Stochastic Modeling of Data-driven Complex Systems Using Machine Learning Tools

by

Dongkun Zhang

B.S., Peking University; Beijing, China P.R., 2013

M.S., Brown University; Providence, RI, 2014

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in the Division of Applied Mathematics at Brown University

PROVIDENCE, RHODE ISLAND

May 2019

This dissertation by Dongkun Zhang is accepted in its present form
by the Division of Applied Mathematics as satisfying the
dissertation requirement for the degree of Doctor of Philosophy.

Date_____ _____

George Em Karniadakis, Ph.D., Advisor

Recommended to the Graduate Council

Date_____ _____

Hui Wang, Ph.D., Reader

Date_____ _____

Themistoklis Sapsis, Ph.D., Reader

Date_____ _____

Hessam Babaee, Ph.D., Reader

Approved by the Graduate Council

Date_____ _____

Andrew G. Campbell, Dean of the Graduate School

Vitae

Dongkun Zhang finished his Bachelor of Science degree, majoring in Theoretical and Applied Mechanics, in July 2013 at Peking University in Beijing, China. He then moved to Providence, Rhode Island, in pursuit of a Doctor of Philosophy degree in Applied Mathematics at Brown University, where he also received a transitional Master of Science degree in Applied Mathematics in 2014.

Dongkun's research interest lies in uncertainty quantification, domain decomposition, data-driven modeling and pertinent fields. Advised by Professor George Em Karniadakis, during his Ph.D. study, Dongkun has completed six publications and was invited to present his work in three international conferences, including the SIAM Conference on Uncertainty Quantification (2016, 2018) and the World Congress on Computational Mechanics (2018).

Dongkun has been teaching assistant in four courses, including Methods in Applied Mathematics I (fall 2014) and II (spring 2015), An introduction to Topics in Probability, Statistics, and Machine Learning (fall 2017), and Operation Research: A Probabilistic Approach (spring 2018). In 2015, he received the Sheridan Center Teaching Certificate I.

Publications

1. **D. Zhang**, L. Guo, and G. E. Karniadakis. Solving time-dependent stochastic differential equations with PINNs. *manuscript in preparation*, 2019.

2. L. Yang, **D. Zhang**, and G. E. Karniadakis. Physics-informed generative adversarial networks for stochastic differential equations. *arXiv preprint arXiv:1811.02033*, 2018.

3. **D. Zhang**, L. Lu, L. Guo, and G. E. Karniadakis. Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. *arXiv preprint arXiv:1809.08327*, 2018.

4. **D. Zhang**, L. Yang, and G. E. Karniadakis. Bi-directional coupling between a PDE-domain and an adjacent data-domain equipped with multi-fidelity sensors. *Journal of Computational Physics*, 374, 121–134, 2018

5. **D. Zhang**, H. Babaee, and G. E. Karniadakis. Stochastic domain decomposition via moment minimization *SIAM Journal on Scientific Computing*, 40(4), A2152–A2173, 2018

6. Y.-H. Tang, **D. Zhang**, and G. E. Karniadakis. An atomistic fingerprint algorithm for learning *Ab Initio* molecular force fields. *The Journal of Chemical Physics*, 148(3), 034101 2018.

Acknowledgments

First and foremost, I would like to thank my advisor, Professor George Em Karniadakis, who provides me with the opportunity to study and conduct research in Crunch Group at Brown Univeristy. Professor Karniadakis brought me into this fascinating area of uncertainty quantification, and was always available to provide me with visionary guidance when I was in doubt. His role as an advisor extends beyond research, and his passion for work and life will inspire me for a lifetime. I would also like to express my gratitude to the other members in my dissertation committee: Professer Themistoklis Sapsis, Professor Hessam Babaee, and Professor Hui Wang for sparing time reading through my dissertation, and providing valuable feedback and suggestions.

Secondly, I would like to thank my collaborators. Professor Hessam Babaee helped me finished my first project, and I could not have gotten on the right track quickly without his counselling. I would like to thank Professor Ling Guo, Dr. Yu-Hang Tang, Mr. Lu Lu, Mr. Liu Yang, Dr. Zhen Li and Dr. Xuhui Meng, and I have always enjoyed the discussion and brainstorm with you.

Further, I would like to send my thankfulness to the current and former Crunch Group members, including Dr. Xiu Yang, Dr. Zhongqiang Zhang, Dr. Heyrim Cho, Dr. Zhiping Mao, Dr. Fangying Song, Dr. Mingge Deng, Dr. Yue Yu, Mr. Minglang Yin, Dr. Guofei Pang, Mr. Ansel Blumers, Ms. Yixiang Deng, Dr. He

Li, Dr. Xiaoning Zheng, etc. Also, I would like to thank all the faculty and staff members in the Division of Applied Mathematics at Brown University. I heartily appreciate living and working in this big supportive family. In addition, my special thanks goes to my friends and my badminton/basketball teammates.

Last but not least, I have my utmost gratitude towards my family members, especially to my mother Lin and my wife Xuan. Without your endless love, support and encouragement, I would not be able to complete the journey of this Ph.D. study.

My dissertation work is supported by the following grants:

- Brown Univeristy graduate school fellowship;
- Brown University teaching assistant fellowship;
- AFOSR MURI, FA9550-09-1-0613;
- ARO, W911NF-14-1-0425;
- ARO MURI, W911NF-15-1-0562;
- ARO, W911NF-18-1-0301.

Contents

List of Tables

List of Figures

Chapter One

Introduction

In recent years we have been witnessing the rise of a data-driven era in which probability and statistics have been the focal point in the development of disruptive technologies, such as probabilistic machine learning. Machine learning tools, e.g., the deep neural networks (DNNs), have regained enormous attention not only because of their extensive expressiveness and huge capability of processing vast amount of data, but also due to the exponentially increasing computing speed and the incessantly development of effective optimization algorithms. How to make the best use of existing data while exploiting the information from classical mathematical models or even empirical correlations developed within a discipline is an important issue, as data-driven modeling is emerging as a powerful paradigm for physical and biological systems. As with constructing models driven by data, it is equally important to quantify the uncertainty associated with these models. This explains why data assimiliation is one of the most popular topics in the uncertainty quantification (UQ) community. However, it is generally very difficult to build effective mathematical models for complex stochastic systems, especially when we also take into account the information from big-data. To achieve this ultimate goal, there are at least two distinct challenges:

1. **The lack of information fusion algorithms.** It happens that the system may not preserve a consistent physical property, then it is not applicable to described the whole system with a unified set of mathematical equations. For example, when dealing with the multi-scale heterogeneous system, a coarse-grained model will not resolve the fine-scale patterns that are important in some region, while a high-resolution model is too expensive to be implemented to the whole domain, thus an algorithm that can propagate uncertainty across between scales and subdomains is needed. It also happens that the system is not only restricted by the prescribed physical law,

which usually takes the form of partial differential equations (PDEs), but also has to be compatible with experimental measurements. Therefore, it is handy to have an algorithm that can merge the information from both the physical laws and the big-data.

