Euro. Jnl of Applied Mathematics (2021), *vol.* 32, *pp.* 397–420 © The Author(s), 2020. Published by Cambridge University Press.
doi:10.1017/S0956792520000169

Learning and meta-learning of stochastic advection–diffusion–reaction systems from sparse measurements

XIAOLI CHEN[1,2], JINQIAO DUAN[3] and GEORGE EM KARNIADAKIS[2,4]

[1] *Center for Mathematical Sciences and School of Mathematics and Statistics, Huazhong University of Science and Technology, Wuhan 430074, China*
email: xlchen@hust.edu.cn
[2] *Division of Applied Mathematics, Brown University, Providence, RI 02912, USA*
email: george_karniadakis@brown.edu
[3] *Department of Applied Mathematics, Illinois Institute of Technology, Chicago, IL 60616, USA*
email: duan@iit.edu
[4] *Pacific Northwest National Laboratory, Richland, WA 99354, USA*

(*Received 18 October 2019*; *revised 9 March 2020*; *accepted 12 May 2020*; *first published online 15 June 2020*)

Physics-informed neural networks (PINNs) were recently proposed in [18] as an alternative way to solve partial differential equations (PDEs). A neural network (NN) represents the solution, while a PDE-induced NN is coupled to the solution NN, and all differential operators are treated using automatic differentiation. Here, we first employ the standard PINN and a stochastic version, sPINN, to solve forward and inverse problems governed by a non-linear advection–diffusion–reaction (ADR) equation, assuming we have some sparse measurements of the concentration field at random or pre-selected locations. Subsequently, we attempt to optimise the hyper-parameters of sPINN by using the Bayesian optimisation method (meta-learning) and compare the results with the empirically selected hyper-parameters of sPINN. In particular, for the first part in solving the inverse deterministic ADR, we assume that we only have a few high-fidelity measurements, whereas the rest of the data is of lower fidelity. Hence, the PINN is trained using a composite *multi-fidelity* network, first introduced in [12], that learns the correlations between the multi-fidelity data and predicts the unknown values of diffusivity, transport velocity and two reaction constants as well as the concentration field. For the stochastic ADR, we employ a Karhunen–Loève (KL) expansion to represent the stochastic diffusivity, and arbitrary polynomial chaos (aPC) to represent the stochastic solution. Correspondingly, we design multiple NNs to represent the *mean* of the solution and learn each aPC mode separately, whereas we employ a separate NN to represent the *mean* of diffusivity and another NN to learn all modes of the KL expansion. For the inverse problem, in addition to stochastic diffusivity and concentration fields, we also aim to obtain the (unknown) deterministic values of transport velocity and reaction constants. The available data correspond to 7 *spatial* points for the diffusivity and 20 *space–time* points for the solution, both sampled 2000 times. We obtain good accuracy for the deterministic parameters of the order of 1–2% and excellent accuracy for the mean and variance of the stochastic fields, better than three digits of accuracy. In the second part, we consider the previous stochastic inverse problem, and we use Bayesian optimisation to find five hyper-parameters of sPINN, namely the width, depth and learning rate of two NNs for learning the modes. We obtain much deeper and wider optimal NNs compared to the manual tuning, leading to even better accuracy, i.e., errors less than 1% for the deterministic values, and about an order of magnitude less for the stochastic fields.

Key words: Physics-informed neural networks, arbitrary polynomial chaos, multi-fidelity data, Karhunen–Loève expansion, uncertainty quantification, Bayesian optimisation, inverse problems

2020 Mathematics Subject Classification: 68T05

1 Introduction

In classical inverse problems, we assume that we have a lot of measurements for the state variables, and we aim to obtain some unknown parameters or space/time-dependent material properties by formulating appropriate objective functions and employing the necessary regularisation techniques. However, in many practical problems, e.g., in sub-surface transport [23, 1], we have to deal with a *mixed* problem, as we typically have some measurements on the material properties and some measurements on the state variables. Here, we consider such 'mixed' problems for a non-linear advection–diffusion–reaction (ADR) describing a concentration field, and we formulate new algorithms inspired by recent developments in machine learning. In particular, we will assume that we have a stochastic diffusivity field, which is partially known only at a few points, and hence we aim to determine the entire stochastic field from only sparse measurements of the concentration field. Moreover, we will assume that the constant transport velocity in the advection term is unknown and that the reaction term is parameterised by two unknown parameters. Hence, the problem set-up we consider is as follows: determine the entire stochastic diffusivity and stochastic concentration fields as well as three (deterministic) parameters from a few multi-fidelity measurements of the concentration field at random points in space time. For simplicity, we will refer to this 'mixed' problem as 'inverse' problem in the following.

The aforementioned problem set-up could be tackled by using Bayesian optimisation (BO) methods as we have done in previous work for other problems, e.g., see [19, 15], but to overcome open issues related to strong nonlinearity and scalability, here we will employ neural networks (NNs) following the works of [7, 3, 4, 20, 17], and in particular the physics-informed neural network (PINN) approach introduced in [18]. In addition, we have to model stochastic fields and in order to avoid optimising expensive Bayesian NNs, we will instead model stochasticity using polynomial chaos expansions following the work of [25]. Another important consideration is how to fuse data of variable fidelity, as some data may be collected by a few high-resolution sensors, whereas the majority of the data may be collected by lower fidelity sensors. This, in turn, implies that we have to train the NN or the PINN with multi-fidelity data, and to this end, we will employ a new composite network recently proposed by [12]. Finally, because of the complexity of the proposed NNs, we also introduce an automated method to optimise the hyper-parameters of PINN using a simple version of meta-learning. Meta-learning is a methodology considered with 'learning to learn' machine learning algorithms and is model-agnostic and compatible with any model trained with gradient descent [6]. In this paper, we use the BO, e.g., see [2, 21, 22].

In order to make progress towards the final goal and to evaluate each of the algorithmic steps separately, we will use a hierarchical approach by introducing complexity incrementally. We will start with multi-fidelity deterministic problems using PINNs, and subsequently, we will introduce randomness in the data and present the stochastic formulation. This will require us to design multiple NNs that learn in modal space. Subsequently, we will formulate an additional optimisation problem for five of the most important hyper-parameters of the multi-NN design and compare its performance with the performance obtained previously by manual tuning.

The organisation of this paper is as follows. In Section 2, we introduce the PINN to solve the deterministic partial differential equation (PDE) and the sPINN to solve the stochastic PDE (SPDE). In Section 3, we present the results of PINN for solving the inverse problem of the deterministic ADR equation. In Section 4, we provide the results of the sPINN method for both the forward and inverse problem. Finally, we employ meta-learning for the last stochastic inverse problem, and we conclude with a short summary.

2 Methodology

2.1 PINNs for deterministic PDEs

First, we briefly review the type of deep neural networks (DNNs) to solve deterministic PDEs and the corresponding inverse problem [18, 14]. The PDE can have the general form:

$$u_t + \mathcal{N}[u(x,t);\eta] = 0,\ x \in \mathcal{D},\ t \in [0,T], \tag{2.1}$$

with the initial and boundary conditions:

$$\begin{aligned} &u(x,0) = u_0(x), && x \in \mathcal{D},\\ &\mathbb{B}_X[u(x,t)] = \tilde{u}(x,t), && x \in \partial\mathcal{D},\ t \in (0,T], \end{aligned} \tag{2.2}$$

where $u(x,t)$ denotes the solution, $u_0(x)$ is the initial condition, $\tilde{u}(x,t)$ is the boundary condition, $\mathcal{N}[\cdot]$ is a non-linear differential operator, η is the parameter in the PDE, $\mathcal{D}$ is a subset of $\mathbb{R}$ and $\partial\mathcal{D}$ is the boundary of $\mathcal{D}$.

The solution, denoted by $u_{NN}(x,t;w,b)$, is constructed as a NN approximation of $u(x,t)$; DNN has the weights (w) and biases (b). We can couple it to another DNN induced by the PDE residual f_{NN} computed based on the NN solution $u_{NN}(x,t;w,b)$ and corresponding to equation (2.1); also, the residual $f(x,t)$ is given by equation (2.1), i.e.,

$$f = u_t + \mathcal{N}[u(x,t);\eta]. \tag{2.3}$$

The inputs of the DNN are the spatial coordinates and time (x,t), while the output is u_{NN}, which has the same dimension as the input. For the output of u_{NN}, we use automatic differentiation techniques to compute all derivatives of the non-linear differential operator (physics part). There are two restrictions on u_{NN}. First, the solution of u_{NN} should be close to the observations u at the training points. Second, every u_{NN} should comply with the physics imposed by equation (2.1). The second part is achieved by defining a residual network:

$$f_{NN}(x,t;w,b,\eta) = (u_{NN})_t + \mathcal{N}[u_{NN}(x,t;w,b);\eta], \tag{2.4}$$

which is computed from u_{NN} straightforwardly with automatic differentiation. This residual network network f_{NN} shares the same parameters (w,b) with the network for u_{NN} and should output a value close to 0 for any input $(x,t) \in \mathcal{D} \times [0,T]$. During training, the shared parameters (w,b) are adjusted by back-propagating the error obtained by minimising a loss function that is the weighted sum of the above two constraints. A sketch of the PINN, consisted of the physics-uninformed and physics-informed DNNs, is shown in Figure 1.

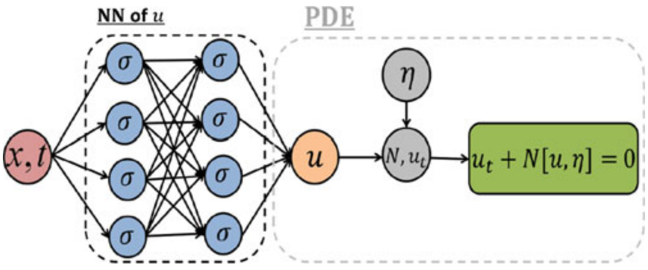

FIGURE 1. Schematic of the PINN for solving deterministic PDEs.

The PINN loss function is defined as

$$MSE = MSE_u + MSE_f, \tag{2.5}$$

where

$$MSE_u = \frac{1}{N_u}\sum_{i=1}^{N_u}(u_{NN}(x_i, t_i; w, b) - u(x_i, t_i))^2,$$
$$MSE_f = \frac{1}{N_f}\sum_{j=1}^{N_f}(f_{NN}(x_j, t_j; w, b, \eta))^2. \tag{2.6}$$

Here, $(x_i, t_i, u_{NN}(x_i, t_i; w, b))_{i=1}^{N_u}$ denotes the initial and boundary conditions of u for the forward problem as well as the training data of u for the inverse problem. The data $u(x_i, t_i)$ are the observation data of u, while $\{(x_j, t_j)\}_{j=1}^{N_f}$ denotes the residual points for penalising $f(x, t)$.

