



HAL
open science

Solving Constrained Pseudoconvex Optimization Problems with Deep Learning-Based Neurodynamic Optimization

Dawen Wu, Abdel Lisser

► **To cite this version:**

Dawen Wu, Abdel Lisser. Solving Constrained Pseudoconvex Optimization Problems with Deep Learning-Based Neurodynamic Optimization. *Mathematics and Computers in Simulation*, 2024, 10.1016/j.matcom.2023.12.032 . hal-04370956

HAL Id: hal-04370956

<https://hal.science/hal-04370956>

Submitted on 3 Jan 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Solving Constrained Pseudoconvex Optimization Problems with Deep Learning-Based Neurodynamic Optimization

Dawen Wu^a (dawen.wu@centralesupelec.fr), Abdel Lisser^a (abdel.lisser@l2s.centralesupelec.fr)

^a Université Paris-Saclay, CNRS, CentraleSupélec, Laboratoire des signaux et systèmes, 91190, Gif-sur-Yvette, France

Corresponding Author:

Dawen Wu

Address: Université Paris-Saclay, CNRS, CentraleSupélec, Laboratoire des signaux et systèmes, 91190, Gif-sur-Yvette, France

Tel: (+33) 750798387

Email: dawen.wu@centralesupelec.fr

Solving Constrained Pseudoconvex Optimization Problems with Deep Learning-Based Neurodynamic Optimization

Dawen Wu^{a,*}, Abdel Lisser^a

^a*Université Paris-Saclay, CNRS, CentraleSupélec, Laboratoire des signaux et systèmes, 91190, Gif-sur-Yvette, France*

Abstract

In this paper, we consider Constrained Pseudoconvex Nonsmooth Optimization Problems (CPNOPs), which are a class of nonconvex optimization problems. Due to their nonconvexity, classical convex optimization algorithms are unable to solve them, while existing methods, i.e., numerical integration methods, are inadequate in terms of computational performance. In this paper, we propose a novel approach for solving CPNOPs that combines neurodynamic optimization with deep learning. We construct an initial value problem (IVP) involving a system of ordinary differential equations for a CPNOP and use a surrogate model based on a neural network to approximate the IVP. Our approach transforms the CPNOP into a neural network training problem, leveraging the power of deep learning infrastructure to improve computational performance and eliminate the need for traditional optimization solvers. Our experimental results show that our approach is superior to numerical integration methods in terms of both solution quality and computational efficiency.

Keywords: Constrained Pseudoconvex Optimization Problems, Neurodynamic Optimization, Neural Networks, Ordinary Differential Equations

1. Introduction

Constrained nonlinear optimization problems involve finding the best solution among a set of possible solutions by minimizing or maximizing an objective function. These problems are prevalent in various fields such as engineering, physics, finance, and management, with a wide range of applications. They can be divided into two groups based on the nature of the objective or constraint functions: convex and nonconvex optimization problems. Convex optimization problems, which include linear programming and quadratic programming, are a special class of nonconvex optimization problems and have been studied extensively. Methods such as the primal-dual interior point method have been developed to solve them efficiently (Boyd et al., 2004; Nocedal & Wright, 2006). Nonconvex optimization problems, however, are more complex and commonly solved through gradient descent-based algorithms, which often struggle to converge to the global optimal solution (Jain et al., 2017; Jiang et al., 2019).

In this paper, we focus on a specific type of constrained nonconvex optimization problem known as

*Corresponding author

Email address: dawen.wu@centralesupelec.fr, abdel.lisser@12s.centralesupelec.fr (Abdel Lisser)

13 constrained pseudoconvex nonsmooth optimization problems (CPNOPs). A CPNOP has an objective func-
14 tion that is both pseudoconvex, meaning that it is not strictly convex, and nonsmooth, meaning that it is
15 not everywhere differentiable. Due to the pseudoconvexity and non-smoothness of the objective function,
16 traditional convex optimization algorithms are not applicable to solve it.

17 CPNOPs are typically solved by neurodynamic optimization, which involves constructing an initial value
18 problem (IVP) consisting of a system of ordinary differential equations (ODEs) (Liu et al., 2012; Xu et al.,
19 2020; Liu et al., 2022). The final state in the solution of this IVP represents the predicted solution of the
20 CPNOP. These ODE systems are highly nonlinear and have no analytical solutions. Therefore, numerical
21 integration methods, such as the Runge-Kutta method (Dormand & Prince, 1980) or the backward differen-
22 tiation formula (Shampine & Reichelt, 1997), are often used to approximate their solutions numerically.

23 **Motivation.** However, numerical integration methods can be computationally expensive and can pro-
24 duce inaccurate solutions, particularly if the ODE system derived from the CPNOP is stiff. In addition,
25 these methods are not well suited to solving CPNOPs as they require the calculation of all previous states
26 to reach the desired final state, making them inefficient. Therefore, a more efficient and accurate approach
27 is needed to solve CPNOPs.

28 *1.1. Related Works*

29 *1.1.1. Neurodynamic Optimization*

30 Neurodynamic optimization is a class of methods that model constrained optimization problems using
31 ODE systems. This approach was first introduced by Hopfield & Tank (1985) to solve the traveling salesman
32 problem. Since then, neurodynamic optimization has been applied to a wide range of optimization problems,
33 including linear and quadratic programming problems (Xia & Wang, 2000), general convex programming
34 problems (Xia & Feng, 2007), biconvex optimization problems (Che & Wang, 2018), global optimization
35 problems (Che & Wang, 2019), and stochastic optimization problems (Tassouli & Lissner, 2023). These
36 methods typically use the Lyapunov stability theorem to prove that the constructed ODE system has a
37 global convergence property. This means that any state solution of the ODE system converges to an optimal
38 solution of the target problem.

39 In particular, neurodynamic optimization for solving pseudoconvex optimization problems has received
40 widespread attention in recent years. Researchers have applied it to many applications, including portfolio
41 optimization, energy efficiency optimization, and production planning (Liu et al., 2012; Yang et al., 2019).
42 Various neurodynamic methods have been proposed to solve pseudoconvex problems with different types
43 of constraints, such as linear equation constraints (Guo et al., 2011), bound constraints (Liu et al., 2012),
44 convex inequality constraints (Bian et al., 2018), and quasiconvex constraints (Liu et al., 2022).

45 *1.1.2. Deep Learning for Solving Differential Equations*

46 Another line of research included in our work is the use of deep learning to solve differential equations.
47 The idea of using neural networks to approximate the solutions of differential equations was first introduced

48 by Dissanayake & Phan-Thien (1994), where training was performed by minimizing a loss function based on
49 the network’s satisfaction of the boundary conditions and the differential equations. Lagaris et al. (1998)
50 showed that the network architecture could be designed to satisfy the boundary conditions, and this method
51 was extended to systems with irregular boundaries (McFall & Mahan, 2009).

52 With the advancement of deep learning, this approach has received renewed attention with the goal of
53 solving high-dimensional nonlinear partial differential equations (PDEs) (Han et al., 2018; Huang et al.,
54 2022). One notable approach is the use of physics-informed neural networks (PINNs) (Raissi et al., 2019a),
55 which incorporate physical laws and boundary conditions into the network architecture and training process.
56 PINNs have been applied to a variety of engineering problems, such as fluid mechanics (Samaniego et al.,
57 2020; Cai et al., 2022). Variations of PINNs have been developed to address different problem scenarios (Lu
58 et al., 2021b; Zhang et al., 2020; Liao & Zhang, 2022) or to improve computational performance (Jagtap &
59 Karniadakis, 2021; Yu et al., 2022; Sharma & Shankar, 2022). Software packages have been developed to
60 facilitate the application of these methods (Lu et al., 2021a; Chen et al., 2020).