2. **The difficulty in quantifying uncertainty that comes from various sources.** There is model uncertainty, where a high-fidelity model is more trust worthy than a low-fidelity one, as the high-fidelity model is less likely to fail. There can be uncertainty caused by different approximation strategies, for example, the uncertainty introduced from truncating an infinite expansion series, or discretizing a continuous function, or using DNNs that are randomly initialized as surrogate models. There is uncertainty associated with data, too. For example, we cannot eliminite the flucuations in repeated measurements due to the random measurement error. The qualitative discription sounds simple and plain, but performing an in-depth quantitative analysis requires innovative UQ strategies.

In this dissertation I attempt to address these challenges by providing original mathematical models/frameworks. They are established based on existing machine learning tools, such as Gaussian Process Regression (GPR) and DNNs, and they turn out to be viable options in modeling data-driven stochastic complex systems. Specifically, I will discuss the following two aspects of modeling data-driven stochastic systems:

- Domain decomposition;
- Data-driven UQ.

In this chapter, I will first give a brief review of several key concepts that

are repeatedly used in this dissertation as building blocks. Among them, the Karhunen-Loève expansion and generalized polynomial chaos are classical tools extensively used when modeling stochastic systems. The Schwarz alternating method inspired me to analyze the uncertainty propagation across domain interfaces, and also led to the proposed PDE-data information fusion algorithm. The Physics-Informed Neural Networks (PINNs), which have just became popular in the last couple of years, laid the foundation of physics-informed machine learning and made data-driven UQ possible. An overview of the included works will be presented at last.

1.1 Karhunen-Loève Expansion of Random Fields

Applying order reduction schemes have been a popular approach for solving UQ problems, and among which the Karhunen-Loève (KL) expansion [29, 53] turns out to be useful especially when dealing with high-dimensional complex systems, as it provides a low-dimensional representation for second-order random fields that is optimal in the mean square sense.

Let $(\Omega, \mathcal{F}, P)$ be the probability space, where Ω is the sample space, $\mathcal{F}$ is the $\sigma-$algebra of subsets of Ω, and P is a probability measure. Let $\omega \in \Omega$ indicate a random event and $u(x;\omega)$ indicate a random process indexed by $x \in \mathcal{D}$, which, for example, can be regarded as the physical space coordinate, thus $u(x;\omega)$ is a random process in the physical space $\mathcal{D}$. The following discussions about the KL expansion are also valid for time and space-time indexed random processes. Given a second-order process $u(x;\omega)$, i.e., $\mathbb{E}[u(x;\omega)^2] < \infty$, the KL expansion reads

$$u(x;\omega) = \bar{u}(x) + \sum_{i=1}^{\infty} \sqrt{\lambda_i}\phi_i(x)\xi_i(\omega), \quad x \in \mathcal{D},\ \omega \in \Omega, \tag{1.1}$$

where the random field $u(x;\omega)$ is decomposed into two parts: (i) the deterministic mean field function $\bar{u}(x) = \mathbb{E}[u(x;\omega)]$; and (ii) the random fluctuation part as a linear combination of infinitly many deterministic orthonormal basis $\phi_i(x)$ multiplied by stochastic coefficients $\xi_i(\omega)$. The basis $\phi_i(x)$ are normalized eigenfunctions of the covariance kernel of $u(x;\omega)$, and they solve the following Fredholm integral equation

$$\int_{\mathcal{D}} C(x,y)\phi_i(x)\,\mathrm{d}x = \lambda_i\phi_i(y), \quad \text{for } x, y \in \mathcal{D},\ i = 1, 2, \ldots, \tag{1.2}$$

where λ_i denotes the eigenvalues corresponding to ϕ_i, and $C(x,y)$ denotes the covariance between $u(x;\omega)$ and $u(y;\omega)$, i.e.,

$$C(x,y) = \mathbb{E}\left[(u(x;\omega) - \bar{u}(x))^T\,(u(y;\omega) - \bar{u}(y))\right], \quad x, y \in \mathcal{D}. \tag{1.3}$$

Since $C(x,y)$ is symmetric and positive definite, the eigenvectors ϕ_i are naturally orthogonal so that $\langle\phi_i, \phi_j\rangle = \delta_{ij}$, where δ_{ij} is the Dirac's delta function. The eigenvalues λ_i are non-negative and are arranged in a descending order. The random variables $\xi_i(\omega)$ have zero mean and are mutually uncorrelated, i.e. $E[\xi_i\xi_j] = \delta_{ij}$, and they can be calculated as follows:

$$\xi_i(\omega) = \frac{1}{\lambda_i}\int_{\mathcal{D}} (u(x;\omega) - \bar{u}(x))\,\phi_i(x)\,\mathrm{d}x\,, \text{ for } i = 1, 2, \ldots. \tag{1.4}$$

The KL expansion, in the form of Eq. 1.1, is of little use because it is an infinite

series. In practice, one adopts a finite series expansion:

$$\hat{u}(x;\omega) = \bar{u}(x) + \sum_{i=1}^{d} \sqrt{\lambda_i}\phi_i(x)\xi_i(\omega), \quad \omega \in \Omega. \tag{1.5}$$

The answers to where to truncate the expansion and how to choose d are closely related to the decay of eigenvalues λ_i as the index i increases. A common approach is to examine the decay of λ_i and keep the first d eigenvalues so that the contribution of the rest of the eigenvalues is negligible.