2.2 sPINNs: PINNs for SPDEs

Next, we briefly review a stochastic version, based on the arbitrary polynomial chaos (aPC) [24, 16] to represent stochasticity, and combine it with a PINN, following the method first introduced in [25]. We consider the following SPDE:

$$u_t + \mathcal{N}[u(x, t; \omega); k(x; \omega)] = 0, \ x \in \mathcal{D}, \ t \in (0, T], \ \omega \in \Omega, \tag{2.7}$$

with the initial and boundary conditions:

$$\begin{aligned} u(x, 0; \omega) &= u_0(x), & x &\in \mathcal{D}, \\ \mathbb{B}_X[u(x, t; \omega)] &= 0, & x &\in \partial\mathcal{D}, \ t \in (0, T]. \end{aligned} \tag{2.8}$$

Here, Ω is the random space. In the following, we describe how to use sPINN to solve stochastic inverse problems since for the forward problem the method is straightforward. We assume that we have N_k sensors for $k(x; \omega)$ placed at $\{x_k^{(i)}\}_{i=1}^{N_k}$ and N_u sensors for $u(x, t; \omega)$ placed at $\{(x_u^{(i)}, t_u^{(i)})\}_{i=1}^{N_u}$. We also choose at random N_f locations $\{(x_f^{(i)}, t_f^{(i)})\}_{i=1}^{N_f}$ that are used to compute the residual of equation (2.7). We assume that the observation data of k are $\{k(x_i; \omega_s)\}$ (denoted by $\{k_s^i\}$), where $i = 1, 2, ..., N_k$, and $s = 1, 2, ..., N$. The observation data of u are $\{u(x_j, t_j; \omega_s)\}$ (denoted by $\{u_s^j\}$), where $j = 1, 2, ..., N_u$, and $s = 1, 2, ..., N$. Here, N denotes the number of samples available for a specific location, and for simplicity we take that to be the same both for $k(x; \omega)$ and for $u(x, t; \omega)$ for all locations.

One of the key questions for the inverse stochastic problem is what type of randomness we encounter in the data and how we represent the stochastic fields. We consider a general setting, i.e., instead of the classical inverse problem where we are given data on $u(x,t;\omega)$ but not on $k(x;\omega)$, here we assume that we have some data on u and some data on k. Hence, in order to choose the type of the distribution required to represent our random variables so that we employ aPC, we use the data samples of either u or k. Here, we assume that we have $N_k = 7$ sensors for k so we can determine the random variables ξ from the k data, as we explain below, see equations (2.11)–(2.12).

We choose M sensors of k to compute the random variables ξ, where $M \le N_k$. Denote the observations of k as $k_1 = (k_1(i,j))$, where the element of $k_1(i,j)$ is the value of $k(x_i;\omega_j)$, and the size of k_1 is $M \times N$, where N is the number of samples. K be the $M \times M$ covariance matrix for the observation data of k_1, i.e.,

$$K_{i,j} = Cov(k_1^{(i)}, k_1^{(j)}). \tag{2.9}$$

Let λ_i and υ_i be the ith eigenvalue and its corresponding normalised eigenvector of K. Using principal component analysis, we obtain

$$K = V^T \Lambda V, \tag{2.10}$$

where $V = [\upsilon_1, \upsilon_2, ..., \upsilon_M]$ is an orthonormal matrix and $\Lambda = diag(\lambda_1, \lambda_2, ..., \lambda_M)$ is a diagonal matrix. The random variable ξ satisfies the following equation:

$$k_1 = \bar{k}_1 + V\sqrt{\Lambda}\xi, \tag{2.11}$$

where $\bar{k}_1$ is the mean of k_1.

Hence,

$$\xi = \sqrt{\Lambda}^{-1} V^T(k_1 - \bar{k}_1), \tag{2.12}$$

where each row of ξ is an uncorrelated random vector, and the size of ξ is $M \times N$.

In the continuous case, the diffusion term $k(x;\omega)$ can be approximated by

$$k_{NN}(x;\omega_j) = k_0(x) + \sum_{i=1}^{M} k_i(x)\sqrt{\lambda_i}\xi_{i,j}, \quad j = 1, ..., N. \tag{2.13}$$

Correspondingly, the solution u at the jth snapshot can be approximated by

$$u_{NN}(x,t;\omega_j) \approx \sum_{\alpha=0}^{P} u_\alpha(x,t)\psi_\alpha(\xi_j), \tag{2.14}$$

where $\{\psi_\alpha\}_{\alpha=1}^{P}$ are the set of multivariate orthonormal polynomial basis and the highest polynomial order is r. The parameter P, r and M satisfy the following formula:

$$P + 1 = \frac{(r+M)!}{r!M!}. \tag{2.15}$$

Similar to the PINN method, we construct the residual network via automatic differentiation and by substituting $u(t,x;\omega)$ and $k(x,\omega)$ in equation (2.7) with $u_{NN}(x,t;\omega)$ and $k_{NN}(x;\omega)$. A sketch of the stochastic PINN (sPINN) is shown in Figure 2.

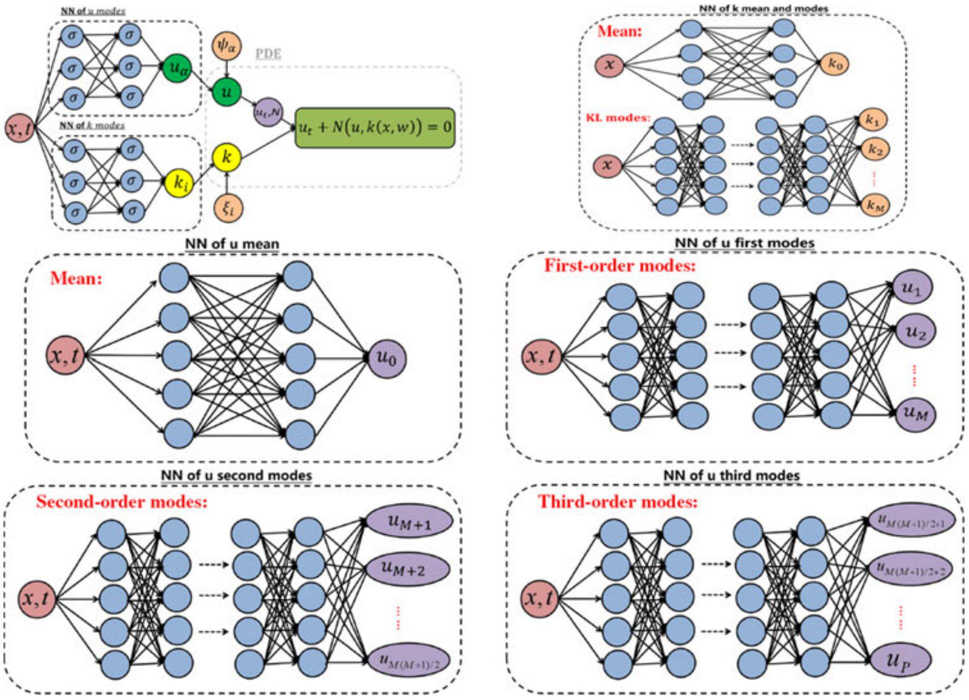

FIGURE 2. Schematic of the sPINN for solving SPDEs. Top left: A composite NN consisting of multiple NNs for computing the mean and modes of the stochastic diffusivity and the solution. The random variable ξ is obtained by the observation data on k, (see equation (2.12)). Top right: Two separate NNs for the mean and all the modes of diffusivity. Middle left: NN to compute the mean of the solution. Middle right and bottom row: Separate NN to compute the modes of the solution. Adopted from reference [25].

The loss function or mean square error (MSE) is defined as

$$MSE = MSE_u + MSE_k + MSE_f, \tag{2.16}$$

where

$$MSE_u = \frac{1}{N*N_u}\sum_{j=1}^{N}\sum_{i=1}^{N_u}[u_{NN}(x_u^{(i)}, t_u^{(i)}; \omega_j) - u(x_u^{(i)}, t_u^{(i)}; \omega_j)]^2,$$

$$MSE_k = \frac{1}{N*N_k}\sum_{j=1}^{N}\sum_{i=1}^{N_k}[k_{NN}(x_k^{(i)}; \omega_j) - k(x_k^{(i)}; \omega_j)]^2,$$

$$MSE_f = \frac{1}{N*N_f}\sum_{j=1}^{N}\sum_{i=1}^{N_f}[f_{NN}(x_f^{(i)}, t_f^{(i)}; \omega_j)]^2.$$

Here, $(x_u^{(i)}, t_u^{(i)}, u_{NN}(x_u^{(i)}, t_u^{(i)}; \omega_j))_{i,j=1}^{N_u,N}$ denotes the training data of u, while the data $(x_u^{(i)}, t_u^{(i)}, u(x_u^{(i)}, t_u^{(i)}; \omega_j))_{i,j=1}^{N_u,N}$ are the observation data of u; also, $(x_k^{(i)}, k_{NN}(x_k^{(i)}; \omega_j))_{i,j=1}^{N_k,N}$ denote the training data of k. The data $(x_k^{(i)}, k(x_k^{(i)}; \omega_j))_{i,j=1}^{N_k,N}$ are the observation data of k, while $\{(x_f^{(i)}, t_f^{(i)}; \omega_j)\}_{i,j=1}^{N_f,N}$ denote the residual points for evaluating $f(x,t)$.

3 Results for the deterministic PDE

We start with a deterministic PDE to demonstrate how can we infer some of the unknown parameters using PINNs. We will assume that we have different types of data of variable fidelity, and we will also demonstrate how we can make use of data of lower fidelity as well using the composite NN first introduced in [12]. We consider the following non-linear ADR equation:

$$\begin{cases} u_t = \nu_1 u_{xx} - \nu_2 u_x + g(u), & (x,t) \in (0,\pi) \times (0,1], \\ u(x,0) = u_0(x), & x \in (0,\pi), \\ u(0,t) = 1, \;\; u_x(\pi,t) = 0, & t \in (0,1]. \end{cases} \tag{3.1}$$

We define the residual $f = u_t - \nu_1 u_{xx} + \nu_2 u_x - g(u)$. The L_2 error of a function h is defined as $E_h = ||h_{NN} - h_{true}||_{L2}$, and the relative L_2 error is defined as $E_h = \frac{||h_{NN} - h_{true}||_{L2}}{||h_{true}||_{L2}}$.

3.1 Single-fidelity data

First, we will use single-fidelity data to infer different parameters and at the same time obtain the solution u. We consider the initial condition $u_0(x) = \exp(-10x)$ and the reaction term $g(u) = \lambda_1 u^{\lambda_2}$. We aim to infer the parameters $\nu_1, \nu_2, \lambda_1, \lambda_2$ given some sparse measurements of u in addition to initial and boundary conditions. The correct values for the 'unknown' parameters are $\nu_1 = 1, \nu_2 = 1, \lambda_1 = -1, \lambda_2 = 2$.