61 *1.2. Contributions*

62 Our main contributions in this work are as follows:

- 63 • We propose a novel approach that combines the advantages of neurodynamic optimization in modeling
64 CPNOPs as ODE systems and the power of deep learning in approximating the solutions to these
65 systems. By transforming the CPNOP into a neural network training problem, our solver eliminates
66 the need for traditional optimization solvers or numerical integration methods.
- 67 • We design a specialized training algorithm that takes advantage of the problem structure of CPNOPs
68 to optimize the performance of our proposed model. The neural network is trained to simultaneously
69 satisfy the ODE system and minimize the CPNOP objective function.
- 70 • In our experimental results, we demonstrate that our proposed approach superiority over numerical
71 integration methods in terms of solution quality and computational efficiency for solving CPNOPs. In
72 addition, we show the performance of the proposed method in solving a large-scale CPNOP which is
73 difficult to be solved by classical numerical integration methods.

74 *1.3. Outline*

75 The remainder of this paper is organized as follows: Section 2 provides essential background knowledge,
76 including an introduction to CPNOPs and neurodynamic optimization. Section 3 presents our proposed
77 neural network model and its application to optimization problems. The training process for the proposed
78 neural network is described in Section 4. In Section 5, experimental results are presented and discussed,
79 comparing our method to numerical integration techniques. Finally, Section 6 offers a summary and outlines
80 future research directions.

81 **2. Preliminaries**

82 *2.1. Constrained Pseudoconvex Nonsmooth Optimization Problem*

83 We consider the following optimization problem:

$$\left\{ \begin{array}{l} \min_x f(x) \\ \text{s.t.} \\ x \in \mathcal{I} = \{x \in \mathbb{R}^n | g_i(x) \leq 0, i = 1, \dots, m\}, \\ x \in \mathcal{S} = \{x \in \mathbb{R}^n | Ax = b\}, \end{array} \right. \quad (1)$$

84 where $x = (x_1, x_2, \dots, x_n)^T \in \mathbb{R}^n$ are the decision variables, $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is the objective function, $g(x) =$
 85 $(g_1(x), g_2(x), \dots, g_m(x))^T : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is the function of the inequality constraints, and $Ax = b$ is the equality
 86 constraints with $A \in \mathbb{R}^{p \times n}$ and $b \in \mathbb{R}^p$. n , m , and p denote the number of decision variables, inequality
 87 constraints, and equality constraints, respectively.

88 In this paper, we consider the case where $f(x)$ is pseudoconvex and nonsmooth, $g(x)$ is convex and
 89 smooth, and A is of full row rank. We denote x^* as the optimal solution of the problem.

90 **Assumption 1.**

- 91 • *There exists a point $x \in \mathbb{R}^n$ such that $x \in \mathcal{S} \cap \text{int}(\mathcal{I})$, where $\text{int}(\cdot)$ denotes the interior of the set.*
- 92 • *The objective function $f(x)$ is Lipschitz continuous and regular on $\mathcal{S} \cap \mathcal{I}$.*

93 *2.2. Neurodynamic Optimization*

94 Now let $y : \mathbb{R} \rightarrow \mathbb{R}^n$ be a time-dependent function. In this context, $y(t)$ represents the state of the system
 95 at time t , and $\frac{dy}{dt} = \Phi(y(t))$ is an ODE system representing the dynamics of the system. A neurodynamic
 96 optimization method aims to construct an ODE system $\frac{dy}{dt} = \Phi(y(t))$ for the control of $y(t)$ such that $y(t)$
 97 settles down to an optimal solution of the considered CPNOP represented in equation (1).

98 **Definition 1** (State solution). *Consider an ODE system $\frac{dy}{dt} = \Phi(y(t))$, where $\Phi : \mathbb{R}^n \rightarrow \mathbb{R}^n$. Given an initial*
 99 *point $y_0 \in \mathbb{R}^n$, a vector value function $y : \mathbb{R} \rightarrow \mathbb{R}^n$ is called a state solution, if it satisfies the ODE system*
 100 *$\frac{dy}{dt} = \Phi(y(t))$ and the initial condition $y(0) = y_0$.*

101 **Definition 2** (Global convergence). *Let $y(t)$ be a state solution of an ODE system $\frac{dy}{dt} = \Phi(y(t))$, and let \mathcal{X}^**
 102 *be the set of optimal solutions of the CPNOP (1).*

The state solution of the ODE system is said to converge globally to \mathcal{X}^ , if $y(t)$ satisfies*

$$\lim_{t \rightarrow \infty} \text{dist}(y(t), \mathcal{X}^*) = 0,$$

103 *where $\text{dist}(y(t), \mathcal{X}^*) = \inf_{x^* \in \mathcal{X}^*} |y(t) - x^*|$, and $\|\cdot\|$ is the Euclidean norm. If the set \mathcal{X}^* contains only one*
 104 *optimal solution x^* , then $\lim_{t \rightarrow \infty} y(t) = x^*$, and the ODE system is globally asymptotically stable at x^* .*

105 In this paper, a modified version of the one-layer neurodynamic approach (Liu et al., 2022) is used to
 106 model the CPNOP. We adapt the original approach to better fit the problem and improve its performance.
 107 The resulting ODE system is given by:

$$\frac{dy}{dt} \in -\theta(t)(I - U) \left(\left\{ \prod_{i=1}^m (1 - \mu(g_i(y(t)))) \right\} \partial f(y(t)) + \partial B(y(t)) \right) - A^T \rho(Ay(t) - b), \quad (2)$$

108 where $I \in \mathbb{R}^{n \times n}$ is the identity matrix, and $U = A^T (AA^T)^{-1} A$. The components of this ODE system are
 109 described as follows:

110 $\theta : \mathbb{R} \rightarrow \mathbb{R}$ is a function that controls the convergence of the state solution $y(t)$ to satisfy the equality
 111 constraints, given by

$$\theta(t) = \begin{cases} 0, & \text{if } t \leq T_0, \\ 1, & \text{otherwise,} \end{cases} \quad (3)$$

112 where $T_0 = 1 + \|Ay_0 - b\|_1 / \lambda_{\min}(AA^T)$, $y_0 \in \mathbb{R}^n$ is an initial point of the ODE system, $\lambda_{\min}(AA^T)$ represents
 113 the smallest eigenvalue of the matrix AA^T .