1.2 Generalized Polynomial Chaos

First defined by Wiener in [85] as the span of Hermite polynomial functionals of a Gaussian random field, polynomial chaos (PC) has been extensively used to model uncertainty in physical applications. A more general extension, named the generalized polynomial chaos (gPC), was proposed by Xiu and Karniadakis [89]. The gPC employs more types of orthogonal polynomials from the Askey family by exploiting the correspondence between the probability density functions (PDFs) of certain random variables and the weight functions of the orthogonal polynomials.

Again, let $(\Omega, \mathcal{F}, P)$ be the probability space, where Ω is the sample space, $\mathcal{F}$ is the $\sigma-$algebra of subsets of Ω, and P is a probability measure. Let $\boldsymbol{\xi} = (\xi_1, ..., \xi_d)$ be an d-dimensional continuous random variable with PDF $f(\boldsymbol{\xi})$. A general second-order random process $R(\omega) \in L_2(\Omega, \mathcal{F}, P)$ can be expressed by gPC as

$$R(\omega) = \sum_{i=0}^{+\infty} a_i \Phi_i(\boldsymbol{\xi}(\omega)), \tag{1.6}$$

where $\omega \in \Omega$ is the random event and $\Phi_i(\boldsymbol{\xi}(\omega))$ denotes the gPC basis in $L_2(\Omega, \mathcal{F}, P)$. They are orthonormal polynomials of $\boldsymbol{\xi}(\omega)$, satisfying

$$\mathbb{E}[\Phi_i \Phi_j] = \mathbb{E}[\Phi_i^2]\delta_{ij}, \tag{1.7}$$

where $\mathbb{E}$ denotes the expectation with respect to the probability measure $\mathrm{d}P(\omega) = f(\boldsymbol{\xi}(\omega))\mathrm{d}\omega$. The index in Eq. 1.6 takes the lexicographical order for a fixed random dimension d that counts the orthogonal polynomials with an increasing degree. In general, the expansion has an infinite number of terms, but in practice, it is truncated at a certain level. For instance, the number of gPC terms of a d-dimensional random input truncated at a designated maximum polynomial degree p is $(d+p)!/(d!\,p!)$, while the number of random dimension d of a stochastic process is usually determined by the Karhunen-Loève expansion based on the decay of eigenvalues. For certain random vector $\boldsymbol{\xi}$, the orthogonal gPC basis $\{\Phi_i\}$ can be chosen such that its weight function has the same form as the PDF $f(\boldsymbol{\xi})$ of $\boldsymbol{\xi}$. The corresponding types of classical orthogonal polynomials and their associated random variable types can be found in [87] and will not be listed here.

As a remark, the PC expansion can be furthur generalized to functions with arbitrary distributed random variables [84, 86] or even data-driven problems [50] and random variables with discrete measures [92].

1.3 The Schwarz Alternating Method

The Schwarz alternating method is a reliable framework for coupling solutions in different domains. It is especially useful to find a C_1 continuous solution in

the entire domain when one only has access to black-box solvers that are valid in disjoint partitions of the domain. The original Schwarz algorithm [71] employs overlapping subdomains, but other versions [43] employ non-overlapping subdomains. Here we present a version first proposed by Funaro et al. [17] and modified in [26].

Take the following elliptic Helmholtz equation as an example:

$$\begin{aligned} -\nabla^2 u + \lambda^2 u - f(x) = 0, \quad & x \in \mathcal{D}, \\ u = 0, \quad & x \in \partial\mathcal{D}. \end{aligned} \tag{1.8}$$

For a 1D domain, consider the decomposition in Figure 1.1. We refer to $\mathcal{D}_m$, whose length is l_m, as the *dominus* domain where we impose a Dirichlet condition at the interface x_b, and refer to $\mathcal{D}_s$, with length l_s, as the *servus* domain with a Neumann interface condition. To this end, we use n to denote the number of iterations and

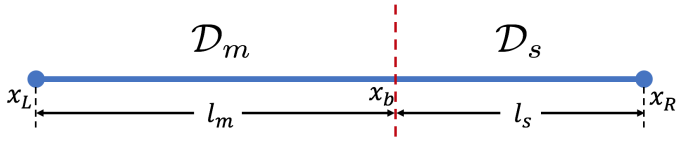


Figure 1.1: 1D domain decomposition diagram, where $\mathcal{D}_m$ is the *dominus* domain with length l_m, and $\mathcal{D}_s$ is the *servus* domain with length l_s. The two subdomains share the boundary point x_b.

set up an iterative scheme that involves the Dirichlet and Neumann problems assigned to the *dominus* and *servus* domains, respectively.

dominus domain:

$$\begin{aligned} -(u_{xx})_m^n + \lambda^2 u_m^n = f(x), \quad & \text{on } \mathcal{D}_m, \\ u(x_L) = 0, & \\ u_m^n(x_b) = u_s^{n-1}(x_b), & \end{aligned} \tag{1.9}$$

servus domain:

$$\begin{aligned} -(u_{xx})_s^n + \lambda^2 u_s^n &= f(x), \quad \text{on } \mathcal{D}_s, \\ u(x_R) &= 0, \\ (u_x)_s^n(x_b) &= (u_x)_m^k(x_b). \end{aligned} \tag{1.10}$$

In Eq. 1.9 and Eq. 1.10, u_m^n and u_s^n denote the solutions for their respective subdomains during the n-th iteration. There are two choices for the index k in Eq. 1.10 depending on whether there is a serial or a parallel implementation. These values are: (1) a serial version $k = n$; and (2) a parallel version $k = n - 1$. The iterative process stops when the change of solution at the interface between successive iterations is smaller than a designated threshold ϵ, i.e.,

$$|u_m^n(x_b) - u_m^{n-1}(x_b)| + |u_s^n(x_b) - u_s^{n-1}(x_b)| < \epsilon. \tag{1.11}$$

The following theorem holds regarding to the convergence of this scheme.

Theorem 1.3.1 (see [31])**.** *The iterative procedure Eq. 1.9 and Eq. 1.10 converges if and only if* $l_m/l_s > 1$.