We employ the following loss function in the PINN:

$$MSE = MSE_u + w_{\nabla_u} * MSE_{\nabla u} + MSE_f, \tag{3.2}$$

where

$$MSE_u = \frac{1}{N_u} \sum_{i=1}^{N_u} |u_{NN}(t_u^i, x_u^i) - u^i|^2,$$

$$MSE_{\nabla u} = \frac{1}{N_u} \sum_{i=1}^{N_u} |\nabla u_{NN}(t_u^i, x_u^i) - \nabla u^i|^2,$$

$$MSE_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |f_{NN}(t_f^i, x_f^i)|^2.$$

The points $\{t_u^i, x_u^i, u_{NN}(t_u^i, x_u^i)\}_{i=1}^{N_u}$ denote the training data for $u(t,x)$, and $N_u = 64, N_f = 1089$, and u^i is the 'reference solution', which is computed by the second-order finite difference method ($\Delta x = \frac{\pi}{1024}$ and $\Delta t = \frac{1}{1600}$); also ∇u is computed with the finite difference method. We use four hidden layers and 20 neurons per layer for the DNN. The error of the parameters is defined as $E_{\nu_1} = \frac{\nu_{1train} - \nu_1}{\nu_1}$, $E_{\nu_2} = \frac{\nu_{2train} - \nu_2}{\nu_2}$, $E_{\lambda_1} = \frac{\lambda_{1train} - \lambda_1}{\lambda_1}$ and $E_{\lambda_2} = \frac{\lambda_{2train} - \lambda_2}{\lambda_2}$.

In the following, we will investigate four different ways to choose the training points as shown in Figure 3. For case I, the training data come from two snapshots at $t = 0.1$ and $t = 0.9$. For case II, the training data come from three snapshots at $t = 0.1$, $t = 0.9$ and $x = \frac{\pi}{2}$. For case III, we choose the training data randomly. For case IV, we assume that we have the training data on a regular lattice in the $x - t$ domain. In all cases, we have 64 training points, and for the weights in the loss function, we investigate both the case with $w_{\nabla u} = 0$ and also the case with $w_{\nabla u} = 1$. In the latter case, we assume that we also have available the gradients of the field u. We present the parameter evolution predictions as the iteration of the optimiser progresses in Figure 4. The

Table 1. *Single-fidelity case: Errors of the solution and of the parameters*

	E_u	$E_{\nu_1}(\%)$	$E_{\nu_2}(\%)$	$E_{\lambda_1}(\%)$	$E_{\lambda_2}(\%)$
Case I ($w_{\nabla u} = 0$)	$1.9514e-03$	1.1098	0.9010	0.2240	0.2370
Case I ($w_{\nabla u} = 1$)	$1.6405e-03$	0.8004	0.9780	0.0522	0.7355
Case II ($w_{\nabla u} = 0$)	$2.7517e-03$	1.4970	2.8120	0.4920	2.2725
Case II ($w_{\nabla u} = 1$)	$2.5277e-03$	0.1130	1.0973	0.1630	1.7915
Case III ($w_{\nabla u} = 0$)	$1.6881e-03$	0.3170	1.6801	0.2430	3.3520
Case III ($w_{\nabla u} = 1$)	$1.4899e-03$	0.7740	1.2520	0.0730	1.2185
Case IV ($w_{\nabla u} = 0$)	$1.5556e-03$	0.5308	1.1720	0.1896	1.2275
Case IV ($w_{\nabla u} = 1$)	$1.3695e-03$	0.6666	1.4820	0.4230	1.2555

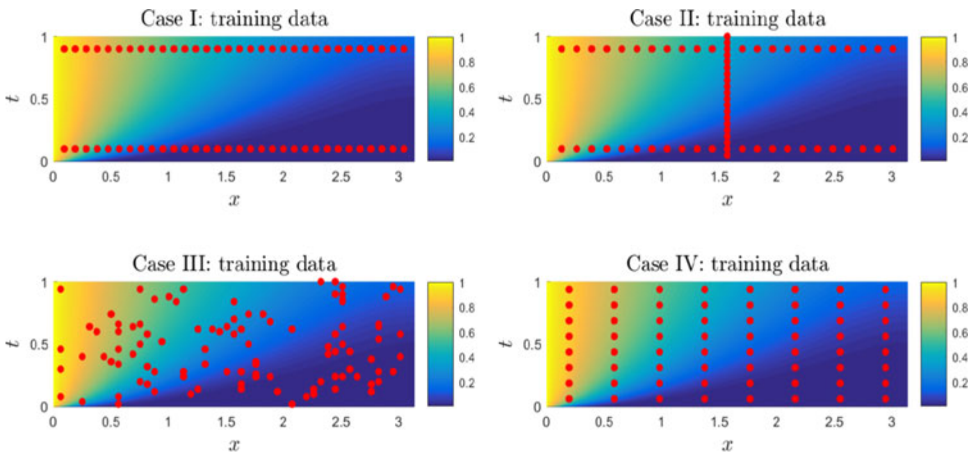

FIGURE 3. Single-fidelity case: The position of training data used in the loss function.

convergence is faster if we include the gradient penalty term ($w_{\nabla_u} = 1$). The convergence with respect to the number of iterations and in actual wall clock time is faster. In both cases, we use the same Adam optimiser. Since we know the target values explicitly, we do not need to specify a convergence criterion. However, in general, this convergence criterion will be problem-dependent, and it is important to search for robust metrics. We summarise the results in terms of the error of the solution $u(x, t)$ and of the parameters in Table 1.

Taken together, the results indicate that even with very few sensors very accurate inference of the parameters as well as the field u is obtained using PINN. Moreover, penalising the gradient of the measurements when possible leads to better accuracy for u although the improvement in the inference of the parameters is mixed.

3.2 Multi-fidelity data

In many real-world applications, the training data are small and possibly inadequate to obtain even a rough estimation of the parameters. Here, we demonstrate how we can resolve this issue

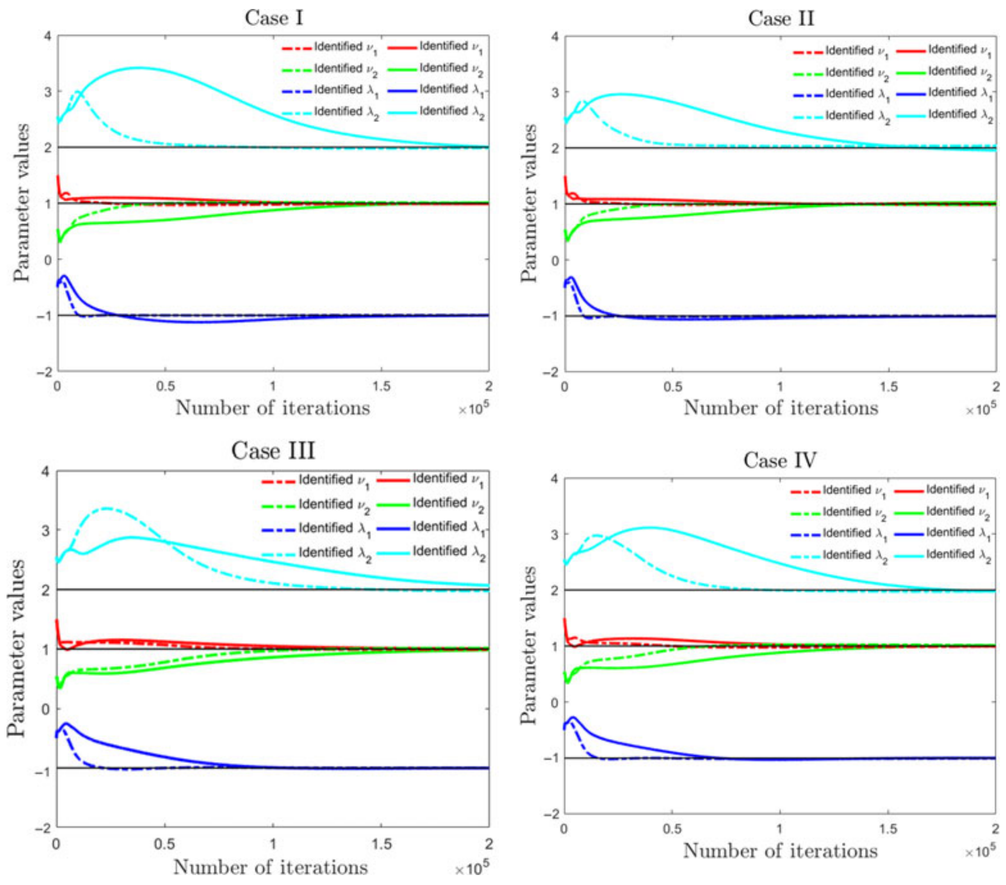

FIGURE 4. Single-fidelity case. Parameter evolution as the iteration of optimiser progresses for four different training data sets. The solid line corresponds to a loss without penalising the gradient term, while the dash line corresponds to a loss that includes the gradient term.

by resorting to supplementary data of lower fidelity that may come from cheaper instruments of lower resolution or from some computational models. We will refer to such data as 'low-fidelity' (LF), and we will assume that we have a large number of such data points unlike the high-fidelity (HF) data. Here, we will employ a composite network inspired by the recent work on multi-fidelity NNs in [12].

The estimator of the HF model using the correlation structure to correct the LF model can be expressed as

$$u_{HF}(x,t) = h(u_{LF}(x,t), x, t), \tag{3.3}$$

where h is a correlation map to be learned, which is based on the correlation between the HF and LF data. Similarly, we have two NNs for LF and HF, respectively, as follows:

$$\begin{aligned} u_{LF} &= \mathcal{NN}_{LF}(x_{LF}, t_{LF}, w_{LF}, b_{LF}), \\ u_{HF} &= \mathcal{NN}_{HF}(x_{HF}, t_{HF}, u_{LF}, w_{HF}, b_{HF}). \end{aligned} \tag{3.4}$$

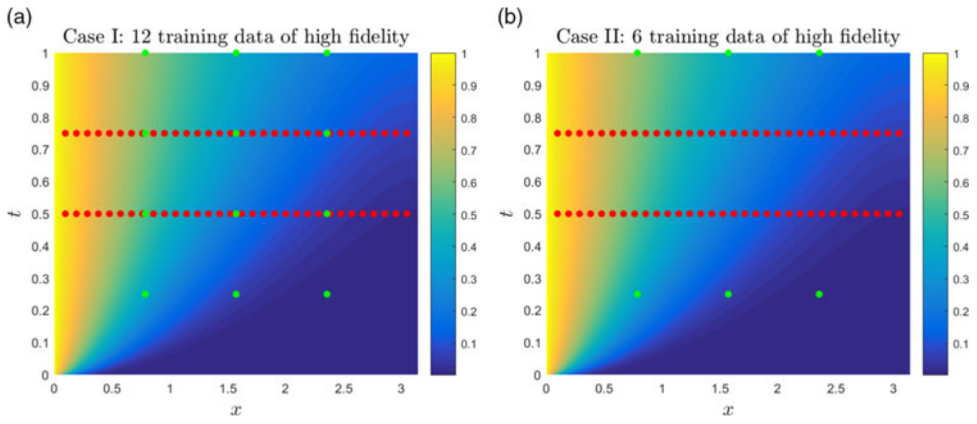

FIGURE 5. Multi-fidelity case: (a) 12 HF training data ($N_{HF} = 12$). (b) 6 HF training data ($N_{HF} = 6$).

