PINNs[46,47,48].

5.4 | Large Scale NPE

Consider the following NPE problem: For $i = 1, 2, \dots, 1000$,

$$\left(x_i^* - \frac{1}{2\sqrt{(Mx^*)_i + q_i}} \right)^+ = x_i^*. \tag{26}$$

The objective is to find an optimal solution $x^* = [x_1^*, x_2^*, \dots, x_{1000}^*] \in \mathbb{R}^{1000}$ that solves Eq. (26). The problem data $M \in \mathbb{R}^{1000\times1000}$ is partitioned as

$$M = \begin{bmatrix} M_1 & M_2 \\ M_3 & M_4 \end{bmatrix}, \tag{27}$$

where $M_1 \in \mathbb{R}^{500\times500}$, $M_2 \in \mathbb{R}^{500\times500}$, $M_3 \in \mathbb{R}^{500\times500}$, and $M_4 \in \mathbb{R}^{500\times500}$ are given by

$$M_1 = \begin{bmatrix} 1.2 & 0.6 & \dots & 0.6 \\ 0.6 & 1.2 & \dots & 0.6 \\ \vdots & \vdots & \ddots & \vdots \\ 0.6 & 0.6 & \dots & 1.2 \end{bmatrix}, \quad M_2 = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}, \quad M_3 = \begin{bmatrix} -1 & 0 & \dots & 0 \\ 0 & -1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & -1 \end{bmatrix}, \quad M_4 = \begin{bmatrix} 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix}. \tag{28}$$

For the problem data $q \in \mathbb{R}^{1000}$, the first half is sampled uniformly from the interval $[-31, -28]$, while the second half is sampled uniformly from the interval $[3, 10]$. We create a problem set by generating ten different q, resulting in ten different NPE problems. All the sampled q can be accessed through the link [1]. Solving these NPE problems is non-trivial due to their high dimensionality with 1,000 variables and the presence of multiple nonlinear operations. Therefore, the RK method may encounter computational inefficiencies when applied to these NPE problems.

Table 8 Performance of the NN approach for solving the set of NPE problems. STD stands for standard deviation. CPU time is measured in seconds.

Iteration	M.S.E. loss (Mean ± STD)	CPU time (Mean ± STD)	NPE error (Mean ± STD)	NPE error (50%-quantile)	NPE error (75%-quantile)	NPE error (95%-quantile)
0	470.29 ± 92.68	0.00 ± 0.00	4839.33 ± 673.10	4720.42	4911.21	6008.41
100	45.85 ± 39.83	2.36 ± 0.60	15.34 ± 16.46	6.65	19.98	46.38
500	32.10 ± 26.70	11.32 ± 0.68	0.67 ± 0.89	0.37	0.46	2.30
1000	26.81 ± 11.53	22.57 ± 1.16	0.31 ± 0.18	0.27	0.36	0.60
3000	15.20 ± 11.23	67.82 ± 1.40	0.19 ± 0.06	0.18	0.24	0.27
5000	12.33 ± 10.44	112.91 ± 2.20	0.13 ± 0.07	0.11	0.17	0.24
7000	8.16 ± 8.13	157.98 ± 2.86	0.10 ± 0.06	0.07	0.14	0.19
10000	2.93 ± 3.38	225.47 ± 3.82	0.05 ± 0.04	0.04	0.07	0.12
30000	0.73 ± 1.18	670.38 ± 5.82	0.01 ± 0.01	0.01	0.01	0.02

We use the proposed NN approach to solve the ten large-scale NPE problems. With respect to the experimental setup, the employed network architecture consisted of a three-layer fully connected network, each layer having 500 neurons and utilizing the tanh activation function. The time range chosen for the experiments is $[0, 10]$, and the initial point is set to a zero vector. All

[1] https://github.com/wuwudawen/IJNME_data_2023/blob/main/Q.npy

Table 9 Performance of the RK method for solving the set of NPE problems STD stands for standard deviation. CPU time is measured in seconds.

Time range	CPU time (Mean ± STD)	NPE error (Mean ± STD)	NPE error (50%-quantile)	NPE error (75%-quantile)	NPE error (95%-quantile)
[0, 2]	369.33 ± 44.27	1.87 ± 0.16	1.86	2.01	2.08
[0, 4]	547.93 ± 74.83	0.48 ± 0.06	0.50	0.51	0.55
[0, 6]	739.74 ± 88.02	0.14 ± 0.02	0.15	0.15	0.16
[0, 8]	1048.92 ± 88.20	0.04 ± 0.01	0.04	0.05	0.05
[0, 10]	1542.07 ± 205.43	0.01 ± 0.01	0.01	0.01	0.02

other hyperparameter settings are kept consistent with those described in Section 5.1. Tables 8 and 9 present the experimental results of our NN approach compared to the RK method, focusing on both accuracy and computational time. In order to offer a comprehensive understanding of the experimental results, we provide statistical descriptors of the outcomes, including the mean, standard deviation (STD), and various percentiles.