In application, a relaxed version of Dirichlet boundary condition [31, 21] is carried out by imposing a proper relaxation parameter η to updating the Dirichlet boundary condition, that is

$$u_m^n(x_b) = \eta u_s^{n-1}(x_b) + (1 - \eta) u_m^{n-1}(x_b). \tag{1.12}$$

This relaxed scheme converges for subdomains of any lengths, as long as the relaxation parameter η is chosen so that

$$0 < \eta < \Theta_S = \frac{2}{1 + G(\lambda)}, \tag{1.13}$$

where $G(\lambda)$ is the error growth factor, defined by

$$G(\lambda) = \begin{cases} \frac{l_s}{l_m}, & \lambda = 0; \\ \frac{\tanh(\lambda l_s)}{\tanh(\lambda l_m)}, & \text{otherwise}, \end{cases} \tag{1.14}$$

and λ^2 is the constant in Eq. 1.8. As mentioned in [31], we could choose $\eta = \Theta_S/2$ to achieve an optimal convergence rate.

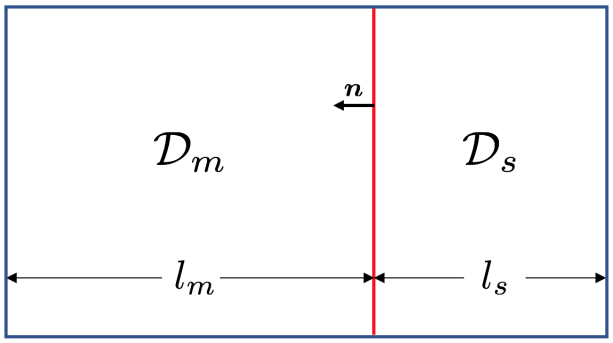


Figure 1.2: 2D domain decomposition diagram, where $\mathcal{D}_m$ is the *dominus* domain, $\mathcal{D}_s$ is the *servus* domain, and $\boldsymbol{n}$ is the outer norm of $\mathcal{D}_s$ at the interface.

Similarly, for a 2D domain decomposition, e.g. Figure 1.2, the serial iterating boundary conditions reads,

$$u_m^n = \eta u_s^{n-1} + (1-\eta) u_m^{n-1}, \quad \boldsymbol{n} \cdot \nabla u_s^n = -\boldsymbol{n} \cdot \nabla u_m^n, \tag{1.15}$$

where $\boldsymbol{n}$ is the outward unit normal vector of the *servus* domain at the interface. The convergence of the iterative scheme in 2D is guaranteed by the following theorem:

Theorem 1.3.2 (see [31, 21])**.** *The serial iterative procedure for two dimensions with the interface conditions Eq. (1.15) converges, irrespective of the initial guess $u_m^0(x_b)$ for a fixed*

relaxation parameter η, if and only if

$$0 < \eta < \frac{2}{1 + G(\lambda_q)}, \tag{1.16}$$

where

$$\lambda_q = \lambda^2 + \frac{q^2\pi^2}{4}, \; q \geq 1. \tag{1.17}$$

The parameter q is a frequency component greater than 1. Note that when $l_m > l_s$, $G(\lambda_q)$ is always smaller than one thus $2/(1 + G(\lambda_q))$ is always greater than one. Therefore, choosing any η from $(0, 1]$ will ensure the convergence of the iterative scheme regardless of q.

1.4 Physics-Informed Neural Networks

In this part, we briefly review the approach of using DNNs to solve differential equations [38, 39, 62], and its generalization in [63] for solving inverse problems. To demonstrate the idea, we solve an ordinary differential equation (ODE) and we remark that the algorithm to be discussed can be effortlessly extended to solving PDEs. Suppose we have a parametrized ODE:

$$\begin{aligned} \mathcal{N}_x[u; \eta] &= 0, \quad x \in \mathcal{D}, \\ \text{B.C.:} \qquad \mathcal{B}_x[u] &= 0, \quad x \in \Gamma, \end{aligned} \tag{1.18}$$

where $\mathcal{N}_x$ is a differential operator, $\mathcal{B}_x$ is the linear boundary condition operator, η denotes the collection of the ODE parameters, and $u(x)$ is the quantity of interest.

A DNN, denoted by $\hat{u}(x; \theta)$, is constructed as a surrogate of the solution $u(x)$.

It takes the coordinate x as the input and outputs a vector that has the same dimensionality as $u(x)$. Here we use θ to represent the DNN parameters that will be tuned at the training stage, namely, for a classical dense neural network, θ contains all the weights $\boldsymbol{w}$ and biases $\boldsymbol{b}$ in $\hat{u}(x;\theta)$. We can take derivatives of this surrogate network $\hat{u}$ with respect to its input by applying the chain rule for differentiating compositions of functions using the automatic differentiation, which is conveniently integrated in many machine learning packages such as Tensorflow [1]. The restrictions on $\hat{u}(x;\theta)$ come in two folds: first, given the data of scattered $u(x)$ observations, the neural network should be able to recover the observed value, when taking the associated x as the input; and second, $\hat{u}(x;\theta)$ should comply with the physical laws imposed by Eq. 1.18. The second part is achieved by defining a residual network:

$$\hat{f}(x;\theta,\eta) \coloneqq \mathcal{N}_x\left[\hat{u}(x;\theta);\eta\right], \tag{1.19}$$

which can be computed from $\hat{u}(x;\theta)$ straightforwardly with automatic differentiation. This residual network $\hat{f}(x;\theta,\eta)$, also named the Physics-Informed Neural Network (PINN), shares the same parameters θ with network $\hat{u}(x;\theta)$ and should output constant 0 for any input $x \in \mathcal{D}$ in ideal situations. Figure 1.3 shows a sketch of the PINN. At the training stage, the shared parameters θ are fine-tuned to minimize a loss function that reflects the above two constraints.

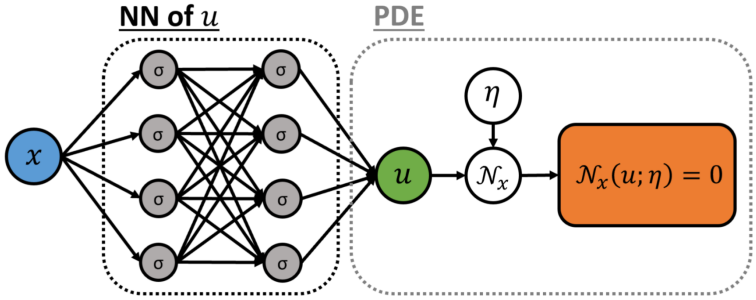


Figure 1.3: Schematic of the PINN for solving differential equations.