114 $B : \mathbb{R}^n \rightarrow \mathbb{R}$ is defined as follows:

$$B(y(t)) = \sum_{i=1}^m \max\{0, g_i(y(t))\} \quad (4)$$

115 Since every $g_i(\cdot)$ is convex, $B(\cdot)$ is also convex, which means that $\partial B(\cdot)$ always exists. To compute $\partial B(\cdot)$, we
 116 first define an activation function $\mu : \mathbb{R} \rightarrow \mathbb{R}$ as follows:

$$\mu(s) = \begin{cases} 1, & \text{if } s > 0, \\ 0, & \text{if } s \leq 0. \end{cases} \quad (5)$$

117 Then, the closed-form for $\partial B(y(t))$ can be given as follows:

$$\partial B(y(t)) = \begin{cases} 0, & \text{if } y(t) \in S \cap \text{int}(\mathcal{I}), \\ \sum_{i \in I^0(y(t))} \mu(g_i(y(t))) \nabla g_i(y(t)), & \text{if } y(t) \in S \cap \text{bd}(\mathcal{I}), \\ \sum_{i \in I^0(y(t))} \mu(g_i(y(t))) \nabla g_i(y(t)) + \sum_{i \in I^+(y(t))} \nabla g_i(y(t)), & \text{if } y(t) \in S \setminus \mathcal{I}, \end{cases} \quad (6)$$

118 where $I^0(y(t)) = \{i \in \{1, 2, \dots, k\} | g_i(y(t)) = 0\}$, $I^+(y(t)) = \{i \in \{1, 2, \dots, k\} | g_i(y(t)) > 0\}$, and $\text{bd}(\cdot)$ de-
 119 notes the boundary of the set.

120 $\rho : \mathbb{R}^p \rightarrow \mathbb{R}^p$ is an activation function defined as $\rho(s) = (\tilde{\rho}(s_1), \tilde{\rho}(s_2), \dots, \tilde{\rho}(s_p))^T$, where

$$\tilde{\rho}(s_i) = \begin{cases} 1 & \text{if } s_i > 0, \\ 0 & \text{if } s_i = 0, \\ -1 & \text{if } s_i < 0. \end{cases} \quad (7)$$

121 **Theorem 1** (Liu et al. (2022)). *Let Assumption 1 hold. Given any initial point $y_0 \in \mathbb{R}^n$, the corresponding*
 122 *state solution $y(t)$ of the ODE system (2) globally converges to an optimal solution of the CPNOP (1).*

123 3. Surrogate Model

124 3.1. Initial Value Problem Setup

125 In this work, we model the CPNOP in equation (1) using the ODE system outlined in equation (2). As
 126 stated in Theorem 1, given any initial point y_0 , the state solution $y(t)$ approaches an optimal solution x^* as
 127 time t goes to infinity, i.e., $\lim_{t \rightarrow \infty} y(t) = x^*$.

128 To determine the state solution, we must first specify an IVP by choosing an initial point and a time
 129 range. For simplicity, we set the initial point to be the all-zero vector, i.e., $y_0 = \mathbf{0}$. The time range is defined
 130 as the interval $[0, T]$, where T is a positive number given in advance.

131 We introduce the following definition:

132 **Definition 3** (Final state). *Given a state solution $y(t)$ and a time range $[0, T]$, the state at the end of the*
 133 *time range, i.e., $y(T)$, is called the final state.*

134 The final state $y(T)$ holds significance as it approximates the optimal solution x^* , i.e.,

$$y(T) \approx x^*. \quad (8)$$

135 The goal of this study is to effectively and accurately determine the final state of a given time range.

136 3.2. Neural Network Based Surrogate Model

137 We propose a neural network based surrogate model to approximate the solution of the constructed IVP.
 138 The structure of the model is shown in Figure 1 (left). The equation below describes the surrogate model:

$$\hat{y}(t; w) = (1 - e^{-t}) N(t; w), \quad t \in [0, T], \quad (9)$$

139 where $N(t; w)$ is a fully connected neural network with trainable parameters w , and $(1 - e^{-t})$ is an auxiliary
 140 function that ensures that the initial condition $\hat{y}(t = 0; w) = \mathbf{0}$ is satisfied.

141 The auxiliary function $(1 - e^{-t})$ is a modified version of the Lagaris method Lagaris et al. (1998), where
 142 the exponential form has been shown to improve convergence in previous studies Mattheakis et al. (2020).

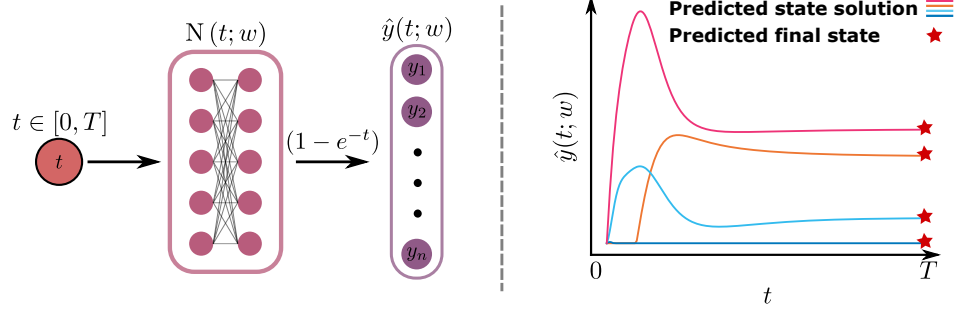


Figure 1: **Left:** The structure of the surrogate model. **Right:** The surrogate model as an approximate state solution on the time range $[0, T]$.

143 In our experiments, we also observed similar benefits from using this function, likely because it reduces the
 144 influence of the neural network further from the initial time.

145 **Approximate State Solution.** The surrogate model is designed to approximate the state solution of
 146 the IVP constructed in Section 3.1. This is represented by the following relationship:

$$\hat{y}(t; w) \approx y(t), \quad t \in [0, T]. \quad (10)$$

147 **Predicted Solution to the CPNOP.** In particular, we denote the final state of the surrogate model
 148 as:

$$\hat{x}(w) = \hat{y}(t = T; w), \quad (11)$$

149 which serves as a prediction of the optimal solution of CPNOP. By combining equation (8) and equation (10)
 150 at $t = T$, we have

$$\hat{x}(w) \approx y(T) \approx x^*. \quad (12)$$

151 Figure 1 (right) shows our proposed surrogate model as an approximate state solution, thus providing a
 152 prediction for the CPNOP. The CPNOP being considered in this figure has four variables. The colored lines
 153 represent the surrogate model, a 4-dimensional vector-valued function that approximates the state solution
 154 of the IVP. The red stars represent the final state, which is a 4-dimensional real vector that serves as the
 155 predicted optimal solution to the CPNOP.

156 4. Model Training

157 4.1. Loss Function

158 We define the loss function for the proposed surrogate model as follows:

$$\mathcal{L}(t, w) = \left\| \frac{\partial \hat{y}(t; w)}{\partial t} - \Phi(\hat{y}(t; w)) \right\|, \quad (13)$$

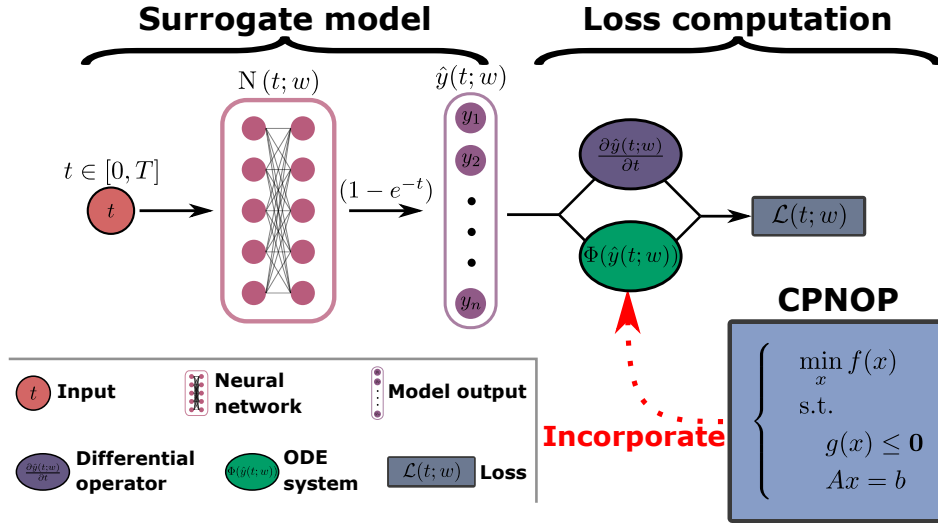


Figure 2: Incorporating the CPNOP into the loss function

159 where $\Phi(\cdot)$ refers to the ODE system outlined in equation (2), which corresponds to a CPNOP to be solved.
 160 $\frac{\partial \hat{y}(t; w)}{\partial t}$ is the derivative of the model output $\hat{y}(t; w)$ with respect to the input time t . This derivative can
 161 be calculated analytically using automatic differentiation tools such as PyTorch or JAX (Paszke et al., 2019;
 162 Bradbury et al., 2018).