In the following, we discuss the differences between our NN approach and the RK method in terms of computational efficiency, stability and convergence.

- Efficiency: As shown in the tables, our NN approach outperforms the RK method in terms of computational efficiency when solving large scale NPE problems. In particular, our NN approach requires less CPU time than the RK method for the same level of accuracy. For example, our approach achieves an NPE error of 0.01, requiring an average of 30,000 iterations and consuming only 670.38 seconds. On the contrary, the RK method requires spanning a time range of [0, 10], which requires a higher average time of 1542.07 seconds. Moreover, the efficiency gap becomes more significant as the required accuracy is relaxed. For example, for an NPE error threshold of less than 0.50, our NN approach takes on average only 500 iterations and 11.32 seconds, making it about 48 times faster than the RK method, which requires 547.93 seconds.

- Stability: Our NN approach demonstrates not only computational efficiency, but also greater stability, reflected in a much lower STD of CPU times - only 5.82 at the 30,000th iteration. This indicates consistent and reliable performance across different NPE problem instances. On the other hand, the RK method exhibits higher variability with an STD of 205.43 for the time range of [0, 10], signifying greater sensitivity to the specific NPE problem at hand.

- Convergence: While there are notable differences in computational efficiency and stability between our NN approach and the RK method, both techniques demonstrate comparable accuracy in the long run, each achieving an NPE error as low as 0.01. As discussed in Section 4.3, the convergence mechanisms of the two approaches are fundamentally different. Our NN approach operates within a fixed time range of [0, 10] and refines its predictive accuracy through iterative training. In contrast, the RK method improves its accuracy by progressively extending the time range.

5.5 | Discussion

Below, we summarize the key features of our proposed NN approach:

- Our NN approach reliably converges to the optimal solutions of the NPE problems. This is supported by Theorem 1 in neurodynamic optimization and the universal approximation theorem of NNs. Experimentally, we have shown that the NN approach successfully found optimal solutions for the three types of NPE problems in Section 5.1, as well as ten large-scale NPE problems in Section 5.4.

- The proposed NN approach outperforms the PINN in solving NPE problems. This improvement is due to some modifications made to the basic PINN approach that allow it to exploit the problem structure of NPEs for better performance. In particular, during each training iteration, the NN model evaluates the accuracy of its end state against the target NPE problem and retains the best performing solution. Additionally, we employ the projection function of Eq. (1) to further boost accuracy.

- The computational performance of our NN approach is significantly affected by the hyperparameter settings. As discussed in Section 5.3, choices regarding the time range and initial point have a strong impact on the convergence result. For example, smaller time ranges may prevent the NN model from converging to an optimal solution, regardless of the training time. Moreover, in our empirical observations, the network architecture and activation functions influence the model convergence rate.
- Our NN approach excels in solving large-scale NPE problems. As presented in Section 5.4, our NN approach outperforms the RK method in terms of computational efficiency. For the same level of accuracy, the CPU time required by our approach is less than that required by the RK method. In addition, our approach exhibits greater stability, with its solution time remaining consistent across different NPE problems.

However, we must acknowledge some limitations of our proposed NN approach. The most notable limitation is that, unlike traditional RK methods, the NN approach lacks rigorous theoretical underpinnings to guarantee convergence, primarily due to the black-box nature of NNs. Furthermore, as elaborated in Section 5.3, the performance of the model is highly sensitive to hyperparameter choices and architectural decisions, requiring careful tuning. Numerical integration methods, which are typically simpler, avoid these complexities.

To address these limitations, future research should focus on improving network design and training methods. This could include developing more effective hyperparameter search strategies or exploring advanced network architectures, such as attention-based models or transformers. In addition, investigating how to incorporate domain-specific knowledge into the network structure may improve solution accuracy and reduce training time.

6 | CONCLUSION

In this paper, we presented an innovative deep learning-based approach for solving NPEs based on neurodynamic optimization and PINNs. We showed how our approach can efficiently solve NPEs and highlighted its advantages over PINNs and the RK method. The proposed approach transforms NPE problems into NN training problems, allowing the use of the latest advances in machine learning and deep learning to solve NPEs. We also identified areas for future research, such as exploring better methods for selecting initial points and time ranges, experimenting with different network architectures, and investigating advanced neurodynamic optimization techniques, among others.

In summary, our proposed framework shows considerable promise for improving computational efficiency in solving NPEs. With ongoing research and development, we expect to further strengthen the robustness of our approach and position it as a valuable computational tool for addressing a wide range of nonlinear optimization challenges in diverse applications.

