A Method for Computing Inverse Parametric PDE Problems with Random-Weight Neural Networks

Suchuan Dong[*] Yiran Wang
Center for Computational and Applied Mathematics
Department of Mathematics
Purdue University, USA

(April 16, 2023)

Abstract

We present a method for computing the inverse parameters and the solution field to inverse parametric partial differential equations (PDE) based on randomized neural networks. This extends the local extreme learning machine technique originally developed for forward PDEs to inverse problems. We develop three algorithms for training the neural network to solve the inverse PDE problem. The first algorithm (termed NLLSQ) determines the inverse parameters and the trainable network parameters all together by the nonlinear least squares method with perturbations (NLLSQ-perturb). The second algorithm (termed VarPro-F1) eliminates the inverse parameters from the overall problem by variable projection to attain a reduced problem about the trainable network parameters only. It solves the reduced problem first by the NLLSQ-perturb algorithm for the trainable network parameters, and then computes the inverse parameters by the linear least squares method. The third algorithm (termed VarPro-F2) eliminates the trainable network parameters from the overall problem by variable projection to attain a reduced problem about the inverse parameters only. It solves the reduced problem for the inverse parameters first, and then computes the trainable network parameters afterwards. VarPro-F1 and VarPro-F2 are reciprocal to each other in some sense. The presented method produces accurate results for inverse PDE problems, as shown by the numerical examples herein. For noise-free data, the errors of the inverse parameters and the solution field decrease exponentially as the number of collocation points or the number of trainable network parameters increases, and can reach a level close to the machine accuracy. For noisy data, the accuracy degrades compared with the case of noise-free data, but the method remains quite accurate. The presented method has been compared with the physics-informed neural network method.

Keywords: *randomized neural networks, extreme learning machine, nonlinear least squares, variable projection, inverse problems, inverse PDE*

1 Introduction

In this work we focus on the simultaneous determination of the parameters (as constants or field distributions) and the solution field to parametric PDEs based on artificial neural networks (ANN/NN), given sparse and noisy measurement data of certain variables. This type of problems is often referred to as the inverse PDE problems in the literature [31]. Typical examples include the determination of the diffusion coefficient given certain concentration data or the computation of the wave speed given sparse measurement of the wave profile. When the parameter values in the PDE are known, approximation of the PDE solution is often referred to as the forward PDE problem. We will adopt these notations in this paper.

Closely related to the inverse PDE problems is the data-driven “discovery” of PDEs [4, 7], in which, given certain measurement data, the PDE functional form is to be discerned. Early works in this area

[*]Author of correspondence. Emails: sdong@purdue.edu (S. Dong), wang2335@purdue.edu (Y. Wang).

include [4, 58] based on symbolic regression and evolutionary algorithms for identifying the hidden physical laws. An alternative approach based on sparse regression/optimization has been investigated in [7, 53, 55, 52], in which a library of candidate functions and their derivatives is constructed first and then key terms are selected from this library to express the dynamics by sparsity promotion techniques (L^1 regularization). The work [69] employs dimensional analysis and sparse Bayesian regression to determine the candidate terms and to approximate their weights in the underlying equations. In [2] the measurement data is first approximated by a neural network in order to attain the derivative data of the measured variables, and then another neural network (with L^1 regularization) is used to approximate the functional form of the underlying equation. A symbolic neural network has been employed to represent the PDE form in [35], thus replacing the library of candidate functions, and the derivatives of the measurement data are computed by convolutions. In [64] the discrete evolution operator, rather than the functional form, for the PDE is learned with deep neural networks. In another recent development [5] the state variables are represented by a neural network, whose output is used to construct the set of candidate functions, and sparse regression is encoded into the loss function of the neural network.

As advocated in [60, 31], data-driven scientific machine learning problems can be viewed in terms of the amount of data that is available and the amount of physics that is known. They are broadly classified into three categories in [31]: (i) those with "lots of physics and small data" (e.g. forward PDE problems), (ii) those with "some physics and some data" (e.g. inverse PDE problems), and (iii) those with "no physics and big data" (e.g. general PDE discovery). The authors of [31] point out that those in the second category are typically the more interesting and representative in real applications, where the physics is partially known and sparse measurements are available. One illustrating example is from multiphase flows, where the conservation laws (mass/momentum conservations) and thermodynamic principles (second law of thermodynamics, Galilean invariance) lead to a thermodynamically-consistent phase field model, but with an incomplete system of governing equations [15, 14]. One has the freedom to choose the form of the free energy, the wall energy, the form and coefficients of the constitutive relation, and the form and coefficient of the interfacial mobility [12, 13, 67]. Different choices will lead to different specific models, which are all thermodynamically consistent. The different models cannot be distinguished by the thermodynamic principles, but can be differentiated with experimental measurements.

The development of machine learning techniques for solving inverse PDE problems has attracted a great deal of interest recently, with a variety of contributions from different researchers. In [49] a method for estimating the parameters in nonlinear PDEs is developed based on Gaussian processes. The physics informed neural network (PINN) method is introduced in the influential work [50] for solving forward and inverse nonlinear PDEs. The residuals for the PDE, the boundary/initial conditions, and the measurement data are encoded into the loss function as soft constraints, and the neural network is trained by gradient descent (or back propagation) type algorithms. The PINN idea has significantly influenced subsequent developments and stimulated applications in many related areas (see e.g. [37, 39, 56, 36, 65, 47], among others). A hybrid method combining finite element and neural networks is developed in [1]. The finite element method (FEM) is used to solve the underlying PDE, which is augmented by a neural network to represent the PDE coefficient [1]. A conservative PINN method is proposed in [29] together with domain decomposition for simulating nonlinear conservation laws, in which the flux continuity is enforced along the sub-domain interfaces, and interesting results are presented for several forward and inverse problems. This method is further developed and extended subsequently with domain decompositions in both space and time [28]; see a recent study of this extended technique for supersonic flows [30]. Interesting applications are described in [51, 9], where PINN is employed to infer the 3D velocity and pressure fields based on scattered flow visualization data or Schlieren images from experiments. In [20] a distributed PINN technique based on domain decomposition is presented, in which for nonlinear PDEs a related linearized equation is solved with certain variables fixed at their initial values. An auxiliary PINN technique is developed in [68] for solving nonlinear integro-differential equations, in which auxiliary variables are introduced to represent the anti-derivatives and thus avoiding the integral computation. We would also like to mention [11, 60, 38, 34] (among others) for inverse applications of neural networks in other related fields. It is noted that in the above works the full set of NN parameters (from the hidden layers and the output layer) are trainable.

In the current work we consider the use of randomized neural networks, also known as extreme learning machines (ELM) [25] (or random vector functional link (RVFL) networks [46]), for solving inverse PDE problems. ELM was originally developed for linear classification and regression problems. It is characterized by two ideas: (i) randomly assigned but fixed (non-trainable) hidden-layer coefficients, and (ii) trainable linear output-layer coefficients determined by linear least squares or by using the Moore-Penrose inverse [25]. This technique has been extended to scientific computing in the past few years, for function approximations and for solving ordinary and partial differential equations (ODE/PDE); see e.g. [66, 45, 21, 16, 17, 10, 22, 57, 19, 43], among others. The random-weight neural networks are universal function approximators. As established by the theoretical results of [27, 26, 40], a single-hidden-layer feed-forward neural network (FNN) having random but fixed (not trained) hidden units can approximate any continuous function to any desired degree of accuracy, provided that the number of hidden units is sufficiently large.

In this paper we present a method for computing inverse PDE problems based on randomized neural networks. This extends the local extreme learning machine (locELM) technique originally developed in [16] for forward PDEs to inverse problems. Because of the coupling between the unknown PDE parameters (referred to as the inverse parameters hereafter) and the solution field, the inverse PDE problem is fully nonlinear with respect to the unknowns, even though the associated forward PDE may be linear. We partition the overall domain into sub-domains, and represent the solution field (and the inverse parameters, if they are field distributions) by a local FNN on each sub-domain, imposing $C^{\mathbf{k}}$ (with appropriate $\mathbf{k}$) continuity conditions across the sub-domain boundaries. The weights/biases in the hidden layers of the local NNs are assigned to random values and fixed (not trainable), and only the output-layer coefficients are trainable. The inverse PDE problem is thus reduced to a nonlinear problem about the inverse parameters and the output-layer coefficients of the solution field, or if the inverse parameters are field distributions, about the output-layer coefficients for the inverse parameters and the solution field.

We develop three algorithms for training the neural network to solve the inverse PDE problem:

- The first algorithm (termed NLLSQ) computes the inverse parameters and the trainable parameters of the local NNs all together by the nonlinear least squares method [3]. This extends the nonlinear least squares method with perturbations (NLLSQ-perturb) from [16] (developed for forward nonlinear PDEs) to inverse PDE problems.

- The second algorithm (termed VarPro-F1) eliminates the inverse parameters from the overall problem based on the variable projection (VarPro) strategy [23, 24] to attain a reduced problem about the trainable network parameters only. It solves the reduced problem first for the trainable parameters of the local NNs by the NLLSQ-perturb algorithm, and then computes the inverse parameters by the linear least squares method.

- The third algorithm (termed VarPro-F2) eliminates the trainable network parameters from the overall inverse problem by variable projection to arrive at a reduced problem about the inverse parameters only. It solves the reduced problem first for the inverse parameters by the NLLSQ-perturb algorithm, and then computes the trainable parameters of the local NNs based on the inverse parameters already obtained. The VarPro-F2 and VarPro-F1 algorithms both employ the variable projection idea and are reciprocal formulations in a sense. For inverse problems with an associated forward nonlinear PDE, VarPro-F2 needs to be combined with a Newton iteration.

The presented method produces accurate solutions to inverse PDE problems, as shown by a number of numerical examples presented herein. For noise-free data, the errors for the inverse parameters and the solution field decrease exponentially as the number of training collocation points or the number of trainable parameters in the neural network increases. These errors can reach a level close to the machine accuracy when the simulation parameters become large. For noisy data, the current method remains quite accurate, although the accuracy degrades compared with the case of noise-free data. We observe that, by scaling the measurement-residual vector by a factor, one can markedly improve the accuracy of the current method for noisy data, while only slightly degrading the accuracy for noise-free data. We have compared the current method with the PINN method (see Appendix E). The current method exhibits an advantage in terms of the accuracy and the computational cost (network training time).

Both the second and the third algorithms developed herein are based on the idea of variable projection (VarPro) [23, 24], as mentioned earlier. VarPro is a classical strategy for solving separable nonlinear least squares problems [23, 32, 24, 44]. These are problems in which the unknown parameters can be separated into two sets, the linear parameters and the nonlinear parameters. VarPro treats the linear parameters as dependent on the nonlinear parameters, and then seeks to eliminate the linear parameters from the problem to arrive at a reduced problem about the nonlinear parameters only. The nonlinear parameters are determined first by solving the reduced problem, and the linear parameters are computed afterwards. The benefits of variable projection include the reduced dimension of parameter space, better conditioning, and faster convergence with the reduced problem [54, 59, 24]. The VarPro approach for training neural networks has been investigated in e.g. [61, 63, 62, 59, 48, 33, 42, 41, 18] (among others). The projection learning method [61, 63, 62] seems to be the earliest work on neural-network training in the spirit of variable projection. The improved conditioning in the problem and faster convergence with VarPro for neural network fitting is shown in [59]. In [48, 33] two-layered neural networks are trained by VarPro together with the Levenberg-Marquardt method. In more recent works [42, 41], VarPro has been extended to handle non-quadratic objective functions (e.g. the cross-entropy function for classification problems) and a stochastic optimization method (slimTrain) based on VarPro has been developed. In [18] the VarPro strategy has been adapted to numerically solving linear and nonlinear (forward) PDEs by a physics informed neural network-like approach, leading to spectral-like accuracy in the computation results.

The method and algorithms developed herein are implemented in Python based on the Tensorflow[1], Keras[2], and the scipy[3] libraries. The numerical simulations are performed on a MAC computer (3.2GHz Intel Core i5 CPU, 24GB memory) in the authors' institution.

The main contribution of this paper lies in the local extreme learning machine based technique together with the three algorithms for solving inverse PDE problems. The exponential convergence behavior exhibited by the current method for inverse problems is particularly interesting, and can be analogized to the

[1]https://www.tensorflow.org/
[2]https://keras.io/
[3]https://scipy.org/

observations in [16] for forward PDEs. For inverse problems such fast convergence seems not available in the existing techniques (e.g. PINN based methods).

The rest of this paper is structured as follows. In Section 2 we first discuss the representation of functions by local randomized neural networks and domain decomposition, and then present the NLLSQ, VarPro-F1 and VarPro-F2 algorithms for training the neural network to solve the inverse PDE. Section 3 uses a number of inverse parametric PDEs to demonstrate the exponential convergence and the accuracy of our method, as well as the effects of the noise and the number of measurement points. Section 4 concludes the discussion with some closing remarks. Appendix A summarizes the NLLSQ-perturb algorithm from [16] (with modifications), which forms the basis for the three algorithms in the current paper for solving inverse PDEs. Appendix B provides the matrices in the NLLSQ and VarPro-F2 algorithms. Appendix C and Appendix D provide additional numerical tests of the current method with the inverse parametric advection equation and Sine-Gordon equation, respectively. Appendix E compares the current method with PINN for several inverse problems from Section 3. Appendix F lists the parameter values in the NLLSQ-perturb algorithm for all the numerical simulations presented in this paper.

2 Algorithms for Inverse PDEs with Randomized Neural Networks

2.1 Inverse Parametric PDEs and Local Randomized Neural Networks

We focus on the inverse problem described by the following parametric PDE, boundary conditions, and measurement operations on some domain $\Omega \subset \mathbb{R}^d$ $(d = 1, 2, 3)$:

$$\alpha_1\mathcal{L}_1(u) + \alpha_2\mathcal{L}_2(u) + \cdots + \alpha_n\mathcal{L}_n(u) + \mathcal{F}(u) = f(\mathbf{x}), \quad \mathbf{x} \in \Omega, \tag{1a}$$

$$\mathcal{B}u(\mathbf{x}) = g(\mathbf{x}), \quad \mathbf{x} \in \partial\Omega, \tag{1b}$$

$$\mathcal{M}u(\boldsymbol{\xi}) = S(\boldsymbol{\xi}), \quad \boldsymbol{\xi} \in \Omega_s \subset \Omega. \tag{1c}$$

In this system, $\mathcal{L}_i$ $(1 \leqslant i \leqslant n)$ and $\mathcal{F}$ are differential or algebraic operators, which can be linear or nonlinear, and f and g are prescribed source terms. $u(\mathbf{x})$ is an unknown scalar field, where $\mathbf{x}$ denotes the coordinates. α_i $(1 \leqslant i \leqslant n)$ are n unknown constants. The case with any α_i being an unknown field distribution will be dealt with later in a remark (Remark 2.7). We assume that the highest derivative term in (1a) is linear with respect to u, while the nonlinear terms with respect to u involve only lower derivatives (if any). $\mathcal{B}$ is a linear differential or algebraic operator, and $\mathcal{B}u$ denotes the boundary condition(s) on the domain boundary $\partial\Omega$. $\mathcal{M}$ is a linear algebraic or differential operator representing the measurement operations. $\mathcal{M}u(\boldsymbol{\xi})$ denotes the measurement of $\mathcal{M}u$ at the point $\boldsymbol{\xi}$, and $S(\boldsymbol{\xi})$ denotes the measurement data. Ω_s denotes the set of measurement points. Given $S(\boldsymbol{\xi})$, the goal here is to determine the parameters α_i $(1 \leqslant i \leqslant n)$ and the solution field $u(\mathbf{x})$. Hereafter we will refer to the parameters $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_n)^T$ as the inverse parameters. Suppose the inverse parameters are given. The boundary value problem consisting of the equations (1a)–(1b) will be referred to as the associated forward PDE problem, with $u(\mathbf{x})$ as the unknown. We assume that the formulation is such that the forward PDE problem is well-posed.

Remark 2.1. *We assume that the operators $\mathcal{L}_i$ $(1 \leqslant i \leqslant n)$ or $\mathcal{F}$ may contain time derivatives (e.g. $\frac{\partial}{\partial t}$, $\frac{\partial^2}{\partial t^2}$, where t denotes time), thus leading to an initial-boundary value problem on a spatial-temporal domain Ω. In this case, we treat t in the same way as the spatial coordinate $\mathbf{x}$, and use the last dimension in $\mathbf{x} = (x_1, x_2, \dots, x_d)$ to denote t (i.e. $x_d \equiv t$). Accordingly, we assume that the equation* (1b) *should include conditions on the appropriate initial boundaries from $\partial\Omega$. The point here is that the system* (1) *may refer to time-dependent problems, and we will not distinguish this case in subsequent discussions.*

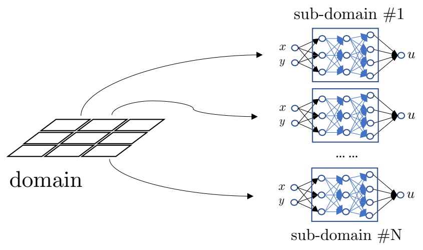


Figure 1: Cartoon illustrating domain decomposition and local random-weight neural networks.

We devise numerical algorithms to compute a least squares solution to the system (1) based on local randomized neural networks (or ELM). We decompose the domain Ω into sub-domains, and represent $u(\mathbf{x})$ on each sub-domain by a local ELM in a way analogous to in [16]. Let $\Omega = \Omega_1 \cup \Omega_2 \cup \cdots \cup \Omega_N$, where Ω_i ($1 \leqslant i \leqslant N$) denote N non-overlapping sub-domains (see Figure 1 for an illustration). Let

$$u(\mathbf{x}) = \begin{cases} u_1(\mathbf{x}), & \mathbf{x} \in \Omega_1, \\ u_2(\mathbf{x}), & \mathbf{x} \in \Omega_2, \\ \cdots \\ u_N(\mathbf{x}), & \mathbf{x} \in \Omega_N, \end{cases} \tag{2}$$

where $u_i(\mathbf{x})$ ($1 \leqslant i \leqslant N$) denotes the solution field restricted to the sub-domain Ω_i. On the interior sub-domain boundaries shared by adjacent sub-domains we impose $C^{\mathbf{k}}$ continuity conditions on $u(\mathbf{x})$, where $\mathbf{k} = (k_1, \dots, k_d)$ denotes a set of appropriate non-negative integers related to the order of the PDE (1a). If the PDE order (highest derivative) is m_i along the x_i ($1 \leqslant i \leqslant d$) direction, we would in general impose $C^{m_i - 1}$ (i.e. $k_i = m_i - 1$) continuity conditions in this direction on the shared sub-domain boundaries.

On Ω_i ($1 \leqslant i \leqslant N$) we employ a local FNN, whose hidden-layer coefficients are randomly assigned and fixed, to represent $u_i(\mathbf{x})$. More specifically, the local neural network is set as follows. The input layer consists of d nodes, representing the input coordinate $\mathbf{x} = (x_1, x_2, \dots, x_d) \in \Omega_i$. The output layer consists of a single node, representing $u_i(\mathbf{x})$. The network contains $(L-1)$ (with integer $L \geqslant 2$) hidden layers in between. Let $\sigma : \mathbb{R} \to \mathbb{R}$ denote the activation function for all the hidden nodes. Hereafter we use the following vector (or list) $\mathbf{M}$ of $(L+1)$ positive integers to represent the architecture of the local NN,

$$\mathbf{M} = [m_0, m_1, \dots, m_{L-1}, m_L], \quad \text{(architectural vector)} \tag{3}$$

where $m_0 = d$ and $m_L = 1$ denote the number of nodes in the input/output layers respectively, and m_i is the number of nodes in the i-th hidden layer ($1 \leqslant i \leqslant L-1$). We refer to $\mathbf{M}$ as an architectural vector.

We make the following assumptions:

- The output layer should contain (i) no bias, and (ii) no activation function (or equivalently, the activation function be $\sigma(x) = x$).
- The weights/biases in all the hidden layers are pre-set to uniform random values on $[-R_m, R_m]$, where $R_m > 0$ is a user-provided constant. The hidden-layer coefficients are fixed once they are set.
- The output-layer weights constitute the the trainable parameters of the local neural network.

We employ the same architecture, same activation function, and the same R_m for the local neural networks on different sub-domains.

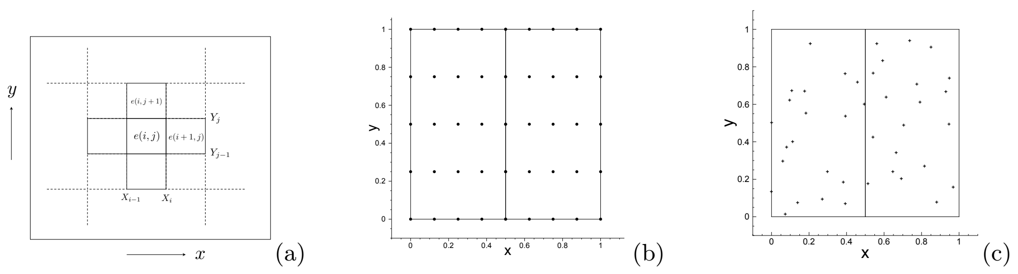


Figure 2: Sub-domains and collocation/measurement points: (a) Sketch of adjacent sub-domains. (b) Sketch of uniform grid points as collocation points (5×5 here) on two adjacent sub-domains. (c) Sketch of 20 random measurement points (shown as "+" symbols) in each sub-domain on two adjacent sub-domains.

In light of these settings, the logic in the output layer of the local NNs leads to the following relation on the sub-domain Ω_i ($1 \leqslant i \leqslant N$),

$$u_i(\mathbf{x}) = \sum_{j=1}^{M} \beta_{ij}\phi_{ij}(\mathbf{x}) = \mathbf{\Phi}_i(\mathbf{x})\boldsymbol{\beta}_i, \tag{4}$$

where $M = m_{L-1}$ denotes the width of the last hidden layer of the local NN, $\phi_{ij}(\mathbf{x})$ ($1 \leqslant j \leqslant M$) denote the set of output fields of the last hidden layer on Ω_i, β_{ij} ($1 \leqslant j \leqslant M$) denote the set of output-layer coefficients (trainable parameters) on Ω_i, and $\mathbf{\Phi}_i = (\phi_{i1}, \phi_{i2}, \dots, \phi_{iM})$ and $\boldsymbol{\beta}_i = (\beta_{i1}, \beta_{i2}, \dots, \beta_{iM})^T$. Note that, once the random hidden-layer coefficients are assigned, $\mathbf{\Phi}_i(\mathbf{x})$ in (4) denotes a set of random (but fixed and known) nonlinear basis functions. Therefore, with local ELMs the output field on each sub-domain is represented by an expansion of a set of random basis functions as given by (4).

With domain decomposition and local ELMs, the system (1) is symbolically transformed into the following form, which includes the continuity conditions across shared sub-domain boundaries:

$$\alpha_1\mathcal{L}_1(u_i) + \alpha_2\mathcal{L}_2(u_i) + \cdots + \alpha_n\mathcal{L}_n(u_i) + \mathcal{F}(u_i) = f(\mathbf{x}), \quad \mathbf{x} \in \Omega_i,\ 1 \leqslant i \leqslant N; \tag{5a}$$

$$\mathcal{B}u_i(\mathbf{x}) = g(\mathbf{x}), \quad \mathbf{x} \in \partial\Omega \cap \Omega_i,\ 1 \leqslant i \leqslant N; \tag{5b}$$

$$\mathcal{M}u_i(\boldsymbol{\xi}) = S(\boldsymbol{\xi}), \quad \boldsymbol{\xi} \in \Omega_s \cap \Omega_i,\ 1 \leqslant i \leqslant N; \tag{5c}$$

$$\mathcal{C}u_i(\mathbf{x}) - \mathcal{C}u_j(\mathbf{x}) = 0, \quad \mathbf{x} \in \partial\Omega_i \cap \partial\Omega_j,\ \text{for all adjacent sub-domains } (\Omega_i, \Omega_j), \quad 1 \leqslant i, j \leqslant N. \tag{5d}$$

In this system $u_i(\mathbf{x})$ is given by (4), and the operator $\mathcal{C}u$ denotes the set of $C^{\mathbf{k}}$ continuity conditions imposed across the shared sub-domain boundaries on u or its derivatives. Define the residual of this system as,

$$\mathbf{R}(\boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{x}, \boldsymbol{\xi}) = \begin{bmatrix} \alpha_1\mathcal{L}_1(u_i) + \alpha_2\mathcal{L}_2(u_i) + \cdots + \alpha_n\mathcal{L}_n(u_i) + \mathcal{F}(u_i) - f(\mathbf{x}),\ \mathbf{x} \in \Omega_i,\ 1 \leqslant i \leqslant N \\ \mathcal{B}u_i(\mathbf{x}) - g(\mathbf{x}),\ \mathbf{x} \in \partial\Omega \cap \Omega_i,\ 1 \leqslant i \leqslant N \\ \mathcal{M}u_i(\boldsymbol{\xi}) - S(\boldsymbol{\xi}),\ \boldsymbol{\xi} \in \Omega_s \cap \Omega_i,\ 1 \leqslant i \leqslant N \\ \mathcal{C}u_i(\mathbf{x}) - \mathcal{C}u_j(\mathbf{x}),\ \mathbf{x} \in \partial\Omega_i \cap \partial\Omega_j,\ \text{for all adjacent } (\Omega_i, \Omega_j),\ 1 \leqslant i, j \leqslant N \end{bmatrix}, \tag{6}$$

where $\boldsymbol{\beta}$ is the vector of all trainable parameters, $\boldsymbol{\beta} = (\boldsymbol{\beta}_1^T, \dots, \boldsymbol{\beta}_N^T)^T = (\beta_{11}, \beta_{12}, \dots, \beta_{1M}, \beta_{21}, \dots, \beta_{NM})^T$.

The system (5) is what we would solve numerically by least squares for the inverse parameters $\boldsymbol{\alpha}$ and the trainable network parameters $\boldsymbol{\beta}$. After $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ are determined, the field solution $u(\mathbf{x})$ is computed by (2) and (4). In what follows we present three algorithms, one based on the nonlinear least squares method with perturbations and the other two based on the variable projection idea, for determining the $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$.

2.2 Nonlinear Least Squares (NLLSQ) Method for Network Training

We first outline a basic algorithm for computing $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ by the nonlinear least squares (NLLSQ) method with perturbations [16]. It forms the basis for the variable projection algorithms presented in the next subsection.

For the simplicity of presentation we focus on rectangular domains, i.e. $\Omega = [a_1, b_1]\times[a_2, b_2]\times\cdots\times[a_d, b_d]$, where a_i and b_i ($1 \leqslant i \leqslant d$) denote the lower/upper bounds of Ω in the x_i direction, and assume that Ω is partitioned into N_i ($N_i \geqslant 1$) sub-domains along x_i ($1 \leqslant i \leqslant d$).

To make the discussion more concrete, we specifically consider a second-order PDE in two dimensions ($d = 2$, $\mathbf{x} = (x_1, x_2) = (x, y)$) as an example in this and the next subsections. In the following discussions we assume that equation (1a) is of second order with respect to both x and y, and we impose C^1 continuity conditions across the sub-domain boundaries in both x and y directions.

Let the vectors $\boldsymbol{\mathcal{X}} = (X_0, X_1, \dots, X_{N_1})$ and $\boldsymbol{\mathcal{Y}} = (Y_0, Y_1, \dots, Y_{N_2})$ denote the sub-domain boundary points along the two directions, respectively, where $(X_0, X_{N_1}) = (a_1, b_1)$ and $(Y_0, Y_{N_2}) = (a_2, b_2)$. The total number of sub-domains is $N = N_1N_2$. We assume that the sub-domain Ω_e ($1 \leqslant e \leqslant N$) is characterized by the partition indices (i, j) along the x and y directions (see Figure 2(a)), with the following relation,

$$\Omega_e = \Omega_{e(i,j)} = [X_{i-1}, X_i] \times [Y_{j-1}, Y_j], \quad e = e(i, j) = (i - 1)N_2 + j, \quad \text{for } 1 \leqslant (i, j) \leqslant (N_1, N_2), \tag{7}$$

where "$1 \leqslant (i, j) \leqslant (N_1, N_2)$" or "$(1, 1) \leqslant (i, j) \leqslant (N_1, N_2)$" stands for $1 \leqslant i \leqslant N_1$ and $1 \leqslant j \leqslant N_2$. We will use this and similar notations hereafter for conciseness.

With these settings the boundary conditions in (5b) are reduced to,

$$\mathcal{B}u_{e(1,j)}(a_1, y) = g(a_1, y), \quad \mathcal{B}u_{e(N_1,j)}(b_1, y) = g(b_1, y), \quad \text{for } 1 \leqslant j \leqslant N_2; \tag{8a}$$

$$\mathcal{B}u_{e(i,1)}(x, a_2) = g(x, a_2), \quad \mathcal{B}u_{e(i,N_2)}(x, b_2) = g(x, b_2), \quad \text{for } 1 \leqslant i \leqslant N_1. \tag{8b}$$

Here $u_{e(i,j)}$ denotes u on $\Omega_{e(i,j)}$, and $e(i, j)$ is given by (7). The C^1 continuity conditions in (5d) reduce to,

$$u_{e(i,j)}(X_i, y) - u_{e(i+1,j)}(X_i, y) = 0, \quad \text{for } 1 \leqslant (i, j) \leqslant (N_1 - 1, N_2); \tag{9a}$$

$$\left.\frac{\partial u_{e(i,j)}}{\partial x}\right|_{(X_i,y)} - \left.\frac{\partial u_{e(i+1,j)}}{\partial x}\right|_{(X_i,y)} = 0, \quad \text{for } 1 \leqslant (i, j) \leqslant (N_1 - 1, N_2); \tag{9b}$$

$$u_{e(i,j)}(x, Y_j) - u_{e(i,j+1)}(x, Y_j) = 0, \quad \text{for } 1 \leqslant (i, j) \leqslant (N_1, N_2 - 1); \tag{9c}$$

$$\left.\frac{\partial u_{e(i,j)}}{\partial y}\right|_{(x,Y_j)} - \left.\frac{\partial u_{e(i,j+1)}}{\partial y}\right|_{(x,Y_j)} = 0, \quad \text{for } 1 \leqslant (i, j) \leqslant (N_1, N_2 - 1). \tag{9d}$$

The equations (9a) and (9c) are the C^0 conditions on the horizontal/vertical sub-domain boundaries, and the equations (9b) and (9d) are the corresponding C^1 conditions.