163 In Figure 2, we show how the CPNOP is integrated into the loss computation, rather than being a
 164 component of the surrogate model structure. The surrogate model itself is an empty framework with no
 165 goal of solving a particular CPNOP. By reformulating CPNOP as an ODE system and embedding it in
 166 the loss function, we train the neural network to let the surrogate model satisfy the ODE system. This
 167 approach is similar to that used in PINN (Raissi et al., 2019b), where physical rules are encoded as PDEs
 168 and incorporated into the loss function for training.

169 The goal of training the surrogate model is to minimize the following objective function:

$$E(w) = \int_0^T \mathcal{L}(t, w) dt, \quad (14)$$

170 which is an integral of the loss function over the pre-given time range $[0, T]$. The loss value $\mathcal{L}(t, w)$ represents
 171 the error of the model at time t , and the objective function $E(w)$ represents the total error of the model over
 172 the time range $[0, T]$.

173 However, it is computationally intractable to compute $E(w)$ due to the integral part. Therefore, in
 174 practice, we train the model by minimizing the following batch loss:

$$\mathcal{L}(\mathbb{T}, w) = \frac{1}{|\mathbb{T}|} \sum_{t \in \mathbb{T}} \mathcal{L}(t, w), \quad (15)$$

175 where \mathbb{T} is a set of collocation time points, and $|\mathbb{T}|$ is the size of the set. In this study, we specifically adopt
 176 a uniform distribution over the interval $[0, T]$ for sampling collocation time points. While there is existing

Algorithm 1 A deep learning-based solver for CPNOP

Input: A CPNOP as defined in (1); A time range $[0, T]$ **Output:** Predicted optimal solution to the CPNOP

```
1: function CPNOP SOLVER:
2:   Derive the ODE system,  $\Phi(\cdot)$ , according to the CPNOP.           ▷ By equation (2).
3:   Instantiate a surrogate model  $\hat{y}(t; w)$ .                           ▷ By equation (9).
4:   Set  $\epsilon_{\text{best}} \leftarrow \epsilon(\hat{x}(w))$                                ▷ Initialize  $\epsilon_{\text{best}}$ .
5:   while iteration  $\leq$  maximum iteration do
6:     Sample a batch of times  $\mathbb{T} \sim U(0, T)$  uniformly from the interval  $[0, T]$ .   ▷ Data generation.
7:     Compute the batch loss  $\mathcal{L}(\mathbb{T}, w)$ .                               ▷ Forward propagation.
8:     Update  $w$  by  $\nabla_w \mathcal{L}(\mathbb{T}, w)$ .                               ▷ Backward propagation.
9:     Project  $\hat{x}(w)$  onto the feasible set  $\mathcal{I}$  by  $\hat{x}(w) \leftarrow P_{eq}(\hat{x}(w))$ .   ▷ To satisfy equality constraints.
10:    Calculate the epsilon value  $\epsilon_{\text{temp}} \leftarrow \epsilon(\hat{x}(w))$ .
11:    if  $\epsilon_{\text{temp}} < \epsilon_{\text{best}}$  then
12:       $\epsilon_{\text{best}} \leftarrow \epsilon_{\text{temp}}$ .                                       ▷ Update  $\epsilon_{\text{best}}$ .
13:       $\hat{x}_{\text{best}} \leftarrow \hat{x}(w)$ .                                       ▷ Update  $\hat{x}_{\text{best}}$ .
14:    end if
15:  end while
16:  return  $\hat{x}_{\text{best}}$ 
17: end function
```

177 research in PINNs exploring more sophisticated collocation point sampling methods to enhance computational
178 efficiency (e.g., Tang et al. (2023); Nabian et al. (2021); Wu et al. (2023); Fang et al. (2023)), our practical
179 experience indicates that the straightforward approach of uniform distribution yields comparable results.

180 4.2. Evaluation Metric

181 We propose a metric to evaluate the performance of the predicted optimal solution $\hat{x}(w)$. This metric,
182 known as the epsilon value, is defined as follows:

$$\epsilon(\hat{x}(w)) = \begin{cases} f(\hat{x}(w)), & \text{if } \hat{x}(w) \in \mathcal{S} \cap \mathcal{I}, \\ +\infty, & \text{otherwise,} \end{cases} \quad (16)$$

183 where $f(\cdot)$, \mathcal{S} , and \mathcal{I} are given in equation (1). If $\hat{x}(w)$ is within the feasible set, the epsilon value is set to the
184 objective value corresponding to $\hat{x}(w)$, i.e., $f(\hat{x}(w))$. Otherwise, the epsilon value is set to $+\infty$ to indicate
185 that the prediction is not within the feasible set.

186 Since it is difficult to strictly satisfy the equality constraints, we use the following technique to project
187 the prediction into \mathcal{S} :

$$P_{eq}(\hat{x}(w)) = \hat{x}(w) - A^T (AA^T)^{-1} (A\hat{x}(w) - b). \quad (17)$$

188 4.3. Pipeline

189 Algorithm 1 outlines the steps for solving a CPNOP using our proposed surrogate model approach. The
190 algorithm starts by constructing an IVP from the given CPNOP and a pre-specified time range of $[0, T]$. A
191 neural network is then initialized as a surrogate model to serve as an approximate state solution for this IVP.

192 The parameters of the network are then optimized by performing gradient descent on the designed batch loss
 193 function, as defined in equation (15), in order to improve the accuracy of the approximation.

194 A key aspect of the proposed algorithm is the use of the evaluation metric, as defined in equation (16),
 195 to assess the performance of the model at each training iteration. The variables ϵ_{best} and ϵ_{temp} represent
 196 the lowest and current epsilon values achieved by the model, respectively. After each update of the network
 197 parameters, the algorithm compares ϵ_{temp} with ϵ_{best} . Suppose ϵ_{temp} is less than ϵ_{best} , indicating that the
 198 model found a better prediction. In this case, the algorithm updates $\epsilon_{\text{best}} = \epsilon_{\text{temp}}$ and saves the current
 199 best prediction $\hat{x}(w)$. This design is similar to traditional optimization algorithms that check for optimality
 200 conditions (e.g., KKT conditions) at each iteration.

201 5. Experiments

202 Our proposed approach was evaluated on the Google Colab Pro+ platform, using PyTorch 1.12.1 with
 203 CUDA 11.2 for the neural network and JAX 0.4.1 (Bradbury et al., 2018) for modeling the ODE system.