We use four hidden layers and 20 neurons per layer for $\mathcal{NN}_{LF}$ and two hidden layers with 10 neurons for $\mathcal{NN}_{HL}$. The learning rate is 5×10^{-5}. We infer the same parameters as in the single-fidelity case and the field u by minimising the mean-squared-error loss function:

$$MSE = MSE_{u_{LF}} + MSE_{u_{HF}} + MSE_{f_{HF}}, \tag{3.5}$$

where

$$MSE_{u_{LF}} = \frac{1}{N_{LF}} \sum_{i=1}^{N_{LF}} |u_{LF}(t^i_{u_{LH}}, x^i_{u_{LF}}) - u^i_{LH}|^2,$$

$$MSE_{u_{HF}} = \frac{1}{N_{HF}} \sum_{i=1}^{N_{HF}} |u_{HF}(t^i_{u_{HF}}, x^i_{u_{HF}}) - u^i_{HF}|^2,$$

$$MSE_{f_{HF}} = \frac{1}{N_f} \sum_{i=1}^{N_f} |f_{HF}(t^i_{f_{HF}}, x^i_{f_{HF}})|^2,$$

and $\{(t^i_{u_{LH}}, x^i_{u_{LH}})\}_{i=1}^{N_{LF}}$ are the point of LF, $\{(t^i_{u_{HF}}, x_{u_{HF}})\}_{i=1}^{N_{HF}}$ are the point of HF and $\{(t^i_{f_{HF}}, x^i_{f_{HF}})\}_{i=1}^{N_f}$ are the residual points where we penalise the residual f. We choose $N_f = 1024$ for the tests here.

We choose the reaction term $g(u) = \lambda_1 u^{\lambda_2}$ and set the true parameters $\nu_1 = 1$, $\nu_2 = 1$, $\lambda_1 = -1$ and $\lambda_2 = 2$. Here, the LF training data are obtained by the second-order finite difference solution of (3.1) with erroneous parameter values, i.e., $\nu_1 = 1.25$, $\nu_2 = 1.25$, $\lambda_1 = -0.75$ and $\lambda_2 = 2.5$, where $\Delta x = \frac{\pi}{32}$ and $\Delta t = \frac{1}{32}$; we choose 64 points of LF of u, i.e., $N_{LF} = 64$. The positions of LF are denoted by the red point in Figure 5. The HF data are obtained by the numerical solution of (3.1) when $\nu_1 = 1$, $\nu_2 = 1$, $\lambda_1 = -1$, and $\lambda_2 = 2$, where $\Delta x = \frac{\pi}{1024}$ and $\Delta t = \frac{1}{1024}$. The positions of HF data are shown by the green points in Figure 5. We choose 12 HF training data ($N_{HF} = 12$) in Figure 5(a) and 6 data as the HF training data ($N_{HF} = 6$) in Figure 5(b).

To test the effect of the LF data, we compare the PINN and multi-fidelity PINN results in Table 2. As we can see, the parameter inference using the multi-fidelity PINN is much better than the single-fidelity predictions. Moreover, if we have a small number of HF data, e.g. $N_{HF} = 6$, the results of the multi-fidelity PINN are still quite accurate.

Table 2. *Multi-fidelity case: Errors of the solution and of the parameters. (mPINN refers to the multi-fidelity PINN)*

	E_u	$E_{\nu_1}(\%)$	$E_{\nu_2}(\%)$	$E_{\lambda_1}(\%)$	$E_{\lambda_2}(\%)$
12 points+PINN	$2.3558e-03$	3.4000	9.0891	1.6330	13.445
12 points+mPINN	$1.1214e-03$	0.6380	1.2772	2.4939	0.8975
6 points+PINN	$6.5386e-03$	9.3640	24.455	14.707	49.445
6 points+mPINN	$1.2425e-03$	1.6190	3.6775	2.1220	2.0635

4 Results for the stochastic case

Next, we test the effectiveness of sPINN for solving forward and inverse problems by considering the following stochastic non-linear ADR equation:

$$\begin{cases} u_t = (k(x;\omega)u_x)_x - \nu_2 u_x + g(u) + f(x,t), & (x,t,\omega) \in (x_0,x_1) \times (0,T] \times \Omega, \\ u(x,0) = 1 - x^2, & x \in (x_0,x_1), \\ u(x_0,t) = 0,\ u(x_1,t) = 0, & t \in (0,T]. \end{cases} \tag{4.1}$$

Here, $x_0 = 0$, $x_1 = 1$, Ω is the random space, and the stochastic diffusivity is modelled as $log(k(x;\omega)) \in GP(k_0(x), Cov(x,x'))$, hence, it is a non-Gaussian random process with mean $k_0(x) = \sin(\pi(x+1)/2)/5$, and covariance function $Cov(x,x') = \sigma^2 * exp\big(-\frac{(x-x')^2}{l_c^2}\big)$ with $l_c = 1$ (GP stands for Gaussian Process here); $\sigma = 0.1$. We also define the residual $f = u_t - (k(x;\omega)u_x)_x + \nu_2 u_x - g(u)$. We consider the reaction term $g(u) = \lambda_1 u^{\lambda_2}$ and $f(x,t) = 2$. The true parameter values are $\nu_2 = 1$, $\lambda_1 = 1$ and $\lambda_2 = 3$.

4.1 Forward problem

We use a sPINN with four hidden layers and 20 neurons per layer for the modes of u, i.e., u_l, $0 \leq l \leq P$. The learning rate is 5×10^{-4}.

We minimise the following mean-squared-error loss function:

$$MSE = MSE_u + MSE_f, \tag{4.2}$$

where MSE_u is the loss function for the initial and boundary conditions of u, MSE_f is the loss function for the PDE, and they are computed as follows:

$$MSE_u = \frac{1}{N * N_u} \sum_{s=1}^{N} \sum_{i=1}^{N_I} [u_{NN}(x_u^{(i)}, 0; \omega_s) - u(x_u^{(i)}, 0; \omega_s)]^2$$
$$+ \frac{1}{N * N_B} \sum_{s=1}^{N} \sum_{i=1}^{N_B} [u_{NN}(x_0, t_u^{(i)}; \omega_s) - u(x_0, t_u^{(i)}; \omega_s)]^2$$

Table 3. *Forward problem: The L_2 error and the relative L_2 error for different values of the order r of aPC*

		L_2 error	Relative L_2 error
$r = 1$	$E[u]$	$2.5090e-03$	$3.0862e-03$
	$Var[u]$	$7.5067e-05$	$3.0841e-02$
$r = 2$	$E[u]$	$1.0230e-03$	$1.2583e-03$
	$Var[u]$	$4.4646e-06$	$1.8343e-03$
$r = 3$	$E[u]$	$6.1008e-04$	$7.5045e-04$
	$Var[u]$	$7.4720e-07$	$3.0699e-04$

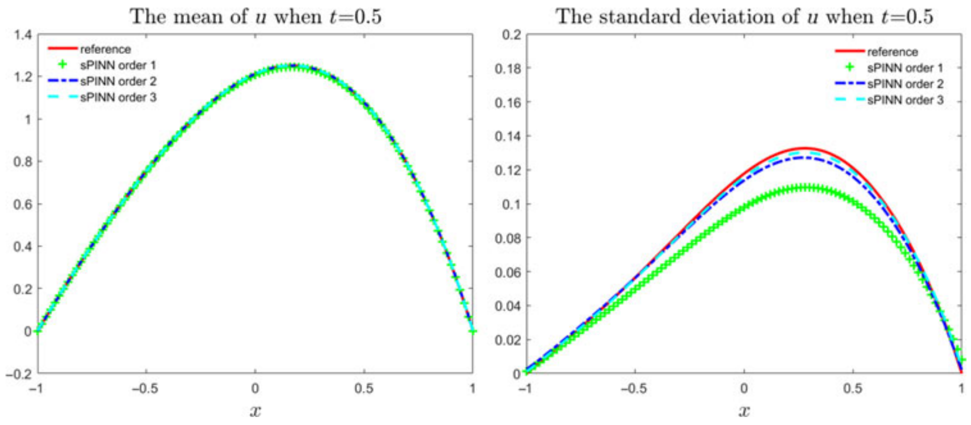

FIGURE 6. Forward problem: predicted mean and standard deviation at $t = 0.5$ when $r = 1,\ 2,\ 3$. The reference solution is obtained by QMC (see Appendix).

$$+\frac{1}{N * N_B}\sum_{s=1}^{N}\sum_{i=1}^{N_B}[u_{NN}(x_1, t_u^{(i)}; \omega_s) - u(x_1, t_u^{(i)}; \omega_s)]^2,$$

$$MSE_f = \frac{1}{N * N_f}\sum_{s=1}^{N}\sum_{i=1}^{N_f}[f_{NN}(x_f^{(i)}, t_f^{(i)}; \omega_s) - f(x_f^{(i)}, t_f^{(i)}; \omega_s)]^2.$$

We set $M = 4$, $N = 1000$, $N_I = 101$, $N_B = 101$ and $N_f = 441$ in the loss function.

We use the first-order ($r = 1, P = 4$), second-order ($r = 2, P = 14$) and third-order ($r = 3, P = 34$) aPC expansion for u in this sub-section. In Table 3, we present the L_2 and relative L_2 errors of the mean and variance of u at $t = 0.5$; using the higher order aPC expansion, we obtain better results. We present in Figure 6 the DNN predictions of the u mean and variance at $t = 0.5$, where the reference solutions are calculated by the Quasi-Monte Carlo (QMC) method (more details are shown in Appendix). We also present in Figure 7 the sPINN prediction of a few modes of u at $t = 0.5$. Taken together, the results show that sPINN can solve forward stochastic problems accurately for more complex (non-linear and time-dependent) SPDEs than the ones considered in the original paper of [25].