The system to solve now consists of equations (5a), (8), (5c), and (9). This is a continuous system. We next enforce this system on a set of collocation points and measurement points to arrive at a discrete system about the parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$.

We choose a set of Q ($Q \geqslant 1$) collocation points on each sub-domain Ω_e ($1 \leqslant e \leqslant N$), denoted by $\mathbf{x}_p^e = (x_p^e, y_p^e)$ ($1 \leqslant p \leqslant Q$), among which Q_b ($1 \leqslant Q_b < Q$) points reside on $\partial\Omega_e$. Let $\mathbb{X}_e$ denote the set of collocation points on Ω_e, and $\mathbb{X}_e^b = \mathbb{X}_e \cap \partial\Omega_e$ denote the set of collocation points residing on the sub-domain boundaries. The boundary collocation points on adjacent sub-domains are required to be compatible. That is, for any two adjacent sub-domains $(\Omega_{e_1}, \Omega_{e_2})$, those boundary collocation points from Ω_{e_1} that reside on the shared boundary $\partial\Omega_{e_1} \cap \partial\Omega_{e_2}$ are required to be identical to those boundary collocation points from Ω_{e_2} that reside on the same boundary.

The collocation points can in principle be chosen based on various distributions (e.g. random, uniform). In this paper we focus on using uniform grid points as the collocation points; see Figure 2(b) for an illustration with a 5×5 uniform grid points as the collocation points on two neighboring sub-domains. Let Q_1 and Q_2 denote the number of uniform grid points along x and y, with $Q = Q_1Q_2$. The uniform collocation points on the sub-domain $\Omega_e = \Omega_{e(m,l)}$ $(1 \leqslant (m,l) \leqslant (N_1, N_2))$ are given by

$$\begin{cases} \mathbf{x}_p^e = \mathbf{x}_{p(i,j)}^{e(m,l)} = \left(x_{p(i,j)}^{e(m,l)}, y_{p(i,j)}^{e(m,l)}\right), \quad x_{p(i,j)}^{e(m,l)} = X_{m-1} + (i-1)(X_m - X_{m-1})/(Q_1 - 1), \\ y_{p(i,j)}^{e(m,l)} = Y_{l-1} + (j-1)(Y_l - Y_{l-1})/(Q_2 - 1), \quad \text{for } 1 \leqslant (m,l,i,j) \leqslant (N_1, N_2, Q_1, Q_2); \\ p = p(i,j) = (i-1)Q_2 + j, \quad \text{for } 1 \leqslant (p,i,j) \leqslant (Q, Q_1, Q_2). \end{cases} \tag{10}$$

We assume that the measurement data is given on a set of Q_s $(Q_s \geqslant 1)$ random measurement points (with a uniform distribution) on each Ω_e $(1 \leqslant e \leqslant N)$, denoted by $\boldsymbol{\xi}_p^e = (\xi_p^e, \eta_p^e)$ $(1 \leqslant p \leqslant Q_s)$. Figure 2(c) shows an example of $Q_s = 20$ random measurement points in each sub-domain on two adjacent sub-domains. We use $\mathbb{Y}_e$ to denote the set of measurement points on Ω_e $(1 \leqslant e \leqslant N)$.

Once the hidden-layer coefficients of local NNs are randomly assigned and the collocation and measurement points are chosen, we compute the last hidden-layer field data $\boldsymbol{\Phi}_e(\mathbf{x}_p^e)$ and their derivatives (up to a certain order), and the data for $\mathcal{M}\boldsymbol{\Phi}_e(\boldsymbol{\xi}_p^e)$, by forward evaluations of the neural network and by automatic differentiations. We then store these data for subsequent use. In light of (4), for any given $\boldsymbol{\beta} = (\boldsymbol{\beta}_1^T, \dots, \boldsymbol{\beta}_N^T)^T$, we have

$$u_e(\mathbf{x}_p^e) = \boldsymbol{\Phi}_e(\mathbf{x}_p^e)\boldsymbol{\beta}_e,\ \mathcal{D}u_e(\mathbf{x}_p^e) = \mathcal{D}\boldsymbol{\Phi}_e(\mathbf{x}_p^e)\boldsymbol{\beta}_e,\ \mathcal{M}u_e(\boldsymbol{\xi}_q^e) = \mathcal{M}\boldsymbol{\Phi}_e(\boldsymbol{\xi}_q^e)\boldsymbol{\beta}_e,\ 1 \leqslant (e,p,q) \leqslant (N, Q, Q_s), \tag{11}$$

where $\mathcal{D}$ is a linear differential operator and $\mathcal{M}$ is the measurement operator.

Remark 2.2. *To compute $\boldsymbol{\Phi}_e(\mathbf{x}_p^e)$, $\mathcal{D}\boldsymbol{\Phi}_e(\mathbf{x}_p^e)$ and $\mathcal{M}\boldsymbol{\Phi}_e(\boldsymbol{\xi}_p^e)$, in the implementation we create a Keras sub-model, referred to as the last-hidden-layer-model, to the local NN for each sub-domain. The input nodes to this sub-model are identical to those of the original local NN, and the output nodes of this sub-model consist of those nodes in the last hidden layer of the original local NN. We compute $\boldsymbol{\Phi}_e(\mathbf{x}_p^e)$ $(1 \leqslant p \leqslant Q)$ and $\boldsymbol{\Phi}_e(\boldsymbol{\xi}_p^e)$ $(1 \leqslant p \leqslant Q_s)$ by a forward evaluation of the last-hidden-layer-model for Ω_e on the input data (collocation points, or measurement points). We compute the derivatives of $\boldsymbol{\Phi}_e$ on $\mathbf{x}_p^e$ or on $\boldsymbol{\xi}_p^e$ by a forward-mode auto-differentiation of the last-hidden-layer-model, implemented by the "ForwardAccumulator" in the Tensorflow library. The forward-mode auto-differentiation is crucial to the performance of the ELM method (see [19]).*

To derive the discrete system we enforce (5a) on all the collocation points in $\mathbb{X}_e$ $(1 \leqslant e \leqslant N)$, enforce (8) on all the boundary collocation points in $\mathbb{X}_e^b \cap \partial\Omega$ for $1 \leqslant e \leqslant N$, enforce (5c) on all the measurement points in $\mathbb{Y}_e$ $(1 \leqslant e \leqslant N)$, and enforce (9) on those collocation points from $\mathbb{X}_e^b$ $(1 \leqslant e \leqslant N)$ that reside on the shared boundaries of adjacent sub-domains.

The discrete system corresponding to (5a) enforced on the collocation points is,

$$\alpha_1 \mathcal{L}_1\left(u_e(\mathbf{x}_p^e)\right) + \cdots + \alpha_n \mathcal{L}_n\left(u_e(\mathbf{x}_p^e)\right) + \mathcal{F}\left(u_e(\mathbf{x}_p^e)\right) - f\left(\mathbf{x}_p^e\right) = 0, \text{ for } \mathbf{x}_p^e \in \mathbb{X}_e,\ 1 \leqslant (e,p) \leqslant (N, Q). \tag{12}$$

The discrete system corresponding to (8) on the boundary collocation points is given by,

$$\mathcal{B}u_{e(1,l)}(a_1, y_{p(1,j)}^{e(1,l)}) - g(a_1, y_{p(1,j)}^{e(1,l)}) = 0, \quad \text{for } 1 \leqslant (l,j) \leqslant (N_2, Q_2); \tag{13a}$$

$$\mathcal{B}u_{e(N_1,l)}(b_1, y_{p(Q_1,j)}^{e(N_1,l)}) - g(b_1, y_{p(Q_1,j)}^{e(N_1,l)}) = 0, \quad \text{for } 1 \leqslant (l,j) \leqslant (N_2, Q_2); \tag{13b}$$

$$\mathcal{B}u_{e(m,1)}(x_{p(i,1)}^{e(m,1)}, a_2) - g(x_{p(i,1)}^{e(m,1)}, a_2) = 0, \quad \text{for } 1 \leqslant (m,i) \leqslant (N_1, Q_1); \tag{13c}$$

$$\mathcal{B}u_{e(m,N_2)}(x_{p(i,Q_2)}^{e(m,N_2)}, b_2) - g(x_{p(i,Q_2)}^{e(m,N_2)}, b_2) = 0, \quad \text{for } 1 \leqslant (m,i) \leqslant (N_1, Q_1). \tag{13d}$$

Here the functions $e(\cdot,\cdot)$ and $p(\cdot,\cdot)$ are defined in (7) and (10), respectively. The discrete system corresponding to (5c) enforced on the measurement points is given by

$$\mathcal{M}u_e(\boldsymbol{\xi}_p^e) - S(\boldsymbol{\xi}_p^e) = 0, \quad \text{for } \boldsymbol{\xi}_p^e \in \mathbb{Y}_e,\ 1 \leqslant (e,p) \leqslant (N, Q_s). \tag{14}$$

The discrete system corresponding to (9) enforced on the interior sub-domain boundary points is,

$$u_{e(m,l)}(X_m, y_{p(Q_1,j)}^{e(m,l)}) - u_{e(m+1,l)}(X_m, y_{p(1,j)}^{e(m+1,l)}) = 0, \quad \text{for } 1 \leqslant (m,l,j) \leqslant (N_1-1, N_2, Q_2); \tag{15a}$$

$$\left.\frac{\partial u_{e(m,l)}}{\partial x}\right|_{(X_m, y_{p(Q_1,j)}^{e(m,l)})} - \left.\frac{\partial u_{e(m+1,l)}}{\partial x}\right|_{(X_m, y_{p(1,j)}^{e(m+1,l)})} = 0, \quad \text{for } 1 \leqslant (m,l,j) \leqslant (N_1-1, N_2, Q_2); \tag{15b}$$

$$u_{e(m,l)}(x_{p(i,Q_2)}^{e(m,l)}, Y_l) - u_{e(m,l+1)}(x_{p(i,1)}^{e(m,l+1)}, Y_l) = 0, \quad \text{for } 1 \leqslant (m,l,i) \leqslant (N_1, N_2-1, Q_1); \tag{15c}$$

$$\left.\frac{\partial u_{e(m,l)}}{\partial y}\right|_{(x_{p(i,Q_2)}^{e(m,l)}, Y_l)} - \left.\frac{\partial u_{e(m,l+1)}}{\partial y}\right|_{(x_{p(i,1)}^{e(m,l+1)}, Y_l)} = 0, \quad \text{for } 1 \leqslant (m,l,i) \leqslant (N_1, N_2-1, Q_1). \tag{15d}$$

In the above equations $\mathbf{x}_p^e$, $x_{p(i,j)}^{e(m,l)}$ and $y_{p(i,j)}^{e(m,l)}$ are defined in (10), and $u_e(\mathbf{x})$ is given by (4) and (11).

The equations (12)–(15d) form the system we would solve to determine the inverse parameters $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_n)^T$ and the trainable network parameters $\boldsymbol{\beta} = (\beta_{11}, \dots, \beta_{NM})^T$. This is a system of nonlinear algebraic equations about $(\boldsymbol{\alpha}, \boldsymbol{\beta})$. Note that the functions $\boldsymbol{\Phi}_e(\mathbf{x})$ $(1 \leqslant e \leqslant N)$ and their derivatives evaluated on the collocation/measurement points, which are involved in the operators such as $\mathcal{L}_i(u_e)$, $\mathcal{F}(u_e)$, $\mathcal{B}u_e$, $\mathcal{M}u_e$, and $\mathcal{C}u_e$, are computed by evaluations of the neural network and auto-differentiations (see Remark 2.2). This system consists of N_c equations and a total of N_a unknowns, where

$$N_c = N(Q + Q_s + 2Q_1 + 2Q_2), \quad N_a = N_L + n = NM + n, \tag{16}$$

and $N_L = NM$ is the total number of trainable parameters in the neural network.

We seek a least squares solution to this system, and solve this system for $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ by the nonlinear least squares (NLLSQ) method [3, 16]. In our implementation we take advantage of the quality implementations of the nonlinear least squares method in the scientific libraries, specficially the "least_squares()" routine from the scipy.optimize package in Python for the current work. This library routine implements the Gauss-Newton method [3] together with a trust region algorithm [6, 8].

Since the nonlinear least squares method is a local optimization algorithm, it can be trapped to a local-minimum solution that is unacceptable. It is therefore crucial to combine the nonlinear least squares method with some perturbation strategy when solving the nonlinear least squares problem, in order to prevent the method from being trapped to the worst local-minimum solutions. In this paper we adopt the strategy for the initial guess perturbation and sub-iteration procedure developed in [16], with some modifications, and combine it with the nonlinear least squares method for solving the current system arising from the inverse PDE problem. We refer to the combined algorithm as the nonlinear least squares method with perturbations (NLLSQ-perturb). The NLLSQ-perturb algorithm is listed in the Appendix A of this paper (as Algorithm 7), which contains explanations of the various input parameters to the algorithm.

The NLLSQ-perturb algorithm (Algorithm 7) requires two routines, one for computing the residual vector and the other for computing the Jacobian matrix for an arbitrary given approximation to the solution. When the system (5) is enforced on the collocation points, the residual function in (6) is reduced to the vector,

$$\mathbf{R}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \begin{bmatrix} \mathbf{R}^{\text{pde}}(\boldsymbol{\alpha}, \boldsymbol{\beta}) \\ \mathbf{R}^{\text{bc}}(\boldsymbol{\beta}) \\ \mathbf{R}^{\text{mea}}(\boldsymbol{\beta}) \\ \mathbf{R}^{\text{ck}}(\boldsymbol{\beta}) \end{bmatrix}_{N_c \times 1}. \tag{17}$$

The vectors $\mathbf{R}^{\text{pde}}$, $\mathbf{R}^{\text{bc}}$, $\mathbf{R}^{\text{mea}}$ and $\mathbf{R}^{\text{ck}}$ in this expression are related to the left hand side (LHS) of the equations (12)–(15d) and their specific forms are provided in the equation (56) of Appendix B.

We therefore compute the residual vector $\mathbf{R}(\boldsymbol{\alpha}, \boldsymbol{\beta})$ as follows. Given arbitrary $(\boldsymbol{\alpha}, \boldsymbol{\beta})$, we compute $u_e(\mathbf{x}_p^e)$ $(\mathbf{x}_p^e \in \mathbb{X}_e)$ for $1 \leqslant e \leqslant N$, and their derivatives by (11). Then we compute the LHSs of the equations (12),

Algorithm 1: Computing the residual $\mathbf{R}(\boldsymbol{\alpha},\boldsymbol{\beta})$ for NLLSQ algorithm

```
input  : vector $\boldsymbol{\theta}=(\boldsymbol{\alpha},\boldsymbol{\beta})$; $\boldsymbol{\Phi}_e(x_p^e)$ and derivatives $(1\leqslant(e,p)\leqslant(N,Q))$; $\mathcal{M}\boldsymbol{\Phi}_e(\boldsymbol{\xi}_p^e)$ $(1\leqslant(e,p)\leqslant(N,Q_s))$.
output: residual vector $\mathbf{R}(\boldsymbol{\theta})$
1 if $\boldsymbol{\theta}=\boldsymbol{\theta}_s$ then
2     retrieve $u_e(\mathbf{x}_p^e)$ $(1\leqslant(e,p)\leqslant(N,Q))$ and their derivatives, and $\mathcal{M}u_e(\boldsymbol{\xi}_p^e)$ $(1\leqslant(e,p)\leqslant(N,Q_s))$
3 else
4     compute $u_e(\mathbf{x}_p^e)$ $(1\leqslant(e,p)\leqslant(N,Q))$, their derivatives (up to a necessary order), and $\mathcal{M}u_e(\boldsymbol{\xi}_p^e)$
        $(1\leqslant(e,p)\leqslant(N,Q_s))$ by (11)
5     set $\boldsymbol{\theta}_s=\boldsymbol{\theta}$, and save $u_e(\mathbf{x}_p^e)$ $(1\leqslant(e,p)\leqslant(N,Q))$, their derivatives, and $\mathcal{M}u_e(\boldsymbol{\xi}_p^e)$ $(1\leqslant(e,p)\leqslant(N,Q_s))$
6 end
7 compute $\mathbf{R}^{\rm pde}(\boldsymbol{\theta})$, $\mathbf{R}^{\rm bc}(\boldsymbol{\theta})$, $\mathbf{R}^{\rm mea}(\boldsymbol{\theta})$, $\mathbf{R}^{\rm ck}(\boldsymbol{\theta})$ by the LHSs of (12)–(15d), as given in (56) of Appendix B
8 form $\mathbf{R}(\boldsymbol{\theta})$ according to (17)
```

Algorithm 2: Computing the Jacobian matrix $\frac{\partial\mathbf{R}}{\partial(\boldsymbol{\alpha},\boldsymbol{\beta})}$ for NLLSQ algorithm

```
input  : vector $\boldsymbol{\theta}=(\boldsymbol{\alpha},\boldsymbol{\beta})$; $\boldsymbol{\Phi}_e(x_p^e)$ and derivatives $(1\leqslant(e,p)\leqslant(N,Q))$; $\mathcal{M}\boldsymbol{\Phi}_e(\boldsymbol{\xi}_p^e)$ $(1\leqslant(e,p)\leqslant(N,Q_s))$.
output: Jacobian matrix $\frac{\partial\mathbf{R}}{\partial\boldsymbol{\theta}}$
1 if $\boldsymbol{\theta}=\boldsymbol{\theta}_s$ then
2     retrieve $u_e(\mathbf{x}_p^e)$ $(1\leqslant(e,p)\leqslant(N,Q))$ and their derivatives, and $\mathcal{M}u_e(\boldsymbol{\xi}_p^e)$ $(1\leqslant(e,p)\leqslant(N,Q_s))$
3 else
4     compute $u_e(\mathbf{x}_p^e)$ $(1\leqslant(e,p)\leqslant(N,Q))$, their derivatives (up to a necessary order), and $\mathcal{M}u_e(\boldsymbol{\xi}_p^e)$
        $(1\leqslant(e,p)\leqslant(N,Q_s))$ by (11)
5     set $\boldsymbol{\theta}_s=\boldsymbol{\theta}$, and save $u_e(\mathbf{x}_p^e)$ $(1\leqslant(e,p)\leqslant(N,Q))$, their derivatives, and $\mathcal{M}u_e(\boldsymbol{\xi}_p^e)$ $(1\leqslant(e,p)\leqslant(N,Q_s))$
6 end
7 compute $\frac{\partial\mathbf{R}^{\rm pde}}{\partial\boldsymbol{\alpha}}$, $\frac{\partial\mathbf{R}^{\rm pde}}{\partial\boldsymbol{\beta}}$, $\frac{\partial\mathbf{R}^{\rm bc}}{\partial\boldsymbol{\beta}}$, $\frac{\partial\mathbf{R}^{\rm mea}}{\partial\boldsymbol{\beta}}$, $\frac{\partial\mathbf{R}^{\rm ck}}{\partial\boldsymbol{\beta}}$ by equations (57)–(61) of Appendix B
8 form $\frac{\partial\mathbf{R}}{\partial\boldsymbol{\theta}}=\frac{\partial\mathbf{R}}{\partial(\boldsymbol{\alpha},\boldsymbol{\beta})}$ by (18)
```

(13a)–(13d), (14), and (15a)–(15d), and assemble them to form the vectors $\mathbf{R}^{\rm pde}$, $\mathbf{R}^{\rm bc}$, $\mathbf{R}^{\rm mea}$ and $\mathbf{R}^{\rm ck}$ according to equation (56) of Appendix B. The residual vector $\mathbf{R}(\boldsymbol{\alpha},\boldsymbol{\beta})$ is finally assembled according to (17). The procedure for computing $\mathbf{R}(\boldsymbol{\alpha},\boldsymbol{\beta})$ is summarized in Algorithm 1.

Remark 2.3. *On line 4 of Algorithm 1, the "necessary order" refers to the order of all the derivative terms of u_e involved in the system consisting of (12)–(15d). For example, if $\frac{\partial^2 u_e}{\partial y^2}$ and $\frac{\partial u_e}{\partial x}$ are involved in this system, one would need to compute these derivatives based on $\frac{\partial^2\boldsymbol{\Phi}_e}{\partial y^2}$ and $\frac{\partial\boldsymbol{\Phi}_e}{\partial x}$ on line 4 of this algorithm.*

The Jacobian matrix is given by

$$\frac{\partial\mathbf{R}}{\partial(\boldsymbol{\alpha},\boldsymbol{\beta})}=\begin{bmatrix}\frac{\partial\mathbf{R}}{\partial\boldsymbol{\alpha}} & \frac{\partial\mathbf{R}}{\partial\boldsymbol{\beta}}\end{bmatrix}=\begin{bmatrix}\frac{\partial\mathbf{R}^{\rm pde}}{\partial\boldsymbol{\alpha}} & \frac{\partial\mathbf{R}^{\rm pde}}{\partial\boldsymbol{\beta}}\\ \mathbf{0} & \frac{\partial\mathbf{R}^{\rm bc}}{\partial\boldsymbol{\beta}}\\ \mathbf{0} & \frac{\partial\mathbf{R}^{\rm mea}}{\partial\boldsymbol{\beta}}\\ \mathbf{0} & \frac{\partial\mathbf{R}^{\rm ck}}{\partial\boldsymbol{\beta}}\end{bmatrix}_{N_c\times N_L}. \tag{18}$$

The specific forms for the matrices $\frac{\partial\mathbf{R}^{\rm pde}}{\partial\boldsymbol{\alpha}}$, $\frac{\partial\mathbf{R}^{\rm pde}}{\partial\boldsymbol{\beta}}$, $\frac{\partial\mathbf{R}^{\rm bc}}{\partial\boldsymbol{\beta}}$, $\frac{\partial\mathbf{R}^{\rm mea}}{\partial\boldsymbol{\beta}}$ and $\frac{\partial\mathbf{R}^{\rm ck}}{\partial\boldsymbol{\beta}}$ involved in the above expression are specified in the equations (57)–(61) of Appendix B.

Therefore the Jacobian matrix can be computed as follows. Given arbitrary $(\boldsymbol{\alpha},\boldsymbol{\beta})$, we compute $u_e(\mathbf{x}_p^e)$ $(1\leqslant(e,p)\leqslant(N,Q))$, their derivatives, and $\mathcal{M}u_e(\boldsymbol{\xi}_p^e)$ $(1\leqslant(e,p)\leqslant(N,Q_s))$ based on $\boldsymbol{\beta}$ and the pre-computed $\boldsymbol{\Phi}_e(\mathbf{x}_p^e)$, their derivatives, and the $\mathcal{M}\boldsymbol{\Phi}_e(\boldsymbol{\xi}_p^e)$ data. Then we compute the Jacobian and related matrices by the equations (18) and (57)–(61). Algorithm 2 summarizes the routine for computing the Jacobian matrix.

Remark 2.4. *In Algorithms 1 and 2 we have stored the data for u, its derivatives, and $\mathcal{M}u$ on the collocation/measurement points corresponding to the $\boldsymbol{\theta} = (\boldsymbol{\alpha}, \boldsymbol{\beta})$ value last computed (denoted by $\boldsymbol{\theta}_s$); see lines 1 to 6 in both algorithms. This saves computations, because in the nonlinear least squares iterations Algorithm 1 is typically invoked first to compute the residual corresponding to some $(\boldsymbol{\alpha}, \boldsymbol{\beta})$, and then Algorithm 2 is invoked to compute the Jacobian for the same $(\boldsymbol{\alpha}, \boldsymbol{\beta})$. Again please note that $\boldsymbol{\theta}_s$ in these two algorithms is used to save the last $\boldsymbol{\theta} = (\boldsymbol{\alpha}, \boldsymbol{\beta})$ value for which the data have been computed. $\boldsymbol{\theta}_s$ should be initialized to "None" at the beginning of the computation.*

Remark 2.5. *In this work the hidden-layer coefficients are assigned to uniform random values generated on the interval $[-R_m, R_m]$, where $R_m > 0$ is a constant. The R_m value influences the accuracy of the simulation results of inverse PDE problems, similar to what has been observed in forward problems (see [16, 19]). In this paper we compute a near-optimal R_m using the method from [19] based on the differential evolution algorithm, and employ this value (or a value nearby) in numerical simulations of inverse PDEs.*

Remark 2.6. *For noisy measurement data $S(\boldsymbol{\xi})$, we observe that scaling the residual vector associated with the measurement ($\mathbf{R}^{mea}$) by a constant factor can improve the accuracy of the results (more robust to noise). Let $\lambda_{mea} > 0$ denote a prescribed constant. We scale the equation (14) by λ_{mea},*

$$\lambda_{mea}\mathcal{M}u_e(\boldsymbol{\xi}_p^e) - \lambda_{mea}S(\boldsymbol{\xi}_p^e) = 0, \quad \textit{for } \boldsymbol{\xi}_p^e \in \mathbb{Y}_e, \ 1 \leqslant (e,p) \leqslant (N, Q_s). \tag{19}$$

Then in the presented method we replace equation (14) by the scaled equation (19), with corresponding changes to the computation of the residual vector and the Jacobian matrix. The scaling factor λ_{mea} will cause some change to the least squares solution to $(\boldsymbol{\alpha}, \boldsymbol{\beta})$. When the data $S(\boldsymbol{\xi})$ is noisy, numerical experiments indicate that employing a constant $0 < \lambda_{mea} < 1$ can in general improve the accuracy of the computed $\boldsymbol{\alpha}$ and $u(\mathbf{x})$ markedly, compared with the case without scaling (i.e. $\lambda_{mea} = 1$). Note that employing the scaled equation (19) is equivalent to using a scaled term $\frac{1}{2}\lambda_{mea}^2\|\mathbf{R}^{mea}\|^2$ in the underlying loss function for the nonlinear least squares method.

Remark 2.7. *The method developed here can be applied to inverse PDEs in which the inverse parameters may be an unknown field distribution. Consider for example,*

$$\gamma(\mathbf{x})\mathcal{L}(u) + \mathcal{F}(u) = f(\mathbf{x}), \tag{20}$$

where the coefficient $\gamma(\mathbf{x})$ is an unknown field and $u(\mathbf{x})$ is the unknown solution to the forward problem. In this case we can expand $\gamma(\mathbf{x})$ in terms of a set of basis functions and transform (20) into a form similar to (1a), in which the expansion coefficients of $\gamma(\mathbf{x})$ become the inverse parameters. Therefore the inverse problem can be computed using the method presented above. In this work we employ the same bases in the expansion for $u(\mathbf{x})$ (see (4)) and for $\gamma(\mathbf{x})$. This translates into two nodes in the output layer of the neural network architecture, one representing $u(\mathbf{x})$ and the other representing $\gamma(\mathbf{x})$. When more inverse coefficient fields are involved, one can correspondingly increase the number of nodes in the output layer of the neural network. We will present a numerical example for an inverse PDE similar to (20) in Section 3.

2.3 Variable Projection Algorithms for Network Training

This subsection outlines two algorithms for computing $(\boldsymbol{\alpha}, \boldsymbol{\beta})$, both based on the variable projection (VarPro) idea [23, 24, 18] but with different formulations. In the first formulation (VarPro-F1), the inverse parameters $(\boldsymbol{\alpha})$ are eliminated from the problem to attain a reduced problem about $\boldsymbol{\beta}$ only. The reduced problem is solved by the nonlinear least squares method first for $\boldsymbol{\beta}$, and then $\boldsymbol{\alpha}$ is computed by the linear least squares method. In the second formulation (VarPro-F2), the field solution (equivalently, the $\boldsymbol{\beta}$ parameters) is eliminated from the problem to attain a reduced problem about $\boldsymbol{\alpha}$ only. The reduced problem is solved first by the nonlinear least squares method for $\boldsymbol{\alpha}$, and then $\boldsymbol{\beta}$ is computed based on the $\boldsymbol{\alpha}$ already obtained. The problem settings and notations here follow those of Section 2.2.

2.3.1 Formulation #1 (VarPro-F1): Eliminating the Inverse Parameters

We start with the discrete system consisting of equations (12)–(15d). We re-arrange this system symbolically into a matrix equation about the parameters $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_n)^T$,

$$\mathbf{H}(\boldsymbol{\beta})\boldsymbol{\alpha} = \mathbf{b}(\boldsymbol{\beta}), \tag{21}$$

where

$$\begin{cases} \mathbf{H}(\boldsymbol{\beta}) = \begin{bmatrix} \mathbf{H}^{\text{pde}}(\boldsymbol{\beta}) \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}_{N_c\times n}, \mathbf{b}(\boldsymbol{\beta}) = \begin{bmatrix} \mathbf{b}^{\text{pde}}(\boldsymbol{\beta}) \\ -\mathbf{R}^{\text{bc}}(\boldsymbol{\beta}) \\ -\mathbf{R}^{\text{mea}}(\boldsymbol{\beta}) \\ -\mathbf{R}^{\text{ck}}(\boldsymbol{\beta}) \end{bmatrix}_{N_c\times 1}, \mathbf{H}^{\text{pde}}(\boldsymbol{\beta}) = \begin{bmatrix} \vdots & & \vdots \\ \mathcal{L}_1\left(u_e(\mathbf{x}_p^e)\right) & \cdots & \mathcal{L}_n\left(u_e(\mathbf{x}_p^e)\right) \\ \vdots & & \vdots \end{bmatrix}_{NQ\times n}, \\ \mathbf{b}^{\text{pde}}(\boldsymbol{\beta}) = \begin{bmatrix} b_{ep}^{\text{pde}} \end{bmatrix}_{NQ\times 1} = \begin{bmatrix} \vdots \\ f\left(\mathbf{x}_p^e\right) - \mathcal{F}\left(u_e(\mathbf{x}_p^e)\right) \\ \vdots \end{bmatrix}_{NQ\times 1}. \end{cases} \tag{22}$$

In these expressions, $\mathbf{R}^{\text{bc}}$, $\mathbf{R}^{\text{mea}}$ and $\mathbf{R}^{\text{ck}}$ are defined in (56) of Appendix B.