The process of solving the ODE Eq. 1.18 is no more than training the neural network $\hat{u}(x;\theta)$ with a properly designed loss function. Suppose we have a total number of N_u observations on u, collected at location $\{x_u^{(i)}\}_{i=1}^{N_u}$, and N_c is the number of collocation points $\{x_f^{(i)}\}_{i=1}^{N_c}$ where we evaluate the residual $\hat{f}(x_f^{(i)};\theta,\eta)$. A straight forward way to write the loss function is to put together the mean squared error (MSE) on $u(x_u^{(i)})$ and the MSE on $f(x_f^{(i)})$, as they provide the boundary conditions (or sometimes the necessary restrictions at ancher points) and the restrictions of physical laws in the form of the ODE, accordingly. As an example, the loss function can be written as:

$$\mathcal{L}(\theta,\eta) = \frac{1}{N_u}\sum_{i=1}^{N_u}\left[\hat{u}(x_u^{(i)};\theta) - u(x_u^{(i)})\right]^2 + \frac{1}{N_c}\sum_{i=1}^{N_c}\hat{f}(x_f^{(i)};\theta,\eta)^2; \tag{1.20}$$

The workflow of solving an ODE using PINN is summarized below:

Algorithm 1: PINN for solving ODEs

Step 1: Specify the training set:

$$\hat{u}(x;\theta) \text{ network: } \{(x_u^{(i)}, u(x_u^{(i)}))\}_{i=1}^{N_u}, \quad \hat{f}(x;\theta,\eta) \text{ network: } \{(x_f^{(i)}, 0)\}_{i=1}^{N_f};$$

Step 2: Construct a DNN $\hat{u}(x;\theta)$ with properly initialized parameters θ;

Step 3: Construct the residual network $\hat{f}(x;\theta,\eta)$ by substituting the surrogate $\hat{u}(x;\theta)$ into the governing equation (Eq. 1.19) via automatic differentiation and arithmetic operations;

Step 4: Specify a loss function (for example, Eq. 1.20) and a proper optimization algorithm (for example, the Adam algorithm [35]) to minimize the loss function;

Step 5: Train the DNNs for a designated number of epochs;

Step 6: Estimate the quantity of interest by evaluating the well-trained neural network $\hat{u}(x;\theta)$ with the interested coordinates as input.

Note that in the situation of solving an inverse problem, where we do not know the value of η, but instead we have additional information on $u(x)$ and we try to generate reasonable estimates for η, the ODE parameters η can be regarded as variables similar to θ, and shall be tuned together with θ at the training stage.

1.5 Overview of Dissertation

The rest of this dissertation is organized as follows. Chapter 2 and Chapter 3 focus on the first aspect of data-driven stochastic modeling, i.e., designing information

fusion algorithms: in Chapter 2, the algorithm for propagating uncertainty across domains via moment minimization (SDD-MM) is presented, and in Chapter 3, I introduce an algorithm for bi-directional coupling of PDE- and Data-domains named the Domain Decomposition with Gaussian Process Regression (GPDD). The second aspect, quantifying uncertainty of data-driven complex systems, is mainly addressed in Chapter 4 and Chapter 5: in Chapter 4, I discuss the a data-driven strategy for solving stochastic forward (model inference) and inverse (model identification) problems, and in Chapter 5, we exhibit the potential for PINNs in solving time-dependent stochastic problems. At last, a summary of this dissertation is given in Chapter 6.

Chapter Two

Stochastic Domain Decomposition via Moment Minimization

2.1 Introduction

We are interested in the uncertainty propagation in stochastic simulations where multi-scale phenomena are present, e.g., atmospheric boundary layers, catalysis, surface nano-patterning, etc.. In many situations, the randomness can manifest itself across scales, yielding a multi-scale random behavior with vastly changing correlation lengths. In practical simulations, we may need to use heterogeneous solvers, and hence we are not guaranteed to have access to the *global* random solution trajectories. In general, it is usually very hard, if not impossible, to solve a hybrid multi-scale stochastic problem as a single stochastic PDE. To address this issue, we develop a stochastic domain decomposition framework that utilizes local solvers and does not require access to global random trajectories. In particular, we develop a framework to derive boundary conditions at the interface of the decomposed domains.

Domain decomposition methods have already been used extensively in deterministic problems [31, 72] to speed up computation, but the idea of stochastic domain decomposition has only been considered fairly recently. Liao and Willcox [41] proposed a domain decomposition uncertainty quantification (DDUQ) method that combines domain decomposition and importance sampling to solve uncertainty quantification (UQ) problems. In DDUQ, local SPDE solutions are pre-calculated during an offline stage, at sample points drawn from an assumed joint distribution of the random input variables and the interface parameters. Then during the online stage, the interface parameters are determined in an iterative manner and a target joint PDF of the input variables and the interface parameters is estimated. The solution to the UQ problem is assembled by reweighing the local SPDE solutions using importance sampling. The DDUQ method allows

users to conduct stochastic analysis at the local level using different UQ strategies. However, estimating PDFs of high dimensional coupled random variables can be very difficult, and characterizing subdomain interfaces with interface parameters requires solving a handful of fully coupled system simulations. In another attempt, Chen, et al. [6] aimed at solving SPDEs with black-box solvers at reduced computational cost. They developed a local polynomial chaos expansion method for *linear* SPDEs, where the local problems are solved in a low dimensional random space and represented as polynomial chaos expansions. For each sample of the global random process, the interface boundary variables are determined by proper subdomain coupling conditions, and then the corresponding local solutions can be obtained. In this method, the computational cost is largely reduced, while high accuracy is also maintained, but it only works for *linear* SPDEs, and requires access to the global random process samples. In order to overcome these limitations, Cho, et al. [9] introduced a two-level domain decomposition approach, where the whole domain is decomposed into overlapping $\mathcal{D}$ subdomains (supporting the solution) and interfacing I subdomains (supporting the stochastic input). The correlation kernel of the entire domain is filtered and the randomness is parametrized locally within each I domain. Since every two adjacent $\mathcal{D}$ domains share one I domain, the moments of the subdomain solutions conditioned on this I domain should be the same. Two different methods were developed, a Schwarz type iterative method (conditional moment interface method) and an optimization method (PDE-constrained interface method), to match the moments. These methods work for both linear and nonlinear problems and produce accurate *mean* solutions. However, filtering the covariance function eliminates long-range correlations, and hence, it does not represent accurately the global correlation structure. Therefore, neither of these two methods in Cho, et al. [9] can generate accurate high order moments.