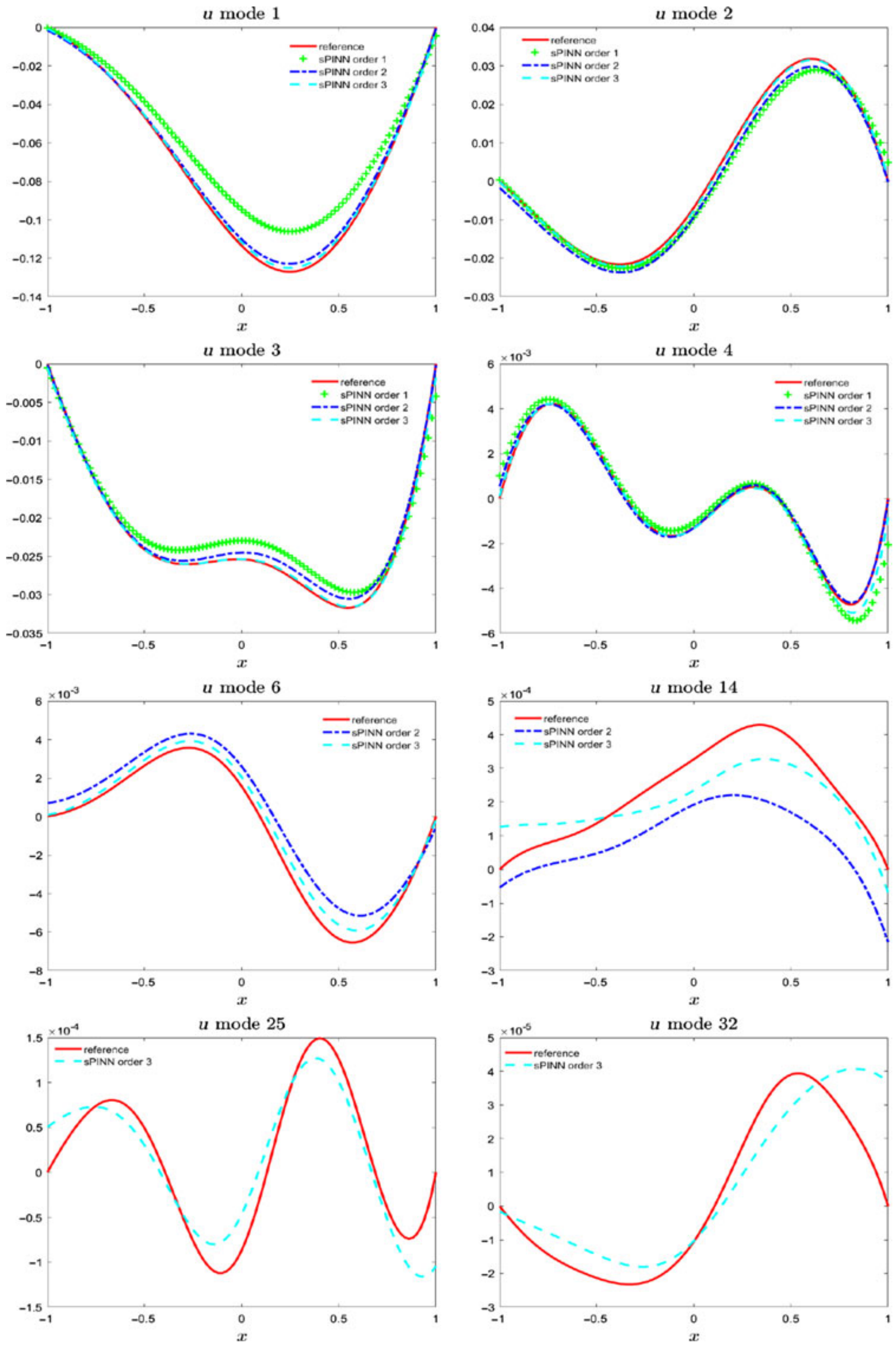

FIGURE 7. Forward problem: some predicted modes of u with aPC expansions versus the reference solutions at $t = 0.5$ for polynomial order $r = 1,\ 2,\ 3$.

4.2 Inverse problem

Next, we will infer the stochastic process $k(x, \omega)$ as well as the parameters $\nu_2, \lambda_1, \lambda_2$ and the solution $u(x, t, \omega)$. We use two hidden layers and 4 neurons per layer for the k mean and $k_i(x)$, $(i = 1, ...M)$ NNs, and four hidden layers and 20 neurons per layer for the $u_\alpha(x, t)$, $(\alpha = 0, 1, ...P)$ NNs, and the learning rate is 5×10^{-4}; we choose $N = 2000$, $N_u = 20$, $N_k = 7$, $N_f = 441$, $w_u = 100$ and $w_k = 16$. These values of weights were chosen based on experimentation and also taken into account the order of magnitude of the various quantities, e.g., mean versus standard deviation.

We minimise the following mean-squared-error loss function:

$$MSE = 10 * (MSE_u + MSE_k) + MSE_f, \tag{4.3}$$

where

$$\begin{aligned}
MSE_u = {} & \frac{1}{N_u}\sum_{i=1}^{N_u}\left[\frac{1}{N}\sum_{s=1}^{N} u_{NN}(x_u^{(i)}, t_u^{(i)}; \omega_s) - \frac{1}{N}\sum_{s=1}^{N} u(x_u^{(i)}, t_u^{(i)}; \omega_s)\right]^2 \\
& + w_u \frac{1}{N * N_u}\sum_{i=1}^{N_u}\sum_{s=1}^{N}[u_{NN}(x_u^{(i)}, t_u^{(i)}; \omega_s) - \frac{1}{N}\sum_{s=1}^{N} u_{NN}(x_u^{(i)}, t_u^{(i)}; \omega_s) \\
& + u(x_u^{(i)}, t_u^{(i)}; \omega_s) - \frac{1}{N}\sum_{s=1}^{N} u(x_u^{(i)}, t_u^{(i)}; \omega_s)]^2, \\
MSE_k = {} & \frac{1}{N_k}\sum_{i=1}^{N_k}\left[\frac{1}{N}\sum_{s=1}^{N} k_{NN}(x_k^{(i)}; \omega_s) - \frac{1}{N}\sum_{s=1}^{N} k(x_k^{(i)}; \omega_s)\right]^2 + w_k \frac{1}{N * N_u}\sum_{i=1}^{N_k}\sum_{s=1}^{N} \\
& [k_{NN}(x_k^{(i)}; \omega_s) - \frac{1}{N}\sum_{s=1}^{N} k_{NN}(x_k^{(i)}; \omega_s) + k(x_k^{(i)}; \omega_s) - \frac{1}{N}\sum_{s=1}^{N} k(x_k^{(i)}; \omega_s)]^2, \\
MSE_f = {} & \frac{1}{N * N_f}\sum_{s=1}^{N}\sum_{i=1}^{N_f}[f_{NN}(x_f^{(i)}, t_f^{(i)}; \omega_s) - f(x_f^{(i)}, t_f^{(i)}; \omega_s)]^2.
\end{aligned}$$

We assume that we have measurements of u at the positions indicated in Figure 8; the positions where k is known are shown directly in the inference plots.

We use the first-order ($r = 1$ and $P = 4$), second-order ($r = 2$ and $P = 14$) and third-order ($r = 3$ and $P = 34$) aPC expansions. The errors of the mean and variance of u and k are shown in Table 4. The errors of the parameters are shown in Table 5. Overall, the results improve by using a higher order aPC expansion.

The predicted mean, standard deviation and the modal functions of k are shown in Figure 9. The predicted mean, standard deviation and the modal functions of u are shown in Figure 10. The results for the solution u are good, but the inaccuracy of the first mode of k affects the accuracy of the standard deviation. We have used a very small NN for k as we observed problems with overfitting; hence, in order to improve the overall learning of k in modal space, we will introduce the meta-learning method next to search for better NN architectures for k but also for u.

Table 4. *L_2 and relative L_2 errors of u and k. BO refers to the meta-learning results*

		$E[u]$	$Var[u]$	$E[k]$	$Var[k]$
$r=1$	L_2 error	$2.0058e-03$	$6.0891e-07$	$2.0721e-03$	$1.1191e-06$
	Relative L_2 error	$2.4673e-03$	$2.5017e-04$	$1.8106e-03$	$8.5212e-05$
$r=2$	L_2 error	$1.8472e-03$	$8.2031e-07$	$1.3080e-03$	$2.3888e-06$
	Relative L_2 error	$2.2722e-03$	$3.3702e-04$	$1.1429e-03$	$1.8189e-04$
$r=3$	L_2 error	$1.7582e-03$	$5.9305e-07$	$2.1860e-03$	$7.8451e-07$
	Relative L_2 error	$2.1628e-03$	$2.4365e-04$	$1.9102e-03$	$5.9735e-05$
BO	L_2 error	$5.9512e-04$	$9.7195e-09$	$3.9250e-04$	$1.5484e-07$
	Relative L_2 error	$7.3204e-04$	$3.9933e-06$	$3.4297e-04$	$1.1790e-05$

Table 5. *The error and the relative error of the parameters*

		E_{ν_2}	$E_{\lambda 1}$	$E_{\lambda 2}$
$r=1$	Error	$1.8855e-02$	$1.2252e-02$	$1.1636e-01$
	Relative error	1.8855 %	1.2252 %	3.8787 %
$r=2$	Error	$2.3376e-02$	$4.2319e-04$	$6.3868e-02$
	Relative error	2.3376 %	0.0423 %	2.1289 %
$r=3$	Error	$1.9281e-02$	$2.306e-03$	$3.9592e-02$
	Relative error	1.9281 %	0.2306 %	1.3197 %
BO	Error	$2.2605e-03$	$9.2113e-04$	$2.1623e-02$
	Relative error	0.2261 %	0.0921 %	0.7207 %

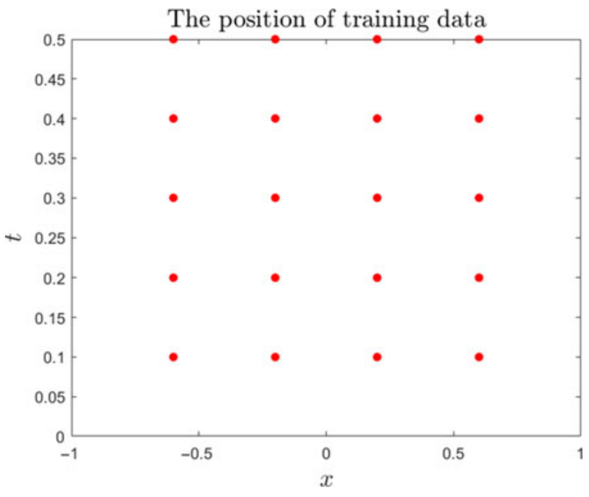

FIGURE 8. Stochastic inverse problem: Space–time positions of the training data for u.