204 Section 5.1 presents a standard CPNOP, where we demonstrate the complete process of solving the
 205 problem using the proposed method and provide a detailed comparison between our approach and numerical
 206 integration methods. In Section 5.2, we showcase a large-scale CPNOP, which cannot be addressed by
 207 classical numerical integration techniques, and illustrate how our proposed method effectively tackles this
 208 challenge. Finally, Section 5.3 discusses the advantages and limitations of the proposed approach.

209 5.1. Comparing CPNOP Solutions: Our Approach vs. Numerical Integration

210 **Example 1:** Consider the following specific instance of a CPNOP:

$$\begin{aligned}
 \min_x f(x) &= \frac{x_1 + x_2 + e^{|x_2-1|} - 40}{(x_1 + x_2 + x_3)^2 + 3} \\
 \text{s.t.} & \\
 g_1(x) &= -3x_1 + 2x_2 - 5 \leq 0 \\
 g_2(x) &= x_1^2 + x_2 - 3 \leq 0 \\
 h(x) &= x_1 + 2x_2 + x_3 - 2 = 0
 \end{aligned} \tag{18}$$

211 According to Theorem 7 in Liu et al. (2012), the objective function in this instance is pseudoconvex. It is
 212 also non-smooth due to the presence of the absolute value function $|x_2 - 1|$.

213 **IVP Construction.** We modeled this specific instance using the ODE system outlined in equation (2).
 214 We set the initial point to $y_0 = \mathbf{0}$ and the time range to the interval $[0, 10]$ to construct an IVP for this
 215 ODE system. The true state solution of this IVP is denoted as $y(t)$, which is unknown and has no analytical
 216 solution. Our experiments showed that this particular instance of CPNOP results in a stiff ODE system,
 217 which leads to poor computational efficiency and accuracy when using traditional numerical integration
 218 methods.

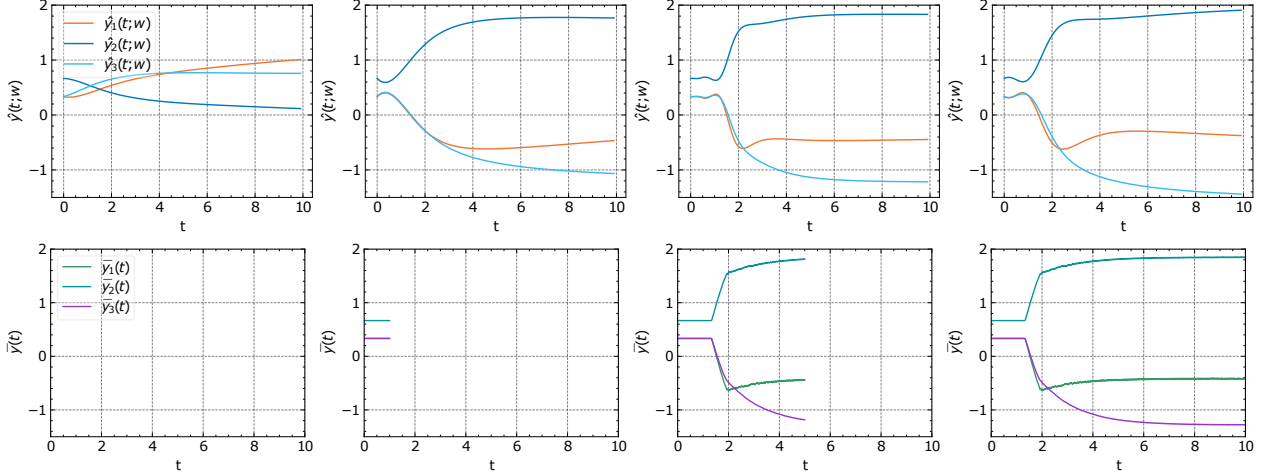


Figure 3: Comparison of solution processes between our approach and RK45. The top panel shows the evolution of $\hat{y}(t; w) = (\hat{y}_1(t; w), \hat{y}_2(t; w), \hat{y}_3(t; w))^T$ obtained by our approach at 0, 100, 1000, and 10000 TIs (from left to right). The bottom panel shows the evolution of $\bar{y}(t) = (\bar{y}_1(t), \bar{y}_2(t), \bar{y}_3(t))^T$ obtained by RK45 at 0, 1000, 5000, and 10000 CPs (from left to right).

219 **Experimental Setup of Our Approach.** To solve the instance presented in equation (18), we instan-
 220 tiated a surrogate model as follows: $\hat{y}(t; w) = (1 - e^{-t})N(t; w)$, where $t \in [0, 10]$, $N(t; w)$ is a fully-connected
 221 neural network with one hidden layer of 100 neurons and the Tanh activation function. We then trained
 222 this surrogate model using Algorithm 1, with a maximum of 10,000 iterations, the Adam optimizer with a
 223 learning rate of 0.001, and a batch size of 128.

224 **Experimental Setup of RK45.** We performed a comparison of our proposed approach with the
 225 Runge-Kutta (RK) method, which is a commonly used technique among numerical integration methods. In
 226 particular, we chose the RK45 solver, which can be accessed via the Scipy library (Virtanen et al., 2020), as
 227 a representative implementation of the RK method. We opted to present only a comparison with RK45 in
 228 Figure 3 and Table 1, as experimental observations indicate that the approximate state solutions obtained
 229 from different numerical integration methods do not show significant differences. We set the number of
 230 collocation points to 10,000, which were uniformly distributed over the time range of $[0, 10]$. The approximate
 231 state solution obtained by RK45 is denoted as $\bar{y}(t)$. Both $\hat{y}(t; w)$ and $\bar{y}(t)$ aim to be as close as possible to
 232 the true state solution $y(t)$ over the interval $[0, 10]$.

233 **Solution Process of Our Approach.** In the top panel of Figure 3, we show the evolution of $\hat{y}(t; w)$ as
 234 it approximates $y(t)$. Our proposed approach achieves this by training the neural network with the parameters
 235 w . As the network parameters w are updated at each training iteration (TI), a new $\hat{y}(t; w)$ is generated. The
 236 upper left corner of Figure 3 shows the initial approximation, while the upper right corner of the figure shows
 237 the final approximation after 10,000 training iterations, which is a more accurate approximation for the IVP
 238 and closer to $y(t)$.

239 **Solution Process of RK45.** In the bottom panel of Figure 3, we show the evolution of $\bar{y}(t)$ as it
 240 approaches $y(t)$. RK45 achieves this by incrementally advancing the collocation point (CP) from the initial
 241 time $t = 0$ to the final time $t = 10$. The bottom panel of the figure illustrates how the approximation of $\bar{y}(t)$

improves as the number of CPs increases, with the plot of $\bar{y}(t)$ at 0, 1000, 5000, and 10,000 CPs corresponding to the solved time ranges [0, 0], [0, 1], [0, 5], and [0, 10], respectively.