For any given $\boldsymbol{\beta}$, the least squares solution to (21) with the minimum norm is given by

$$\boldsymbol{\alpha} = \mathbf{H}^{+}(\boldsymbol{\beta})\mathbf{b}(\boldsymbol{\beta}), \tag{23}$$

where $\mathbf{H}^{+}(\boldsymbol{\beta})$ denotes the Moore-Penrose inverse of $\mathbf{H}(\boldsymbol{\beta})$. Substituting this expression into (21) gives rise to a reduced system about $\boldsymbol{\beta}$ only. The residual of this reduced system (see also (17)) is given by

$$\mathbf{r}(\boldsymbol{\beta}) = \mathbf{R}(\boldsymbol{\alpha},\boldsymbol{\beta}) = \mathbf{H}(\boldsymbol{\beta})\boldsymbol{\alpha} - \mathbf{b}(\boldsymbol{\beta}) = \mathbf{H}(\boldsymbol{\beta})\mathbf{H}^{+}(\boldsymbol{\beta})\mathbf{b}(\boldsymbol{\beta}) - \mathbf{b}(\boldsymbol{\beta}). \tag{24}$$

We determine the optimum $\boldsymbol{\beta}^*$ by minimizing the Euclidean norm of this residual,

$$\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta}} \frac{1}{2}\|\mathbf{r}(\boldsymbol{\beta})\|^2 = \arg\min_{\boldsymbol{\beta}} \frac{1}{2}\|\mathbf{H}(\boldsymbol{\beta})\mathbf{H}^{+}(\boldsymbol{\beta})\mathbf{b}(\boldsymbol{\beta}) - \mathbf{b}(\boldsymbol{\beta})\|^2 \tag{25}$$

where $\|\cdot\|$ denotes the Euclidean norm. With $\boldsymbol{\beta}$ determined by (25), we solve the system (21) for $\boldsymbol{\alpha}$ by the linear least squares method with the minimum-norm solution (or by directly using (23)).

Equation (25) represents a nonlinear least squares problem about $\boldsymbol{\beta}$. We solve this problem by the NLLSQ-perturb algorithm (Algorithm 7 in Appendix A). As noted previously, two routines are required for this algorithm, one for computing the reduced residual $\mathbf{r}(\boldsymbol{\beta})$ and the other for computing the Jacobian matrix of the reduced problem, $\frac{\partial \mathbf{r}}{\partial \boldsymbol{\beta}}$, for any given $\boldsymbol{\beta}$.

We compute the reduced residual as follows. For any given $\boldsymbol{\beta}$, we solve equation (21) for $\boldsymbol{\alpha}$ (with minimum norm) by the linear least squares method. Let $\boldsymbol{\alpha}^{LS}$ denote this solution. Then the residual is given by

$$\mathbf{r}(\boldsymbol{\beta}) = \mathbf{H}(\boldsymbol{\beta})\boldsymbol{\alpha}^{LS} - \mathbf{b}(\boldsymbol{\beta}). \tag{26}$$

Algorithm 3 summarizes the procedure for computing the reduced residual.

Algorithm 3: Computing reduced residual $\mathbf{r}(\boldsymbol{\beta})$ for VarPro-F1.

```
input  : $\boldsymbol{\beta}$; $\boldsymbol{\Phi}_e(x_p^e)$ and derivatives $(1 \leqslant (e,p) \leqslant (N,Q))$; $\mathcal{M}\boldsymbol{\Phi}_e(\boldsymbol{\xi}_p^e)$ $(1 \leqslant (e,p) \leqslant (N,Q_s))$.
output: reduced residual $\mathbf{r}(\boldsymbol{\beta})$
1  if $\boldsymbol{\beta} = \boldsymbol{\beta}_s$ then
2      retrieve $\mathbf{H}(\boldsymbol{\beta}_s)$, $\mathbf{b}(\boldsymbol{\beta}_s)$, $\boldsymbol{\alpha}^{LS}$
3      set $\mathbf{H}(\boldsymbol{\beta}) = \mathbf{H}(\boldsymbol{\beta}_s)$ and $\mathbf{b}(\boldsymbol{\beta}) = \mathbf{b}(\boldsymbol{\beta}_s)$
4  else
5      compute $u_e(\mathbf{x}_p^e)$ $(1 \leqslant (e,p) \leqslant (N,Q))$, their derivatives (up to a necessary order), and $\mathcal{M}u_e(\boldsymbol{\xi}_p^e)$
         $(1 \leqslant (e,p) \leqslant (N,Q_s))$ by (11)
6      compute $\mathbf{H}(\boldsymbol{\beta})$ and $\mathbf{b}(\boldsymbol{\beta})$ by (22) and (56) of Appendix B
7      solve equation (21) for $\boldsymbol{\alpha}$ by the linear least squares method, and let $\boldsymbol{\alpha}^{LS} = \boldsymbol{\alpha}$
8      set $\boldsymbol{\beta}_s = \boldsymbol{\beta}$, and save $\mathbf{H}(\boldsymbol{\beta})$, $\mathbf{b}(\boldsymbol{\beta})$, $\boldsymbol{\alpha}^{LS}$
9  end
10 compute $\mathbf{r}(\boldsymbol{\beta})$ by equation (26)
```

To compute the Jacobian of the reduced residual, we note the following formula owing to [23],

$$\begin{aligned}\frac{\partial}{\partial\boldsymbol{\theta}}\left[\mathbf{H}(\boldsymbol{\theta})\mathbf{H}^{+}(\boldsymbol{\theta})\right] &= \left[\mathbf{I}-\mathbf{H}(\boldsymbol{\theta})\mathbf{H}^{+}(\boldsymbol{\theta})\right]\frac{\partial\mathbf{H}}{\partial\boldsymbol{\theta}}\mathbf{H}^{+}(\boldsymbol{\theta}) + \left[\mathbf{H}^{T}(\boldsymbol{\theta})\right]^{+}\frac{\partial\mathbf{H}^{T}}{\partial\boldsymbol{\theta}}\left[\mathbf{I}-\mathbf{H}(\boldsymbol{\theta})\mathbf{H}^{+}(\boldsymbol{\theta})\right]\\ &\approx \left[\mathbf{I}-\mathbf{H}(\boldsymbol{\theta})\mathbf{H}^{+}(\boldsymbol{\theta})\right]\frac{\partial\mathbf{H}}{\partial\boldsymbol{\theta}}\mathbf{H}^{+}(\boldsymbol{\theta}),\end{aligned} \tag{27}$$

where $\mathbf{I}$ is the identity matrix and on the second line we have kept only the first term in the formula as an approximation to the LHS, thanks to the suggestion of [32]. In light of (24) and (27), we have

$$\begin{aligned}\frac{\partial\mathbf{r}}{\partial\boldsymbol{\beta}} &= \left(\frac{\partial}{\partial\boldsymbol{\beta}}\left[\mathbf{H}(\boldsymbol{\beta})\mathbf{H}^{+}(\boldsymbol{\beta})\right]\right)\mathbf{b}(\boldsymbol{\beta}) + \left[\mathbf{H}(\boldsymbol{\beta})\mathbf{H}^{+}(\boldsymbol{\beta})\right]\frac{\partial\mathbf{b}}{\partial\boldsymbol{\beta}} - \frac{\partial\mathbf{b}}{\partial\boldsymbol{\beta}}\\ &\approx \frac{\partial\mathbf{H}}{\partial\boldsymbol{\beta}}\mathbf{H}^{+}(\boldsymbol{\beta})\mathbf{b}(\boldsymbol{\beta}) - \mathbf{H}(\boldsymbol{\beta})\mathbf{H}^{+}(\boldsymbol{\beta})\frac{\partial\mathbf{H}}{\partial\boldsymbol{\beta}}\mathbf{H}^{+}(\boldsymbol{\beta})\mathbf{b}(\boldsymbol{\beta}) + \mathbf{H}(\boldsymbol{\beta})\mathbf{H}^{+}(\boldsymbol{\beta})\frac{\partial\mathbf{b}}{\partial\boldsymbol{\beta}} - \frac{\partial\mathbf{b}}{\partial\boldsymbol{\beta}}\\ &= \left(\frac{\partial\mathbf{H}}{\partial\boldsymbol{\beta}}\mathbf{H}^{+}(\boldsymbol{\beta})\mathbf{b}(\boldsymbol{\beta}) - \frac{\partial\mathbf{b}}{\partial\boldsymbol{\beta}}\right) - \mathbf{H}(\boldsymbol{\beta})\mathbf{H}^{+}(\boldsymbol{\beta})\left(\frac{\partial\mathbf{H}}{\partial\boldsymbol{\beta}}\mathbf{H}^{+}(\boldsymbol{\beta})\mathbf{b}(\boldsymbol{\beta}) - \frac{\partial\mathbf{b}}{\partial\boldsymbol{\beta}}\right)\\ &= \mathbf{J}_1(\boldsymbol{\beta}) - \mathbf{J}_2(\boldsymbol{\beta}),\end{aligned} \tag{28}$$

where

$$\mathbf{J}_1(\boldsymbol{\beta}) = \mathbf{J}_0(\boldsymbol{\beta}) - \frac{\partial\mathbf{b}}{\partial\boldsymbol{\beta}}, \quad \mathbf{J}_2(\boldsymbol{\beta}) = \mathbf{H}(\boldsymbol{\beta})\mathbf{H}^{+}(\boldsymbol{\beta})\mathbf{J}_1(\boldsymbol{\beta}), \quad \mathbf{J}_0(\boldsymbol{\beta}) = \frac{\partial\mathbf{H}}{\partial\boldsymbol{\beta}}\mathbf{H}^{+}(\boldsymbol{\beta})\mathbf{b}(\boldsymbol{\beta}). \tag{29}$$

Therefore, we need a procedure for computing $\mathbf{J}_0(\boldsymbol{\beta})$, $\frac{\partial\mathbf{b}}{\partial\boldsymbol{\beta}}$ and $\mathbf{J}_2(\boldsymbol{\beta})$. $\mathbf{J}_0(\boldsymbol{\beta})$ can be computed as follows,

$$\mathbf{J}_0(\boldsymbol{\beta}) = \frac{\partial\mathbf{H}}{\partial\boldsymbol{\beta}}\mathbf{H}^{+}(\boldsymbol{\beta})\mathbf{b}(\boldsymbol{\beta}) = \frac{\partial\mathbf{H}}{\partial\boldsymbol{\beta}}\boldsymbol{\alpha}^{LS} = \frac{\partial\left[\mathbf{H}(\boldsymbol{\beta})\boldsymbol{\alpha}^{LS}\right]}{\partial\boldsymbol{\beta}} = \begin{bmatrix}\frac{\partial\mathbf{R}^{\text{pdeI}}}{\partial\boldsymbol{\beta}}\\ \mathbf{0}\\ \mathbf{0}\\ \mathbf{0}\end{bmatrix}_{N_c\times NM}. \tag{30}$$

In this equation, $\boldsymbol{\alpha}^{LS} = (\alpha_1^{LS},\dots,\alpha_n^{LS})^T$ is the minimum-norm solution to (21) computed by the linear least squares method, and

$$\mathbf{R}^{\text{pdeI}}(\boldsymbol{\beta}) = \left[R_{ep}^{\text{pdeI}}\right]_{NQ\times1} = \begin{bmatrix}\vdots\\ \alpha_1^{LS}\mathcal{L}_1(u_e(\mathbf{x}_p^e)) + \cdots + \alpha_n^{LS}\mathcal{L}_n(u_e(\mathbf{x}_p^e))\\ \vdots\end{bmatrix}_{NQ\times1}, \tag{31a}$$

$$\frac{\partial\mathbf{R}^{\text{pdeI}}}{\partial\boldsymbol{\beta}} = \left[\frac{\partial R_{ep}^{\text{pdeI}}}{\partial\beta_{ij}}\right]_{NQ\times NM}. \tag{31b}$$

Algorithm 4: Computing Jacobian matrix $\frac{\partial \mathbf{r}}{\partial \boldsymbol{\beta}}$ for VarPro-F1.

```
input  : $\boldsymbol{\beta}$; $\boldsymbol{\Phi}_e(x_p^e)$ and derivatives $(1 \leqslant (e,p) \leqslant (N,Q))$; $\mathcal{M}\boldsymbol{\Phi}_e(\boldsymbol{\xi}_p^e)$ $(1 \leqslant (e,p) \leqslant (N,Q_s))$.
output: Jacobian matrix $\frac{\partial \mathbf{r}}{\partial \boldsymbol{\beta}}$
1  if $\boldsymbol{\beta} = \boldsymbol{\beta}_s$ then
2      retrieve $\mathbf{H}(\boldsymbol{\beta}_s)$, $\mathbf{b}(\boldsymbol{\beta}_s)$, $\boldsymbol{\alpha}^{LS}$
3      set $\mathbf{H}(\boldsymbol{\beta}) = \mathbf{H}(\boldsymbol{\beta}_s)$ and $\mathbf{b}(\boldsymbol{\beta}) = \mathbf{b}(\boldsymbol{\beta}_s)$
4  else
5      compute $u_e(\mathbf{x}_p^e)$ $(1 \leqslant (e,p) \leqslant (N,Q))$, their derivatives (up to a necessary order), and $\mathcal{M}u_e(\boldsymbol{\xi}_p^e)$
         $(1 \leqslant (e,p) \leqslant (N,Q_s))$ by (11)
6      compute $\mathbf{H}(\boldsymbol{\beta})$ and $\mathbf{b}(\boldsymbol{\beta})$ by (22) and (56) of Appendix B
7      solve equation (21) for $\boldsymbol{\alpha}$ by the linear least squares method, and let $\boldsymbol{\alpha}^{LS} = \boldsymbol{\alpha}$
8      set $\boldsymbol{\beta}_s = \boldsymbol{\beta}$, and save $\mathbf{H}(\boldsymbol{\beta})$, $\mathbf{b}(\boldsymbol{\beta})$, $\boldsymbol{\alpha}^{LS}$
9  end
10 compute $\mathbf{J}_0(\boldsymbol{\beta})$ by equations (30)–(32)
11 compute $\frac{\partial \mathbf{b}}{\partial \boldsymbol{\beta}}$ by (33), (34), (57), and (59)–(61) of Appendix B
12 compute $\mathbf{J}_1(\boldsymbol{\beta})$ by (29)
13 compute $\mathbf{J}_2(\boldsymbol{\beta})$ by (35)–(36)
14 compute $\frac{\partial \mathbf{r}}{\partial \boldsymbol{\beta}}$ by (28)
```

In the matrix $\frac{\partial \mathbf{R}^{\text{pdeI}}}{\partial \boldsymbol{\beta}}$ the only non-zero terms are,

$$\frac{\partial R_{ep}^{\text{pdeI}}}{\partial \beta_{ej}} = \alpha_1^{LS}\mathcal{L}_1'(u_e(\mathbf{x}_p^e))\phi_{ej}(\mathbf{x}_p^e) + \cdots + \alpha_n^{LS}\mathcal{L}_n'(u_e(\mathbf{x}_p^e))\phi_{ej}(\mathbf{x}_p^e), \quad \text{for } 1 \leqslant (e,p,j) \leqslant (N,Q,M). \tag{32}$$

It is important to note that, when computing $\frac{\partial \mathbf{R}^{\text{pdeI}}}{\partial \boldsymbol{\beta}}$, we treat $\boldsymbol{\alpha}^{LS}$ as a constant vector independent of $\boldsymbol{\beta}$.

$\frac{\partial \mathbf{b}}{\partial \boldsymbol{\beta}}$ is computed as follows,

$$\frac{\partial \mathbf{b}}{\partial \boldsymbol{\beta}} = \begin{bmatrix} \frac{\partial \mathbf{b}^{\text{pde}}}{\partial \boldsymbol{\beta}} \\ -\frac{\partial \mathbf{R}^{\text{bc}}}{\partial \boldsymbol{\beta}} \\ -\frac{\partial \mathbf{R}^{\text{mea}}}{\partial \boldsymbol{\beta}} \\ -\frac{\partial \mathbf{R}^{\text{ck}}}{\partial \boldsymbol{\beta}} \end{bmatrix}, \quad \frac{\partial \mathbf{b}^{\text{pde}}}{\partial \boldsymbol{\beta}} = \left[\frac{\partial b_{ep}^{\text{pde}}}{\partial \beta_{ij}}\right]_{NQ\times NM}, \tag{33}$$

where $\frac{\partial \mathbf{R}^{\text{bc}}}{\partial \boldsymbol{\beta}}$, $\frac{\partial \mathbf{R}^{\text{mea}}}{\partial \boldsymbol{\beta}}$ and $\frac{\partial \mathbf{R}^{\text{ck}}}{\partial \boldsymbol{\beta}}$ are given in (57) and (59)–(61) of Appendix B. The only non-zero terms in $\frac{\partial \mathbf{b}^{\text{pde}}}{\partial \boldsymbol{\beta}}$ are,

$$\frac{\partial b_{ep}^{\text{pde}}}{\partial \beta_{ej}} = -\mathcal{F}'\left(u_e(\mathbf{x}_p^e)\right)\phi_{ej}(\mathbf{x}_p^e), \quad \text{for } 1 \leqslant (e,p,j) \leqslant (N,Q,M). \tag{34}$$

With $J_0(\boldsymbol{\beta})$ and $\frac{\partial \mathbf{b}}{\partial \boldsymbol{\beta}}$ determined, we can compute $\mathbf{J}_1(\boldsymbol{\beta})$ by (29).

In light of (29), we compute $\mathbf{J}_2(\boldsymbol{\beta})$ by the following equations,

$$\mathbf{H}(\boldsymbol{\beta})\mathbf{K} = \mathbf{J}_1(\boldsymbol{\beta}), \tag{35}$$

$$\mathbf{J}_2(\boldsymbol{\beta}) = \mathbf{H}(\boldsymbol{\beta})\mathbf{K}. \tag{36}$$

We first solve equation (35) for the $n \times NM$ matrix $\mathbf{K}$ by the linear least squares method, and then compute $\mathbf{J}_2(\boldsymbol{\beta})$ by equation (36) with a matrix multiplication.

Therefore, given an arbitrary $\boldsymbol{\beta}$, we compute $\mathbf{J}_0(\boldsymbol{\beta})$ by (30)–(32), $\frac{\partial \mathbf{b}}{\partial \boldsymbol{\beta}}$ by (33) and (34), and $\mathbf{J}_1(\boldsymbol{\beta})$ by (29). Then we compute $\mathbf{J}_2(\boldsymbol{\beta})$ by (35)–(36). The (approximate) Jacobian matrix of the reduced problem is then given by (28). The procedure for computing the Jacobian matrix is summarized in the Algorithm 4. In

Algorithms 3 and 4, it should be noted that $\boldsymbol{\beta}_s$ denotes the last $\boldsymbol{\beta}$ value for which the data $\mathbf{H}(\boldsymbol{\beta})$, $\mathbf{b}(\boldsymbol{\beta})$ and $\boldsymbol{\alpha}^{LS}$ have been computed. $\boldsymbol{\beta}_s$ should be initialized to "None" at the beginning of the computation.

The overall VarPro-F1 algorithm for solving the inverse problem consists of two steps: (i) Invoke the NLLSQ-perturb algorithm (Algorithm 7 in Appendix A) to compute $\boldsymbol{\beta}$ from the reduced problem (25), with the routines given in Algorithms 3 and 4 as input. (ii) Solve (21) for $\boldsymbol{\alpha}$ by the linear least squares method.

Remark 2.8. *In the VarPro-F1 algorithm, one only needs to solve linear systems by the linear least squares method. The Moore-Penrose inverse of the coefficient matrix is not explicitly computed. In our implementation we employ the linear least squares routine scipy.linalg.lstsq() from the scipy package in Python, which in turn uses the linear least squares implementation in the LAPACK library.*

2.3.2 Formulation #2 (VarPro-F2): Eliminating the Field Function

We next present an alternative formulation (VarPro-F2) of variable projection, which is reciprocal to the VarPro-F1 algorithm of Section 2.3.1. In this formulation, we eliminate the field function u (or the parameters $\boldsymbol{\beta}$) from the problem to attain a reduced problem about $\boldsymbol{\alpha}$ only. We then solve the reduced problem first for $\boldsymbol{\alpha}$, and compute the parameters $\boldsymbol{\beta}$ afterwards.

This formulation applies to cases in which the operators $\mathcal{L}_i$ ($1 \leqslant i \leqslant n$) and $\mathcal{F}$ are all linear with respect to u. We first present the algorithm with regard to this case below. Then we outline an extension in a remark (Remark 2.9) by combining this algorithm with a Newton iteration to deal with cases in which these operators are nonlinear with respect to u.

Let us now assume that $\mathcal{L}_i$ ($1 \leqslant i \leqslant n$) and $\mathcal{F}$ are all linear operators, and we again start with the discrete system consisting of the equations (12)–(15d). We re-arrange this system into a matrix equation about the trainable network parameters $\boldsymbol{\beta} = (\boldsymbol{\beta}_1^T, \dots, \boldsymbol{\beta}_N^T)^T = (\beta_{11}, \dots, \beta_{NM})^T$,

$$\mathbf{H}(\boldsymbol{\alpha})\boldsymbol{\beta} = \mathbf{b}, \tag{37}$$

where

$$\mathbf{H}(\boldsymbol{\alpha}) = \begin{bmatrix} \mathbf{H}^{\text{pde}}(\boldsymbol{\alpha}) \\ \mathbf{H}^{\text{bc}} \\ \mathbf{H}^{\text{mea}} \\ \mathbf{H}^{\text{ck}} \end{bmatrix}_{N_c \times NM}, \quad \mathbf{b} = \begin{bmatrix} \mathbf{b}^{\text{pde}} \\ \mathbf{b}^{\text{bc}} \\ \mathbf{b}^{\text{mea}} \\ \mathbf{0} \end{bmatrix}_{N_c \times 1}, \quad \mathbf{H}^{\text{bc}} = \begin{bmatrix} \mathbf{H}^{\text{bc1}} \\ \mathbf{H}^{\text{bc2}} \\ \mathbf{H}^{\text{bc3}} \\ \mathbf{H}^{\text{bc4}} \end{bmatrix}, \quad \mathbf{H}^{\text{ck}} = \begin{bmatrix} \mathbf{H}^{\text{ck1}} \\ \mathbf{H}^{\text{ck2}} \\ \mathbf{H}^{\text{ck3}} \\ \mathbf{H}^{\text{ck4}} \end{bmatrix}, \quad \mathbf{b}^{\text{bc}} = \begin{bmatrix} \mathbf{b}^{\text{bc1}} \\ \mathbf{b}^{\text{bc2}} \\ \mathbf{b}^{\text{bc3}} \\ \mathbf{b}^{\text{bc4}} \end{bmatrix}, \tag{38}$$

and the specific forms for these matrices are provided in the equations (62)–(65) of Appendix B.

For any given $\boldsymbol{\alpha}$ the least squares solution (with minimum norm) to the system (37) is,

$$\boldsymbol{\beta} = \mathbf{H}^+(\boldsymbol{\alpha})\mathbf{b}. \tag{39}$$

Substitution of this expression into (37) results in a reduced system about $\boldsymbol{\alpha}$ only, with a residual given by

$$\mathbf{r}(\boldsymbol{\alpha}) = \mathbf{H}(\boldsymbol{\alpha})\mathbf{H}^+(\boldsymbol{\alpha})\mathbf{b} - \mathbf{b}. \tag{40}$$

We determine the optimum $\boldsymbol{\alpha}^*$ by minimizing the Euclidean norm of this residual,

$$\boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha}} \frac{1}{2}\|\mathbf{r}(\boldsymbol{\alpha})\|^2 = \arg\min_{\boldsymbol{\alpha}} \frac{1}{2}\|\mathbf{H}(\boldsymbol{\alpha})\mathbf{H}^+(\boldsymbol{\alpha})\mathbf{b} - \mathbf{b}\|^2. \tag{41}$$

After $\boldsymbol{\alpha}$ is obtained, we compute $\boldsymbol{\beta}$ by solving the system (37) with the linear least squares method.

The problem (41) is a nonlinear least squares problem about $\boldsymbol{\alpha}$. We employ the NLLSQ-perturb algorithm (Algorithm 7) to solve this problem. In light of (27), we can obtain the Jacobian matrix for this problem,

$$\frac{\partial \mathbf{r}}{\partial \boldsymbol{\alpha}} \approx \mathbf{J}_0(\boldsymbol{\alpha}) - \mathbf{J}_1(\boldsymbol{\alpha}), \quad \mathbf{J}_0(\boldsymbol{\alpha}) = \frac{\partial \mathbf{H}}{\partial \boldsymbol{\alpha}}\mathbf{H}^+(\boldsymbol{\alpha})\mathbf{b}, \quad \mathbf{J}_1(\boldsymbol{\alpha}) = \mathbf{H}(\boldsymbol{\alpha})\mathbf{H}^+(\boldsymbol{\alpha})\mathbf{J}_0(\boldsymbol{\alpha}). \tag{42}$$

Algorithm 5: Computing reduced residual $\mathbf{r}(\boldsymbol{\alpha})$ for VarPro-F2.

```
input  : $\boldsymbol{\alpha}$; $\boldsymbol{\Phi}_e(x_p^e)$ and derivatives $(1 \leqslant (e,p) \leqslant (N,Q))$; $\mathcal{M}\boldsymbol{\Phi}_e(\boldsymbol{\xi}_p^e)$ $(1 \leqslant (e,p) \leqslant (N,Q_s))$.
output: reduced residual $\mathbf{r}(\boldsymbol{\alpha})$
1 if $\boldsymbol{\alpha} = \boldsymbol{\alpha}_s$ then
2     retrieve $\mathbf{H}(\boldsymbol{\alpha}_s)$, $\mathbf{b}$, $\boldsymbol{\beta}^{LS}$
3     set $\mathbf{H}(\boldsymbol{\alpha}) = \mathbf{H}(\boldsymbol{\alpha}_s)$
4 else
5     compute $\mathbf{H}(\boldsymbol{\alpha})$ and $\mathbf{b}$ by (38) and the equations (62)–(65) in Appendix B
6     solve equation (37) for $\boldsymbol{\beta}$ by the linear least squares method, and let $\boldsymbol{\beta}^{LS} = \boldsymbol{\beta}$
7     set $\boldsymbol{\alpha}_s = \boldsymbol{\alpha}$, and save $\mathbf{H}(\boldsymbol{\alpha})$, $\mathbf{b}$, and $\boldsymbol{\beta}^{LS}$
8 end
9 compute $\mathbf{r}(\boldsymbol{\alpha})$ by $\mathbf{r}(\boldsymbol{\alpha}) = \mathbf{H}(\boldsymbol{\alpha})\boldsymbol{\beta}^{LS} - \mathbf{b}$
```

$\mathbf{J}_0(\boldsymbol{\alpha})$ can be computed as follows. For any given $\boldsymbol{\alpha}$, let $\boldsymbol{\beta}^{LS} = \mathbf{H}^{+}(\boldsymbol{\alpha})\mathbf{b} = ((\boldsymbol{\beta}_1^{LS})^T, \dots, (\boldsymbol{\beta}_N^{LS})^T)^T$ denote a *constant* vector. Then

$$\begin{cases} \mathbf{J}_0(\boldsymbol{\alpha}) = \dfrac{\partial\left[\mathbf{H}(\boldsymbol{\alpha})\boldsymbol{\beta}^{LS}\right]}{\partial\boldsymbol{\alpha}} = \dfrac{\partial}{\partial\boldsymbol{\alpha}}\begin{bmatrix}\mathbf{H}^{\mathrm{pde}}(\boldsymbol{\alpha})\boldsymbol{\beta}^{LS}\\ \mathbf{H}^{\mathrm{bc}}\boldsymbol{\beta}^{LS}\\ \mathbf{H}^{\mathrm{mea}}\boldsymbol{\beta}^{LS}\\ \mathbf{H}^{\mathrm{ck}}\boldsymbol{\beta}^{LS}\end{bmatrix}_{N_c\times 1} = \begin{bmatrix}\frac{\partial\left[\mathbf{H}^{\mathrm{pde}}(\boldsymbol{\alpha})\boldsymbol{\beta}^{LS}\right]}{\partial\boldsymbol{\alpha}}\\ \mathbf{0}\\ \mathbf{0}\\ \mathbf{0}\end{bmatrix}_{N_c\times n}, \\ \dfrac{\partial}{\partial\boldsymbol{\alpha}}\left[\mathbf{H}^{\mathrm{pde}}(\boldsymbol{\alpha})\boldsymbol{\beta}^{LS}\right] = \dfrac{\partial}{\partial\boldsymbol{\alpha}}\begin{bmatrix}\vdots\\ \alpha_1\mathcal{L}_1 u_e^{LS}(\mathbf{x}_p^e) + \cdots + \alpha_n\mathcal{L}_n u_e^{LS}(\mathbf{x}_p^e) + \mathcal{F}u_e^{LS}(\mathbf{x}_p^e)\\ \vdots\end{bmatrix}_{NQ\times 1} \\ \qquad = \begin{bmatrix}\vdots & & \vdots\\ \mathcal{L}_1 u_e^{LS}(\mathbf{x}_p^e) & \cdots & \mathcal{L}_n u_e^{LS}(\mathbf{x}_p^e)\\ \vdots & & \vdots\end{bmatrix}_{NQ\times n}, \end{cases} \tag{43}$$

where $u_e^{LS}(\mathbf{x}) = \boldsymbol{\Phi}_e(\mathbf{x})\boldsymbol{\beta}_e^{LS}$ for $1 \leqslant e \leqslant N$. We compute $\mathbf{J}_1(\boldsymbol{\alpha})$ by the following two equations,

$$\mathbf{H}(\boldsymbol{\alpha})\mathbf{K} = \mathbf{J}_0(\boldsymbol{\alpha}) \tag{44a}$$

$$\mathbf{J}_1(\boldsymbol{\alpha}) = \mathbf{H}(\boldsymbol{\alpha})\mathbf{K}. \tag{44b}$$

We first solve (44a) for the $n \times n$ matrix $\mathbf{K}$ by the linear least squares method, and then compute $\mathbf{J}_1(\boldsymbol{\alpha})$ by (44b) with a matrix multiplication.

The procedures for computing the residual $\mathbf{r}(\boldsymbol{\alpha})$ and the Jacobian matrix $\frac{\partial\mathbf{r}}{\partial\boldsymbol{\alpha}}$ for the reduced problem (41) are summarized in the Algorithms 5 and 6. In these two algorithms, it should be noted that $\boldsymbol{\alpha}_s$ denotes the last $\boldsymbol{\alpha}$ value for which the data $\mathbf{H}(\boldsymbol{\alpha})$, $\mathbf{b}$ and $\boldsymbol{\beta}^{LS}$ have been computed. $\boldsymbol{\alpha}_s$ should be initialized to "None" at the beginning of the computation.

The overall VarPro-F2 algorithm consists of two steps: (i) Invoke the NLLSQ-perturb algorithm (Algorithm 7 in Appendix A) to solve the problem (41) for $\boldsymbol{\alpha}$, with the routines in Algorithms 5 and 6 as input arguments. (ii) Solve equation (37) for $\boldsymbol{\beta}$ by the linear least squares method.