We propose a stochastic domain decomposition method (SDD-MM) based on a new interface condition, derived from minimizing the second-order statistical moments of the difference of local solutions at the subdomain interface. The new interface condition is general, and works with diverse UQ methods, e.g., generalized polynomial chaos (gPC) [89], Dynamically Orthogonal (DO) [69, 11] or Bi-Orthogonal (BO) [81] expansions, multi-level Monte Carlo method, etc.. Unlike the conditional moment interface method of Cho, et al. [9], which imposes conditions on the solutions samples in the overlapping domains, we impose conditions on the expansion coefficients derived explicitly via moment minimization in non-overlapping domains. We adopt the same test problems to compare the performance of the SDD-MM method against the conditional moment interface method, and we also extend the numerical examples to solve a two-dimensional Allen-Cahn equation. For demonstration purposes, we use gPC expansion of different orders in different domains in the numerical examples.

The organization of the chapter is as follows. In Section 2.2, we set up the general SPDE problem and the domain decomposition. In Section 2.3, we explain how we parametrize randomness locally. In Section 2.4, which is the core of this chapter, we present the stochastic domain decomposition via the moment minimization algorithm, and derive the moment minimizing interface condition. We also analyze the computational complexity of SDD-MM in this section. In Section 2.5 we present our numerical results, and we conclude in Section 2.6.

2.2 Problem Setup

Let us consider the standard setup of a UQ problem,

$$\begin{cases} \mathcal{L}(x,t,u(x,t;\omega);\omega) = f(x,t;\omega), & x \in \mathcal{D}; \\ \mathcal{B}(u(x,t;\omega)) = g(x,t;\omega), & x \in \partial\mathcal{D}; \\ u(x,0;\omega) = u_0(x;\omega), & x \in \mathcal{D}. \end{cases} \tag{2.1}$$

Here, $\mathcal{D} \subset \mathbb{R}^d$ is a bounded spatial domain where the SPDE is defined, and $x \in \mathbb{R}^d$ denotes the spatial variable. We refer to $\mathcal{D}$ as the global domain, and let $\partial\mathcal{D}$ be the boundary of $\mathcal{D}$. $\mathcal{L}$ is a stochastic differential operator, $\mathcal{B}$ is the boundary condition operator, and $u_0(x;\omega)$ is the initial condition; $u(x,t;\omega)$ is the solution to this SPDE, and we refer to it as the global solution, with

$$u(x,t;\omega) : \mathcal{D} \times [0,T] \times \Omega \to \mathbb{R},$$

where $T > 0$ is the end time of computation, and Ω is the random space. The random input can come from the stochastic differential operator, the boundary conditions, or the initial conditions. In practice, $\mathcal{L}$ may not have an explicit expression and a global solver for Eq. 2.1 may not exist. In general, we assume that $\mathcal{D}$ is partitioned into *non-overlapping* subdomains with heterogeneous solvers. For the sake of simplicity, we consider two subdomains where we divide $\mathcal{D}$ into non-overlapping subdomains $\mathcal{D}_1$ and $\mathcal{D}_2$, such that

$$\overline{\mathcal{D}} = \overline{\mathcal{D}_1} \cup \overline{\mathcal{D}_2}, \quad \mathcal{D}_1 \cap \mathcal{D}_2 = \emptyset. \tag{2.2}$$

Let $u_1(x,t;\omega)$ and $u_2(x,t;\omega)$ be the SPDE solutions in $\mathcal{D}_1$ and $\mathcal{D}_2$, and we will

refer to them as local solutions. Hence, $u_1(x,t;\omega)$ and $u_2(x,t;\omega)$ satisfy

$$\begin{cases} \mathcal{L}(x,t,u_i(x,t;\omega);\omega) = f(x,t;\omega), & x \in \mathcal{D}_i \\ \mathcal{B}(u_i(x,t;\omega)) = g(x,t;\omega), & x \in \partial\mathcal{D}_i \cap \partial\mathcal{D} \\ u_i(x,0;\omega) = u_0(x;\omega), & x \in \mathcal{D}_i \\ \mathcal{I}_i(u_i(x,t;\omega)) = \beta_i(x,t;\omega), & x \in \partial\mathcal{D}_i \backslash \partial\mathcal{D} \end{cases}, \quad i \in \{1,2\}. \tag{2.3}$$

We assume that after decomposing $\mathcal{D}$ into subdomains $\mathcal{D}_1$ and $\mathcal{D}_2$, the subdomain problems Eq. 2.3 are still well-posed and can be solved independently given proper interface conditions β_i.

2.3 Local Randomness Parametrization

Suppose the random inputs in Eq. 2.1 take the form of random processes. In the situation when we have a multi-scale problem or a hybrid stochastic system, sampling the whole trajectory of the random process might be impossible. Therefore, an important aspect of our domain decomposition framework is to avoid using global random process trajectories, which in the end requires the local parametrization of random process. We propose a local randomness parametrization method that not only reflects the local stochastic behavior, but also takes into account the cross-correlations between different subdomains.

2.3.1 Embedded Random Process

Let $a(x;\omega) : \mathcal{D} \times \Omega \to \mathbb{R}$ be a random process in the global domain $\mathcal{D}$. Assume its expectation $\bar{a}(x)$ and covariance function $K(x,y)$ are provided as part of the problem setup, that is:

$$\bar{a}(x) = \mathbb{E}[a(x;\omega)],$$
$$K(x,y) = \mathbb{E}[(a(x;\omega) - \bar{a}(x))(a(y;\omega) - \bar{a}(y))], \quad x, y \in \mathcal{D}.$$

By definition, $K(x,y)$ is real symmetric and positive definite. For random variables X and Y, define $\mathrm{Cov}(X,Y)$ as $\mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])]$, so $K(x,y)$ equals $\mathrm{Cov}(a(x;\omega), a(y;\omega))$.