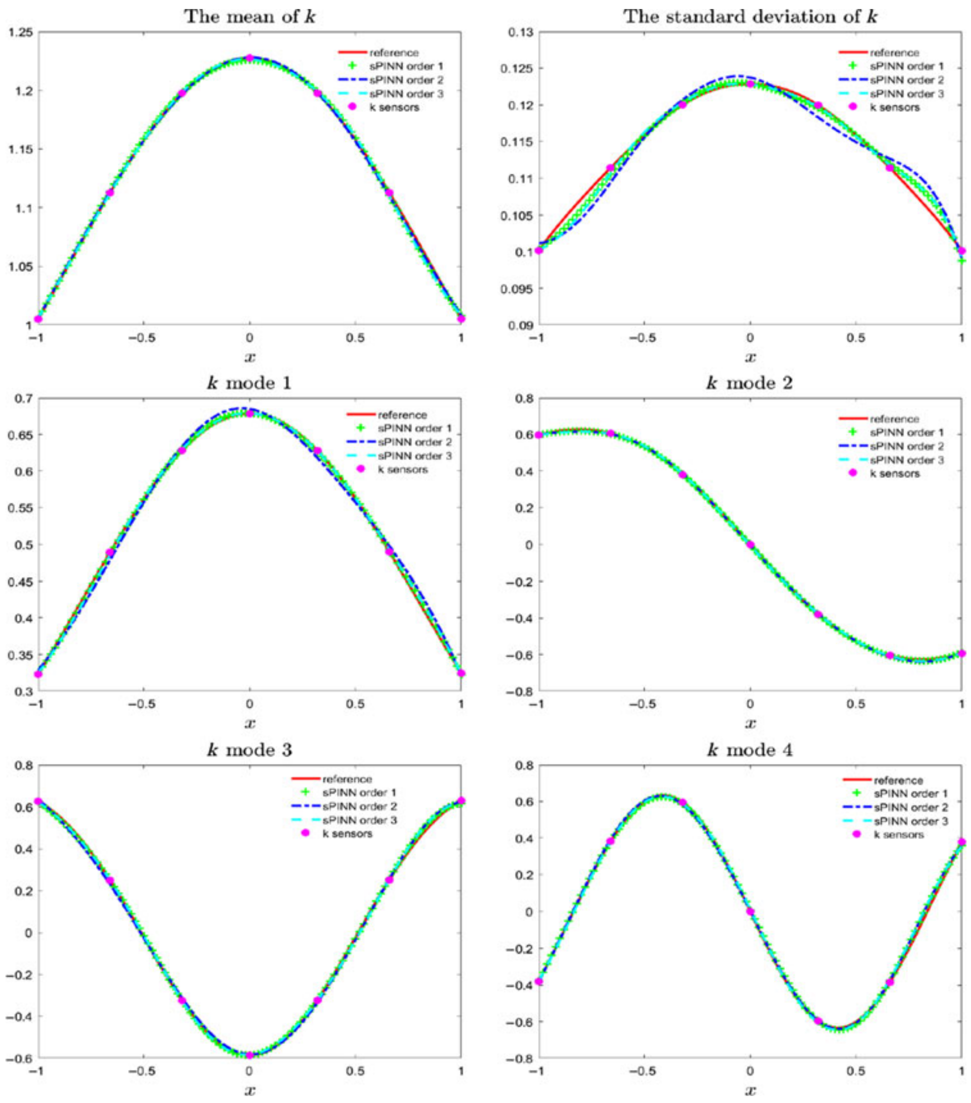

FIGURE 9. Stochastic inverse problem: predicted mean, standard deviation and modes of k versus the reference solutions when $r = 1,\ 2,\ 3$. The locations of the k sensors are denoted by red points.

4.3 Meta-learning

There are many types of meta-learning methods. To reduce the empiricism of selecting the sPINN architecture, in this section, we employ BO to learn the optimum structure of the NNs. We use d_K hidden layers and w_K neurons per layer for k mean NN, and d_U hidden layers and w_U neurons per layer for $k_i(x),\ (i = 1, ...M)$ and $u_\alpha(x,t),\ (\alpha = 0, 1, ...P)$ NN. The learning rate is l_r. The target is

$$Target = 10 * (MSE_u + MSE_k) + E_{v_2} + E_{\lambda_1} + E_{\lambda_2}. \tag{4.4}$$

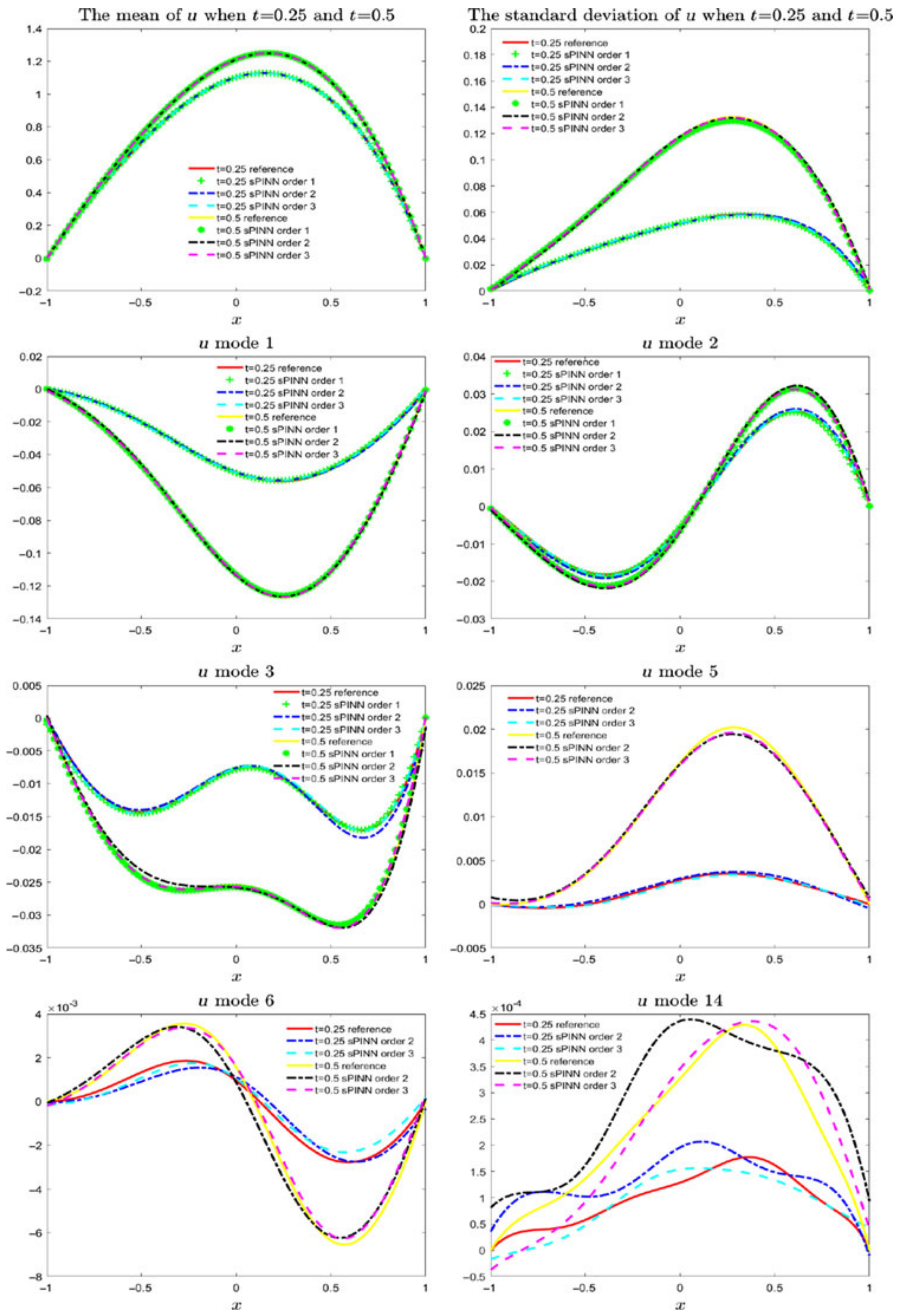

FIGURE 10. Stochastic inverse problem: predicted mean, standard deviation and modes of u at $t = 0.25$ and $t = 0.5$ when $r = 1,\ 2,\ 3$.

Table 6. *The hyper-parameters and architecture choices for the fully connected NNs*

Hyper-parameter	Range	Log-transform
Hidden layers (d_K)	$[1, 10]$	No
Units per layer (w_K)	$[1, 64]$	Yes
Hidden layers (d_U)	$[1, 30]$	No
Units per layer (w_U)	$[1, 128]$	Yes
Learning rate ($log(l_r)$)	$[-5, -2]$	Yes

Table 7. *Top 10 results of meta-learning using BO*

Number	Target	d_K	w_K	d_U	w_U	$log(l_r)$
1	0.03898	3	23	10	113	−3.5670
2	0.04678	3	21	9	107	−2.3724
3	0.04704	3	23	10	113	−3.6665
4	0.04915	2	2	6	7	−2.3740
5	0.04962	3	21	9	127	−3.4890
6	0.05007	3	25	9	118	−3.7266
7	0.05429	3	22	9	110	−3.6487
8	0.05451	3	22	9	118	−3.6606
9	0.05691	3	20	9	107	−3.7656
10	0.05735	2	2	6	7	−3.1180
*	0.04704	3	4	15	80	−3.4890

So the target is a function: $\chi \to \mathbb{R}$, and $\chi = \{d_K, w_K, d_U, w_U, l_r\}$. The acquisition function is upper confidence bound and the tradeoff parameter is 2.576.

Table 6 gives the range of the hyper-parameters we choose. We use the log-transform for the width of the NN and the learning rare. We compute 80 times to optimize the target function and get the optimization hyper-parameters. The result is show in Figure 11. The top 10 good results are shown in Table 7; the $*$ result denotes that we do not use the log-transform. These results suggest that we need a larger NN for both k and u.

Next, we use the best structure of the NN and learning rate to re-compute the previous stochastic inverse problem, i.e. the depth of k mean NN is 3, and the width is 23. For the NN of the k modal functions, the u mean and the u modal functions, the depth is 10 and the width is 113. The learning rate is $10^{-3.5670}$. The results are shown in Figures 12–13.