Remark 2.9. *Let us now discuss an extension of the above algorithm to deal with the case in which some (or all) of the operators of $\mathcal{L}_i$ $(1 \leqslant i \leqslant n)$ and $\mathcal{F}$ are nonlinear with respect to u. In this case, we can first use a Newton iteration to linearize the nonlinear operators, and then solve the linearized system by the VarPro-F2 algorithm as discussed above. Upon convergence of the Newton iteration, the solution for $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ to the original system will be attained. To make the discussion more concrete and without loss of generality,*

Algorithm 6: Computing Jacobian matrix $\frac{\partial \mathbf{r}}{\partial \boldsymbol{\alpha}}$ for VarPro-F2.

```
input  : $\boldsymbol{\alpha}$; $\boldsymbol{\Phi}_e(x_p^e)$ and derivatives $(1 \leqslant (e,p) \leqslant (N,Q))$; $\mathcal{M}\boldsymbol{\Phi}_e(\boldsymbol{\xi}_p^e)$ $(1 \leqslant (e,p) \leqslant (N,Q_s))$.
output: Jacobian matrix $\frac{\partial \mathbf{r}}{\partial \boldsymbol{\alpha}}$
1  if $\boldsymbol{\alpha} = \boldsymbol{\alpha}_s$ then
2      retrieve $\mathbf{H}(\boldsymbol{\alpha}_s)$, $\mathbf{b}$, $\boldsymbol{\beta}^{LS}$
3      set $\mathbf{H}(\boldsymbol{\alpha}) = \mathbf{H}(\boldsymbol{\alpha}_s)$
4  else
5      compute $\mathbf{H}(\boldsymbol{\alpha})$ and $\mathbf{b}$ by (38) and the equations (62)–(65) in Appendix B
6      solve equation (37) for $\boldsymbol{\beta}$ by the linear least squares method, and let $\boldsymbol{\beta}^{LS} = \boldsymbol{\beta}$
7      set $\boldsymbol{\alpha}_s = \boldsymbol{\alpha}$, and save $\mathbf{H}(\boldsymbol{\alpha})$, $\mathbf{b}$, and $\boldsymbol{\beta}^{LS}$
8  end
9  compute $u_e^{LS}(\mathbf{x}_p^e)$ $(1 \leqslant (e,p) \leqslant (N,Q))$ and their derivatives by (11) based on $\boldsymbol{\beta}^{LS}$
10 compute $\mathbf{J}_0(\boldsymbol{\alpha})$ by (43)
11 compute $\mathbf{J}_1(\boldsymbol{\alpha})$ by (44a)–(44b)
12 compute $\frac{\partial \mathbf{r}}{\partial \boldsymbol{\alpha}}$ by (42)
```

let us assume that $\mathcal{L}_1$ and $\mathcal{F}$ are nonlinear while the other operators are linear. Let $u_e^k(\mathbf{x})$ $(1 \leqslant e \leqslant N)$ denote the approximation of $u_e(\mathbf{x})$ at the k-th Newton step. Equation (12) *is nonlinear with respect to u, and its linearized form is given by,*

$$\begin{aligned}
&\alpha_1\mathcal{L}_1'(u_e^k(\mathbf{x}_p^e))u_e^{k+1}(\mathbf{x}_p^e) + \alpha_2\mathcal{L}_2u_e^{k+1}(\mathbf{x}_p^e) + \cdots + \alpha_n\mathcal{L}_nu_e^{k+1}(\mathbf{x}_p^e) + \mathcal{F}'(u_e^k(\mathbf{x}_p^e))u_e^{k+1}(\mathbf{x}_p^e) \\
&\quad - \Big[f(\mathbf{x}_p^e) - \alpha_1\mathcal{L}_1(u_e^k(\mathbf{x}_p^e)) + \alpha_1\mathcal{L}_1'(u_e^k(\mathbf{x}_p^e))u_e^k(\mathbf{x}_p^e) - \mathcal{F}(u_e^k(\boldsymbol{x}_p^e)) + \mathcal{F}'(u_e^k(\mathbf{x}_p^e))u_e^k(\mathbf{x}_p^e)\Big] = 0, \\
&\quad \text{for } 1 \leqslant (e,p) \leqslant (N,Q).
\end{aligned} \tag{45}$$

Notice that this equation is linear with respect to u_e^{k+1}. The equations (13)–(15) *are linear with respect to u_e, and we enforce them on the* $(k+1)$*-th Newton step (i.e. replacing u_e by u_e^{k+1} in these equations). The system consisting of* (45) *and the equations* (13)–(15) *(written in terms of u_e^{k+1}) are linear with respect to the updated approximation field u_e^{k+1}. With the expansion $u_e^{k+1}(\mathbf{x}) = \boldsymbol{\Phi}_e(\mathbf{x})\boldsymbol{\beta}_e^{k+1}$, we can solve this system for $(\boldsymbol{\alpha}, \boldsymbol{\beta}^{k+1})$ by the VarPro-F2 algorithm as discussed above. Upon convergence of the Newton iteration, the solution to $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ is given by the converged result, and the neural network coefficients contains the representation for the field solution $u(\mathbf{x})$ to the original nonlinear system. For inverse nonlinear PDEs with respect to u, the combination of the Newton iteration and the VarPro-F2 algorithm in general works quite well. We have also observed from numerical experiments that for certain problems it appears to be somewhat less robust than the VarPro-F1 and NLLSQ methods, leading to less accurate results than VarPro-F1 and NLLSQ.*

3 Numerical Examples

In this section we test the presented method and algorithms using several inverse PDE problems in two dimensions (2D) or in one spatial dimension (1D) plus time. The Gaussian activation function, $\sigma(x) = e^{-x^2}$, is employed in all the neural networks. We fix the seed value at 25 in the random number generator for all the test problems, so that the reported results here are exactly reproducible. Note that λ_{mea} denotes the scaling coefficient for the measurement residual (see Remark 2.6), with $\lambda_{mea} = 1$ corresponding to the case of no scaling. The network training time reported in the following subsections includes the preprocessing time (generation of the input collocation points, the random measurement points and the measurement data with noise) and the actual computation time with the nonlinear least squares iterations or the variable-projection iterations. It does not include the time for evaluating the neural network to generate the solution data, after

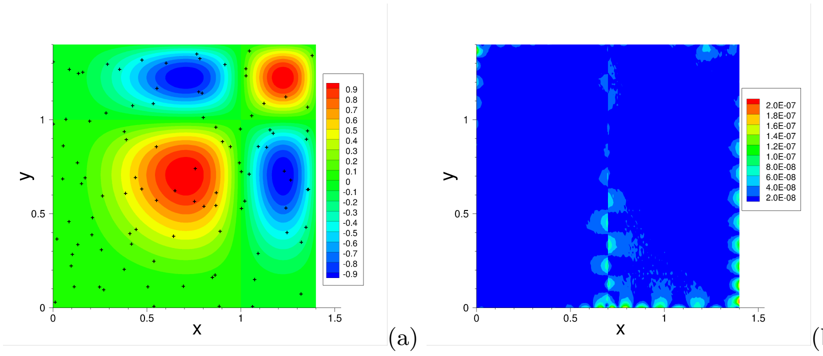


Figure 3: Inverse Poisson problem: distributions of (a) the NLLSQ solution and (b) its point-wise absolute error, with the random measurement points shown in (a) as "+" symbols. Two uniform sub-domains (along x), local NN $[2, 400, 1]$, $Q = 25 \times 25$, $Q_s = 50$, $R_m = 2.0$, λ_{mea}=1, ϵ=0 (no noise in measurement data).

the neural network is trained, for comparison with the exact solution to compute the errors. We refer the reader to the Appendix C and Appendix D for additional numerical tests of the current algorithms, and Appendix E for a comparison between the current method and the PINN method.

3.1 Parametric Poisson Equation

Consider the domain $(x, y) \in \Omega = [0, 1.4] \times [0, 1.4]$, and the inverse problem,

$$\frac{\partial^2 u}{\partial x^2} + \alpha \frac{\partial^2 u}{\partial y^2} = f(x, y), \tag{46a}$$

$$u(0, y) = g_1(y), \quad u(1.4, y) = g_2(y), \quad u(x, 0) = g_3(x), \quad u(x, 1.4) = g_4(x), \tag{46b}$$

$$u(\xi_i, \eta_i) = S(\xi_i, \eta_i), \quad (\xi_i, \eta_i) \in \mathbb{Y}, \quad 1 \leqslant i \leqslant NQ_s, \tag{46c}$$

where f and g_i $(1 \leqslant i \leqslant 4)$ denote a source term and the boundary data respectively, $\mathbb{Y} \subset \Omega$ denotes the set of random measurement points, α and $u(x, y)$ are the unknowns to be solved for, N denotes the number of sub-domains, and Q_s is the number of measurement points per sub-domain. We use the following manufactured solution to this problem,

$$\alpha_{ex} = 1, \quad u_{ex}(x, y) = \sin(\pi x^2) \sin(\pi y^2). \tag{47}$$

The source term and the boundary data are chosen such that the expressions in (47) satisfy (46a)–(46b). The measurement data are taken to be

$$S(\xi_i, \eta_i) = u_{ex}(\xi_i, \eta_i)(1 + \epsilon \zeta_i), \quad 1 \leqslant i \leqslant NQ_s, \tag{48}$$

where ζ_i denotes a uniform random number from $[-1, 1]$ representing the noise and the constant $\epsilon \geqslant 0$ denotes the relative level of the noise.

Henceforth Q denotes the number of uniform collocation points per sub-domain, Q_s denotes the number of random measurement points per sub-domain, ϵ denotes the noise level, and M denotes the number of trainable parameters of each local NN. R_m denotes a constant, and the hidden-layer coefficients are assigned to uniform random values generated on $[-R_m, R_m]$. The R_m values employed in the tests are obtained by the method from [19], as noted in Remark 2.5. After the NN is trained, it is evaluated on another set of $Q_{eval} = 101 \times 101$ uniform grid points (evaluation points) on each sub-domain to obtain u, which is compared

Q	α (NLLSQ)	α (VarPro-F1)	α (VarPro-F2)
5×5	1.076466245043E+0	9.982719409724E-1	0.000000000000E+0
10×10	9.999867935849E-1	9.999965494049E-1	-3.188390321381E-5
15×15	1.000000029498E+0	9.999999954822E-1	9.999999998978E-1
20×20	9.999999999701E-1	9.999999999592E-1	9.999999999536E-1
25×25	9.999999987249E-1	1.000000000817E+0	1.000000001279E+0
30×30	1.000000002811E+0	1.000000000906E+0	1.000000000002E+0
35×35	1.000000001708E+0	1.000000000670E+0	1.000000000237E+0
40×40	1.000000001552E+0	1.000000000717E+0	1.000000000183E+0

Table 1: Inverse Poisson problem: computed α by the NLLSQ, VarPro-F1 and VarPro-F2 algorithms versus Q (number of collocation points). Single sub-domain, NN $[2, 600, 1]$, $Q_s = 100$, λ_{mea}=1, ϵ=0; $R_m = 3.0$ with NLLSQ, $R_m = 2.8$ with VarPro-F1, and $R_m = 2.0$ with VarPro-F2.

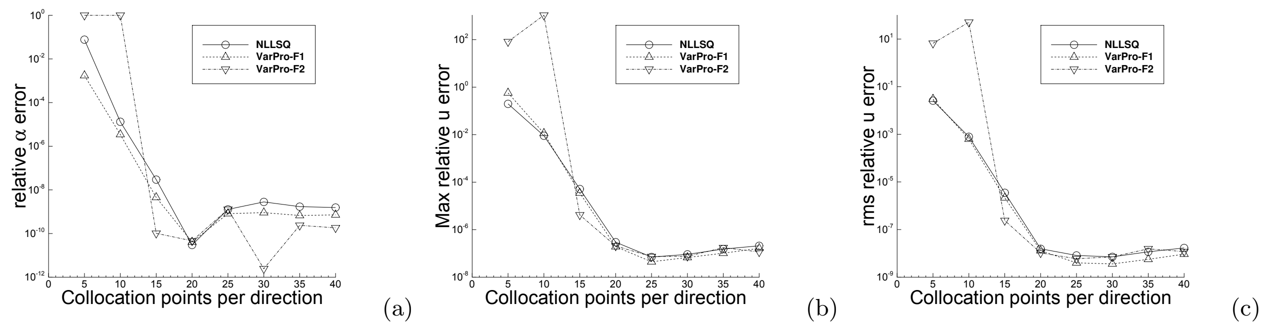


Figure 4: Inverse Poisson problem: relative errors of α and u versus Q_1 ($Q = Q_1 \times Q_1$) computed by the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. Single sub-domain, NN $[2, 600, 1]$, $Q_s = 100$, λ_{mea}=1, ϵ=0; $R_m = 3.0$ with NLLSQ, $R_m = 2.8$ with VarPro-F1, and $R_m = 2.0$ with VarPro-F2.

with (47) to compute the errors. The relative errors of α (e_α) and u (l^∞ and l^2 norms) are defined as,

$$e_\alpha = \frac{|\alpha - \alpha_{ex}|}{|\alpha_{ex}|}, \ l^\infty\text{-u} = \frac{\max\left\{|u(\mathbf{x}_i) - u_{ex}(\mathbf{x}_i)|\right\}_{i=1}^{NQ_{eval}}}{\sqrt{\frac{1}{NQ_{eval}}\sum_{i=1}^{NQ_{eval}} u_{ex}^2(\mathbf{x}_i)}}, \ l^2\text{-u} = \frac{\sqrt{\frac{1}{NQ_{eval}}\sum_{i=1}^{NQ_{eval}}[u(\mathbf{x}_i) - u_{ex}(\mathbf{x}_i)]^2}}{\sqrt{\frac{1}{NQ_{eval}}\sum_{i=1}^{NQ_{eval}} u_{ex}^2(\mathbf{x}_i)}}, \quad (49)$$

where N is the number of sub-domains and $\mathbf{x}_i$ denotes the evaluation points.

Figure 3 illustrates $u(x, y)$ and its point-wise absolute error obtained by the NLLSQ algorithm with 2 sub-domains. The caption lists the main simulation parameters. In particular, the random measurement points (100 total) are shown in Figure 3(a), and there is no noise in the measurement data. The NLLSQ solution for u is quite accurate, with a maximum error on the order of 10^{-7} in the domain. The relative (or absolute) error of the computed α is 9.03×10^{-9}.

The convergence of the computation results with respect to Q (number of collocation points) is illustrated by Table 1 and Figure 4. Table 1 lists the computed α values versus Q by the NLLSQ, VarPro-F1 and VarPro-F2 methods. Figure 4 shows the relative errors of α and u with respect to Q_1 (where $Q = Q_1 \times Q_1$) from the three methods. The main parameter values for these tests are provided in the table and figure captions. The α and the u errors decrease approximately exponentially with increasing Q, until Q reaches a certain level. The errors generally stagnate as Q further increases beyond that point. It is observed that the convergence behavior of VarPro-F2 is not as regular as those of NLLSQ and VarPro-F1.

The convergence of the NLLSQ, VarPro-F1 and VarPro-F2 algorithms with respect to the number of trainable parameters M is illustrated by Figure 5. A single sub-domain and a single hidden layer in the neural network are employed in the simulations, where the number of hidden nodes (M) is varied. The figure

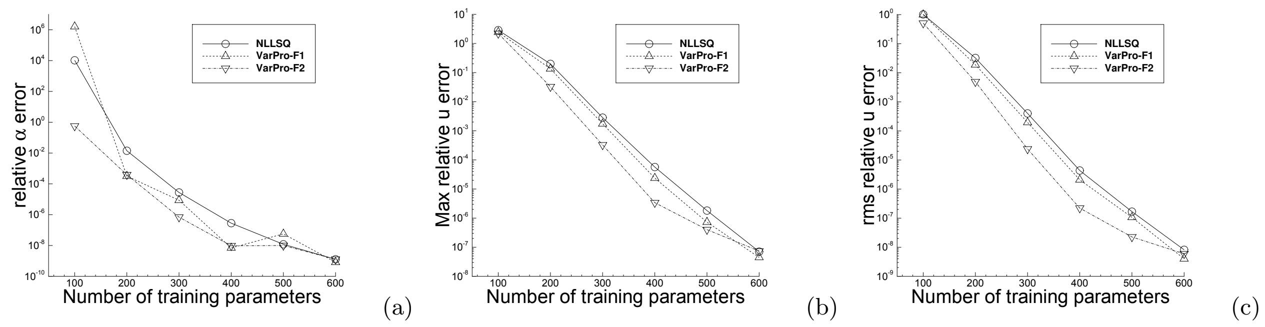


Figure 5: Inverse Poisson problem: α and u relative errors versus M (number of training parameters) obtained by the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. Single sub-domain, NN $[2, M, 1]$, $Q = 25\times25$, $Q_s = 100$, λ_{mea}=1, ϵ=0; $R_m = 3.0$ with NLLSQ, $R_m = 2.8$ with VarPro-F1, and $R_m = 2.0$ with VarPro-F2.

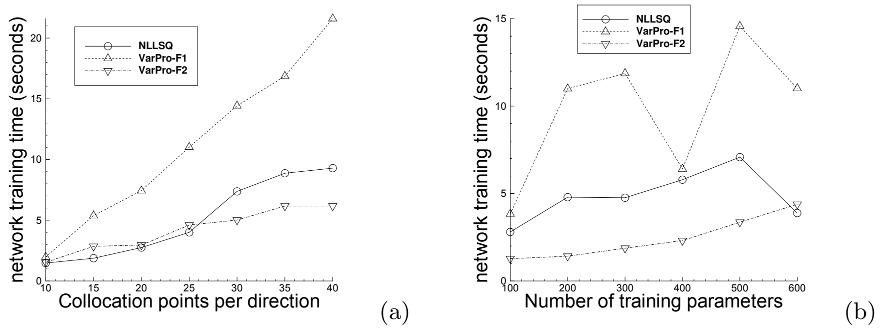


Figure 6: Inverse Poisson problem: Network training time as a function of (a) the number of collocation points per direction, and (b) the number of training parameters, for the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. The test settings and parameters in (a) follow those of Figure 4, and in (b) follow those of Figure 5.

caption lists the crucial parameter values. It is evident that the errors for α and u decrease exponentially (or approximately exponentially) with increasing number of training parameters.

Figure 6 illustrates the computational cost of the NLLSQ, VarPro-F1 and VarPro-F2 algorithms for solving the inverse Poisson problem. It shows the network training time as a function of the number of collocation points per direction (Figure 6(a)) and the number of training parameters in the neural network (Figure 6(b)) for the three algorithms. The problem settings and the simulation parameters employed in NLLSQ, VarPro-F1 and VarPro-F2 in Figures 6(a) and (b) follow those of Figure 4 and Figure 5, respectively. The network training time for all three algorithms appears to grow approximately linearly with respect to the number of collocation points per direction and to the number of training parameters in the network. The VarPro-F1 algorithm is more costly than NLLSQ and VarPro-F2 for this problem, while the cost of NLLSQ seems to be larger than or comparable to that of VarPro-F2. Figure 6(b) indicates that the training time exhibits some irregularity with respect to the number of training parameters for NLLSQ and VarPro-F1. This is due to the triggering of sub-iterations in Algorithm 7 and the irregularity in the actual number of nonlinear least squares iterations to meet the stopping criteria.

Table 2 illustrates the effect of the number of random measurement points (Q_s) on the α and u errors computed by the NLLSQ, VarPro-F1 and VarProf-F2 algorithms. When Q_s is very small, the computed α and u are inaccurate or less accurate. On the other hand, when Q_s reaches a certain value ($Q_s = 3$ for this

	NLLSQ			VarPro-F1			VarPro-F2		
Q_s	e_α	l^∞-u	l^2-u	e_α	l^∞-u	l^2-u	e_α	l^∞-u	l^2-u
1	1.02E+0	1.01E+0	3.27E-1	1.63E+0	2.09E+1	7.27E+0	6.61E-4	1.19E-3	3.59E-4
2	5.01E-7	3.67E-6	3.80E-7	1.70E+0	3.10E+1	1.15E+1	1.67E-8	5.60E-7	3.49E-8
3	5.06E-8	3.67E-6	2.64E-7	9.64E-8	9.11E-7	9.59E-8	4.17E-9	5.46E-7	3.26E-8
5	3.06E-8	3.67E-6	2.62E-7	2.81E-8	8.88E-7	8.26E-8	7.30E-9	5.33E-7	3.25E-8
10	1.39E-8	3.67E-6	2.61E-7	1.34E-8	9.05E-7	8.12E-8	4.72E-8	5.39E-7	4.16E-8
20	5.14E-8	3.67E-6	2.63E-7	1.19E-8	9.04E-7	8.10E-8	2.62E-9	5.47E-7	3.46E-8
50	1.07E-8	3.67E-6	2.62E-7	6.63E-9	8.56E-7	8.02E-8	4.81E-10	5.33E-7	3.23E-8
100	3.26E-8	3.72E-6	2.62E-7	3.15E-9	9.29E-7	7.95E-8	1.17E-8	5.48E-7	3.27E-8

Table 2: Inverse Poisson problem: α and u relative errors versus Q_s (number of measurement points) for the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. Single sub-domain, neural network $[2, 500, 1]$, $Q = 30 \times 30$; $R_m = 3.0$ with NLLSQ, $R_m = 2.8$ with VarPro-F1, and $R_m = 2.0$ with VarPro-F2; λ_{mea}=1, ϵ=0.

ϵ	computed-α	ϵ	computed-α	ϵ	computed-α
0.0	9.99999993208E-1	0.01	9.9875752E-1	0.1	9.8779390E-1
0.001	9.9987537E-1	0.03	9.9630066E-1	0.2	9.7602056E-1
0.002	9.9975066E-1	0.05	9.9383633E-1	0.5	9.4329282E-1
0.005	9.9937764E-1	0.07	9.9139103E-1	0.7	9.2316247E-1
0.007	9.9912874E-1	0.09	9.8897497E-1	1.0	8.9557261E-1

Table 3: Inverse Poisson problem: α obtained by the NLLSQ algorithm corresponding to several noise levels (ϵ). Single sub-domain, NN $[2, 500, 1]$, $Q = 25 \times 25$, $Q_s = 50$, $R_m = 3.0$, λ_{mea}=1.

problem) and beyond, the three algorithms produce highly accurate results. This seems to be a common characteristic of these algorithms for all the test problems considered in this work.

In the foregoing tests no noise is considered in the measurement data ($\epsilon = 0$). Tables 3 and 4 and Figure 7 demonstrate the effect of noisy measurement data on the computation results. Table 3 shows the computed α values by the NLLSQ algorithm corresponding to different noise levels, ranging from $\epsilon = 0$ (0%) to $\epsilon = 1.0$ (100%). Table 4 lists the α errors and the u errors corresponding to several noise levels obtained by the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. Figure 7 provides the α and u relative errors as a function of ϵ for several λ_{mea} (scaling factor of measurement residual) values with the NLLSQ algorithm. The presence of noise degrades the simulation accuracy. But the current method and these algorithms appear to be quite robust. For example, with 10% ($\epsilon = 0.1$) noise in the measurement data the relative error of α is around 1% for the three methods. With 100% ($\epsilon = 1.0$) noise in the data, the computed α exhibits a relative error around 10% with these algorithms. For noisy data, scaling the measurement residual by λ_{mea} can improve the accuracy of computation results and make the method more robust (see Figure 7), compared with the

	NLLSQ			VarPro-F1			VarPro-F2		
ϵ	e_α	l^∞-u	l^2-u	e_α	l^∞-u	l^2-u	e_α	l^∞-u	l^2-u
0.0	6.79E-9	1.81E-6	1.93E-7	5.93E-8	7.59E-7	1.38E-7	4.20E-10	4.50E-7	2.31E-8
0.001	1.25E-4	2.79E-4	8.04E-5	1.33E-4	2.82E-4	8.50E-5	1.23E-4	2.81E-4	7.85E-5
0.005	6.22E-4	1.39E-3	4.01E-4	6.73E-4	1.42E-3	4.29E-4	6.10E-4	1.41E-3	3.92E-4
0.01	1.24E-3	2.79E-3	8.02E-4	1.35E-3	2.85E-3	8.61E-4	1.22E-3	2.81E-3	7.82E-4
0.05	6.16E-3	1.39E-2	4.00E-3	6.63E-3	1.42E-2	4.26E-3	6.49E-3	1.41E-2	4.07E-3
0.1	1.22E-2	2.79E-2	7.98E-3	1.33E-2	2.86E-2	8.58E-3	1.19E-2	2.81E-2	7.75E-3
0.5	5.67E-2	1.42E-1	3.90E-2	6.08E-2	1.44E-1	4.17E-2	5.52E-2	1.43E-1	3.78E-2
1.0	1.04E-1	2.88E-1	7.63E-2	1.11E-1	2.93E-1	8.14E-2	1.08E-1	2.90E-1	7.62E-2

Table 4: Inverse Poisson problem: α and u relative errors versus ϵ (noise level) computed by the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. Single sub-domain, NN $[2, 500, 1]$, $Q = 25 \times 25$, $Q_s = 50$, λ_{mea}=1; $R_m = 3.0$ with NLLSQ, $R_m = 2.8$ with VarPro-F1, and $R_m = 2.0$ with VarPro-F2.

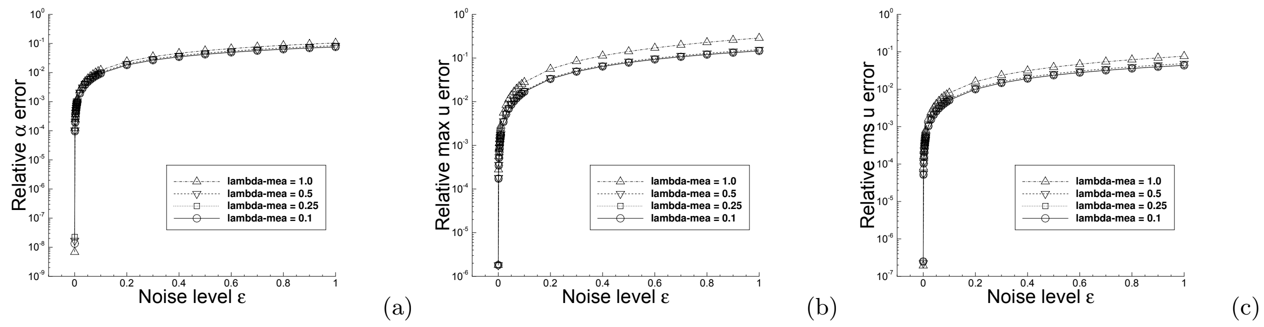


Figure 7: Inverse Poisson problem: α and u (l^∞-u and l^2-u) relative errors versus ϵ for several λ_{mea} (scaling coefficient of measurement residual) values computed by the NLLSQ algorithm. Single sub-domain, NN $[2, 500, 1]$, $Q = 25 \times 25$, $Q_s = 50$, $R_m = 3.0$.

case of no scaling. A smaller λ_{mea} in general leads to a better accuracy.

3.2 Parametric Nonlinear Helmholtz Equation

Consider the 2D domain, $(x, y) \in \Omega = [0, 1.4] \times [0, 1.4]$, and the inverse problem on Ω,

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} - \alpha_1 u + \alpha_2 \cos(2u) = f(x, y), \tag{50a}$$

$$u(0, y) = g_1(y), \quad u(1.4, y) = g_2(y), \quad u(x, 0) = g_3(x), \quad u(x, 1.4) = g_4(x), \tag{50b}$$

$$u(\xi_i, \eta_i) = S(\xi_i, \eta_i), \quad (\xi_i, \eta_i) \in \mathbb{Y} \subset \Omega,\ 1 \leqslant i \leqslant NQ_s, \tag{50c}$$

where f and g_i ($1 \leqslant i \leqslant 4$) are prescribed source term and boundary data, $\mathbb{Y}$ denotes the set of random measurement points in Ω, and the parameters (α_1, α_2) and the field $u(x, y)$ are the unknowns to be determined. We consider the following manufactured solution to this problem in the tests,

$$\alpha_1^{ex} = 100, \quad \alpha_2^{ex} = 5, \quad u_{ex}(x, y) = \cos(\pi x^2)\cos(\pi y^2). \tag{51}$$

The measurement data $S(\xi_i, \eta_i)$ ($1 \leqslant i \leqslant NQ_s$) are given by (48), in which u_{ex} is given by (51). The u errors are computed on a set of 101×101 uniform grid points in each sub-domain after the neural network is trained. The notations below follow those of the previous sub-section.

Figure 8 shows distributions of the $u(x, y)$ solution and its point-wise absolute error computed by the VarPro-F1 algorithm on 4 uniform sub-domains, with the 120 random measurement points in total ($Q_s = 30$ points per sub-domain) displayed in Figure 8(a). The figure caption lists the crucial simulation parameters for this test. VarPro-F1 exhibits a high accuracy, with the maximum u error on the order of 10^{-8}. The relative errors of the computed α_1 and α_2 are 3.52×10^{-11} and 4.76×10^{-10}, respectively, in this test.

The convergence of the simulation results with respect to the number of collocation points (Q) is illustrated by Table 5 and Figure 9. Table 5 lists the computed α_1 and α_2 by the NLLSQ algorithm corresponding to a range of Q values. Figure 9 shows the relative α_1 and α_2 errors and the l^2 norm of the relative u error corresponding to different Q obtained by the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. The crucial simulation parameter values are provided in the table/figure captions. A general exponential convergence in the errors with respect to Q can be observed. One can also observe that the convergence of the VarPro-F2 algorithm appears to be less regular. The VarPro-F2 results are inaccurate with a small Q ($Q = 15 \times 15$ or less), and its errors abruptly drop to $10^{-7} \sim 10^{-8}$ as the collocation points reach $Q = 20 \times 20$ and beyond.

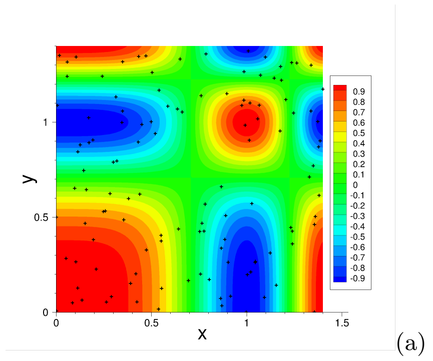
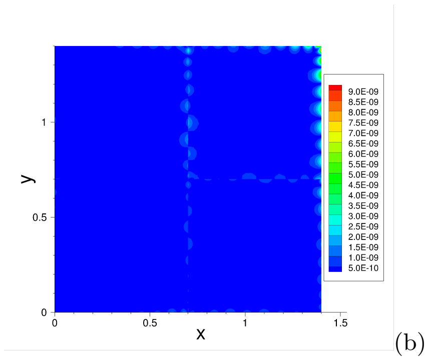
 

Figure 8: Inverse nonlinear Helmholtz problem: distributions of (a) the VarPro-F1 solution for $u(x, y)$ and (b) its point-wise absolute error, with the random measurement points shown in (a) as "+" symbols. Four uniform sub-domains (2 in each direction), local NN $[2, 300, 1]$, $Q = 20 \times 20$, $Q_s = 30$, $R_m = 1.5$, λ_{mea}=1, ϵ=0 (no noise in measurement data).