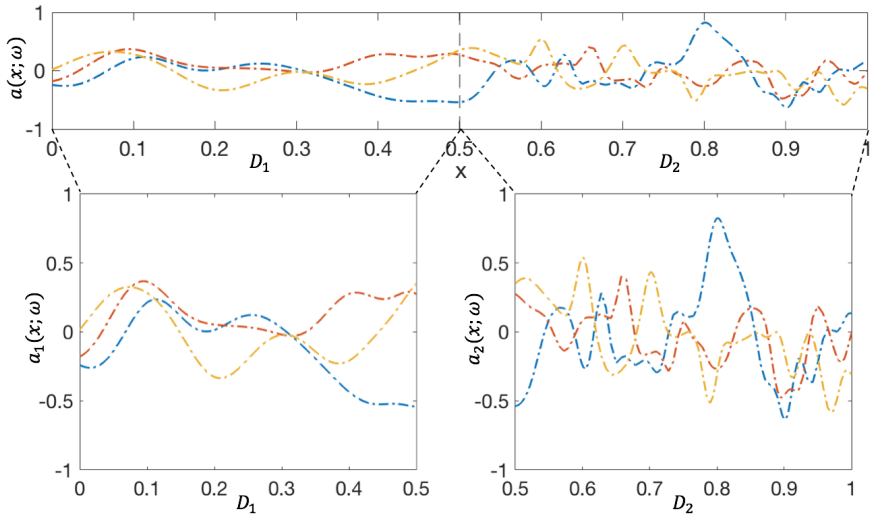


Figure 2.1: Decomposing domain $[0,1]$ into two non-overlapping subdomains $\mathcal{D}_1$ and $\mathcal{D}_2$, where $a(x;\omega)$ is a global random process with varying correlation lengths; $a_1(x;\omega)$ and $a_2(x;\omega)$ are the embedded local random processes.

Given the domain decomposition Eq. 2.2, we can embed $a(x;\omega)$ into the subdomains in a natural way,

$$a(x;\omega) = \sum_{i=1}^{2} a_i(x;\omega)\mathbb{1}_{\mathcal{D}_i}(x), \tag{2.4}$$

where $\mathbb{1}$ is the indicator function satisfying $\mathbb{1}_S(x) = 1$ if $x \in S$, and $\mathbb{1}_S(x) = 0$ if $x \notin S$. Figure 2.1 displays the non-overlapping domain decomposition of the interval $[0, 1]$, and how we embed the stochastic process into the subdomains. Accordingly, $K(x, y)$ can be written as

$$\begin{aligned} K(x,y) = K_{11}(x,y)\mathbb{1}_{\mathcal{D}_1\times\mathcal{D}_1}(x,y) + K_{22}(x,y)\mathbb{1}_{\mathcal{D}_2\times\mathcal{D}_2}(x,y) \\ + K_{12}(x,y)\mathbb{1}_{\mathcal{D}_1\times\mathcal{D}_2}(x,y) + K_{21}(x,y)\mathbb{1}_{\mathcal{D}_2\times\mathcal{D}_1}(x,y), \end{aligned} \tag{2.5}$$

where

$$\begin{aligned} K_{ii}(x,y) &= \mathrm{Cov}\left(a_i(x;\omega), a_i(y;\omega)\right), \quad x, y \in \mathcal{D}_i, \quad i \in \{1,2\}; \\ K_{12}(x,y) &= \mathrm{Cov}\left(a_1(x;\omega), a_2(y;\omega)\right), \quad x \in \mathcal{D}_1, \quad y \in \mathcal{D}_2; \\ K_{21}(x,y) &= \mathrm{Cov}\left(a_2(x;\omega), a_1(y;\omega)\right), \quad x \in \mathcal{D}_2, \quad y \in \mathcal{D}_1. \end{aligned} \tag{2.6}$$

2.3.2 Local Karhunen-Loève Expansion

First we look only at one subdomain $\mathcal{D}_i$ $(i = 1, 2)$. Obviously, $K_{ii}(x, y)$ is positive definite, so the parametrization of a random process $a_i(x;\omega)$ within $\mathcal{D}_i$ follows the standard Karhunen-Loève (KL) expansion:

$$a_i(x;\omega) \approx \bar{a}_i(x) + \sum_{k=1}^{N_i} \phi_{i,k}(x)\sqrt{\lambda_{i,k}}\xi_{i,k}(\omega), \quad x \in \mathcal{D}_i, \quad N_i \in \mathbb{Z}^+, \tag{2.7}$$

where $\{\lambda_{i,k}, \phi_{i,k}(x)\}$ are the eigen-pairs of the eigenvalue problem

$$\int_{\mathcal{D}_i} K_{ii}(x,y)\phi_{i,k}(y)dy = \lambda_{i,k}\phi_{i,k}(x), \tag{2.8}$$

and $\xi_{i,k}$ are uncorrelated random variables defined by

$$\xi_{i,k}(\omega) = \frac{1}{\sqrt{\lambda_{i,k}}}\int_{\mathcal{D}_i} (a_i(x;\omega) - \bar{a}_i(x))\phi_{i,k}(x)dx, \quad k = 1, 2, ..., N_i. \tag{2.9}$$

If $a_i(x;\omega)$ is a Gaussian random process, all $\xi_{i,k}$ $(k = 1, ..., N_i)$ are independent standard normal random variables. In this chapter, we consider the case where $a(x;\omega)$ is a Gaussian random process.

Eq. 2.7 can be rewritten in vector form as

$$a_i(x;\omega) = \bar{a}_i(x) + \Phi_i(x)\sqrt{\Lambda_i}\boldsymbol{\xi_i}, \tag{2.10}$$

where Λ_i is a diagonal matrix of eigenvalues $\Lambda_i = \text{diag}([\lambda_{i,1}, \lambda_{i,2}, ..., \lambda_{i,N_i}])$, $\Phi_i(x)$ is the matrix of eigenfunctions $\Phi_i(x) = (\phi_{i,1}(x), \phi_{i,2}(x), ..., \phi_{i,N_i}(x))$, and $\boldsymbol{\xi_i}$ is a random vector $\boldsymbol{\xi_i} = (\xi_{i,1}, \xi_{i,2}, ..., \xi_{i,N_i})^T$.