DATA AVAILABILITY STATEMENT

The problem data used in Section 5.4 are available in https://github.com/wuwudawen/IJNME_data_2023/blob/main/Q.npy.

References

1. Bertsekas DP. Nonlinear programming. *Journal of the Operational Research Society* 1997; 48(3): 334.
2. Harker PT, Pang JS. Finite-dimensional variational inequality and nonlinear complementarity problems: a survey of theory, algorithms and applications. *Mathematical programming* 1990; 48(1-3): 161–220.
3. Robinson SM. Normal maps induced by linear transformations. *Mathematics of Operations Research* 1992; 17(3): 691–714.
4. Xia Y, Feng G. A new neural network for solving nonlinear projection equations. *Neural Networks* 2007; 20(5): 577–589.
5. Xia Y, Wang J. A Bi-Projection Neural Network for Solving Constrained Quadratic Optimization Problems. *IEEE Transactions on Neural Networks and Learning Systems* 2016; 27(2): 214–224. doi: 10.1109/TNNLS.2015.2500618

6. Che H, Wang J. A collaborative neurodynamic approach to global and combinatorial optimization. *Neural Networks* 2019; 114: 15–27.

7. Burden RL, Faires JD, Burden AM. *Numerical analysis*. Cengage learning . 2015.

8. Tank DW, Hopfield JJ. SIMPLE 'NEURAL' OPTIMIZATION NETWORKS: AN A/D CONVERTER, SIGNAL DECISION CIRCUIT, AND A LINEAR PROGRAMMING CIRCUIT.. *IEEE transactions on circuits and systems* 1986; CAS-33(5): 533–541. doi: 10.1109/tcs.1986.1085953

9. Liu Q, Wang J. A one-layer recurrent neural network with a discontinuous hard-limiting activation function for quadratic programming. *IEEE transactions on neural networks* 2008; 19(4): 558–570.

10. Kennedy MP, Chua LO. Neural Networks for Nonlinear Programming. *IEEE Transactions on Circuits and Systems* 1988; 35(5): 554–562. doi: 10.1109/31.1783

11. Xia Y, Leung H, Wang J. A projection neural network and its application to constrained optimization problems. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 2002; 49(4): 447–458.

12. Guo Z, Liu Q, Wang J. A one-layer recurrent neural network for pseudoconvex optimization subject to linear equality constraints. *IEEE Transactions on Neural Networks* 2011; 22(12): 1892–1900.

13. Che H, Wang J. A two-timescale duplex neurodynamic approach to biconvex optimization. *IEEE Transactions on Neural Networks and Learning Systems* 2018; 30(8): 2503–2514.

14. Jia W, Liu N, Qin S. An Adaptive Continuous-Time Algorithm for Nonsmooth Convex Resource Allocation Optimization. *IEEE Transactions on Automatic Control* 2022; 67(11): 6038–6044. doi: 10.1109/TAC.2021.3137054

15. Liu N, Wang J, Qin S. A one-layer recurrent neural network for nonsmooth pseudoconvex optimization with quasiconvex inequality and affine equality constraints. *Neural Networks* 2022; 147: 1–9.

16. Lagaris IE, Likas A, Fotiadis DI. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks* 1998; 9(5): 987–1000. doi: 10.1109/72.712178

17. McFall KS, Mahan JR. Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions. *IEEE Transactions on Neural Networks* 2009; 20(8): 1221–1233.

18. Han J, Jentzen A, E W. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences* 2018; 115(34): 8505–8510.

19. Huang S, Feng W, Tang C, Lv J. Partial Differential Equations Meet Deep Neural Networks: A Survey. *arXiv preprint arXiv:2211.05567* 2022.

20. Raissi M, Perdikaris P, Karniadakis GE. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics* 2019; 378: 686–707. doi: https://doi.org/10.1016/j.jcp.2018.10.045

21. Samaniego E, Anitescu C, Goswami S, et al. An energy approach to the solution of partial differential equations in computational mechanics via machine learning: Concepts, implementation and applications. *Computer Methods in Applied Mechanics and Engineering* 2020; 362: 112790.

22. Mao Z, Jagtap AD, Karniadakis GE. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering* 2020; 360: 112789.

23. Cai S, Mao Z, Wang Z, Yin M, Karniadakis GE. Physics-informed neural networks (PINNs) for fluid mechanics: A review. *Acta Mechanica Sinica* 2022: 1–12.

24. Abueidda DW, Lu Q, Koric S. Meshless physics-informed deep learning method for three-dimensional solid mechanics. *International Journal for Numerical Methods in Engineering* 2021; 122(23): 7182–7201.