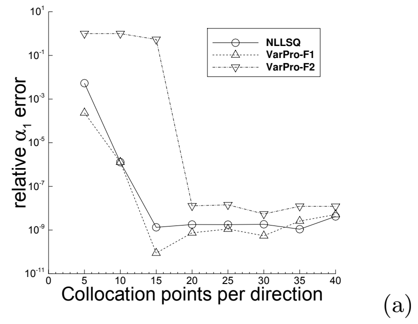
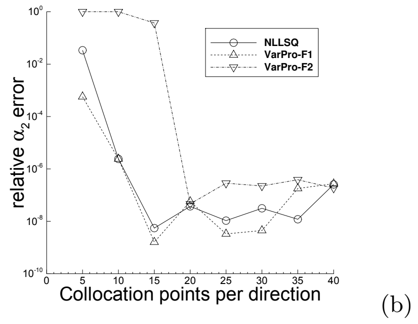
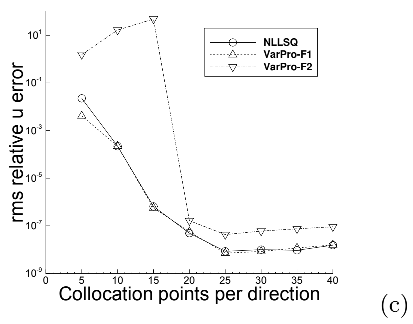
  

Figure 9: Inverse nonlinear Helmholtz problem: relative errors of α_1, α_2 and l^2-u versus Q_1 ($Q = Q_1 \times Q_1$) obtained by the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. Single sub-domain, NN $[2, 500, 1]$, $Q_s = 100$; $R_m = 2.25$ with NLLSQ and VarPro-F1, and $R_m = 2.5$ with VarPro-F2; λ_{mea}=1, ϵ=0. e_{α_1} and e_{α_2} denote the relative errors of α_1 and α_2.

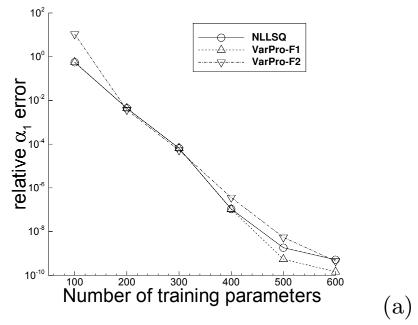
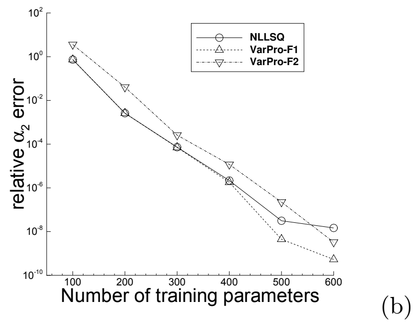
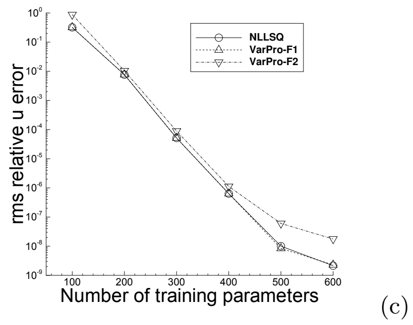
  

Figure 10: Inverse nonlinear Helmholtz problem: relative errors of α_1, α_2 and l^2-u versus M (number of training parameters) obtained by the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. Single sub-domain, NN $[2, M, 1]$, $Q_s = 100$, $Q = 30 \times 30$; $R_m = 2.25$ with NLLSQ and VarPro-F1, and $R_m = 2.5$ with VarPro-F2; ϵ=0, λ_{mea}=1.

Q	computed α_1	computed α_2
5×5	9.946591149073E+1	5.169760481373E+0
10×10	9.999987125506E+1	4.999987933629E+0
15×15	9.999999986638E+1	5.000000027512E+0
20×20	9.999999982078E+1	4.999999813483E+0
25×25	1.000000001774E+2	4.999999946859E+0
30×30	1.000000001832E+2	4.999999843159E+0
35×35	9.999999989070E+1	5.000000059829E+0
40×40	9.999999957958E+1	5.000001280912E+0

Table 5: Inverse nonlinear Helmholtz problem: α_1 and α_2 versus Q (number of collocation points) obtained by the NLLSQ algorithm. Single sub-domain, NN $[2, 500, 1]$, $Q_s = 100$, $R_m = 2.25$, $\epsilon = 0$, λ_{mea}=1.

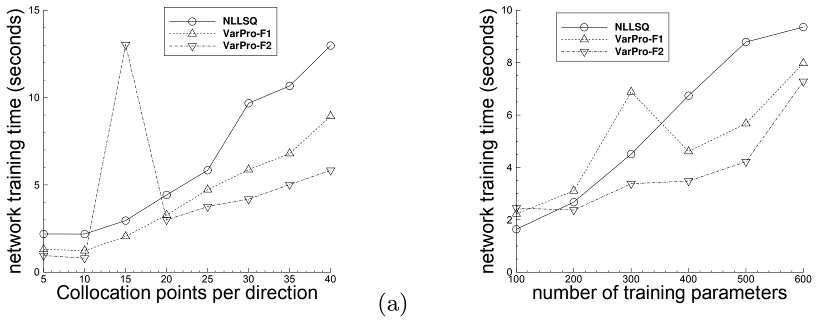


Figure 11: Inverse nonlinear Helmholtz problem: Network training time as a function of (a) the number of collocation points per direction, and (b) the number of training parameters, for the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. The test settings and parameters in (a) follow those of Figure 9, and in (b) follow those of Figure 10.

Figure 10 illustrates the convergence of the α_1, α_2 and u errors, obtained by the NLLSQ, VarPro-F1 and VarPro-F2 algorithms, with respect to the training parameters (M). The figure caption lists values of the main simulation parameters. The relative errors of α_1, α_2 and u decrease exponentially with increasing M.

Figure 11 shows the network training time with the NLLSQ, VarPro-F1 and VarPro-F2 algorithms as a function of the number of collocation points per direction (plot (a)) and the number of training parameters (plot (b)) for the inverse nonlinear Helmholtz problem. The problem settings and the simulation parameters employed in the three algorithms for these two plots correspond to those of Figures 9 and 10, respectively. We observe a quasi-linear growth in the network training time with the increase of the collocation points or the training parameters. In general, the cost of NLLSQ appears a little larger than that of VarPro-F1, which in turn appears a little larger than VarPro-F2 for this problem. We observe an outlier in Figure 11(a) with VarPro-F2 (corresponding to 15×15 collocation points), and in Figure 11(b) with VarPro-F1 (corresponding to 300 training parameters). This is caused by the larger number of actual Newton iterations in VarPro-F2 for the outlier case in Figure 11(a), and by the triggering of subiterations in Algorithm 7 with VarPro-F1 for the case in Figure 11(b).

Table 6 shows the computed α_1 and α_2 relative errors, and the u relative errors (l^∞ and l^2 norms) obtained by the NLLSQ algorithm corresponding to a range of Q_s (number of random measurement points). The effect of Q_s on the errors appears to be not significant, unless Q_s is very small. This is similar to what has been observed with linear forward PDEs (see e.g. Section 3.1).

No noise is considered in the measurement data in the foregoing tests. Figure 12 illustrates the effect of the noise level (ϵ) on the accuracy of the computed α_1, α_2 and u by the NLLSQ, VarPro-F1 and VarPro-F2

Q_s	e_{α_1}	e_{α_2}	l^∞-u	l^2-u
5	5.41E-9	1.03E-6	7.79E-8	4.13E-8
10	7.30E-10	1.24E-7	6.03E-8	8.65E-9
20	1.18E-9	5.71E-9	9.67E-8	8.94E-9
30	6.74E-10	7.83E-8	7.97E-8	8.18E-9
50	1.99E-9	1.32E-7	7.56E-8	1.05E-8
100	1.83E-9	3.14E-8	9.37E-8	1.00E-8

Table 6: Inverse nonlinear Helmholtz problem: the relative-errors of α_1 and α_2, and the u relative errors, versus the number of random measurement points (Q_s), computed by the NLLSQ algorithm. Single sub-domain, NN $[2, 500, 1]$, $Q = 30 \times 30$, $R_m = 2.25$, λ_{mea}=1, ϵ=0.

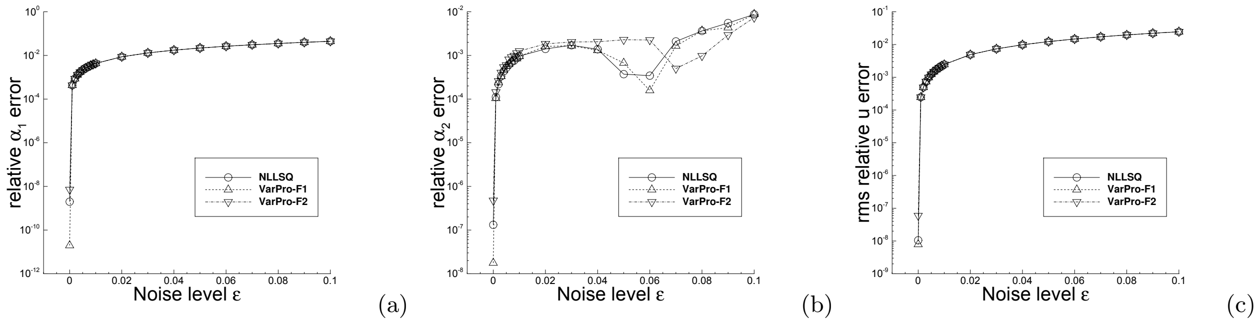


Figure 12: Inverse nonlinear Helmholtz problem: relative errors of α_1, α_2 and l^2-u versus the noise level (ϵ) in the measurement data, obtained by the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. Single sub-domain, NN $[2, 500, 1]$, $Q = 30 \times 30$, $Q_s = 50$, λ_{mea}=1; $R_m = 2.25$ with NLLSQ and VarPro-F1, and $R_m = 2.5$ with VarPro-F2.

algorithms. The main parameters for these simulations are listed in the figure caption. The simulation errors generally increase with increasing noise level in the measurement data. However, the α_2 error appears to be somewhat less regular for a range of noise levels (around $\epsilon \approx 0.06$) for this problem. The accuracy of these algorithms appears quite robust to the noise. For example, with 1% noise ($\epsilon = 0.01$) in the measurement data the relative errors for α_1 and α_2 obtained by the three methods are on the order of 0.1%, and with 10% noise ($\epsilon = 0.1$) in the measurement data the relative errors for α_1 and α_2 are on the order of $1 \sim 4\%$.

3.3 Parametric Viscous Burgers Equation

Consider the spatial-temporal domain, $(x, t) \in \Omega = [0, 2] \times [0, 1.5]$, and the inverse problem with the parametric Burgers' equation,

$$\frac{\partial u}{\partial t} + \alpha_1 u \frac{\partial u}{\partial x} = \alpha_2 \frac{\partial^2 u}{\partial x^2} + f(x, t), \tag{52a}$$

$$u(0, t) = g_1(t), \quad u(2, t) = g_2(t), \quad u(x, 0) = h(x), \tag{52b}$$

$$u(\xi_i, \eta_i) = S(\xi_i, \eta_i), \quad (\xi_i, \eta_i) \in \mathbb{Y} \subset \Omega, \ 1 \leqslant i \leqslant NQ_s, \tag{52c}$$

where f is a prescribed source term, g_1 and g_2 are prescribed Dirichlet boundary data, h is the initial distribution, the constants α_i ($i = 1, 2$) and the field $u(x, t)$ are the unknowns to be solved for, $\mathbb{Y}$ denotes the set of random measurement points, N is the number of sub-domains, and Q_s is the number of measurement

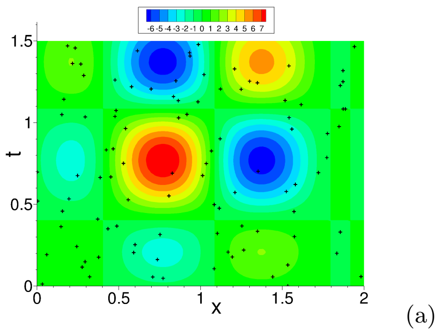
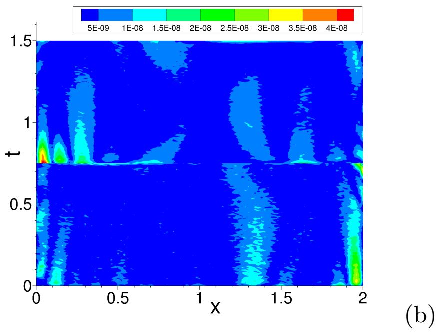



Figure 13: Inverse Burgers' problem: distributions of (a) the NLLSQ solution and (b) its point-wise absolute error, with the random measurement points shown by the "+" symbols in (a). Two uniform sub-domains (along t), local NN $[2, 300, 1]$, $Q = 25 \times 25$, $Q_s = 50$, $R_m = 1.5$, ϵ=0 (no noise in measurement data), λ_{mea}=1.

Q	α_1	α_2
5×5	9.999660775275E-2	9.994514983290E-3
10×10	9.999998874379E-2	9.999992607237E-3
15×15	1.000000000074E-1	1.000000000018E-2
20×20	1.000000000060E-1	1.000000000487E-2
25×25	9.999999999967E-2	9.999999999698E-3
30×30	1.000000000049E-1	9.999999998052E-3

Table 7: Inverse Burgers' problem: the computed α_1 and α_2 versus Q (number of collocation points) obtained with the NLLSQ algorithm. Single sub-domain, NN $[2, 400, 1]$, $Q_s = 100$, $R_m = 1.9$, λ_{mea}=1, ϵ=0.

points per sub-domain. We employ the following manufactured solution in the tests,

$$\begin{cases} \alpha_1^{ex} = 0.1, \quad \alpha_2^{ex} = 0.01, \\ u_{ex}(x,t) = \left(1+\frac{x}{20}\right)\left(1+\frac{t}{20}\right)\left[\frac{3}{2}\cos\left(\pi x + \frac{7\pi}{20}\right) + \frac{27}{20}\cos\left(2\pi x - \frac{3\pi}{5}\right)\right]\left[\frac{3}{2}\cos\left(\pi t + \frac{7\pi}{20}\right)\right. \\ \qquad \left. + \frac{27}{20}\cos\left(2\pi t - \frac{3\pi}{5}\right)\right]. \end{cases} \tag{53}$$

The source term f and the boundary/initial data are chosen such that the expressions in (53) satisfy the equations (52a)–(52b). The measurement data $S(\xi_i, \eta_i)$ is assumed to be given by (48), in which u_{ex} is given by (53). In the following the u errors are computed on a 101×101 uniform grid points in each sub-domain, and we adopt the same notations (e.g. Q, Q_s, M, R_m and ϵ) as in previous sub-sections.

Figure 13 illustrates the $u(x,t)$ solution and its point-wise absolute error computed by the NLLSQ algorithm with two uniform sub-domains along t, and the 100 random measurement points (50 points per sub-domain) in the domain are shown in Figure 13(a). The figure caption provides the main parameter values in this simulation. The results signify a high accuracy for the computed u solution, with the maximum error on the order of 10^{-8}. The relative errors of the computed α_1 and α_2 are 1.30×10^{-9} and 1.48×10^{-8}, respectively, for this simulation.

Table 7 and Figure 14 illustrate the convergence behavior of the NLLSQ, VarPro-F1 and VarPro-F2 algorithms with respect to the number of collocation points (Q). Table 7 shows the computed α_1 and α_2 values by the NLLSQ algorithm for several Q. Figure 14 shows the relative errors of α_1 and α_2 and the l^2 norm of relative u error corresponding to different Q. We refer the reader to the table/figure captions for the simulation parameter values. One can observe the familiar exponential convergence with respect to Q (before stagnation when Q reaches a certain level). The VarPro-F2 algorithm appears not as accurate as NLLSQ/VarPro-F1 with a small number of collocation points (below $Q = 20 \times 20$). But its errors drop to a

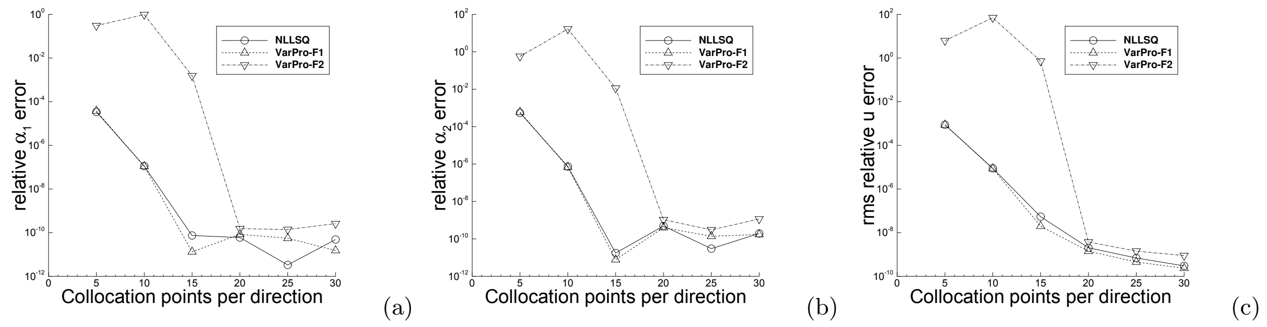


Figure 14: Inverse Burgers' problem: relative errors of (a) α_1, (b) α_2, and (c) l^2-u versus Q_1 ($Q = Q_1 \times Q_1$) obtained by the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. Single sub-domain, NN $[2, 400, 1]$, $Q_s = 100$; $R_m = 1.9$ with NLLSQ and VarPro-F1, and $R_m = 2.0$ with VarPro-F2; ϵ=0, λ_{mea}=1.

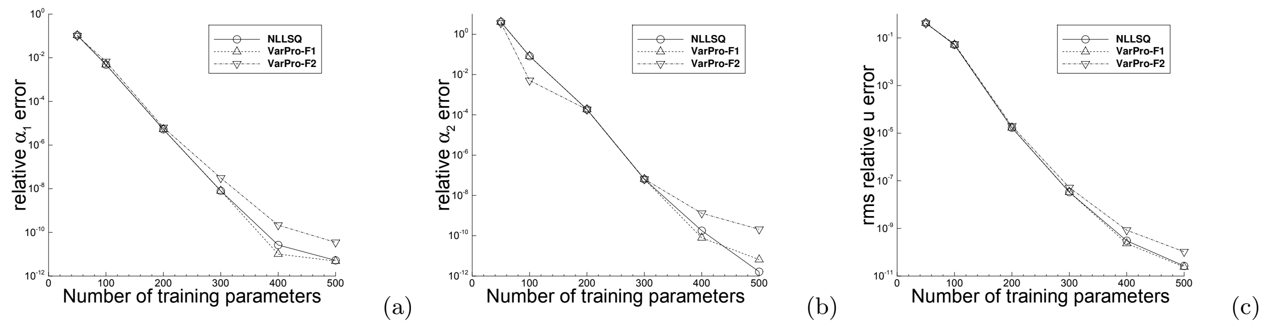


Figure 15: Inverse Burgers' problem: relative errors of (a) α_1, (b) α_2, and (c) u versus M (number of training parameters) obtained by the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. Single sub-domain, NN $[2, M, 1]$ (M varied), Q_s=150, $Q = 30 \times 30$, ϵ=0, λ_{mea}=1; $R_m = 1.9$ with NLLSQ and VarPro-F1, and $R_m = 2.0$ with VarPro-F2.

level similar to those of NLLSQ and VarPro-F1 for $Q = 20 \times 20$ and beyond.

The exponential convergence of the simulation results with respect to the number of training parameters (M) for the NLLSQ, VarPro-F1 and VarPro-F2 algorithms is illustrated by Figure 15. This figure shows the relative errors of the computed α_1, α_2 and u obtained by the three algorithms. One should again refer to the caption for the main settings and simulation parameters.

Figure 16 shows the network training time of the NLLSQ/VarPro-F1/VarPro-F2 algorithms as a function of the number of collocation points per direction and the number of training parameters for the inverse Burgers' problem. The settings here correspond to those of Figures 14 and 15, respectively. One can again observe a near-linear growth in the computational cost. The network training time of VarPro-F2 is significantly larger than those of NLLSQ/VarPro-F1, while the cost of NLLSQ and VarPro-F1 appears to be comparable. Some irregularity is observed in the training time with VarPro-F2 in Figure 16(a), due to the irregularity in the actual number of outer Newton iterations with VarPro-F2 for this problem.

Figure 17 illustrates the effect of the noisy measurement data (ϵ) on the simulation accuracy of the NLLSQ, VarPro-F1 and VarPro-F2 algorithms for the inverse Burgers problem. It shows the relative errors of α_1, α_2 and l^2-u as a function of the noise level ϵ in the measurement data. It is observed that the accuracy of these algorithms is quite robust to the noise. For example, with 1% noise in the measurement data the relative errors of these methods are around 0.026% for the computed α_1 and around 0.2% for the computed

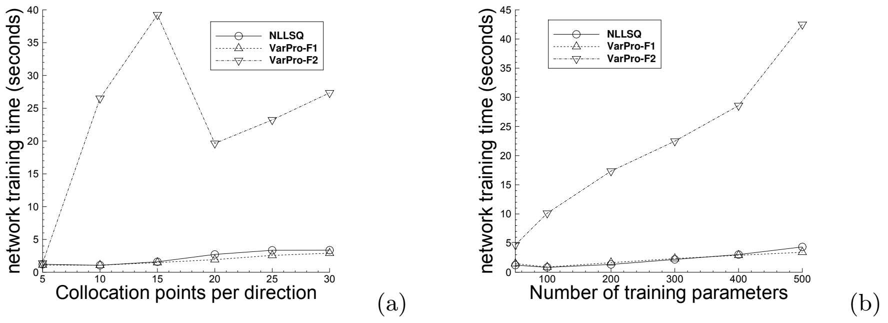


Figure 16: Inverse Burgers' problem: Network training time as a function of (a) the number of collocation points per direction, and (b) the number of training parameters, for the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. The test settings and parameters in (a) follow those of Figure 14, and in (b) follow those of Figure 15.

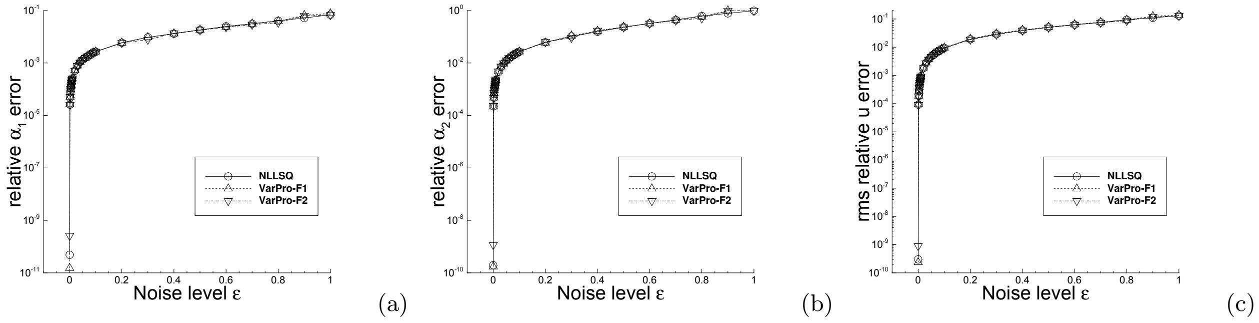


Figure 17: Inverse Burgers' problem: relative errors of (a) α_1, (b) α_2, and (c) l^2-u versus ϵ (noise level) obtained with the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. Single sub-domain, NN $[2, 400, 1]$, $Q = 30 \times 30$, Q_s=100, λ_{mea}=1; $R_m = 1.9$ with NLLSQ and VarPro-F1, and $R_m = 2.0$ with VarPro-F2.

α_2; with 10% noise in the measurement the relative errors are around 0.27% for α_1 and around 2.7% for α_2.

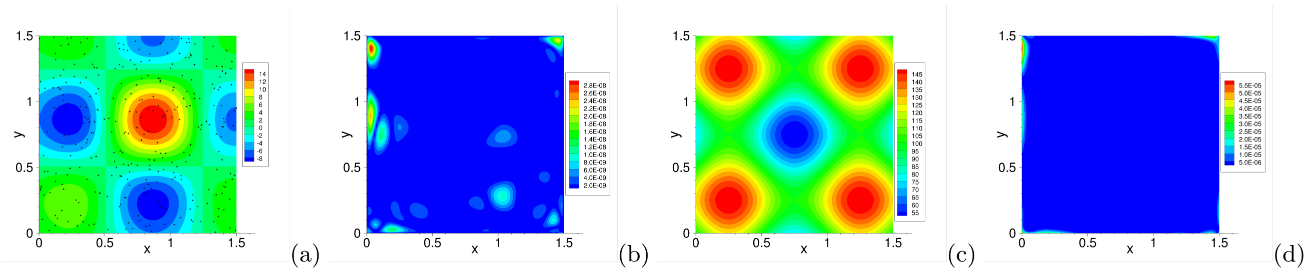


Figure 18: Inverse variable-coefficient Helmholtz problem: distributions of (a) the NLLSQ solution for $u(x, y)$ and (b) its point-wise absolute error, (c) the NLLSQ solution for $\gamma(x, y)$ and (d) its point-wise absolute error, with the $Q_s = 300$ random measurement points shown in (a) as "+" symbols. Single sub-domain, NN $[2, 400, 1]$, $Q = 30 \times 30$, $R_m = 1.5$, $\lambda_{mea} = 1$, $\epsilon = 0$ (no noise), $\lambda_1 = \lambda_2 = 0$ (no regularization).

3.4 Helmholtz Equation with Inverse Variable Coefficient

In this example, we use our method to study a problem involving an inverse coefficient field. Consider the domain $(x, y) \in \Omega = [0, 1.5] \times [0, 1.5]$ and the inverse problem on Ω,

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} - \gamma(x, y)u = f(x, y), \tag{54a}$$

$$u(a_1, y) = g_1(y), \quad u(b_1, y) = g_2(y), \quad u(x, a_2) = g_3(x), \quad u(x, b_2) = g_4(x), \tag{54b}$$

$$u(\xi_i, \eta_i) = S(\xi_i, \eta_i), \quad (\xi_i, \eta_i) \in \mathbb{Y} \subset \Omega, \ 1 \leqslant i \leqslant NQ_s, \tag{54c}$$

where f is a prescribed source term, g_i $(1 \leqslant i \leqslant 4)$ denote the prescribed boundary data, $\mathbb{Y}$ is the set of measurement points, $S(\xi_i, \eta_i)$ denotes the measurement data at the random point (ξ_i, η_i), and $\gamma(x, y)$ and $u(x, y)$ are two field functions to be determined. We employ the following manufactured solutions,

$$\begin{cases} \gamma_{ex}(x, y) = 100 \left[1 + \frac{1}{4}\sin(2\pi x) + \frac{1}{4}\sin(2\pi y) \right], \\ u_{ex}(x, y) = \left[\frac{5}{2}\sin\left(\pi x - \frac{2\pi}{5}\right) + \frac{3}{2}\cos\left(2\pi x + \frac{3\pi}{10}\right) \right] \left[\frac{5}{2}\sin\left(\pi y - \frac{2\pi}{5}\right) + \frac{3}{2}\cos\left(2\pi y + \frac{3\pi}{10}\right) \right]. \end{cases} \tag{55}$$

f and g_i $(1 \leqslant i \leqslant 4)$ are chosen accordingly such that the expressions in (55) satisfy the equations (54a)–(54b). The measurement data $S(\xi_i, \eta_i)$ are given by equation (48), in which the u_{ex} is given in (55). The relative errors for $u(x, y)$ are defined in (49), and the relative errors for $\gamma(x, y)$ are defined analogously. The $\gamma(x, y)$ and $u(x, y)$ errors reported below are computed on a uniform $Q_{eval} = 101 \times 101$ grid in each sub-domain. The notations here follow those of previous subsections.

We employ the algorithm modification as outlined in Remark 2.7 for solving this problem. Compared with previous subsections, the main change here lies in that the local neural network on each sub-domain contains two nodes in the output layer, one representing $u(x, y)$ and the other $\gamma(x, y)$. The output-layer coefficients contributing to $\gamma(x, y)$ play the role of the inverse parameters. We also find it preferable to regularize the the output-layer coefficients that contribute to $\gamma(x, y)$ (or $u(x, y)$) for this problem. For regularization we employ the extra terms for the underlying loss function, $\frac{\lambda_1^2}{2}\|\boldsymbol{\alpha}\|^2 + \frac{\lambda_2^2}{2}\|\boldsymbol{\beta}\|^2$, where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ denote the vectors of output-layer coefficients for $\gamma(x, y)$ and $u(x, y)$, respectively, and the prescribed non-negative constants λ_1 and λ_2 are the corresponding regularization coefficients.

Figure 18 shows distributions of the solutions for $u(x, y)$ and $\gamma(x, y)$, and their point-wise absolute errors, obtained by the NLLSQ algorithm. The random measurement points are also shown in Figure 18(a). The figure caption lists the main parameter values for this simulation. We observe a fairly high accuracy, with the maximum u error on the order of 10^{-8} and the maximum γ error on the order of 10^{-5} in the domain.

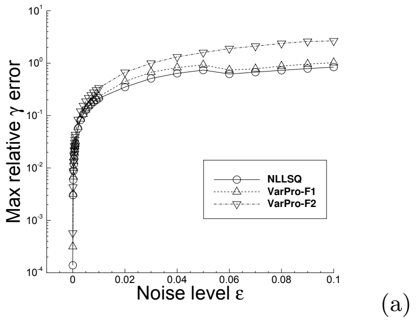
Table 8 lists the relative errors for γ and u (l^∞ and l^2 norms) computed by the NLLSQ algorithm in several sets of tests, with respect to Q (number of collocation points), M (number of training parameters), Q_s (number of random measurement points), and ϵ (noise level). The settings and the simulation parameter values are provided in the table caption for each set of tests. One can observe an approximately exponential decrease in the γ and u errors with respect to Q, M and Q_s (before saturation). One can also observe the deterioration in the simulation accuracy with increasing noise level in the measurement data. Note that no regularization is employed in these simulations. The noise appears to affect the γ results more significantly than u. For example, with 1% noise ($\epsilon = 0.01$) in the measurement data, the maximum (l^∞) relative error for $\gamma(x, y)$ is around 43% and the l^2 relative error is around 5%, while for $u(x, y)$ these errors are around 4% and 0.6% respectively.