5 Summary

We addressed here a special inverse problem governed by a stochastic non-linear ADR equation, where given some samples of the solution $u(x, t; \omega)$ at a relatively few locations (here four spatial locations and five time instants) but also given some samples of the stochastic diffusivity $k(x; \omega)$ at seven locations, we aim to obtain the full stochastic fields for u and k as well as three other unknown parameters. We designed composite NNs, including NNs induced by the stochastic

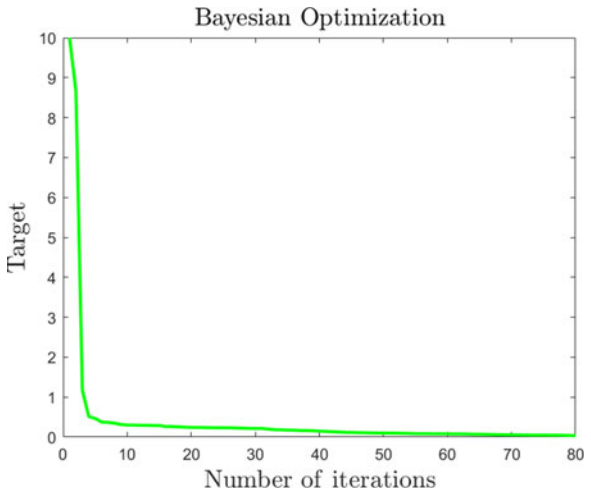

FIGURE 11. Convergence of the target for the BO (meta-learning) as a function of the number of iteration.

ADR equation, and relied on spectral expansions to represent stochasticity in order to deal with the sparsity in data. We also presented a BO method for learning the hyper-parameters of this composite NN as it is time-consuming to find the proper NNs by trial and error. We followed a hierarchical approach in testing the various components of the NNs, including training from multi-fidelity data, investigating possible good locations in space–time for collecting the training data, and evaluating different weights in the loss functions for the multiple terms representing data and physics. To the best of our knowledge, this is the first time that such a study is undertaken with the purpose of evaluating the potential of NNs to learn from sparse data of variable fidelity and with uncertainty.

An important component missing in our study is quantifying the uncertainty of the NN approximation as was first done in related work in [25] addressing the *total* uncertainty. This is a serious but complex issue requiring the use of multiple methods to interpret this uncertainty in an objective way, and we will pursue this line of research in future work. The present work is also the first study that uses meta-learning for PINNs, i.e., to optimise the composite NN, which in our case consists of multiple NNs, as would be the case in simulating multi-physics dynamics. In addition to the BO employed here, one could also consider using several other methods, including genetic algorithms [13], the greedy method [11], hyperband [10, 5] as well as blended versions of the aforementioned methods or even another NN, like an Recurrent neural network in conjunction with reinforcement learning [9], to search for the best architecture. This has already been done for classification work, and it is part of AutoML [8] but not for regression tasks.

Acknowledgements

This work was supported by China Scholarship Council scholarship and the PhILMs grant DE-SC0019453. In addition, we would like to thank Dr. Guofei Pang, Dr. Lu Lu, Dr. Xuhui Meng and Dr. Dongkun Zhang in the Division of Applied Mathematics at Brown University for their helpful suggestions.

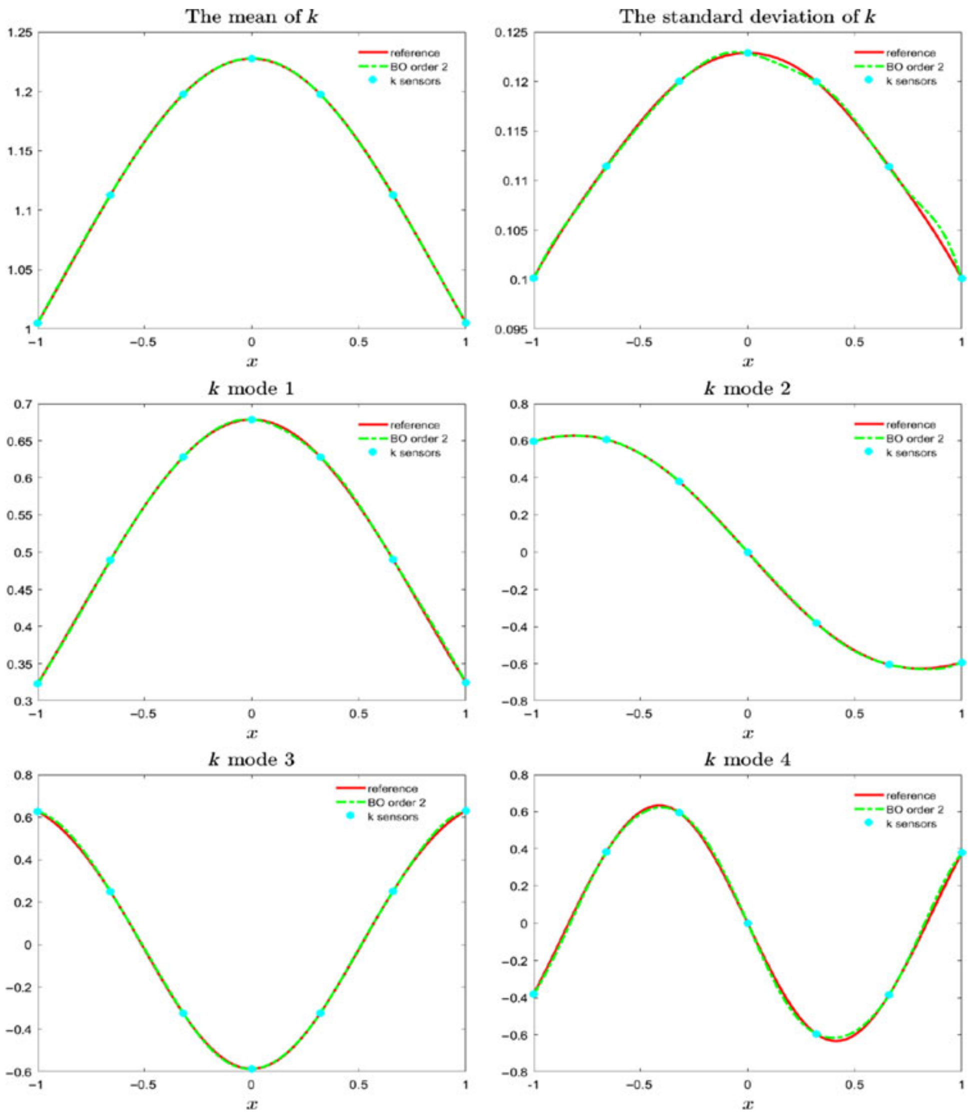

FIGURE 12. Stochastic inverse problem: Predicted results of sPINN for k against the reference solution using the optimum hyper-parameters obtained via BO (meta-learning).

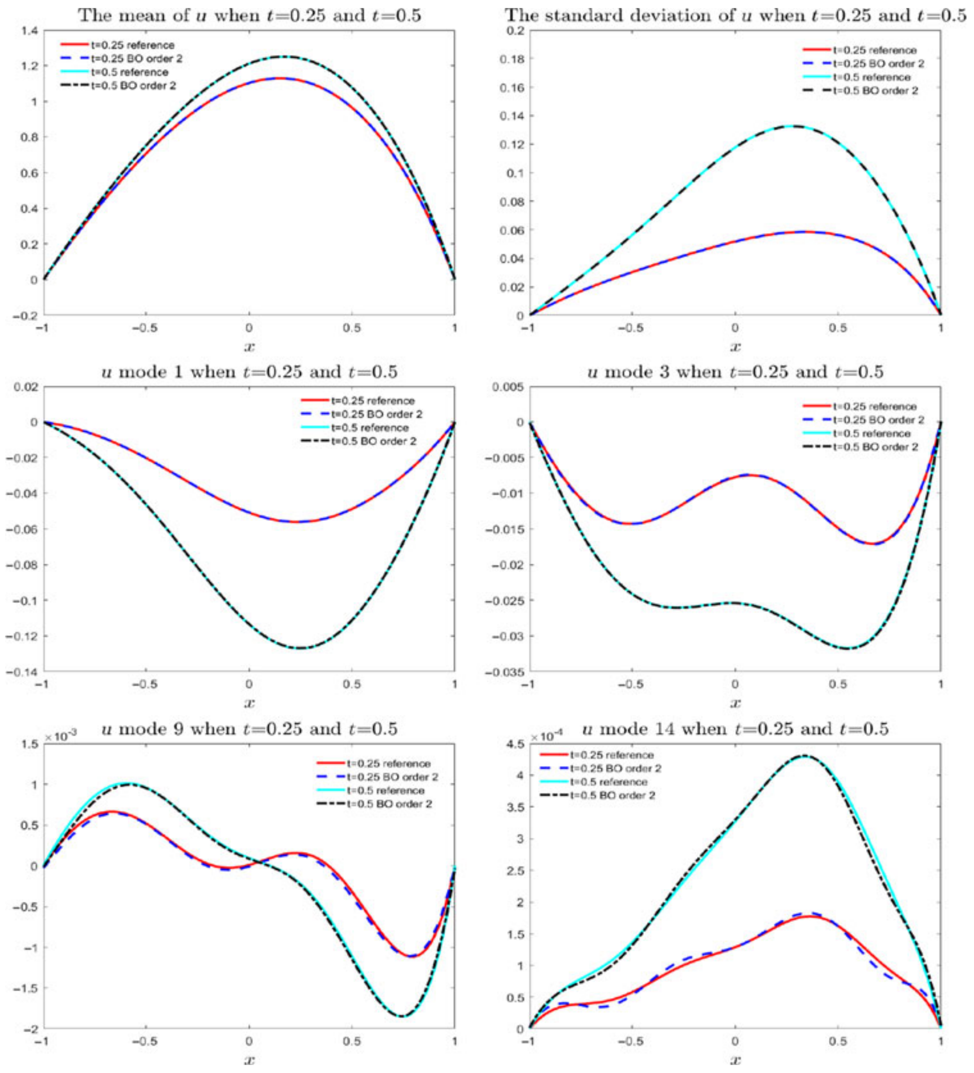

FIGURE 13. Stochastic inverse problem: predicted results of sPINN for u against the reference solutions at $t = 0.25$ and $t = 0.5$ using the optimum hyper-parameters obtained via BO (meta-learning).

Conflict of interest

None

References

[1] Barajas-Solano, D. A. & Tartakovsky, A. M. (2019) Approximate bayesian model inversion for pdes with heterogeneous and state-dependent coefficients. *J. Comput. Phys.* **395**, 247–262.

[2] Bergstra, J. S., Bardenet, R., Bengio, Y. & Kégl, B. (2011) Algorithms for hyper-parameter optimization. In: *Advances in Neural Information Processing Systems,* pp. 2546–2554.

[3] Chen, T. Q., Rubanova, Y., Bettencourt, J. & Duvenaud, D. K. (2018) Neural ordinary differential equations. In: *Advances in Neural Information Processing Systems,* pp. 6571–6583.

[4] Chaudhari, P., Oberman, A., Osher, S., Soatto, S. & Carlier, G. (2018) Deep relaxation: partial differential equations for optimizing deep neural networks. *Res. Math. Sci.* **5**(3), 30.