25. Ghaffari Motlagh Y, Jimack PK, Borst dR. Deep learning phase-field model for brittle fractures. *International Journal for Numerical Methods in Engineering* 2022.

26. Lu L, Pestourie R, Yao W, Wang Z, Verdugo F, Johnson SG. Physics-Informed Neural Networks with Hard Constraints for Inverse Design. *SIAM Journal on Scientific Computing* 2021; 43(6): B1105–B1132. doi: 10.1137/21M1397908

27. Zhang D, Guo L, Karniadakis GE. Learning in Modal Space: Solving Time-Dependent Stochastic PDEs Using Physics-Informed Neural Networks. *SIAM Journal on Scientific Computing* 2020; 42(2): A639–A665. doi: 10.1137/19M1260141

28. Yu J, Lu L, Meng X, Karniadakis GE. Gradient-enhanced physics-informed neural networks for forward and inverse PDE problems. *Computer Methods in Applied Mechanics and Engineering* 2022; 393: 114823.

29. Sharma R, Shankar V. Accelerated Training of Physics Informed Neural Networks (PINNs) using Meshless Discretizations. *arXiv preprint arXiv:2205.09332* 2022.

30. Rezaei S, Harandi A, Moeineddin A, Xu BX, Reese S. A mixed formulation for physics-informed neural networks as a potential solver for engineering problems in heterogeneous domains: Comparison with finite element method. *Computer Methods in Applied Mechanics and Engineering* 2022; 401: 115616. doi: 10.1016/j.cma.2022.115616

31. Abueidda DW, Koric S, Guleryuz E, Sobh NA. Enhanced physics-informed neural networks for hyperelasticity. *International Journal for Numerical Methods in Engineering* 2023; 124(7): 1585–1601.

32. He J, Abueidda D, Al-Rub RA, Koric S, Jasiuk I. A deep learning energy-based method for classical elastoplasticity. *International Journal of Plasticity* 2023; 162: 103531.

33. Samaniego E, Anitescu C, Goswami S, et al. An energy approach to the solution of partial differential equations in computational mechanics via machine learning: Concepts, implementation and applications. *Computer Methods in Applied Mechanics and Engineering* 2020; 362: 112790.

34. Yu B, others . The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics* 2018; 6(1): 1–12.

35. Lu L, Meng X, Mao Z, Karniadakis GE. DeepXDE: A Deep Learning Library for Solving Differential Equations. *SIAM Review* 2021; 63(1): 208–228. doi: 10.1137/19m1274067

36. Chen F, Sondak D, Protopapas P, et al. NeuroDiffEq: A Python package for solving differential equations with neural networks. *Journal of Open Source Software* 2020; 5(46): 1931. doi: 10.21105/joss.01931

37. Billups SC, Murty KG. Complementarity problems. *Journal of Computational and Applied Mathematics* 2000; 124(1-2): 303–318. doi: 10.1016/S0377-0427(00)00432-5

38. Singh VV, Lisser A. Variational inequality formulation for the games with random payoffs. *Journal of Global Optimization* 2018; 72(4): 743–760. doi: 10.1007/s10898-018-0664-8

39. Mattheakis M, Sondak D, Dogra AS, Protopapas P. Hamiltonian neural networks for solving equations of motion. *arXiv preprint arXiv:2001.11107* 2020.

40. Paszke A, Gross S, Massa F, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Curran Associates, Inc. 2019 (pp. 8024–8035).

41. Bradbury J, Frostig R, Hawkins P, et al. JAX: composable transformations of Python+NumPy programs. 2018.

42. Dormand JR, Prince PJ. A family of embedded Runge-Kutta formulae. *Journal of computational and applied mathematics* 1980; 6(1): 19–26.

43. Virtanen P, Gommers R, Oliphant TE, et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods* 2020; 17(3): 261–272. doi: 10.1038/s41592-019-0686-2

44. Jagtap AD, Kharazmi E, Karniadakis GE. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering* 2020; 365: 113028.

45. Jagtap AD, Karniadakis GE. Extended Physics-informed Neural Networks (XPINNs): A Generalized Space-Time Domain Decomposition based Deep Learning Framework for Nonlinear Partial Differential Equations.. 2021.

46. Jagtap AD, Kawaguchi K, Karniadakis GE. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics* 2020; 404: 109136.

47. De Ryck T, Lanthaler S, Mishra S. On the approximation of functions by tanh neural networks. *Neural Networks* 2021; 143: 732–750.

48. Mishra S, Molinaro R. Estimates on the generalization error of physics-informed neural networks for approximating a class of inverse problems for PDEs. *IMA Journal of Numerical Analysis* 2022; 42(2): 981–1022.