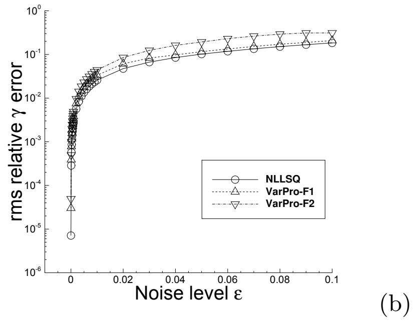
			l^∞-γ	l^2-γ	l^∞-u	l^2-u
collocation point test	$Q =$	5×5	9.84E-2	3.15E-2	7.94E-3	4.38E-4
		10×10	4.05E-3	2.90E-4	1.63E-4	1.02E-5
		15×15	1.15E-4	4.94E-6	3.38E-6	2.99E-7
		20×20	4.99E-6	4.84E-7	2.65E-7	2.70E-8
		25×25	4.42E-6	3.57E-7	3.34E-7	2.90E-8
		30×30	1.56E-6	1.46E-7	1.19E-7	1.08E-8
		35×35	1.61E-6	1.73E-7	1.26E-7	1.15E-8
training parameter test	$M =$	50	6.79E+0	1.67E+0	1.63E+0	5.67E-1
		100	8.64E-2	1.10E-2	8.44E-3	2.00E-3
		200	1.98E-4	2.08E-5	6.46E-6	6.42E-7
		300	2.07E-6	1.26E-7	3.13E-8	2.46E-9
		400	5.08E-7	2.95E-8	5.76E-9	5.29E-10
		500	1.61E-7	1.18E-8	1.95E-9	1.91E-10
measurement point test	$Q_s =$	10	1.36E-3	3.11E-4	2.49E-4	6.60E-5
		30	1.93E-4	3.69E-5	2.03E-5	4.24E-6
		50	2.69E-5	4.08E-6	2.03E-6	3.83E-7
		100	3.18E-6	2.95E-7	2.25E-7	1.78E-8
		200	5.30E-6	2.53E-7	8.21E-8	5.48E-9
		300	1.45E-6	9.93E-8	2.45E-8	2.00E-9
		400	2.65E-6	8.80E-8	1.40E-8	1.79E-9
noise level test	$\epsilon =$	0.0	1.61E-6	1.73E-7	1.26E-7	1.15E-8
		0.0005	5.15E-2	5.54E-3	3.32E-3	4.54E-4
		0.001	7.57E-2	7.31E-3	5.46E-3	6.63E-4
		0.005	2.69E-1	2.92E-2	2.28E-2	3.12E-3
		0.01	4.26E-1	4.96E-2	3.94E-2	5.98E-3
		0.05	1.50E+0	2.00E-1	1.44E-1	2.71E-2
		0.1	1.85E+0	2.97E-1	2.26E-1	4.97E-2

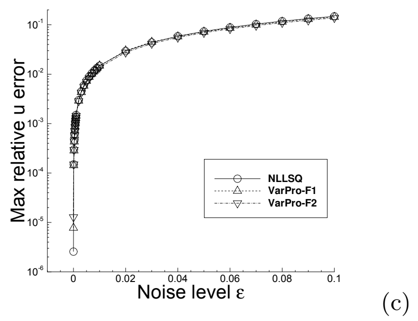
Table 8: Inverse variable-coefficient Helmholtz problem: relative l^∞ and l^2 errors of $\gamma(x, y)$ and $u(x, y)$ in several tests with the NLLSQ algorithm. Single sub-domain, NN $[2, M, 2]$. In collocation point test, $M = 400$, $Q_s = 100$, $\epsilon = 0$, Q is varied. In training parameter test, $Q = 30 \times 30$, $Q_s = 300$, $\epsilon = 0$, M is varied. In measurement point test, $Q = 25 \times 25$, $M = 300$, $\epsilon = 0$, Q_s is varied. In noise level test, $Q = 35 \times 35$, $M = 400$, $Q_s = 100$, ϵ is varied. $R_m = 1.5$ and $\lambda_{mea} = 1$ in all tests. No regularization ($\lambda_1 = \lambda_2 = 0$). l^∞-γ and l^2-γ denote the relative errors (l^∞ and l^2 norms) of $\gamma(x, y)$, respectively.

Figure 19 illustrates the effect of noise in the measurement data on the accuracy of the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. The relative errors for γ and u corresponding to different noise levels have been shown. In these simulations the output-layer coefficients for $\gamma(x, y)$ (and also for $u(x, y)$ with NLLSQ) have been regularized, with the regularization coefficients and the other simulation parameter values given in the table caption. The regularization generally improves the accuracy in the presence of noise. For $\gamma(x, y)$, the NLLSQ results appear to be generally more accurate than those obtained with VarPro-F1 and VarPro-F2. On the other hand, the $u(x, y)$ results obtained with the three methods appear to have a comparable accuracy (with VarPro-F2 slightly better).

Figure 20 shows the network training time of the NLLSQ/VarPro-F1/VarPro-F2 algorithms as a function of the noise level ϵ in the same group of tests as Figure 19. The increase in the noise level in the measurement data appears to have little effect on the network training time, or appears to cause the training time to slightly increase (see e.g. the curves with NLLSQ and VarPro-F2 in Figure 20).







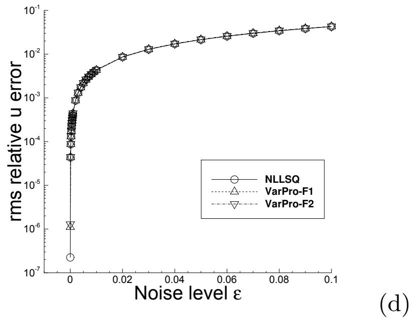


Figure 19: Inverse variable-coefficient Helmholtz problem: relative l^∞ and l^2 errors of $\gamma(x,y)$ and $u(x,y)$ versus ϵ (noise level) by the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. Single sub-domain, NN [2,400,2], $Q = 30 \times 30$, $Q_s = 300$, $\lambda_{mea} = 1$; $R_m = 1.5$ with NLLSQ, $R_m = 2.5$ with VarPro-F1, $R_m = 3.0$ with VarPro-F2; Regularization coefficients: $(\lambda_1, \lambda_2) = (1E-8, 1E-8)$ with NLLSQ, $(\lambda_1, \lambda_2) = (1E-7, 0)$ with VarPro-F1, and $(\lambda_1, \lambda_2) = (1E-6, 0)$ with VarPro-F2.

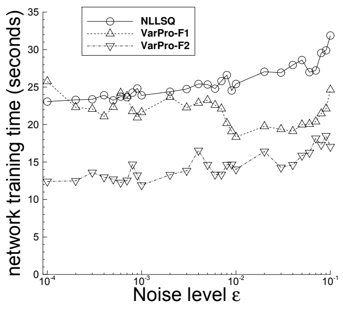


Figure 20: Inverse variable-coefficient Helmholtz problem: Network training time as a function of the noise level ϵ for the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. The test settings and parameters follow those of Figure 19.

4 Concluding Remarks

In this paper we have presented a method for solving inverse parametric PDE problems based on randomized neural networks. This method extends the local extreme learning machine (locELM) technique to inverse PDEs. The field solution is represented by a set of local random-weight neural networks (randomly assigned but fixed hidden-layer coefficients, trainable output-layer coefficients), one for each sub-domain. The local neural networks are coupled through the C^k (with k related to the PDE order) continuity conditions on the shared sub-domain boundaries. The inverse parameters of the PDE and the trainable parameters of the local neural networks are the unknowns to be determined in the system.

Three algorithms are developed for training the neural network to solve the inverse problem. The first algorithm (NLLSQ) computes the inverse parameters and the trainable network parameters all together by the nonlinear least squares method and is an extension of the nonlinear least squares method with perturbations (NLLSQ-perturb) of [16] (developed for forward PDEs) to inverse PDE problems. The second and the third algorithms are based on the variable projection idea. The second algorithm (VarPro-F1) employs variable projection to eliminate the inverse parameters from the problem and attain a reduced problem about the trainable network parameters only. Then the reduced problem is solved first by the NLLSQ-perturb algorithm for the trainable network parameters, and the inverse parameters are computed afterwards by the linear least squares method. The third algorithm (VarPro-F2) provides a reciprocal formulation with variable projection. It eliminates the trainable network parameters (or equivalently the field solution) from the problem first to arrive at a reduced problem about the inverse parameters only. Then the inverse parameters are computed first by solving the reduced problem with the NLLSQ-perturb algorithm, and afterwards the trainable network parameters are computed based on the inverse parameters already obtained. The VarPro-F2 algorithm is suitable for parametric PDEs that are linear with respect to the field solution. For PDEs that are nonlinear with respect to the field solution, this algorithm needs to be combined with a Newton iteration.

The presented method is numerically tested using several inverse parametric PDE problems (We refer the reader to the Appendices C and D for additional test problems). It is also compared with the PINN method (see Appendix E). For smooth solutions and noise-free data, the errors for the inverse parameters and the field solution computed by the NLLSQ, VarPro-F1 and VarPro-F2 algorithms decrease exponentially with respect to the number of collocation points and the number of training parameters. When these parameters become large, the errors can reach a level close to the machine accuracy. These characteristics are in some sense analogous to those observed for the forward PDE problems in [16, 19]. For noisy measurement data, these algorithms can produce computation results with good accuracy, indicating robustness of the method. We observe that, in the presence of noise, by scaling the measurement residual by a factor λ_{mea} $(0 < \lambda_{mea} < 1)$ one can in general improve the simulation accuracy of the current method markedly, while this scaling only slightly degrades the accuracy for the noise-free data. The comparison with PINN shows that the current method has an advantage in terms of both accuracy and the network training time. In particular, for the noise-free data the current method exhibits an accuracy significantly higher than PINN.

In terms of the computational cost, the predominant operations of all three algorithms lie in the nonlinear least squares computation (Algorithm 7) of either the overall inverse problem (with NLLSQ) or the reduced problem (with VarPro-F1 and VarPro-F2). The nonlinear least squares computation (as implemented in the Scipy library and adopted in the current method) consists of the Gauss-Newton iterations, and each iteration generally involves the computation of the residual vector and the Jacobian matrix, the solution of a linear least squares problem and the approximate solution of a trust-region problem. In addition, if the perturbation/sub-iteration is triggered in Algorithm 7, which occurs when the returned cost of the Gauss-Newton iteration fails to meet a tolerance, this will increase the computational cost. We have looked into the network training time, which includes the cost for the nonlinear least squares and associated computations, with the NLLSQ, VarPro-F1 and VarPro-F2 algorithms for different test problems. In general, the network training time grows approximately linearly as the number of collocation points or the number of training parameters increases for all three algorithms. In terms of the relative cost of these three algorithms, the picture seems to be mixed. Among the three, no single algorithm is consistently faster than the others for all the test problems considered here. For the test problems with an associated forward PDE that is linear, the VarPro-F2 algorithm seems to be generally faster than NLLSQ and VarPro-F1. We note that for the test problems and the problem sizes considered in the current paper, the network training time ranges from a few seconds to dozens of seconds with the three algorithms.

These test results suggest that the method developed herein is an effective and promising technique for computing inverse PDEs. The exponential convergence exhibited by the method is especially interesting, suggesting a high accuracy of this technique. We anticipate that this technique will be a useful and meaningful addition to the arsenal for tackling this class of problems and be instrumental in computational science and engineering applications.

Acknowledgment

This work was partially supported by NSF (DMS-2012415).

Appendix A. Nonlinear Least Squares Algorithm with Perturbations (NLLSQ-perturb)

We summarize the nonlinear least squares algorithm with perturbations (NLLSQ-perturb) below in Algorithm 7, which is adapted from the one developed in [16] with certain modifications.

In this algorithm, δ controls the maximum magnitude of each component of the random perturbation vector $\Delta\boldsymbol{\theta}$. The vector $\boldsymbol{\theta}_0$ provides the initial guess to the solution of the nonlinear least squares problem. If the returned solution from the scipy least_squares() routine corresponding to $\boldsymbol{\theta}_0$ is not acceptable (i.e. the returned cost exceeding the tolerance ε), then a sub-iteration procedure is triggered in which new initial guesses ($\boldsymbol{\vartheta}_0$) are generated by perturbing either the origin or the best approximation to the solution obtained so far with a random vector. The scipy least_squares() routine is invoked with the new initial guesses until an acceptable solution is obtained or until the maximum number of sub-iterations is reached. The integer flag η controls around which point the perturbation is performed. If $\eta = 0$ the new initial guess is generated by perturbing the origin. Otherwise, the current best approximation to the solution is perturbed to generate a new initial guess. The parameter "max-nllsq-iterations" controls the maximum number of iterations (e.g. the maximum number of residual function evaluations) in the scipy least_squares() routine. The parameter "max-sub-iterations" controls the maximum number of sub-iterations for the initial guess perturbation. One can turn off the perturbation in the NLLSQ-perturb algorithm by setting max-sub-iterations to zero. Note that the scipy least_squares() function requires two routines in the input, one for computing the residual and the other for computing the Jacobian matrix for an arbitrary given approximation to the solution.

Algorithm 7: NLLSQ-perturb (nonlinear least squares with perturbations) algorithm

```
input  : max perturbation magnitude $\delta > 0$; initial guess vector $\boldsymbol{\theta}_0$; routine for computing residual; routine
         for computing Jacobian matrix; perturbation flag $\eta$ (integer, 0 or 1); tolerance $\varepsilon > 0$;
         max-nllsq-iterations (positive integer); max-sub-iterations (non-negative integer).
output: solution vector $\boldsymbol{\theta}$, associated cost $c$

1  invoke the scipy.optimize.least_squares routine, with the inputs (initial guess $\boldsymbol{\theta}_0$, routines for
     residual/jacobian-matrix calculations, and max-nllsq-iterations)
2  set $\boldsymbol{\theta} \leftarrow$ returned solution, and set $c \leftarrow$ returned cost
3  if $c$ is below $\varepsilon$ then
4      return $\boldsymbol{\theta}$ and $c$
5  end
6  for $i \leftarrow 1$ to max-sub-iterations do
7      generate a uniform random number $\xi$ on the interval $[0, 1]$
8      set $\delta_1 \leftarrow \xi\delta$
9      generate a uniform random vector $\Delta\boldsymbol{\theta}$ of the same shape as $\boldsymbol{\theta}$ on the interval $[-\delta_1, \delta_1]$
10     if $\eta$ is 0 then
11         $\boldsymbol{\vartheta}_0 \leftarrow \Delta\boldsymbol{\theta}$
12     else
13         $\boldsymbol{\vartheta}_0 \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
14     end
15     invoke the scipy.optimize.least_squares routine, with the inputs (initial guess $\boldsymbol{\vartheta}_0$, routines for
         residual/jacobian-matrix calculations, and max-nllsq-iterations)
16     if the returned cost is less than $c$ then
17         set $\boldsymbol{\theta} \leftarrow$ the returned solution, and set $c \leftarrow$ the returned cost
18         if $c$ is below $\varepsilon$ then
19             break
20         end
21     end
22 end
23 return $\boldsymbol{\theta}$ and $c$
```

Appendix B. Matrices in the NLLSQ and VarPro-F2 Algorithms

NLLSQ Algorithm:

The vectors in the expression (17) are given by

$$\begin{cases} \mathbf{R}^{\rm pde} = \begin{bmatrix} \vdots \\ R_{ep}^{\rm pde} \\ \vdots \end{bmatrix}_{NQ\times 1}; \ \mathbf{R}^{\rm mea} = \begin{bmatrix} \vdots \\ R_{ep}^{\rm mea} \\ \vdots \end{bmatrix}_{NQ_s\times 1}; \ \mathbf{R}^{\rm bc} = \begin{bmatrix} \mathbf{R}^{\rm bc1} \\ \mathbf{R}^{\rm bc2} \\ \mathbf{R}^{\rm bc3} \\ \mathbf{R}^{\rm bc4} \end{bmatrix}; \ \mathbf{R}^{\rm ck} = \begin{bmatrix} \mathbf{R}^{\rm ck1} \\ \mathbf{R}^{\rm ck2} \\ \mathbf{R}^{\rm ck3} \\ \mathbf{R}^{\rm ck4} \end{bmatrix}; \ \mathbf{R}^{\rm bc1} = \begin{bmatrix} \vdots \\ R_{lj}^{bc1} \\ \vdots \end{bmatrix}_{N_2Q_2\times 1}, \\ \mathbf{R}^{\rm bc2} = \begin{bmatrix} \vdots \\ R_{lj}^{bc2} \\ \vdots \end{bmatrix}_{N_2Q_2\times 1}, \ \mathbf{R}^{\rm bc3} = \begin{bmatrix} \vdots \\ R_{mi}^{bc3} \\ \vdots \end{bmatrix}_{N_1Q_1\times 1}, \ \mathbf{R}^{\rm bc4} = \begin{bmatrix} \vdots \\ R_{mi}^{bc4} \\ \vdots \end{bmatrix}_{N_1Q_1\times 1}; \ \mathbf{R}^{\rm ck1} = \begin{bmatrix} \vdots \\ R_{mlj}^{ck1} \\ \vdots \end{bmatrix}_{(N-N_2)Q_2\times 1}, \\ \mathbf{R}^{\rm ck2} = \begin{bmatrix} \vdots \\ R_{mlj}^{ck2} \\ \vdots \end{bmatrix}_{(N-N_2)Q_2\times 1}, \ \mathbf{R}^{\rm ck3} = \begin{bmatrix} \vdots \\ R_{mli}^{ck3} \\ \vdots \end{bmatrix}_{(N-N_1)Q_1\times 1}, \ \mathbf{R}^{\rm ck4} = \begin{bmatrix} \vdots \\ R_{mli}^{ck4} \\ \vdots \end{bmatrix}_{(N-N_1)Q_1\times 1}. \end{cases} \tag{56}$$

In the above expressions, $R_{ep}^{\rm pde}$ is the left hand side (LHS) of (12), and $R_{ep}^{\rm mea}$ is the LHS of (14). R_{lj}^{bc1}, R_{lj}^{bc2}, R_{mi}^{bc3} and R_{mi}^{bc4} are the LHSs of (13a)–(13d), respectively. R_{mlj}^{ck1}, R_{mlj}^{ck2}, R_{mli}^{ck3} and R_{mli}^{ck4} are the LHSs of (15a)–(15d), respectively.

The matrices in the expression (18) are given by,

$$\left\{\begin{aligned}
&\frac{\partial \mathbf{R}^{\mathrm{pde}}}{\partial \boldsymbol{\alpha}} = \left[\frac{\partial R^{\mathrm{pde}}_{ep}}{\partial \alpha_i}\right]_{NQ\times n} = \left[\mathcal{L}_1(u_e(\mathbf{x}^e_p)) \quad \dots \quad \mathcal{L}_n(u_e(\mathbf{x}^e_p))\right]_{NQ\times n}, \ \frac{\partial \mathbf{R}^{\mathrm{pde}}}{\partial \boldsymbol{\beta}} = \left[\frac{\partial R^{\mathrm{pde}}_{ep}}{\partial \beta_{ij}}\right]_{NQ\times NM},\\
&\frac{\partial \mathbf{R}^{\mathrm{mea}}}{\partial \boldsymbol{\beta}} = \left[\frac{\partial R^{\mathrm{mea}}_{ep}}{\partial \beta_{ij}}\right]_{NQ_s\times NM}, \ \frac{\partial \mathbf{R}^{\mathrm{bc}}}{\partial \boldsymbol{\beta}} = \begin{bmatrix}\frac{\partial \mathbf{R}^{\mathrm{bc1}}}{\partial \boldsymbol{\beta}}\\ \frac{\partial \mathbf{R}^{\mathrm{bc2}}}{\partial \boldsymbol{\beta}}\\ \frac{\partial \mathbf{R}^{\mathrm{bc3}}}{\partial \boldsymbol{\beta}}\\ \frac{\partial \mathbf{R}^{\mathrm{bc4}}}{\partial \boldsymbol{\beta}}\end{bmatrix}, \ \frac{\partial \mathbf{R}^{\mathrm{ck}}}{\partial \boldsymbol{\beta}} = \begin{bmatrix}\frac{\partial \mathbf{R}^{\mathrm{ck1}}}{\partial \boldsymbol{\beta}}\\ \frac{\partial \mathbf{R}^{\mathrm{ck2}}}{\partial \boldsymbol{\beta}}\\ \frac{\partial \mathbf{R}^{\mathrm{ck3}}}{\partial \boldsymbol{\beta}}\\ \frac{\partial \mathbf{R}^{\mathrm{ck4}}}{\partial \boldsymbol{\beta}}\end{bmatrix}, \ \frac{\partial \mathbf{R}^{\mathrm{bc1}}}{\partial \boldsymbol{\beta}} = \left[\frac{\partial R^{\mathrm{bc1}}_{lj}}{\partial \beta_{ik}}\right]_{N_2Q_2\times NM},\\
&\frac{\partial \mathbf{R}^{\mathrm{bc2}}}{\partial \boldsymbol{\beta}} = \left[\frac{\partial R^{\mathrm{bc2}}_{lj}}{\partial \beta_{ik}}\right]_{N_2Q_2\times NM}, \ \frac{\partial \mathbf{R}^{\mathrm{bc3}}}{\partial \boldsymbol{\beta}} = \left[\frac{\partial R^{\mathrm{bc3}}_{mi}}{\partial \beta_{lk}}\right]_{N_1Q_1\times NM}, \ \frac{\partial \mathbf{R}^{\mathrm{bc4}}}{\partial \boldsymbol{\beta}} = \left[\frac{\partial R^{\mathrm{bc4}}_{mi}}{\partial \beta_{lk}}\right]_{N_1Q_1\times NM},\\
&\frac{\partial \mathbf{R}^{\mathrm{ck1}}}{\partial \boldsymbol{\beta}} = \left[\frac{\partial R^{\mathrm{ck1}}_{mlj}}{\partial \beta_{iq}}\right]_{(N-N_2)Q_2\times NM}, \ \frac{\partial \mathbf{R}^{\mathrm{ck2}}}{\partial \boldsymbol{\beta}} = \left[\frac{\partial R^{\mathrm{ck2}}_{mlj}}{\partial \beta_{iq}}\right]_{(N-N_2)Q_2\times NM}, \ \frac{\partial \mathbf{R}^{\mathrm{ck3}}}{\partial \boldsymbol{\beta}} = \left[\frac{\partial R^{\mathrm{ck3}}_{mli}}{\partial \beta_{jq}}\right]_{(N-N_1)Q_1\times NM},\\
&\frac{\partial \mathbf{R}^{\mathrm{ck4}}}{\partial \boldsymbol{\beta}} = \left[\frac{\partial R^{\mathrm{ck4}}_{mli}}{\partial \beta_{jq}}\right]_{(N-N_1)Q_1\times NM}.
\end{aligned}\right. \tag{57}$$

In the matrix $\frac{\partial \mathbf{R}^{\mathrm{pde}}}{\partial \boldsymbol{\beta}}$ the only non-zero terms are

$$\frac{\partial R^{\mathrm{pde}}_{ep}}{\partial \beta_{ej}} = \alpha_1 \mathcal{L}'_1(u_e(\mathbf{x}^e_p))\phi_{ej}(\mathbf{x}^e_p) + \dots + \alpha_n \mathcal{L}'_n(u_e(\mathbf{x}^e_p))\phi_{ej}(\mathbf{x}^e_p) + \mathcal{F}'(u_e(\mathbf{x}^e_p))\phi_{ej}(\mathbf{x}^e_p),$$
$$\text{for } 1 \leqslant (e,p,j) \leqslant (N,Q,M), \tag{58}$$

where $\mathcal{L}'_i(u)$ $(1 \leqslant i \leqslant n)$ denote the derivatives of $\mathcal{L}_i(u)$ with respect to u, and $\mathcal{F}'(u)$ denotes the derivative of $\mathcal{F}(u)$ with respect to u. In the matrix $\frac{\partial \mathbf{R}^{\mathrm{mea}}}{\partial \boldsymbol{\beta}}$ the only non-zero terms are

$$\frac{\partial R^{\mathrm{mea}}_{ep}}{\partial \beta_{ej}} = \mathcal{M}\phi_{ej}(\boldsymbol{\xi}^e_p), \quad \text{for } 1 \leqslant (e,p,j) \leqslant (N,Q_s,M). \tag{59}$$

In the matrices $\frac{\partial \mathbf{R}^{\mathrm{bc1}}}{\partial \boldsymbol{\beta}}$, $\frac{\partial \mathbf{R}^{\mathrm{bc2}}}{\partial \boldsymbol{\beta}}$, $\frac{\partial \mathbf{R}^{\mathrm{bc3}}}{\partial \boldsymbol{\beta}}$ and $\frac{\partial \mathbf{R}^{\mathrm{bc4}}}{\partial \boldsymbol{\beta}}$ the only non-zero terms are,

$$\left\{\begin{aligned}
&\frac{\partial R^{\mathrm{bc1}}_{lj}}{\partial \beta_{lq}} = \mathcal{B}\phi_{eq}(a_1, y^e_p), \text{ where } e = e(1,l), \ p = p(1,j), \quad \text{for } 1 \leqslant (l,j,q) \leqslant (N_2,Q_2,M);\\
&\frac{\partial R^{\mathrm{bc2}}_{lj}}{\partial \beta_{lq}} = \mathcal{B}\phi_{eq}(b_1, y^e_p), \text{ where } e = e(N_1,l), \ p = p(Q_1,j), \quad \text{for } 1 \leqslant (l,j,q) \leqslant (N_2,Q_2,M);\\
&\frac{\partial R^{\mathrm{bc3}}_{mi}}{\partial \beta_{mq}} = \mathcal{B}\phi_{eq}(x^e_p, a_2), \text{ where } e = e(m,1), \ p = p(i,1), \quad \text{for } 1 \leqslant (m,i,q) \leqslant (N_1,Q_1,M);\\
&\frac{\partial R^{\mathrm{bc4}}_{mi}}{\partial \beta_{mq}} = \mathcal{B}\phi_{eq}(x^e_p, b_2), \text{ where } e = e(m,N_2), \ p = p(i,Q_2), \quad \text{for } 1 \leqslant (m,i,q) \leqslant (N_1,Q_1,M).
\end{aligned}\right. \tag{60}$$

In the matrices $\frac{\partial \mathbf{R}^{\mathrm{ck1}}}{\partial \boldsymbol{\beta}}$, $\frac{\partial \mathbf{R}^{\mathrm{ck2}}}{\partial \boldsymbol{\beta}}$, $\frac{\partial \mathbf{R}^{\mathrm{ck3}}}{\partial \boldsymbol{\beta}}$ and $\frac{\partial \mathbf{R}^{\mathrm{ck4}}}{\partial \boldsymbol{\beta}}$ the only non-zero terms are,

$$\left\{\begin{aligned}
&\frac{\partial R^{\mathrm{ck1}}_{mlj}}{\partial \beta_{e_1q}} = \phi_{e_1q}(X_m, y^{e_1}_{p_1}), \quad \frac{\partial R^{\mathrm{ck1}}_{mlj}}{\partial \beta_{e_2q}} = -\phi_{e_2q}(X_m, y^{e_2}_{p_2}), \quad \text{where } e_1 = e(m,l), \ e_2 = e(m+1,l),\\
&\qquad\qquad p_1 = p(Q_1,j), \ p_2 = p(1,j), \ \text{for } 1 \leqslant (m,l,j,q) \leqslant (N_1-1,N_2,Q_2,M);\\
&\frac{\partial R^{\mathrm{ck2}}_{mlj}}{\partial \beta_{e_1q}} = \left.\frac{\partial \phi_{e_1q}}{\partial x}\right|_{(X_m, y^{e_1}_{p_1})}, \quad \frac{\partial R^{\mathrm{ck2}}_{mlj}}{\partial \beta_{e_2q}} = -\left.\frac{\partial \phi_{e_2q}}{\partial x}\right|_{(X_m, y^{e_2}_{p_2})}, \text{ where } e_1 = e(m,l), \ e_2 = e(m+1,l),\\
&\qquad\qquad p_1 = p(Q_1,j), \ p_2 = p(1,j), \ \text{for } 1 \leqslant (m,l,j,q) \leqslant (N_1-1,N_2,Q_2,M);\\
&\frac{\partial R^{\mathrm{ck3}}_{mli}}{\partial \beta_{e_1q}} = \phi_{e_1q}(x^{e_1}_{p_1}, Y_l), \quad \frac{\partial R^{\mathrm{ck3}}_{mli}}{\partial \beta_{e_2q}} = -\phi_{e_2q}(x^{e_2}_{p_2}, Y_l), \quad \text{where } e_1 = e(m,l), \ e_2 = e(m,l+1),\\
&\qquad\qquad p_1 = p(i,Q_2), \ p_2 = p(i,1), \ \text{for } 1 \leqslant (m,l,i,q) \leqslant (N_1,N_2-1,Q_1,M);\\
&\frac{\partial R^{\mathrm{ck4}}_{mli}}{\partial \beta_{e_1q}} = \left.\frac{\partial \phi_{e_1q}}{\partial y}\right|_{(x^{e_1}_{p_1}, Y_l)}, \quad \frac{\partial R^{\mathrm{ck4}}_{mli}}{\partial \beta_{e_2q}} = -\left.\frac{\partial \phi_{e_2q}}{\partial y}\right|_{(x^{e_2}_{p_2}, Y_l)}, \text{ where } e_1 = e(m,l), \ e_2 = e(m,l+1),\\
&\qquad\qquad p_1 = p(i,Q_2), \ p_2 = p(i,1), \ \text{for } 1 \leqslant (m,l,i,q) \leqslant (N_1,N_2-1,Q_1,M).
\end{aligned}\right. \tag{61}$$

Var-F2 Algorithm:

The matrices in the expression (38) are given by,

$$
\begin{aligned}
&\mathbf{b}^{\text{pde}}=\begin{bmatrix}\vdots\\ f(\mathbf{x}_p^e)\\ \vdots\end{bmatrix}_{NQ\times 1},\ \mathbf{b}^{\text{mea}}=\begin{bmatrix}\vdots\\ S(\boldsymbol{\xi}_p^e)\\ \vdots\end{bmatrix}_{NQ_s\times 1},\ \mathbf{b}^{\text{bc1}}=\begin{bmatrix}\vdots\\ g(a_1,y_{p(1,j)}^{e(1,l)})\\ \vdots\end{bmatrix}_{N_2Q_2\times 1},\\
&\mathbf{b}^{\text{bc2}}=\begin{bmatrix}\vdots\\ g(b_1,y_{p(Q_1,j)}^{e(N_1,l)})\\ \vdots\end{bmatrix}_{N_2Q_2\times 1},\ \mathbf{b}^{\text{bc3}}=\begin{bmatrix}\vdots\\ g(x_{p(i,1)}^{e(m,1)},a_2)\\ \vdots\end{bmatrix}_{N_1Q_1\times 1},\ \mathbf{b}^{\text{bc4}}=\begin{bmatrix}\vdots\\ g(x_{p(i,Q_2)}^{e(m,N_2)},a_2)\\ \vdots\end{bmatrix}_{N_1Q_1\times 1},\\
&\mathbf{H}^{\text{pde}}=\left[h_{ij}^{\text{pde}}\right]_{NQ\times NM},\ \mathbf{H}^{\text{mea}}=\left[h_{ij}^{\text{mea}}\right]_{NQ_s\times NM},\ \mathbf{H}^{\text{bc1}}=\left[h_{ij}^{\text{bc1}}\right]_{N_2Q_2\times NM},\ \mathbf{H}^{\text{bc2}}=\left[h_{ij}^{\text{bc2}}\right]_{N_2Q_2\times NM},\\
&\mathbf{H}^{\text{bc3}}=\left[h_{ij}^{\text{bc3}}\right]_{N_1Q_1\times NM},\ \mathbf{H}^{\text{bc4}}=\left[h_{ij}^{\text{bc4}}\right]_{N_1Q_1\times NM},\ \mathbf{H}^{\text{ck1}}=\left[h_{ij}^{\text{ck1}}\right]_{(N-N_2)Q_2\times NM},\\
&\mathbf{H}^{\text{ck2}}=\left[h_{ij}^{\text{ck2}}\right]_{(N-N_2)Q_2\times NM},\ \mathbf{H}^{\text{ck3}}=\left[h_{ij}^{\text{ck3}}\right]_{(N-N_1)Q_1\times NM},\ \mathbf{H}^{\text{ck4}}=\left[h_{ij}^{\text{ck4}}\right]_{(N-N_1)Q_1\times NM}.
\end{aligned}
\tag{62}
$$

In the matrices $\mathbf{H}^{\text{pde}}$ and $\mathbf{H}^{\text{mea}}$ the only non-zero terms are,

$$
\begin{cases}
h_{ij}^{\text{pde}}=\alpha_1\mathcal{L}_1\phi_{eq}(\mathbf{x}_p^e)+\cdots+\alpha_n\mathcal{L}_n\phi_{eq}(\mathbf{x}_p^e)+\mathcal{F}\phi_{eq}(\mathbf{x}_p^e),\\
\qquad i=(e-1)Q+p,\ j=(e-1)M+q,\quad \text{for } 1\leqslant(e,p,q)\leqslant(N,Q,M);\\
h_{ij}^{\text{mea}}=\mathcal{M}\phi_{eq}(\boldsymbol{\xi}_p^e),\ \ i=(e-1)Q_s+p,\ j=(e-1)M+q,\quad \text{for } 1\leqslant(e,p,q)\leqslant(N,Q_s,M).
\end{cases}
\tag{63}
$$

In the matrices $\mathbf{H}^{\text{bc1}}$, $\mathbf{H}^{\text{bc2}}$, $\mathbf{H}^{\text{bc3}}$ and $\mathbf{H}^{\text{bc4}}$ the only non-zero terms are,

$$
\begin{cases}
h_{ij}^{\text{bc1}}=\mathcal{B}\phi_{eq}(a_1,y_p^e),\ e=e(1,l),\ p=p(1,k),\ i=(l-1)Q_2+k,\ j=(e-1)M+q,\ \text{for } 1\leqslant(l,k)\leqslant(N_2,Q_2);\\
h_{ij}^{\text{bc2}}=\mathcal{B}\phi_{eq}(b_1,y_p^e),\ e=e(N_1,l),\ p=p(Q_1,k),\ i=(l-1)Q_2+k,\ j=(e-1)M+q,\ \text{for } 1\leqslant(l,k)\leqslant(N_2,Q_2);\\
h_{ij}^{\text{bc3}}=\mathcal{B}\phi_{eq}(x_p^e,a_2),\ e=e(m,1),\ p=p(k,1),\ i=(m-1)Q_1+k,\ j=(e-1)M+q,\ \text{for } 1\leqslant(m,k)\leqslant(N_1,Q_1);\\
h_{ij}^{\text{bc4}}=\mathcal{B}\phi_{eq}(x_p^e,b_2),\ e=e(m,N_2),\ p=p(k,Q_2),\ i=(m-1)Q_1+k,\ j=(e-1)M+q,\ \text{for } 1\leqslant(m,k)\leqslant(N_1,Q_1),
\end{cases}
\tag{64}
$$

where the functions $e(\cdot,\cdot)$ and $p(\cdot,\cdot)$ are given by (7) and (10). In the matrices $\mathbf{H}^{\text{ck1}}$, $\mathbf{H}^{\text{ck2}}$, $\mathbf{H}^{\text{ck3}}$ and $\mathbf{H}^{\text{ck4}}$ the only non-zero terms are,

$$
\begin{cases}
h_{ij_1}^{\text{ck1}}=\phi_{e_1q}(X_m,y_{p_1}^{e_1}),\ h_{ij_2}^{\text{ck1}}=-\phi_{e_2q}(X_m,y_{p_2}^{e_2}),\ e_1=e(m,l),\ p_1=p(Q_1,k),\ e_2=e(m+1,l),\\
\qquad p_2=p(1,k),\ i=(m-1)N_2Q_2+(l-1)Q_2+k,\ j_1=(e_1-1)M+q,\ j_2=(e_2-1)M+q,\\
\qquad \text{for } 1\leqslant(m,l,k,q)\leqslant(N_1-1,N_2,Q_2,M);\\
h_{ij_1}^{\text{ck2}}=\left.\dfrac{\partial\phi_{e_1q}}{\partial x}\right|_{(X_m,y_{p_1}^{e_1})},\ h_{ij_2}^{\text{ck2}}=-\left.\dfrac{\partial\phi_{e_2q}}{\partial x}\right|_{(X_m,y_{p_2}^{e_2})},\ e_1=e(m,l),\ p_1=p(Q_1,k),\ e_2=e(m+1,l),\\
\qquad p_2=p(1,k),\ i=(m-1)N_2Q_2+(l-1)Q_2+k,\ j_1=(e_1-1)M+q,\ j_2=(e_2-1)M+q,\\
\qquad \text{for } 1\leqslant(m,l,k,q)\leqslant(N_1-1,N_2,Q_2,M);\\
h_{ij_1}^{\text{ck3}}=\phi_{e_1q}(x_{p_1}^{e_1},Y_l),\ h_{ij_2}^{\text{ck3}}=-\phi_{e_2q}(x_{p_2}^{e_2},Y_l),\ e_1=e(m,l),\ p_1=p(k,Q_2),\ e_2=e(m,l+1),\\
\qquad p_2=p(k,1),\ i=(l-1)N_1Q_1+(m-1)Q_1+k,\ j_1=(e_1-1)M+q,\ j_2=(e_2-1)M+q,\\
\qquad \text{for } 1\leqslant(m,l,k,q)\leqslant(N_1,N_2-1,Q_1,M);\\
h_{ij_1}^{\text{ck4}}=\left.\dfrac{\partial\phi_{e_1q}}{\partial y}\right|_{(x_{p_1}^{e_1},Y_l)},\ h_{ij_2}^{\text{ck4}}=-\left.\dfrac{\partial\phi_{e_2q}}{\partial y}\right|_{(x_{p_2}^{e_2},Y_l)},\ e_1=e(m,l),\ p_1=p(k,Q_2),\ e_2=e(m,l+1),\\
\qquad p_2=p(k,1),\ i=(l-1)N_1Q_1+(m-1)Q_1+k,\ j_1=(e_1-1)M+q,\ j_2=(e_2-1)M+q,\\
\qquad \text{for } 1\leqslant(m,l,k,q)\leqslant(N_1,N_2-1,Q_1,M).
\end{cases}
\tag{65}
$$

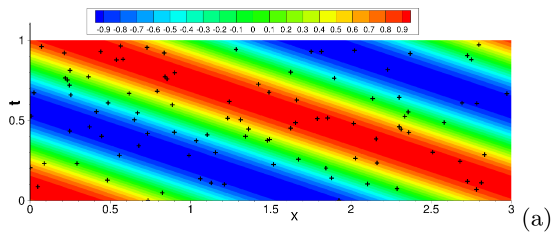
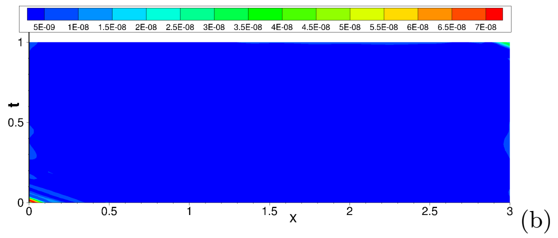



Figure 21: Inverse advection problem: distributions of (a) the NLLSQ solution for $u(x,t)$ and (b) its point-wise absolute error, with the random measurement points shown in (a) as "+" symbols. Single sub-domain, NN $[2,400,1]$, $Q = 25 \times 25$ (collocation points), $Q_s = 100$ (measurement points), $R_m = 2.5$, λ_{mea}=1, $\epsilon = 0$ (no noise in measurement).

Q	c (NLLSQ)	c (VarPro-F1)	c (VarPro-F2)
5×5	3.000074167561E+0	2.999935510214E+0	6.785575335360E-1
10×10	2.999998340831E+0	3.000000635012E+0	6.785578125741E-1
15×15	2.999999999982E+0	2.999999999967E+0	-7.284017530389E-2
20×20	3.000000000029E+0	3.000000000041E+0	3.000000000378E+0
25×25	3.000000000845E+0	3.000000000869E+0	3.000000000025E+0
30×30	3.000000000534E+0	3.000000000542E+0	3.000000001047E+0
35×35	3.000000000596E+0	3.000000000596E+0	3.000000001295E+0
40×40	3.000000000771E+0	3.000000000770E+0	3.000000001534E+0

Table 9: Inverse advection problem: the computed c versus Q obtained by the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. Single sub-domain, NN $[2,400,1]$, $Q_s = 100$; $R_m = 2.5$ with NLLSQ and VarPro-F1, and $R_m = 2.0$ with VarPro-F2; λ_{mea}=1, $\epsilon = 0$.

Appendix C. Parametric Advection Equation

This appendix provides a further test of the algorithms developed herein with the inverse parametric advection equation. Consider the spatial-temporal domain, $(x,t) \in \Omega = [0,3] \times [0,1]$, and the following inverse problem,

$$\frac{\partial u}{\partial t} - c\frac{\partial u}{\partial x} = 0, \tag{66a}$$

$$u(0,t) = u(3,t), \quad u(x,0) = 10\sinh\left[\frac{1}{10}\sin\frac{2\pi}{3}\left(x - \frac{5}{2}\right)\right], \tag{66b}$$

$$u(\xi_i,\eta_i) = S(\xi_i,\eta_i), \quad (\xi_i,\eta_i) \in \mathbb{Y} \subset \Omega,\ 1 \leqslant i \leqslant NQ_s, \tag{66c}$$

where $\mathbb{Y}$ denotes the set of measurement points in Ω. The wave speed c and the field $u(x,t)$ are the unknowns to be determined in this problem. We employ the following exact solution to this problem in the tests,

$$c_{ex} = 3, \quad u_{ex}(x,t) = 10\sinh\left[\frac{1}{10}\sin\frac{2\pi}{3}\left(x + 3t - \frac{5}{2}\right)\right]. \tag{67}$$

We employ random measurement points in Ω, and the measurement data are given by (48), in which u_{ex} is given by (67). The notations adopted below (e.g. Q, M, N, Q_s, R_m, ϵ) are the same as in Section 3.1. The l^∞ and l^2 norms of the u relative error reported below are computed on a set of $Q_{eval} = 101 \times 101$ uniform grid points in each sub-domain after the network is trained.

Figure 21 illustrates the distributions of the NLLSQ solution for $u(x,t)$ and its point-wise absolute error in Ω. The crucial simulation parameters are listed in the figure caption. The solution is highly accurate, with a maximum error on the level 10^{-8} in the domain. The computed wave speed c has a relative error 2.82×10^{-10} for this case.

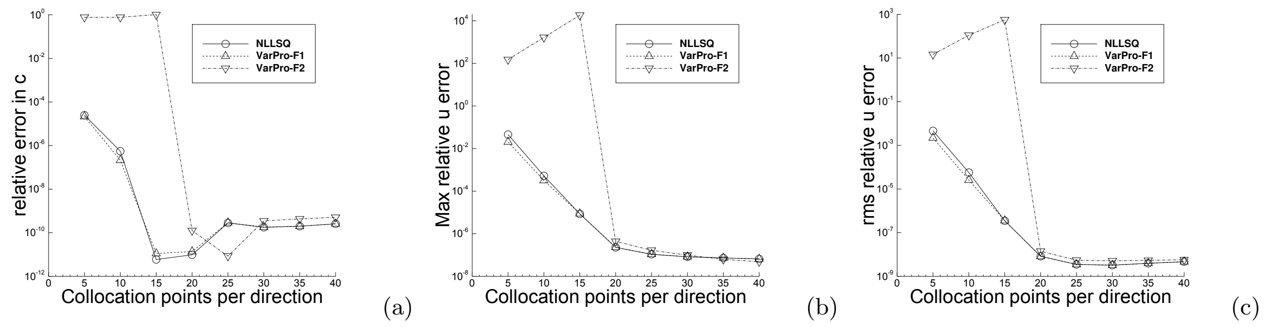


Figure 22: Inverse Advection problem: c and u (l^∞-u, l^2-u) relative errors versus Q_1 ($Q = Q_1 \times Q_1$) obtained by the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. Single sub-domain, NN $[2, 400, 1]$, $Q_s = 100$, λ_{mea}=1, ϵ=0; $R_m = 2.5$ with NLLSQ and VarPro-F1, and $R_m = 2.0$ with VarPro-F2.

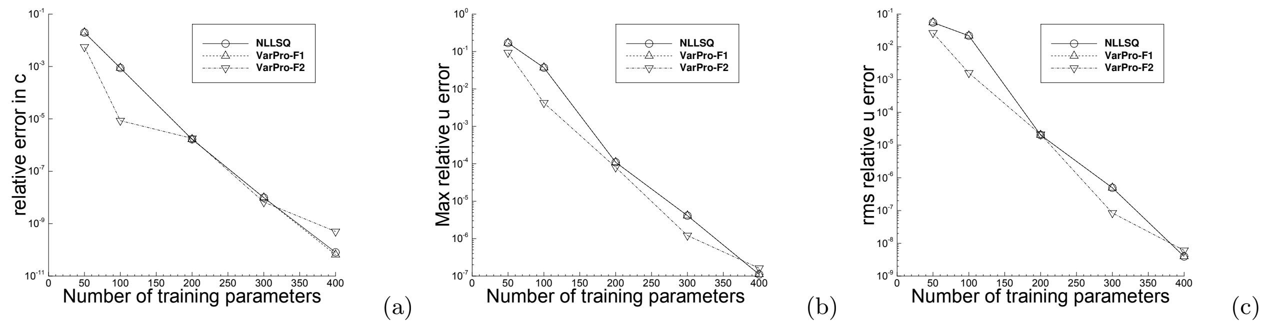
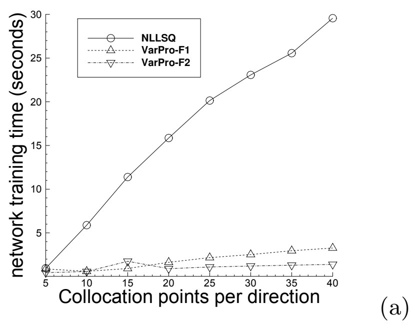


Figure 23: Inverse advection problem: c and u (l^∞-u, l^2-u) relative errors versus M (number of training parameters) obtained with the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. Single sub-domain, NN $[2, M, 1]$, $Q = 25 \times 25$, $Q_s = 50$, λ_{mea}=1, ϵ=0; $R_m = 2.5$ with NLLSQ and VarPro-F1, and $R_m = 2.0$ with VarPro-F2.

The convergence behaviors of the computed c and u with respect to the collocation points (Q) and to the training parameters (M) are illustrated in Table 9 and Figures 22 and 23 (without noise). Table 9 and Figure 22 show the computed c values, and the relative errors of c and u, for several sets of uniform collocation points obtained by the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. Figure 23 shows the c errors and the u errors for several sets of training parameters with the three algorithms. One can observe the general exponential convergence of the c and u errors with respect to Q and to M. Table 9 and Figure 22 indicate that the convergence of VarPro-F2 with respect to Q is not quite regular. If the set of collocation points is too small ($Q = 15 \times 15$ and below), the computed VarPro-F2 results are not accurate.

Figure 24 illustrates the computational cost of the NLLSQ/VarPro-F1/VarPro-F2 algorithms for solving the inverse advection problem by showing the network training time versus the number of collocation points and the number of training parameters. The test configurations and the simulation parameters in the two plots correspond to those of Figures 22 and 23, respectively. A near-linear growth in the network training time can be observed as the number of training parameters or the number of collocation points increases. The cost of NLLSQ is significantly larger than those of VarPro-F1/VarPro-F2 for this problem, while the cost of VarPro-F1 appears generally larger than that of VarPro-F2.

The effects of noisy measurement data on the computation accuracy are illustrated by Tables 10 and 11 and Figure 25. Table 10 lists the computed c by the NLLSQ algorithm corresponding to several noise levels



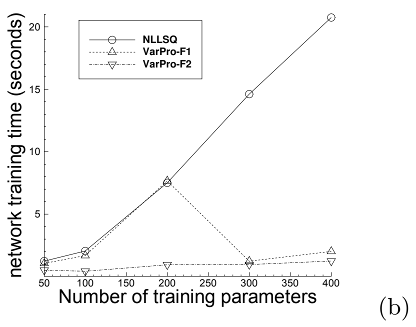


Figure 24: Inverse advection problem: Network training time as a function of (a) the number of collocation points per direction, and (b) the number of training parameters, for the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. The test settings and parameters in (a) follow those of Figure 22, and in (b) follow those of Figure 23.

ϵ	computed c	ϵ	computed c	ϵ	computed c
0.0	3.000000000534E+0	0.01	2.9997368E+0	0.1	2.9971795E+0
0.001	2.9999739E+0	0.03	2.9991975E+0	0.2	2.9937846E+0
0.002	2.9999477E+0	0.05	2.9986396E+0	0.5	2.9779459E+0
0.005	2.9998688E+0	0.07	2.9980677E+0	0.7	2.9570522E+0
0.007	2.9998158E+0	0.09	2.9974839E+0	1.0	2.8441808E+0

Table 10: Inverse advection problem: c computed by the NLLSQ algorithm corresponding to several noise levels ϵ. Single sub-domain, NN $[2, 400, 1]$, $Q = 30 \times 30$, $Q_s = 100$, $R_m = 2.5$, λ_{mea}=1.

ϵ in the measurement data. Table 11 shows the c and u relative errors corresponding to different noise levels, computed by the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. Figure 25 shows the relative errors for c and u as a function of the noise level ϵ for several λ_{mea} values, illustrating the effect of scaling the measurement residual (see Remark 2.6). The computation results are observed to be quite robust to the noise in the measurement. For example, with 10% noise ($\epsilon = 0.1$) in the measurement, the relative errors of c computed by these methods are generally on the level of 0.1% (see Table 11). Scaling the measurement residual by $\lambda_{mea} < 1$ markedly improves the simulation accuracy in the presence of noise, while only slightly degrading the accuracy for the noise-free data; see Figure 25.

	NLLSQ			VarPro-F1			VarPro-F2		
ϵ	e_c	l^∞-u	l^2-u	e_c	l^∞-u	l^2-u	e_c	l^∞-u	l^2-u
0.0	1.78E-10	8.29E-8	3.23E-9	1.81E-10	8.29E-8	3.24E-9	3.49E-10	9.89E-8	5.13E-9
0.001	8.72E-6	4.91E-4	1.77E-4	8.73E-6	4.91E-4	1.77E-4	4.67E-6	6.91E-4	1.76E-4
0.005	4.37E-5	2.46E-3	8.85E-4	4.41E-5	2.46E-3	8.84E-4	2.26E-5	3.49E-3	8.80E-4
0.01	8.77E-5	4.91E-3	1.77E-3	8.81E-5	4.91E-3	1.77E-3	4.61E-5	7.00E-3	1.76E-3
0.05	4.53E-4	2.46E-2	8.84E-3	4.55E-4	2.45E-2	8.84E-3	2.49E-4	3.50E-2	8.79E-3
0.1	9.40E-4	4.92E-2	1.77E-2	9.54E-4	4.92E-2	1.77E-2	5.53E-4	6.95E-2	1.76E-2
0.5	7.35E-3	2.47E-1	8.90E-2	7.40E-3	2.47E-1	8.90E-2	5.42E-3	3.59E-1	8.86E-2
1.0	5.19E-2	5.04E-1	2.06E-1	5.19E-2	5.04E-1	2.06E-1	3.78E-2	8.03E-1	1.95E-1

Table 11: Inverse advection problem: c and u relative errors versus ϵ obtained with the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. Single sub-domain, NN $[2, 400, 1]$, $Q = 30 \times 30$, $Q_s = 100$, λ_{mea}=1; $R_m = 2.5$ with NLLSQ and VarPro-F1, and $R_m = 2.0$ with VarPro-F2.

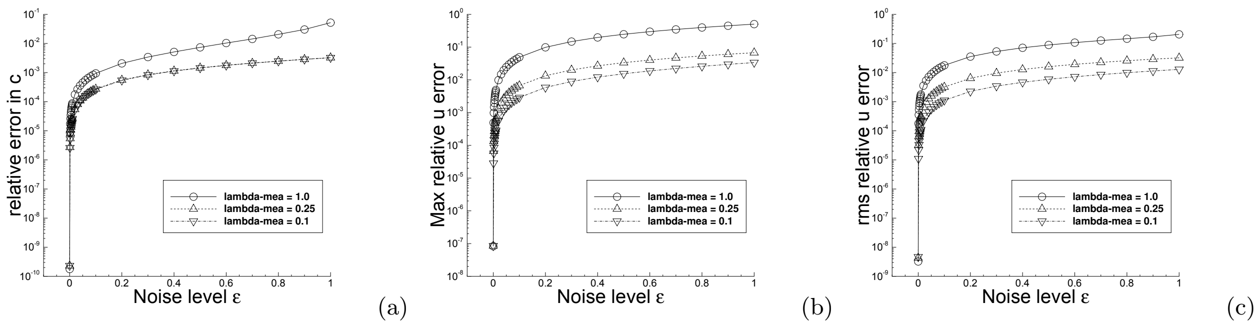


Figure 25: Inverse advection problem: c and u (l^∞-u, l^2-u) relative errors versus ϵ and λ_{mea} (scaling coefficient of measurement residual) obtained with the NLLSQ algorithm. Single sub-domain, NN $[2, 400, 1]$, $Q = 30 \times 30$, $Q_s = 100$, $R_m = 2.5$. These can be compared with the NLLSQ data in Table 11 for λ_{mea}=1.

Appendix D. Parametric Sine-Gordon Equation

This appendix provides a further test of the proposed method with the parametric Sine-Gordon equation. Consider the inverse parametric Sine-Gordon equation on the domain $(x, t) \in \Omega = [0, 1] \times [0, 1]$,

$$\frac{\partial^2 u}{\partial t^2} - \alpha_1 \frac{\partial^2 u}{\partial x^2} + \alpha_2 u + \alpha_3 \sin(u) = f(x, t), \tag{68a}$$

$$u(0, t) = g_1(t), \quad u(1, t) = g_2(t), \quad u(x, 0) = h_1(x), \quad \left.\frac{\partial u}{\partial t}\right|_{(x,0)} = h_2(x), \tag{68b}$$

$$u(\xi_i, \eta_i) = S(\xi_i, \eta_i), \quad (\xi_i, \eta_i) \in \mathbb{Y} \subset \Omega,\ 1 \leqslant i \leqslant NQ_s, \tag{68c}$$

where f is a prescribed source term, g_i $(i = 1, 2)$ and h_i $(i = 1, 2)$ are prescribed boundary and initial conditions, $\mathbb{Y}$ is the set of random measurement points, and the constants α_i $(i = 1, 2, 3)$ and the field $u(x, t)$ are the unknowns to be determined. We employ the following manufactured analytic solution in the tests,

$$\begin{cases} \alpha_1^{ex} = \alpha_2^{ex} = \alpha_3^{ex} = 1, \\ u_{ex}(x,t) = \left[\frac{5}{2}\cos\left(\pi x - \frac{2\pi}{5}\right) + \frac{3}{2}\cos\left(2\pi x + \frac{3\pi}{10}\right)\right]\left[\frac{5}{2}\cos\left(\pi t - \frac{2\pi}{5}\right) + \frac{3}{2}\cos\left(2\pi t + \frac{3\pi}{10}\right)\right]. \end{cases} \tag{69}$$

Accordingly, f, g_i $(i = 1, 2)$, and h_i $(i = 1, 2)$ are chosen such that the expressions in (69) satisfy (68a)–(68b). The measurement data are given by equation (48), in which u_{ex} is given in (69). The u errors are computed on a uniform 101×101 grid in each sub-domain. The notations here follow those of previous numerical examples.

Figure 26 shows distributions of the $u(x, t)$ solution and its point-wise absolute error in Ω obtained by the VarPro-F2 algorithm, with 50 random measurement points (no noise). The other parameter values are provided in the figure caption. We can observe a high accuracy in the solution, with the maximum error on the order of 10^{-8} in the domain. In this simulation the relative errors for the computed α_1, α_2 and α_3 are 2.07×10^{-10}, 7.54×10^{-9} and 2.39×10^{-8}, respectively.

The convergence of the simulation results obtained by the NLLSQ, VarPro-F1 and VarPro-F2 algorithms is demonstrated by the data in Table 12 and Figure 27. In these tests the number of training parameters (M) is varied systematically (no noise in measurement), while the other simulation parameters are fixed and their values are provided in the table/figure captions. Table 12 lists the computed α_i $(i = 1, 2, 3)$ values by the NLLSQ algorithm corresponding to a set of M. Figure 27 lists the relative errors of α_1, α_2 and α_3, as well as the l^∞ and l^2 norms of the relative error for $u(x, t)$, computed by NLLSQ, VarPro-F1 and VarPro-F2

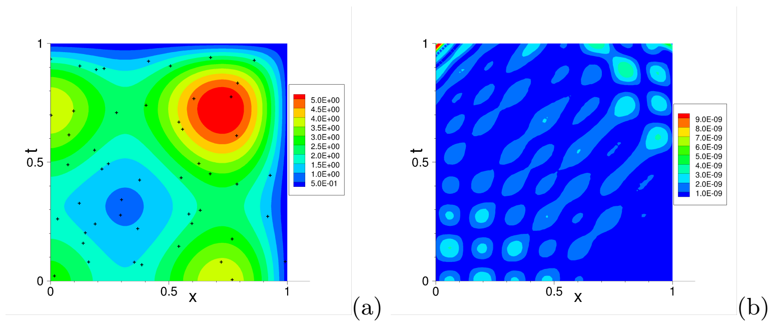


Figure 26: Inverse Sine-Gordon problem: distributions of (a) the VarPro-F2 solution for $u(x,t)$ and (b) its point-wise absolute error, with the measurement points shown as "+" symbols in (a). Single sub-domain, NN $[2,300,1]$, $Q = 25 \times 25$, $Q_s = 50$, $R_m = 1.3$, λ_{mea}=1, $\epsilon = 0$ (no noise in measurement data).

M	α_1	α_2	α_3
50	-3.525085809204E+1	-6.376776028198E+0	6.877670126056E+1
100	1.006414746681E+0	9.536423027578E-1	1.058573631607E+0
200	1.000000694463E+0	9.999830213887E-1	1.000056517879E+0
300	9.999999995066E-1	1.000000013620E+0	9.999999510014E-1
400	1.000000000001E+0	9.999999999962E-1	1.000000000096E+0

Table 12: Inverse Sine-Gordon problem: α_i $(i = 1,2,3)$ versus M (number of training parameters) obtained by the NLLSQ algorithm. Single sub-domain, NN $[2, M, 1]$, $Q = 25 \times 25$, $Q_s = 100$, $R_m = 1.5$, λ_{mea}=1, ϵ=0.

corresponding to different M. It is evident that the errors decrease exponentially with increasing number of training parameters with these algorithms.

Figure 28 illustrates the computational cost of the three algorithms for solving the inverse Sine-Gordon problem by showing the network training time as a function of the number of training parameters in the same set of tests as Figure 27. The data suggest a general quasi-linear growth in the training time with increasing number of training parameters. The VarPro-F2 algorithm is more costly than VarPro-F1, which in turn is more costly than NLLSQ for this problem. The training time with VarPro-F1 and VarPro-F2, especially VarPro-F2, is not quite regular. One can observe a fluctuation in the timing curves corresponding to these methods.