[5] Falkner, S., Klein, A. & Hutter, F. (2018) BOHB: robust and efficient hyperparameter optimization at scale. arXiv preprint arXiv:1807.01774.

[6] Finn, C., Abbeel, P. & Levine, S. (2017) Model-agnostic meta-learning for fast adaptation of deep networks. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70,* JMLR.org, pp. 1126–1135.

[7] Han, J., Jentzen, A. & Weinan, E. (2018) Solving high-dimensional partial differential equations using deep learning. *Proc. Natl. Acad. Sci.* **115,** 8505–8510.

[8] He, Y., Lin, J., Liu, Z., Wang, H., Li, L.-J. & Han, S. (2018) AMC: AutoML for model compression and acceleration on mobile devices. In: *Proceedings of the European Conference on Computer Vision (ECCV),* pp. 784–800.

[9] Jaafra, Y., Laurent, J. L., Deruyver, A. & Naceur, M. S. (2018) A review of meta-reinforcement learning for deep neural networks architecture search. arXiv preprint arXiv:1812.07995.

[10] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A. & Talwalkar, A. (2018) Hyperband: a novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.* **18,** 1–51.

[11] Li, Y. & Osher, S. (2009) Coordinate descent optimization for l1 minimization with application to compressed sensing; a greedy algorithm. *Inverse Problemsd Imaging* **3,** 487–503.

[12] Meng, X. & Karniadakis, G. E. (2020) A composite neural network that learns from multi-fidelity data: application to function approximation and inverse PDE problems. *J. Comput. Phys.* **401**, 109020.

[13] Mitchell, M. (1998) *An Introduction to Genetic Algorithms,* MIT Press.

[14] Pang, G., Lu, L. & Karniadakis, G. E. (2019) fpinns: fractional physics-informed neural networks. *SIAM J. Sci. Comput.* **41,** A2603–A2626.

[15] Pang, G., Yang, L. & Karniadakis, G. E. (2019) Neural-net-induced gaussian process regression for function approximation and pde solution. *J. Comput. Phys.* **384,** 270–288.

[16] Paulson, J. A., Buehler, E. A. & Mesbah, A. (2017) Arbitrary polynomial chaos for uncertainty propagation of correlated random variables in dynamic systems. *IFAC-PapersOnLine* **50,** 3548–3553.

[17] Qin, T., Wu, K. & Xiu, D. (2019) Data driven governing equations approximation using deep neural networks. *J. Comput. Phys.* **395**, 620–635.

[18] Raissi, M., Perdikaris, P. & Karniadakis, G. E. (2019) Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* **378**, 686–707.

[19] Raissi, M., Perdikaris, P. & Karniadakis, G. E. (2018) Numerical gaussian processes for time-dependent and nonlinear partial differential equations. *SIAM J. Sci. Comput.* **40,** A172–A198.

[20] Sirignano, J. & Spiliopoulos, K. (2018) Dgm: a deep learning algorithm for solving partial differential equations. *J. Comput. Phys.* **375,** 1339–1364.

[21] Snoek, J., Larochelle, H. & Adams, R. P. (2012) Practical bayesian optimization of machine learning algorithms. *Adv. Neural Inf. Process Syst.* **25**, 2951–2959.

[22] Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M. & Adams, R. (2015) Scalable bayesian optimization using deep neural networks. *Int. Conf. Mach. Learn.* **37**, 2171–2180.

[23] Tartakovsky, A. M., Marrero, C. O., Tartakovsky, D. & Barajas-Solano, D. (2018) Learning parameters and constitutive relationships with physics informed deep neural networks. arXiv preprint arXiv:1808.03398.

[24] Wan, X. & Karniadakis, G. E. (2006) Multi-element generalized polynomial chaos for arbitrary probability measures. *SIAM J. Sci. Comput.* **28,** 901–928.

[25] Zhang, D., Lu, L., Guo, L. & Karniadakis, G. E. (2019) Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems, *J. Comput. Phys.* **397,** 108850.

Appendix

In Section 4, we use the finite difference method with the QMC method to obtain the reference modes of k and u. In order to estimate how many samples we need for a converged solution, we

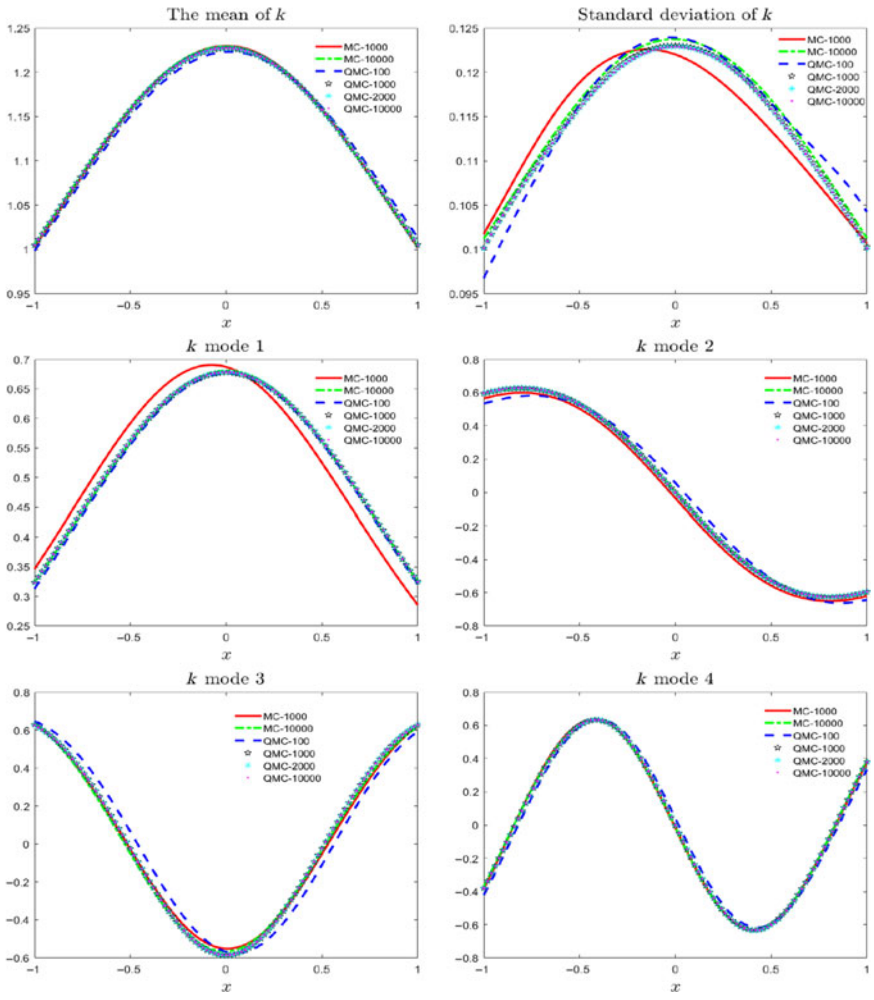

Figure A.1. The mean, standard deviation and mode functions of k: MC vs. QMC.

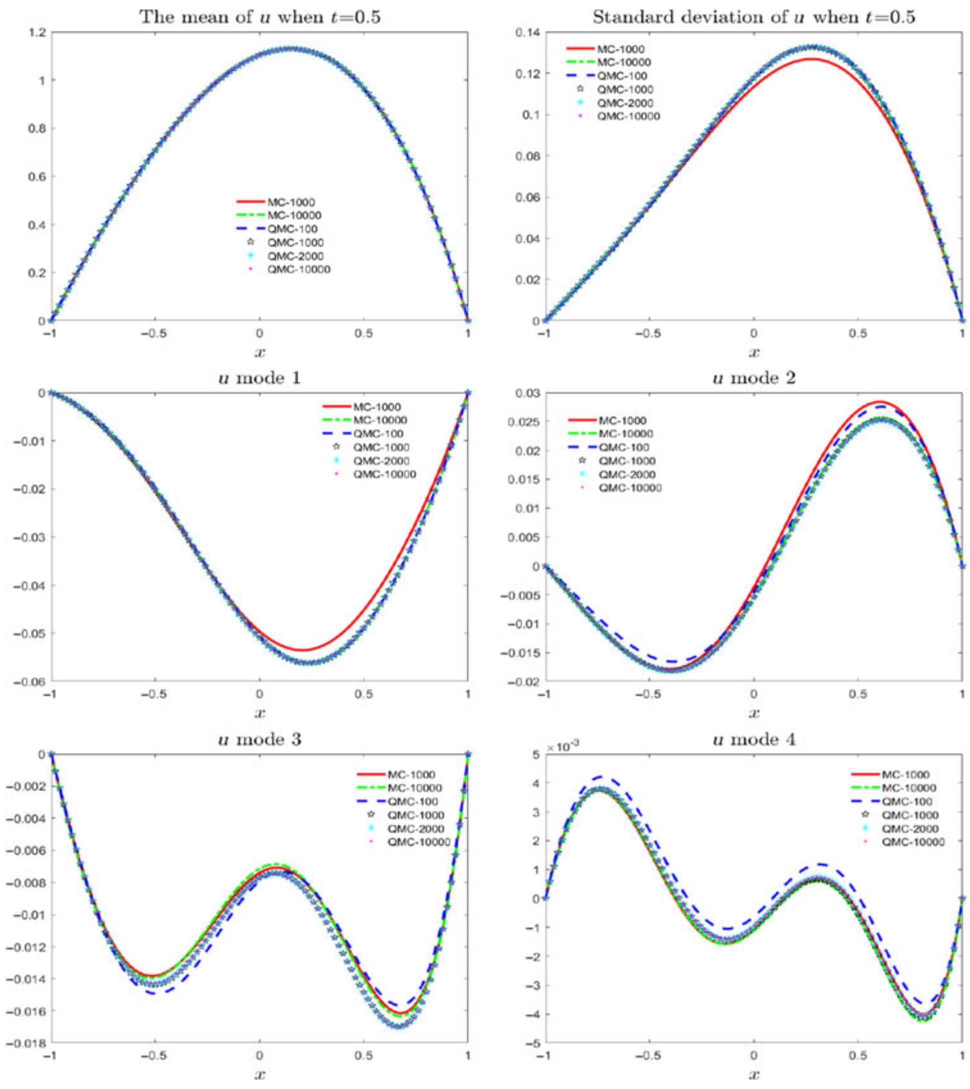

FIGURE A.2. The mean, standard deviation and mode functions of u: MC vs. QMC.

compare the results with different samples using the Monte Carlo (MC) and the QMC methods. In Figures A.1 and A.2, we present the corresponding results using the MC and QMC methods. We can see that the QMC method converges much faster than the MC method. For our examples, we use 2000 QMC samples for training data, and to obtain the reference solutions, we use 10,000 samples.