Our approach			RK45		
TI	Solution	epsilon	CP	Solution	epsilon
0	[1.014 0.114 0.757]	-5.562	0	[0.333 0.667 0.333]	-7.872
10	[1.014 0.114 0.757]	-5.562	10	[0.333 0.667 0.333]	-7.872
100	[-0.462 1.764 -1.067]	-11.963	100	[0.333 0.667 0.333]	-7.872
1000	[-0.462 1.764 -1.067]	-11.963	1000	[0.333 0.667 0.333]	-7.872
3000	[-0.377 1.909 -1.441]	-11.962	3000	[-0.505 1.69 -0.876]	-11.895
5000	[-0.377 1.909 -1.441]	-11.962	5000	[-0.446 1.816 -1.185]	-11.986
7000	[-0.446 1.831 -1.216]	-11.992	7000	[-0.42 1.838 -1.257]	-11.987
9999	[-0.446 1.831 -1.216]	-11.992	9999	[-0.415 1.846 -1.276]	-11.984

Table 1: Evaluation of solution quality between our approach and RK45. The solution represents the predicted optimal solution for the CPNOP instance given in equation (18). The epsilon value serves as the evaluation metric, as defined in equation (16).

Comparison of Solution Quality. In Table 1, we present a comparison of the predicted optimal solutions for the CPNOP instance obtained using our approach and RK45, along with their corresponding epsilon values, where a lower epsilon indicates a better solution. The solution of our approach is represented by the final state of the surrogate model, i.e., $\hat{y}(t = 10, w)$, where each TI corresponds to a different value of $\hat{y}(t = 10, w)$. The solution of RK45 is given by $\bar{y}(t = 10 * (CP/10000))$, which is the final state of a solved time range. For example, when $CP = 5000$, the corresponding solved time range is [0, 5], and the solution of RK45 at this CP is the final state of this time range, i.e., $\bar{y}(t = 5)$.

The results in Table 1 suggest the following:

- Our approach converges faster than RK45. In other words, our approach is able to find a satisfactory solution at an early stage of the solution process. For example, at the 100th TI, our solution has an epsilon value of -11.963, while the solution of RK45 only reaches an epsilon value of -7.872 at the 100th CP.
- Our approach yields a more accurate final solution than RK45. At the end of the solution process, i.e., at TI=10000 for our approach or CP=10000 for RK45, our solution reaches an epsilon value of -11.992, which is superior to that of RK45.

Comparison of Computational Efficiency. In Table 2, we evaluate the computational efficiency of our approach in comparison to a selection of popular numerical integration methods, such as RK45, RK23, DOP853, Radau, BDF, and LSODA, which are available through the Scipy library. The results are presented in terms of the CPU time required to solve the CPNOP, with lower values indicating superior efficiency.

The results in Table 2 indicate that our approach is significantly more efficient than the numerical integration methods. Specifically, our approach took only 80 seconds to complete 10,000 TIs, while the fastest numerical integration method (RK45) took more than 7 minutes to complete 10,000 collocation points. Some numerical integration methods, such as LSODA, and BDF, even failed to complete the task. Combined with

Our approach		Numerical integration methods						
TI	Runtime	CP	RK45 Runtime	RK23 Runtime	DOP853 Runtime	Radau Runtime	BDF Runtime	LSODA Runtime
10	202ms	10	811ms	433ms	1.77s	1.06s	Fail	458ms
100	893ms	100	980ms	1.23s	2.3s	1.33s	Fail	1.47s
1000	8.47s	1000	11s	13.2s	3min 25s	>2h	Fail	20min 20s
5000	40s	5000	3min 31s	5min 5s	29min 43s	>2h	Fail	Fail
10000	80s	10000	7min 45s	12min 15s	49min 55s	>2h	Fail	Fail

Table 2: Evaluation of computational efficiency between our approach and various numerical integration methods. ms, s and h denote milliseconds, seconds and hours, respectively.

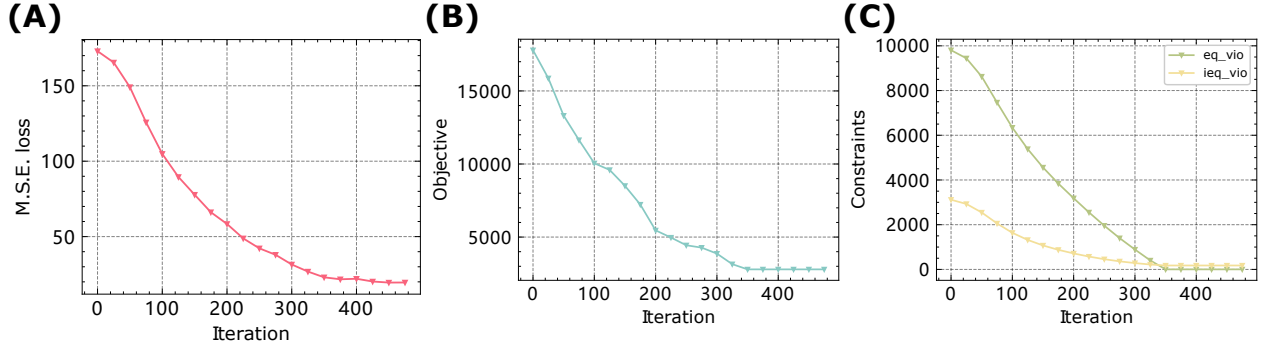


Figure 4: Performance of the proposed neural network method for Example 2: (A) Loss value, (B) Objective value, and (C) Constraint violation metrics. eq_vio and ieq_vio are defined in (20) and (21), respectively.

267 the quality of the solutions reported in Table 1, it is clear that our approach is able to produce faster and
268 better solutions for the CPNOP instance in equation (18) than the numerical integration methods.

269 5.2. Large-Scale CPNOP

270 **Example 2:** Consider the following large-scale CPNOP:

$$\begin{aligned}
 \min_x f(x) &= \sum_{i=1}^{999} \left(x_i^2 - \frac{x_i^2}{x_{i+1}} \right) \\
 \text{s.t.} & \\
 g_i(x) &= x_{10(i-1)+1}^2 + x_{10(i-1)+2}^2 + \dots + x_{10(i-1)+10}^2 - 20 \leq 0, \\
 & i = 1, 2, \dots, 100. \\
 h(x) &= Ax - b = 0
 \end{aligned} \tag{19}$$

271 This large-scale CPNOP is a modification from (Bian et al., 2018), which has 1000 variables, 100 inequality
272 nonlinear constraints, and 1 equality constraint. The objective function $f(x)$ is pseudoconvex, and the
273 inequality constraints are nonlinear and convex. In the equality constraint, $b = 16$, and the vector A can be
274 accessed from the link ¹.

¹https://drive.google.com/file/d/1duUyhCtW0qViVfX0VwBQcoiv9QTWvBe0/view?usp=share_link

275 **IVP Construction.** We construct an IVP by specifying the time range $[0, 10]$ and an initial point y_0 ,
 276 which can be accessed from the link ². We use a fully connected neural network with four hidden layers, each
 277 containing 150 neurons, to solve this problem. Other training settings are consistent with Section 5.1.

278 **Difficulty in Solving with Numerical Integration.** The large-scale nature of Example 2 leads to
 279 an IVP with 1000 state variables, making it difficult to solve using classical numerical integration methods
 280 due to low efficiency. We attempted to solve the problem using methods such as RK45, but they took over
 281 an hour without providing effective predictions.

282 **Constraint Violation Metrics.** We define two metrics to describe the degree to which the predictions
 283 satisfy the equality and inequality constraints:

$$eq_vio(x_{\text{pred}}) = \|Ax_{\text{pred}} - b\|, \quad (20)$$

284

$$ieq_vio(x_{\text{pred}}) = \|g(x_{\text{pred}})^+\|, \quad (21)$$

285 where $g(x_{\text{pred}})^+ = \min\{\mathbf{0}, g(x_{\text{pred}})\}$.