The effect of noise in the measurement data on the simulation accuracy is illustrated by Tables 13 and 14 for the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. The relative errors of α_1, α_2, α_3, and $u(x,t)$ corresponding to a range of noise levels are provided in these two tables. The other crucial simulation

	NLLSQ			VarPro-F1			VarPro-F2		
ϵ	e_{α_1}	e_{α_2}	e_{α_3}	e_{α_1}	e_{α_2}	e_{α_3}	e_{α_1}	e_{α_2}	e_{α_3}
0.0	1.93E-12	4.90E-11	1.35E-10	1.50E-12	4.42E-12	2.23E-11	3.61E-11	9.72E-10	3.02E-9
0.001	6.90E-4	2.66E-3	3.00E-3	6.88E-4	2.64E-3	3.09E-3	6.86E-4	2.63E-3	3.08E-3
0.005	3.44E-3	1.31E-2	1.45E-2	3.44E-3	1.32E-2	1.54E-2	3.43E-3	1.32E-2	1.54E-2
0.01	6.88E-3	2.63E-2	2.93E-2	6.86E-3	2.63E-2	3.08E-2	6.84E-3	2.62E-2	3.04E-2
0.05	3.38E-2	1.27E-1	1.39E-1	3.38E-2	1.30E-1	1.53E-1	3.39E-2	1.32E-1	1.60E-1
0.1	6.65E-2	2.49E-1	2.76E-1	6.65E-2	2.55E-1	3.05E-1	6.65E-2	2.57E-1	3.12E-1
0.5	2.67E-1	8.07E-1	5.90E-1	2.65E-1	7.69E-1	4.39E-1	2.66E-1	7.95E-1	5.41E-1
1.0	4.09E-1	1.01E+0	2.18E-1	4.12E-1	1.07E+0	5.43E-1	4.15E-1	1.10E+0	6.27E-1

Table 13: Inverse Sine-Gordon problem: α_1, α_2 and α_3 relative errors versus the noise level (ϵ) obtained by the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. Single sub-domain, NN: $[2,400,1]$, $Q = 30 \times 30$, $Q_s = 50$, λ_{mea}=1; $R_m = 1.5$ with NLLSQ, $R_m = 1.3$ with VarPro-F1 and VarPro-F2;

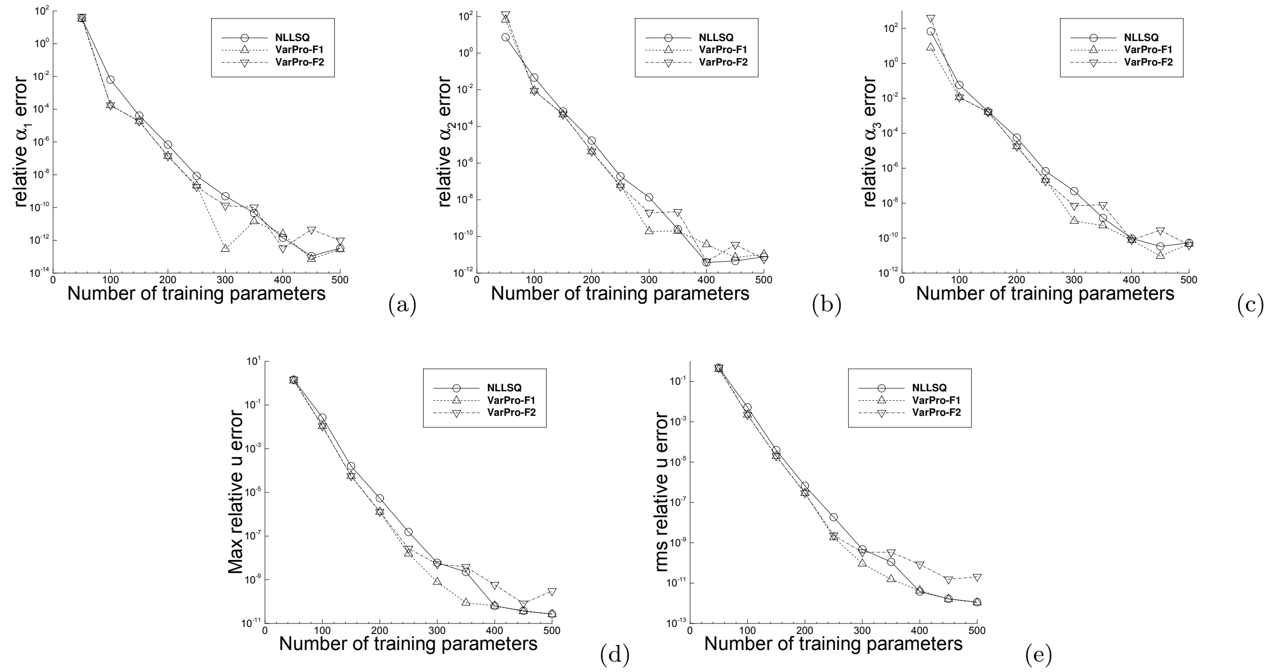


Figure 27: Inverse Sine-Gordon problem: relative errors of α_1, α_2, α_3 and u (l^∞-u, l^2-u) versus M (number of training parameters) obtained by the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. Single sub-domain, NN $[2, M, 1]$, $Q_s = 100$, $Q = 25 \times 25$, λ_{mea}=1, ϵ=0; $R_m = 1.5$ with NLLSQ, $R_m = 1.3$ with VarPro-F1 and VarPro-F2.

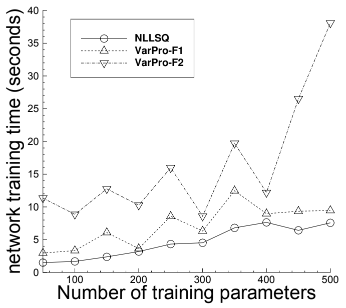


Figure 28: Inverse Sine-Gordon problem: Network training time as a function of the number of training parameters for the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. The test settings and parameters follow those of Figure 27.

	NLLSQ		VarPro-F1		VarPro-F2	
ϵ	l^∞-u	l^2-u	l^∞-u	l^2-u	l^∞-u	l^2-u
0.0	3.44E-11	4.16E-12	7.03E-11	5.01E-12	7.73E-10	1.57E-10
0.001	8.76E-4	3.71E-4	8.49E-4	3.70E-4	8.51E-4	3.69E-4
0.005	4.39E-3	1.86E-3	4.26E-3	1.85E-3	4.25E-3	1.85E-3
0.01	8.77E-3	3.71E-3	8.50E-3	3.70E-3	8.51E-3	3.70E-3
0.05	4.35E-2	1.87E-2	4.25E-2	1.87E-2	4.24E-2	1.86E-2
0.1	8.65E-2	3.78E-2	8.46E-2	3.77E-2	8.43E-2	3.76E-2
0.5	4.10E-1	1.95E-1	4.08E-1	1.95E-1	4.07E-1	1.95E-1
1.0	8.90E-1	3.83E-1	8.97E-1	3.84E-1	9.07E-1	3.86E-1

Table 14: Inverse Sine-Gordon problem: u relative errors versus ϵ obtained by the NLLSQ, VarPro-F1 and VarPro-F2 algorithms. Simulation settings and parameters follow those of Table 13.

	ϵ=0				ϵ=0.01			
method	e_α	l^∞-u	l^2-u	time(sec)	e_α	l^∞-u	l^2-u	time(sec)
PINN (Adam)	6.31E-3	1.08E-2	3.56E-3	134.5	5.53E-3	1.03E-2	3.30E-3	130.9
current (NLLSQ)	1.66E-8	3.66E-6	2.62E-7	11.5	9.72E-4	1.76E-3	5.26E-4	10.4

Table 15: Inverse Poisson problem: relative errors of α and u and the network training time (seconds) obtained by PINN (Adam) and the current NLLSQ algorithm. In both PINN and NLLSQ, Q=30×30, Q_s=100, Gaussian activation function. In PINN, neural network $[2, 30, 30, 30, 1]$; $20,000$ training epochs; $\gamma_{bc} = 0.99$; learning rate decreasing linearly from 0.01 to 1.0E-4 in first $10,000$ epochs, and fixed at 1.0E-4 afterwards. In NLLSQ, single sub-domain, neural network $[2, 500, 1]$, R_m=3.0, λ_{mea}=0.1.

parameters are provided in the caption of Table 13. The accuracy in the computation results deteriorates as the measurement data becomes more noisy. With 1% measurement noise ($\epsilon = 0.01$) the relative errors of the computed α_i ($i = 1, 2, 3$) are around $0.7 \sim 3\%$, and the relative error of u (l^2 norm) is around 0.4% with the three algorithms. With 5% measurement noise ($\epsilon = 0.05$) the relative errors of the computed α_i are around $3 \sim 15\%$ and the relative error of u (l^2 norm) is less than 2%.

Appendix E. Comparison with PINN

This appendix provides a comparison of the simulation results obtained by the current method (NLLSQ algorithm) and the physics-informed neural network (PINN) method [50] for several test problems. The PINN method is also implemented in Python based on the Tensorflow and Keras libraries. The PINN loss function consists of those contributions from the parametric PDE, the measurement, and the boundary/initial conditions (BC/IC). Let $\gamma_{bc} \in (0, 1)$ denote the penalty coefficient in front of the BC/IC loss term, and we employ $(1 - \gamma_{bc})$ as the penalty coefficient for the PDE and measurement loss terms. We have varied γ_{bc}, the learning rate schedule, and the random initialization for the weights/biases of PINN systematically. PINN is trained by the Adam optimizer. The PINN/Adam results reported below are the best we have obtained for these problems using PINN. We have also tried the L-BFGS optimizer with PINN, and its results for these inverse problems are quite poor and worse than the Adam results.

Tables 15 through 19 summarize the errors of the inverse parameters and the solution field, as well as the network training time, obtained by the current and the PINN methods for the inverse Poisson, advection, nonlinear Helmholtz, Burgers', and the Sine-Gordon problems. The table captions provide the respective parameter values in these simulations for the two methods. We observe that the current method produces more accurate results than PINN for both the inverse parameters and the solution field, and that the network training time of the current method is markedly smaller than that of PINN. For the noise-free data, the

	ϵ=0				ϵ=0.01			
method	e_c	l^∞-u	l^2-u	time(sec)	e_c	l^∞-u	l^2-u	time(sec)
PINN (Adam)	1.18E-5	7.18E-3	7.63E-4	133.5	1.47E-4	9.15E-3	1.67E-3	134.9
current (NLLSQ)	2.32E-10	8.51E-8	4.66E-9	29.6	2.61E-5	2.85E-4	1.10E-4	39.3

Table 16: Inverse advection problem: relative errors of c and u and the network training time (seconds) obtained by the PINN (Adam) and the current NLLSQ algorithm. In both PINN and NLLSQ, Q=30×30, Q_s=100, Gaussian activation function. In PINN, neural network $[2, 30, 30, 30, 30, 1]$; $20,000$ training epochs; γ_{bc} = 0.2; learning rate decreasing linearly from 0.01 to 1.0E-4 in first $10,000$ epochs, and fixed at 1.0E-4 afterwards. In NLLSQ, single sub-domain, neural network $[2, 400, 1]$, R_m=2.5, λ_{mea}=0.1.

noise level	method	e_{α_1}	e_{α_2}	l^∞-u	l^2-u	training-time(sec)
$\epsilon = 0$	PINN (Adam)	7.08E-1	2.68E-1	1.48E+0	5.65E-1	3049.2
	current (NLLSQ)	5.71E-9	3.05E-7	5.98E-8	1.49E-8	10.3
$\epsilon = 0.01$	PINN (Adam)	6.74E-1	7.76E-1	1.56E+0	6.79E-1	2742.9
	current (NLLSQ)	4.34E-3	7.13E-4	5.25E-3	2.38E-3	10.0

Table 17: Inverse nonlinear Helmholtz problem: relative errors of α_1, α_2 and u and the network training time (seconds) obtained by PINN (Adam) and the current NLLSQ algorithm. In both PINN and NLLSQ, Q=30×30, Q_s=100, Gaussian activation function. In PINN, neural network $[2, 30, 30, 30, 30, 30, 30, 1]$; $200,000$ training epochs; γ_{bc} = 0.99; learning rate decreasing linearly from 0.01 to 1.0E-4 in first $10,000$ epochs, and fixed at 1.0E-4 afterwards. In NLLSQ, single sub-domain, neural network $[2, 500, 1]$, R_m=2.25, λ_{mea}=0.25.

current method is significantly more accurate (typically by several orders of magnitude) than PINN.

Appendix F. Parameter Values in Algorithm 7 for Numerical Tests

Section 3.1 (Parametric Poisson Equation):

For NLLSQ:

In Figure 3, Table 1, Figures 4 and 5, Tables 3 and 4: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,2,1E-8,1.0,1,0).

In Table 2: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,5,1E-8,1.0,1,0).

For VarPro-F1:

In Figure 4 and Table 4: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,2,1E-8,1.0,1,0).

In Figure 5: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,5,1E-8,1.0,1,0).

In Table 2: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(100,2,1E-8,4.0,1,0).

For VarPro-F2:

In Figures 4 and 5, and Table 4: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,2,1E-8,1.0,1,0).

In Table 2: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,5,1E-8,1.0,1,0).

noise level	method	e_{α_1}	e_{α_2}	l^∞-u	l^2-u	training-time(sec)
$\epsilon = 0$	PINN (Adam)	2.40E-4	7.26E-4	1.59E-3	2.06E-4	529.0
	current (NLLSQ)	4.31E-10	2.33E-9	3.31E-9	5.86E-10	4.2
$\epsilon = 0.01$	PINN (Adam)	1.80E-3	2.66E-3	5.65E-3	7.83E-4	540.1
	current (NLLSQ)	1.12E-5	1.23E-4	9.87E-5	3.75E-5	6.4

Table 18: Inverse Burgers' problem: relative errors of α_1, α_2 and u and the network training time (seconds) obtained by PINN (Adam) and the current NLLSQ algorithm. In both PINN and NLLSQ, Q=30×30, Q_s=100, Gaussian activation function. In PINN, neural network $[2, 30, 30, 30, 30, 1]$; $50,000$ training epochs; γ_{bc} = 0.9; learning rate decreasing linearly from 0.01 to 1.0E-4 in first $10,000$ epochs, and fixed at 1.0E-4 afterwards. In NLLSQ, single sub-domain, neural network $[2, 400, 1]$, R_m=1.9, λ_{mea}=0.1.

noise level	method	e_{α_1}	e_{α_2}	e_{α_3}	l^∞-u	l^2-u	training-time(sec)
$\epsilon = 0$	PINN (Adam)	9.21E-3	2.30E-1	7.33E-1	1.86E-2	3.32E-3	1853.3
	current (NLLSQ)	7.65E-10	4.49E-9	6.60E-9	7.97E-10	3.50E-10	23.6
$\epsilon = 0.01$	PINN (Adam)	1.16E-2	1.35E-1	3.51E-1	1.18E-2	2.97E-3	1833.2
	current (NLLSQ)	5.45E-3	2.59E-2	5.09E-3	5.76E-3	2.63E-3	30.1

Table 19: Inverse Sine-Gordon problem: relative errors of α_i ($i = 1, 2, 3$) and u and the network training time (seconds) obtained by PINN (Adam) and the current NLLSQ algorithm. In both PINN and NLLSQ, Q=30×30, Q_s=100, Gaussian activation function. In PINN, neural network $[2, 30, 30, 30, 30, 1]$; $200,000$ training epochs; $\gamma_{bc} = 0.99$, learning rate decreasing linearly from 0.01 to 1.0E-4 in first $10,000$ epochs, and fixed at 1.0E-4 afterwards. In NLLSQ, single sub-domain, neural network $[2, 400, 1]$, R_m=1.5, λ_{mea}=0.01.

Section 3.2 (Parametric Nonlinear Helmholtz Equation):

For NLLSQ:

In Table 5, Figures 9 and 10, Table 6, Figure 12: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,2,1E-8,0.5,1,0).

For VarPro-F1:

In Figures 8, 9, 10 and 12: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,2,1E-8,0.5,1,0).

For VarPro-F2:

In Figuress 9, 10 and 12: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,0,1E-8,0.5,1,0); max-newton-iterations=15.

Section 3.3 (Parametric Viscous Burgers' Equation):

For NLLSQ:

In Figure 13, Table 7, Figures 14, 15 and 17: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,2,1E-8,0.5,1,0).

For VarPro-F1:

In Figures 14, 15 and 17: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,2,1E-8,1.0,1,0).

For VarPro-F2:

In Figures 14, 15 and 17: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,2,1E-12,1.0,0,$\boldsymbol{\xi}_0$); max-newton-iterations=15. Here $\boldsymbol{\xi}_0$ is a uniform random vector from [-1,1].

Section 3.4 (Helmholtz Equation with Inverse Variable Coefficient):

For NLLSQ:

In Figure 18, Table 8 and Figure 19: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,2,1E-8,1.0,1,0).

For VarPro-F1:

In Figure 19: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,2,1E-8,0.01,1,0).

For VarPro-F2:

In Figure 19: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(50,2,1E-8,0.5,1,0).

Appendix C (Parametric Advection Equation):

For NLLSQ:

In Figures 21, 22, 23 and 25, Tables 9, 10, and 11: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,10,1E-8,10.0,0,$\boldsymbol{\vartheta}_0$).

For VarPro-F1:

In Figure 22: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,2,1E-8,5.0,0,$\boldsymbol{\vartheta}_0$).

In Figure 23 and Table 11: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(100,5,1E-8,5.0,0,$\boldsymbol{\vartheta}_0$).

For VarPro-F2:

In Figure 22: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,2,1E-8,5.0,0,$\boldsymbol{\xi}_0$).
In Figure 23: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,2,1E-8,1.0,1,$\boldsymbol{\xi}_0$).
In Table 11: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,5,1E-8,1.0,1,$\boldsymbol{\xi}_0$).
In the above, $\boldsymbol{\xi}_0$ is a uniform random vector from [-1,1]. $\boldsymbol{\vartheta}_0$ is a uniform random vector generated by the lines 7 through 14 of Algorithm 7 with the δ as specified above and $\eta = 0$.

Appendix D (Parametric Sine-Gordon Equation):

For NLLSQ:
In Tables 12, 13 and 14, and Figure 27: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,5,1E-8,5.0,0,0).
For VarPro-F1:
In Figure 27, and Tables 13 and 14: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,5,1E-8,5.0,0,0).
For VarPro-F2:
In Figures 26 and 27, and Tables 13 and 14: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,5,1E-8,1.0,0,0); max-newton-iterations=15.

Appendix E (Comparison with PINN):

For NLLSQ:
In Table 15: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,5,1E-8,1.0,1,0).
In Table 16: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,10,1E-8,10.0,0,$\boldsymbol{\vartheta}_0$). Here $\boldsymbol{\vartheta}_0$ is a uniform random vector generated by the lines 7 through 14 of Algorithm 7 with the δ as specified here and $\eta = 0$.
In Table 17: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,2,1E-8,0.5,1,0).
In Table 18: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,2,1E-8,0.5,1,0).
In Table 19: (max-nllsq-iterations,max-sub-iterations,ε,δ,η,$\boldsymbol{\theta}_0$)=(80,5,1E-8,5.0,0,0).

References

[1] J. Berg and K. Nystrom. Neural network augmented inverse problems for PDEs. *arXiv:1712.09685*, 2018.

[2] J. Berg and K. Nystrom. Data-driven discovery of PDEs in complex datasets. *Journal of Computational Physics*, 384:239–252, 2019.

[3] A. Bjorck. *Numerical Methods for Least Squares Problems*. SIAM, 1996.

[4] J. Bongard and H. Lipton. Automated reverse engineering of nonlinear dynamical systems. *Proceedings of National Academy of Sciences USA*, 104:9943–9948, 2007.

[5] G.-J. Both, S. Choudhury, P. Sens, and R. Kusters. DeepMoD: deep learning for model discovery in noisy data. *Journal of Computational Physics*, 428:109985, 2021.

[6] M.A. Branch, T.F. Coleman, and Y. Li. A subspace, interior, and conjugate gradient method for large-scale bound-constrained minimization problems. *SIAM Journal on Scientific Computing*, 21:1–23, 1999.

[7] S.L. Brunton, J.L. Proctor, and J.N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of National Academy of Sciences USA*, 113:3932–3937, 2016.

[8] R.H. Byrd, R.B. Schnabel, and G.A. Shultz. Approximate solution of the trust region problem by minimization over two-dimensional subspaces. *Math. Programming*, 40:247–263, 1988.

[9] S. Cai, Z. Wang, F. Fuest, Y.J. Jeon, C. Gray, and G.E. Karniadakis. Flow over an espresso cup:

inferring 3-d velocity and pressure fields from tomographic background oriented schileren via physics-informed neural networks. *Journal of Fluid Mechanics*, 915:A102, 2021.

[10] F. Calabro, G. Fabiani, and C. Siettos. Extreme learning machine collocation for the numerical solution of elliptic PDEs with sharp gradients. *Computer Methods in Applied Mechanics and Engineering*, 387:114188, 2021.

[11] Y. Chen, L. Lu, G.E. Karniadakis, and L.D. Negro. Physics-informed neural networks for inverse problems in nano-optics and metamaterials. *Optics Express*, 28:11618–11633, 2020.

[12] S. Dong. An efficient algorithm for incompressible N-phase flows. *Journal of Computational Physics*, 276:691–728, 2014.

[13] S. Dong. Physical formulation and numerical algorithm for simulating N immiscible incompressible fluids involving general order parameters. *Journal of Computational Physics*, 283:98–128, 2015.

[14] S. Dong. Wall-bounded multiphase flows of N immiscible incompressible fluids: consistency and contact-angle boundary condition. *Journal of Computational Physics*, 338:21–67, 2017.

[15] S. Dong. Multiphase flows of N immiscible incompressible fluids: a reduction-consistent and thermodynamically-consistent formulation and associated algorithm. *Journal of Computational Physics*, 361:1–49, 2018.

[16] S. Dong and Z. Li. Local extreme learning machines and domain decomposition for solving linear and nonlinear partial differential equations. *Computer Methods in Applied Mechanics and Engineering*, 387:114129, 2021. (also arXiv:2012.02895).

[17] S. Dong and Z. Li. A modified batch intrinsic plascity method for pre-training the random coefficients of extreme learning machines. *Journal of Computational Physics*, 445:110585, 2021. (also arXiv:2103.08042).

[18] S. Dong and J. Yang. Numerical approximation of partial differential equations by a variable projection method with artificial neural networks. *Computer Methods in Applied Mechanics and Engineering*, 398:115284, 2022. (also arXiv:2201.09989).

[19] S. Dong and J. Yang. On computing the hyperparameter of extreme learning machines: algorithms and applications to computational PDEs, and comparison with classical and high-order finite elements. *Journal of Computational Physics*, 463:111290, 2022. (also arXiv:2110.14121).

[20] V. Dwivedi, N. Parashar, and B. Srinivasan. Distributed learning machines for solving forward and inverse problems in partial differential equations. *Neurocomputing*, 420:299–316, 2021.

[21] V. Dwivedi and B. Srinivasan. Physics informed extreme learning machine (pielm) – a rapid method for the numerical solution of partial differential equations. *Neurocomputing*, 391:96–118, 2020.

[22] G. Fabiani, F. Calabro, L. Russo, and C. Siettos. Numerical solution and bifurcation analysis of nonlinear partial differential equations with extreme learning machines. *Journal of Scientific Computing*, 89:44, 2021.

[23] G.H. Golub and V. Pereyra. The differentiation of pseudo-inverse and nonlinear least squares problems whose variables separate. *SIAM J. Numer. Anal.*, 10:413–432, 1973.

[24] G.H. Golub and V. Pereyra. Separable nonlinear least squares: the variable projection method and its applications. *Inverse Problems*, 19:R1–R26, 2003.

[25] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: theory and applications. *Neurocomputing*, 70:489–501, 2006.

[26] G.B. Huang, L. Chen, and C.-K. Siew. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Transactions on Neural Networks*, 17:879–892,

2006.

[27] B. Igelnik and Y.H. Pao. Stochastic choice of basis functions in adaptive function approximation and the functional-link net. *IEEE Transactions on Neural Networks*, 6:1320–1329, 1995.

[28] A.D. Jagtap and G.E. Karniadakis. Extended physics-informed neural network (XPINNs): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. *Communications in Computational Physics*, 28:2002–2041, 2020.

[29] A.D. Jagtap, E. Kharazmi, and G.E. Karniadakis. Conservative physics-informed neural networks on discrete domains for conservation laws: applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 365:113028, 2020.

[30] A.D. Jagtap, Z. Mao, N. Adams, and G.E. Karniadakis. Physics-informed neural networks for inverse problems in supersonic flows. *Journal of Computational Physics*, 466:111402, 2022.

[31] G.E. Karniadakis, G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3:422–440, 2021.

[32] L. Kaufman. A variable projection method for solving separable nonlinear least squares problems. *BIT*, 15:49–57, 1975.

[33] C.-T. Kim and J.-J. Lee. Training two-layered feedforward networks with variable projection method. *IEEE Transactions on Neural Networks*, 19:371–375, 2008.

[34] D. Li, K. Xu, J.M. Harris, and E. Darve. Coupled time-lapse full-waveform inversion for subsurface flow problems using intrusive automatic differentiation. *Water Resour. Res.*, 56:e2019WR027032, 2020.

[35] Z. Long, Y. Lu, and B. Dong. PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, 399:108925, 2019.

[36] L. Lu, X. Meng, Z. Mao, and G.E. Karniadakis. DeepXDE: a deep learning library for solving differential equations. *SIAM Review*, 63:208–228, 2021.

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

[38] A. Mathews, M. Francisquez, J. Hghes, and D. Hatch. Uncovering edge plasma dynamics via deep learning from partial observations. *arXiv:2009.05005*, 2020.

[39] X. Meng and G.E. Karniadakis. A composite neural network that learns form multi-fidelity data: application to function approximation and inverse pde problems. *Journal of Computational Physics*, 401:109020, 2020.

[40] D. Needell, A.A. Nelson, R. Saab, and P. Salanevich. Random vector functional link networks for function approximation on manifolds. *arXiv:2007.15776*, 2020.

[41] E. Newman, J. Chung, M. Chung, and L. Ruthotto. SlimTrain – a stochastic approximation method for training separable deep neural networks. *SIAM J. Sci. Comput.*, 44:A2322–A2348, 2022.

[42] E. Newman, L. Ruthotto, J. Hart, and B. van Bloemen Waanders. Train like a (Var)Pro: Efficient training of neural networks with variable projection. *SIAM J. Math. Data Sci.*, 3:1041–1066, 2021.

[43] N. Ni and S. Dong. Numerical computation of partial differential equations by hidden-layer concatenated extreme learning machine. *Journal of Scientific Computing*, 95:35, 2023.

[44] D.P. O'Leary and B.W. Rust. Variable projection for nonlinear least squares problems. *Comput. Optim. Appl.*, 54:579–593, 2013.

[45] S. Panghal and M. Kumar. Optimization free neural network approach for solving ordinary and partial differential equations. *Engineering with Computers*, 37:2989–3002, 2021.

[46] Y.H. Pao, G.H. Park, and D.J. Sobajic. Learning and generalization characteristics of the random vector

functional-link net. *Neurocomputing*, 6:163–180, 1994.

[47] R.G. Patel, I. Manickam, N.A. Trask, M.A. Wood, M. Lee, I. Tomas, and E.C. Cyr. Thermodynamically consistent physics-informed neural networks for hyperbolic systems. *Journal of Computational Physics*, 449:110754, 2022.

[48] V. Pereyra, G. Scherer, and F. Wong. Variable projections neural network training. *Mathematics and Computers in Simulation*, 73:231–243, 2006.

[49] M. Raissi and G.E. Karniadakis. Hidden physics models: machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, 357:125–141, 2018.

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

[51] M. Raissi, A. Yazdani, and G.E. Karniadakis. Hidden fluid mechanics: learning velocity and pressure fields from flow visualizations. *Science*, 367:1026–1030, 2020.

[52] S. Rudy, A. Alla, S.L. Brunton, and J.N. Kutz. Data-driven identification of parametric partial differential equations. *SIAM J. Applied Dynamical Systems*, 18:643–660, 2019.

[53] S.H. Rudy, S.L. Brunton, J.L. Proctor, and J.N. Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 3:e1602614, 2017.

[54] A. Ruhe and P.A. Wedin. Algorithms for separable nonlinear least squares problems. *SIAM Review*, 22:318–337, 1980.

[55] H. Schaeffer. Learning partial differential equations via data discovery and sparse optimization. *Proc. R. Soc. A*, 473:20160446, 2017.

[56] E. Schiassi, A. D'Ambrosio, M. De Florio, R. Furfaro, and F. Curti. Physics-informed extreme theory of functional connections applied to data-driven parameters discovery of epidemiological compartmental models. *arXiv:2008.05554*, 2020.

[57] E. Schiassi, R. Furfaro, C. Leake, M. De Florio, H. Johnson, and D. Mortari. Extreme theory of functional connections: a fast physics-informed neural network method for solving ordinary and partial differential equations. *Neurocomputing*, 457:334–356, 2021.

[58] M. Schmidt and H. Lipton. Distilling free-form natural laws from experimental data. *Science*, 324:81–85, 2009.

[59] J. Sjoberg and M. Viberg. Separable nonlinear least squares minimization - possible improvements for neural net fitting. *Neural Networks for Signal Processing VII. Proceedings of IEEE Signal Processing Workshop*, 1997.

[60] A.M. Tartakovsky, C.O. Marrero, P. Perdikaris, G.D. Tartakovsky, and D. Barajas-Solano. Physics-informed deep neural networks for learning parameters and constitutive relationships in subsurface flow problems. *Water Resource Research*, 56:e2019WR026731, 2020.

[61] K. Weigl and M. Berthod. Neural networks as dynamical bases in function space. *Report No 2124, INRIA, Sophis-Antipolis, France*, 1993. URL: https://hal.inria.fr/inria-00074548/document.

[62] K. Weigl and M. Berthod. Projection learning: alternative approach to the computation of the projection. *Proc. European Symp. on Artificial Neural Networks, Brussels, Belgium*, pages 19–24, 1994.

[63] K. Weigl, G. Giraudon, and M. Berthod. Application of projection learning to the detection of urban areas in SPOT satellite images. *Report No 2143, INRIA, Sophia-Antipolis, France*, 1993. URL: https://hal.inria.fr/inria-00074529.

[64] K. Wu and D. Xiu. Data-driven deep learning of partial differential equations in modal space. *Journal*

of Computational Physics, 408:109307, 2020.

[65] L. Yang, X. Meng, and G.E. Karniadakis. B-PINNs: Bayesian physics-informed neural netwotks for forward and inverse pde problems with noisy data. *Journal of Computational Physics*, 425:109913, 2021.

[66] Y. Yang, M. Hou, and J. Luo. A novel improved extreme learning machine algorithm in solving ordinary differential equations by legendre neural network methods. *Advances in Differential Equations*, 469:1–24, 2018.

[67] Z. Yang and S. Dong. Multiphase flows of N immiscible incompressible fluids: an outflow/open boundary condition and algorithm. *Journal of Computational Physics*, 366:33–70, 2018.

[68] L. Yuan, Y.-Q. Ni, X.-Y. Deng, and S. Hao. A-PINN: auxiliary physics informed neural networks for forward and inverse problems of nonlinear integro-differential equations. *Journal of Computational Physics*, 462:111260, 2022.

[69] S. Zhang and G. Lin. Robust data-driven discovery of governing physical laws with error bars. *Prov. R. Soc. A*, 474:20180305, 2018.