286 **Effectiveness of the Proposed Method in Solving Large-Scale CPNOP.** Figure 4 shows the
 287 performance of our neural network method in solving Example 2. We can draw the following conclusions
 288 from the experimental results:

- 289 • As shown in Figure 4-(A), the loss function decreased from 175 to 20, indicating that the proposed
 290 model effectively solved the constructed IVP, even for large-scale optimization problems. The model
 291 provided an acceptable predicted state solution.
- 292 • The proposed method progressively approaches the optimal solution. As shown in Figure 4-(B), the
 293 objective value decreased from 17,000 to 2,500. Although the obtained predicted value is not the optimal
 294 solution under the current experimental settings, it is already very close and provides a meaningful upper
 295 bound for the original problem.
- 296 • The proposed method effectively finds feasible solutions to the problem. As shown in Figure 4-(C), after
 297 only 500 iterations, the equality constraint violation metric eq_vio decreased from an initial 18,000 to 12,
 298 and the inequality constraint violation metric ieq_vio decreased from an initial 15,000 to 0. Furthermore,
 299 the projection transformation (17) can adjust the predicted values to satisfy the equality constraints,
 300 reducing the eq_vio metric to zero.

301 5.3. Discussion

302 **Advantages.** In comparison to numerical integration methods for solving CPNOPs, our approach
 303 presents three distinct computational benefits:

²https://drive.google.com/file/d/1V098TrlLgPH-WHr1gkX7njVLy-J9FBc2/view?usp=share_link

- 304 • Our method predicts the final state at each TI, whereas numerical integration techniques only offer a
305 prediction upon completion of the entire solution process. This results in faster convergence and more
306 efficient computation, as demonstrated in Tables 1 and 2.
- 307 • By transforming the CPNOP into a neural network training problem, our approach allows us to capi-
308 talize on deep learning infrastructure and methodologies, enhancing computational performance.
- 309 • As evidenced in Section 5.2, our method is capable of solving large-scale CPNOPs, while classical
310 numerical integration techniques demand significantly more computational time and still fail to deliver
311 a feasible predictions

312 **Limitations.** While our approach has demonstrated some advantages, it also has some limitations that
313 must be acknowledged. One limitation is that it requires more complicated tuning of the hyperparameters to
314 achieve optimal performance, unlike numerical integration methods, which are generally more straightforward
315 and require minimal tuning. In addition, when dealing with a complex instance of CPNOP, such as those
316 with many variables and constraints, it may be necessary to use a more advanced neural network architecture
317 and allocate more resources to training in order to achieve a high-quality solution.

318 **Large-Scale CPNOP.** We believe that the proposed neural network approach may open new possi-
319 bilities for solving large-scale CPNOPs. Traditional solvers, when applied to large-scale problems, often face
320 inefficiencies due to memory limitations or iterative solving processes. In contrast, the method proposed here
321 potentially circumvents these issues by transforming the problem into training a neural network. Training
322 large-scale neural networks is a relatively common and well-researched task in the field of deep learning.
323 However, one challenge with the neural network method is the lack of guaranteed convergence. This is due
324 to the fact that neural network training generally involves non-convex optimisation, where continuous loss
325 reduction cannot be guaranteed. Nevertheless, we can use the neural network prediction as a starting point
326 to warm start a traditional solver, thereby obtaining the convergent solution.

327 6. Conclusion

328 In this paper, we presented a novel approach for solving CPNOPs that combines neurodynamic optimiza-
329 tion with deep learning techniques, specifically using deep learning-based differential equation solvers such
330 as PINN. Our results demonstrated that this approach leads to faster convergence and superior solutions
331 compared to traditional numerical integration methods. Our work establishes a link between CPNOPs and
332 deep learning, opening up new possibilities in the field of nonlinear constrained programming.

333 However, it is important to note that the approach is still in its early stages and has limitations, such
334 as potential reliability and robustness issues. In future work, we plan to improve the performance of our
335 approach by exploring advanced deep learning-based PDE solvers and alternative neurodynamic optimization
336 approaches, and to apply the method to other types of nonlinear programming problems.

337 Bibliography

- 338 Bian, W., Ma, L., Qin, S., & Xue, X. (2018). Neural network for nonsmooth pseudoconvex optimization with
339 general convex constraints. *Neural Networks*, *101*, 1–14.
- 340 Boyd, S., Boyd, S. P., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.
- 341 Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke,
342 A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2018). JAX: composable transformations of
343 Python+NumPy programs. URL: <http://github.com/google/jax>.
- 344 Cai, S., Mao, Z., Wang, Z., Yin, M., & Karniadakis, G. E. (2022). Physics-informed neural networks (pinns)
345 for fluid mechanics: A review. *Acta Mechanica Sinica*, (pp. 1–12).
- 346 Che, H., & Wang, J. (2018). A two-timescale duplex neurodynamic approach to biconvex optimization. *IEEE*
347 *Transactions on Neural Networks and Learning Systems*, *30*, 2503–2514.
- 348 Che, H., & Wang, J. (2019). A collaborative neurodynamic approach to global and combinatorial optimiza-
349 tion. *Neural Networks*, *114*, 15–27.
- 350 Chen, F., Sondak, D., Protopapas, P., Mattheakis, M., Liu, S., Agarwal, D., & Giovanni, M. D. (2020).
351 Neurodifreq: A python package for solving differential equations with neural networks. *Journal of Open*
352 *Source Software*, *5*, 1931. URL: <https://doi.org/10.21105/joss.01931>. doi:10.21105/joss.01931.
- 353 Dissanayake, M. W. M. G., & Phan-Thien, N. (1994). Neural-network-based ap-
354 proximations for solving partial differential equations. *Communications in Numer-*
355 *ical Methods in Engineering*, *10*, 195–201. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/cnm.1640100303>.
356 doi:<https://doi.org/10.1002/cnm.1640100303>.
357 arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1002/cnm.1640100303>.
- 358 Dormand, J. R., & Prince, P. J. (1980). A family of embedded runge-kutta formulae. *Journal of computational*
359 *and applied mathematics*, *6*, 19–26.
- 360 Fang, Q., Mou, X., & Li, S. (2023). A physics-informed neural network based on mixed data sampling for
361 solving modified diffusion equations. *Scientific Reports*, *13*, 2491.
- 362 Guo, Z., Liu, Q., & Wang, J. (2011). A one-layer recurrent neural network for pseudoconvex optimization
363 subject to linear equality constraints. *IEEE Transactions on Neural Networks*, *22*, 1892–1900.
- 364 Han, J., Jentzen, A., & E, W. (2018). Solving high-dimensional partial differential equations using deep
365 learning. *Proceedings of the National Academy of Sciences*, *115*, 8505–8510.
- 366 Hopfield, J. J., & Tank, D. W. (1985). “neural” computation of decisions in optimization problems. *Biological*
367 *cybernetics*, *52*, 141–152.

- 368 Huang, S., Feng, W., Tang, C., & Lv, J. (2022). Partial differential equations meet deep neural networks: A
369 survey. *arXiv preprint arXiv:2211.05567*, .
- 370 Jagtap, A. D., & Karniadakis, G. E. (2021). Extended physics-informed neural networks (xpinns): A gen-
371 eralized space-time domain decomposition based deep learning framework for nonlinear partial differential
372 equations. In *AAAI Spring Symposium: MLPS*.
- 373 Jain, P., Kar, P. et al. (2017). Non-convex optimization for machine learning. *Foundations and Trends® in*
374 *Machine Learning*, 10, 142–363.
- 375 Jiang, B., Lin, T., Ma, S., & Zhang, S. (2019). Structured nonconvex and nonsmooth optimization: algorithms
376 and iteration complexity analysis. *Computational Optimization and Applications*, 72, 115–157.
- 377 Lagaris, I., Likas, A., & Fotiadis, D. (1998). Artificial neural networks for solving ordinary and partial
378 differential equations. *IEEE Transactions on Neural Networks*, 9, 987–1000. doi:10.1109/72.712178.
- 379 Liao, G., & Zhang, L. (2022). Solving flows of dynamical systems by deep neural networks and a novel deep
380 learning algorithm. *Mathematics and Computers in Simulation*, 202, 331–342.
- 381 Liu, N., Wang, J., & Qin, S. (2022). A one-layer recurrent neural network for nonsmooth pseudoconvex
382 optimization with quasiconvex inequality and affine equality constraints. *Neural Networks*, 147, 1–9.
- 383 Liu, Q., Guo, Z., & Wang, J. (2012). A one-layer recurrent neural network for constrained pseudoconvex
384 optimization and its application for dynamic portfolio optimization. *Neural Networks*, 26, 99–109.
- 385 Lu, L., Meng, X., Mao, Z., & Karniadakis, G. E. (2021a). Deepxde: A deep learning library for solv-
386 ing differential equations. *SIAM Review*, 63, 208–228. URL: <http://dx.doi.org/10.1137/19M1274067>.
387 doi:10.1137/19m1274067.
- 388 Lu, L., Pestourie, R., Yao, W., Wang, Z., Verdugo, F., & Johnson, S. G. (2021b). Physics-
389 informed neural networks with hard constraints for inverse design. *SIAM Journal on Scientific*
390 *Computing*, 43, B1105–B1132. URL: <https://doi.org/10.1137/21M1397908>. doi:10.1137/21M1397908.
391 arXiv:<https://doi.org/10.1137/21M1397908>.
- 392 Mattheakis, M., Sondak, D., Dogra, A. S., & Protopapas, P. (2020). Hamiltonian neural networks for solving
393 equations of motion. *arXiv preprint arXiv:2001.11107*, .
- 394 McFall, K. S., & Mahan, J. R. (2009). Artificial neural network method for solution of boundary value
395 problems with exact satisfaction of arbitrary boundary conditions. *IEEE Transactions on Neural Networks*,
396 20, 1221–1233.
- 397 Nabian, M. A., Gladstone, R. J., & Meidani, H. (2021). Efficient training of physics-informed neural networks
398 via importance sampling. *Computer-Aided Civil and Infrastructure Engineering*, 36, 962–977.

- 399 Nocedal, J., & Wright, S. J. (2006). Numerical optimization. In *Springer Series in Operations Research and*
400 *Financial Engineering* (pp. 1–664). Springer New York. doi:10.1201/b19115-11.
- 401 Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein,
402 N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chil-
403 amkurthy, S., Steiner, B., Fang, L., Bai, J., & Chintala, S. (2019). PyTorch: An impera-
404 tive style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer,
405 F. d Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in Neural Information Processing Sys-*
406 *tems*. Curran Associates, Inc. volume 32. URL: [https://proceedings.neurips.cc/paper/2019/file/](https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf)
407 [bdbca288fee7f92f2bfa9f7012727740-Paper.pdf](https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf). arXiv:1912.01703.
- 408 Raissi, M., Perdikaris, P., & Karniadakis, G. (2019a). Physics-informed neural networks: A deep learning
409 framework for solving forward and inverse problems involving nonlinear partial differential equations. *Jour-*
410 *nal of Computational Physics*, 378, 686–707. URL: [https://www.sciencedirect.com/science/article/](https://www.sciencedirect.com/science/article/pii/S0021999118307125)
411 [pii/S0021999118307125](https://www.sciencedirect.com/science/article/pii/S0021999118307125). doi:<https://doi.org/10.1016/j.jcp.2018.10.045>.
- 412 Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019b). Physics-informed neural networks: A deep learn-
413 ing framework for solving forward and inverse problems involving nonlinear partial differential equations.
414 *Journal of Computational Physics*, 378, 686–707.
- 415 Samaniego, E., Anitescu, C., Goswami, S., Nguyen-Thanh, V. M., Guo, H., Hamdia, K., Zhuang, X., &
416 Rabczuk, T. (2020). An energy approach to the solution of partial differential equations in computational
417 mechanics via machine learning: Concepts, implementation and applications. *Computer Methods in Applied*
418 *Mechanics and Engineering*, 362, 112790.
- 419 Shampine, L. F., & Reichelt, M. W. (1997). The matlab ode suite. *SIAM journal on scientific computing*,
420 18, 1–22.
- 421 Sharma, R., & Shankar, V. (2022). Accelerated training of physics informed neural networks (pinns) using
422 meshless discretizations. *arXiv preprint arXiv:2205.09332*, .
- 423 Tang, K., Wan, X., & Yang, C. (2023). Das-pinns: A deep adaptive sampling method for solving high-
424 dimensional partial differential equations. *Journal of Computational Physics*, 476, 111868.
- 425 Tassouli, S., & Lisser, A. (2023). A neural network approach to solve geometric programs with joint proba-
426 bilistic constraints. *Mathematics and Computers in Simulation*, 205, 765–777.
- 427 Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E.,
428 Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J.,
429 Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore,
430 E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris,
431 C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., & SciPy 1.0 Contributors (2020).

- 432 SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17, 261–272.
433 doi:10.1038/s41592-019-0686-2.
- 434 Wu, C., Zhu, M., Tan, Q., Kartha, Y., & Lu, L. (2023). A comprehensive study of non-adaptive and residual-
435 based adaptive sampling for physics-informed neural networks. *Computer Methods in Applied Mechanics*
436 *and Engineering*, 403, 115671.
- 437 Xia, Y., & Feng, G. (2007). A new neural network for solving nonlinear projection equations. *Neural*
438 *Networks*, 20, 577–589.
- 439 Xia, Y., & Wang, J. (2000). A recurrent neural network for solving linear projection equations. *Neural*
440 *Networks*, 13, 337–350.
- 441 Xu, C., Chai, Y., Qin, S., Wang, Z., & Feng, J. (2020). A neurodynamic approach to nonsmooth constrained
442 pseudoconvex optimization problem. *Neural Networks*, 124, 180–192.
- 443 Yang, Y., Pesavento, M., Chatzinotas, S., & Ottersten, B. (2019). Energy efficiency optimization in mimo
444 interference channels: A successive pseudoconvex approximation approach. *IEEE Transactions on Signal*
445 *Processing*, 67, 4107–4121.
- 446 Yu, J., Lu, L., Meng, X., & Karniadakis, G. E. (2022). Gradient-enhanced physics-informed neural networks
447 for forward and inverse pde problems. *Computer Methods in Applied Mechanics and Engineering*, 393,
448 114823.
- 449 Zhang, D., Guo, L., & Karniadakis, G. E. (2020). Learning in modal space: Solving time-
450 dependent stochastic pdes using physics-informed neural networks. *SIAM Journal on Scientific*
451 *Computing*, 42, A639–A665. URL: <https://doi.org/10.1137/19M1260141>. doi:10.1137/19M1260141.
452 arXiv:<https://doi.org/10.1137/19M1260141>.