2.3.3 Correlation Parametrization

Consider the cross-correlations of the random process between $\mathcal{D}_1$ and $\mathcal{D}_2$. This is especially important when the entries in K_{12} are large, and thus these entries will have a non-negligible impact on the eigenvalues and eigenvectors of the global covariance function $K(x, y)$. In the previous part, the local random processes are parametrized as random vectors $(\xi_{1,1}, \xi_{1,2}, ..., \xi_{1,N_1})^T$ and $(\xi_{2,1}, \xi_{2,2}, ..., \xi_{2,N_2})^T$. Both random vectors are independent within their respective domains, but due to the effect of K_{12}, cross-correlations exist between them. Combining Eq. 2.6 and Eq. 2.10, we obtain

$$\begin{aligned} K_{12}(x, y) &= \text{Cov}\left(a_1(x;\omega), a_2(y;\omega)\right) \\ &= \text{Cov}\left(\Phi_1(x)\sqrt{\Lambda_1}\boldsymbol{\xi_1}(\omega), \Phi_2(y)\sqrt{\Lambda_2}\boldsymbol{\xi_2}(\omega)\right) \\ &= (\Phi_1(x)\sqrt{\Lambda_1})\,\text{Cov}\left(\boldsymbol{\xi_1}, \boldsymbol{\xi_2}\right)(\sqrt{\Lambda_2}\Phi_2(y)^T). \end{aligned} \tag{2.11}$$

In practice, given a spatial discretization rule, $\Phi_1(x)$ and $\Phi_2(y)$ are unitary matrices, and their pseudo-inverses $\Phi_1(x)^{-1}$ and $\Phi_2(x)^{-1}$ can be calculated. Also, $K_{12}(x, y)$ is

the matrix related to the cross-correlation. Eq. 2.12 calculates the cross-correlation of $\boldsymbol{\xi_1}$ and $\boldsymbol{\xi_2}$, and the correlation structure of $\boldsymbol{\xi_1}$ and $\boldsymbol{\xi_2}$ is depicted in Figure 2.2.

$$\text{Cov}\,(\boldsymbol{\xi_1}, \boldsymbol{\xi_2}) = \sqrt{\Lambda_1^{-1}}\Phi_1(x)^{-1}K_{12}(x,y)(\Phi_2(y)^{-1})^T\sqrt{\Lambda_2^{-1}}. \tag{2.12}$$

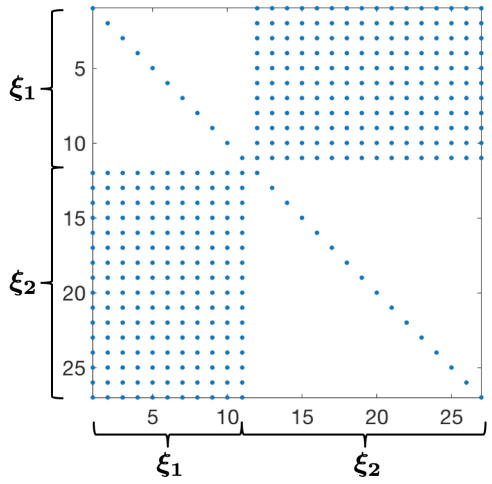


Figure 2.2: Correlation structure of $\boldsymbol{\xi_1}$ and $\boldsymbol{\xi_2}$. The dots show non-zero entries, thereby cross-correlations between random variables.

The question remains whether we can approximate the global random field $a(x;\omega)$ by generating samples of local random vectors $\boldsymbol{\xi_1}$ and $\boldsymbol{\xi_2}$, satisfying the above correlation. Let $\tilde{a}(x;\omega)$ be defined as

$$\tilde{a}(x;\omega) = \bar{a}(x) + \sum_{k=1}^{N_1}\phi_{1,k}(x)\sqrt{\lambda_{1,k}}\xi_{1,k}\mathbb{1}_{x\in\mathcal{D}_1} + \sum_{k=1}^{N_2}\phi_{2,k}(x)\sqrt{\lambda_{2,k}}\xi_{2,k}\mathbb{1}_{x\in\mathcal{D}_2}. \tag{2.13}$$

We assume that $\tilde{a}(x;\omega)$ is a reduced order reconstruction of the global random field $a(x;\omega)$. This assertion shall be verified by answering the following two questions:

1. Is $\{\phi_{i,k}\}$ a complete set of bases for $a(x;\omega)$ in $\mathcal{D}_i$?

2. Is $\tilde{a}(x;\omega)$ a Gaussian process, and does it have the same covariance function as $a(x;\omega)$?

Completeness of Local Bases

Since $\{\phi_{i,k}\}$ are eigenfunctions of the positive definite kernel K_{ii}, the proof of completeness comes directly from the Karhunen-Loève theorem. To numerically prove the completeness of $\{\phi_{i,k}\}$, we implement a spectral element discretization of the global domain $[0, 1]$, with 10 elements and 20^{th} order polynomial in each element. The global domain is divided into subdomains $\mathcal{D}_1$ and $\mathcal{D}_2$ of the same length, as shown in Figure 2.1. We generate samples of the global Gaussian random process $a(x; \omega)$ using a level two sparse grid stochastic collocation method [88]. In each $\mathcal{D}_i$ $(i = 1, 2)$, we project samples of the global random process to the local bases $\{\phi_{i,k}\}$ to get $\tilde{a}(x; \omega)$. Then, we evaluate the expectations of the projection error ϵ, i.e.,

$$\epsilon = \mathbb{E}\left[\|a(x; \omega) - \tilde{a}(x; \omega)\|_{L_2}\right].$$

As we can see in Figure 2.3a, by increasing the number of random variables used in each domain, i.e. the random dimension N_1 and N_2 (we have taken $N_1 = N_2$), the projection error decays exponentially.

Covariance Structure of $\tilde{a}(x; \omega)$

The random process $\tilde{a}(x; \omega)$ is Gaussian directly from its definition in Eq. 2.13. Also, the covariance function of $\tilde{a}(x; \omega)$ coincides with $K(x, y)$ directly from Eq. 2.11. The latter is verified numerically by showing that the finite dimensional representations of $\tilde{a}(x; \omega)$ and $a(x; \omega)$ have random coefficients of the same distribution. Here, we conduct the same spatial discretization as in the previous test. First, by sampling the correlated random variables $\boldsymbol{\xi_1}$ and $\boldsymbol{\xi_2}$, we generate plenty of samples of $\tilde{a}(x; \omega)$ from its definition Eq. 2.13. Samples of $\boldsymbol{\xi_1}$ and $\boldsymbol{\xi_2}$ are obtained by mul-

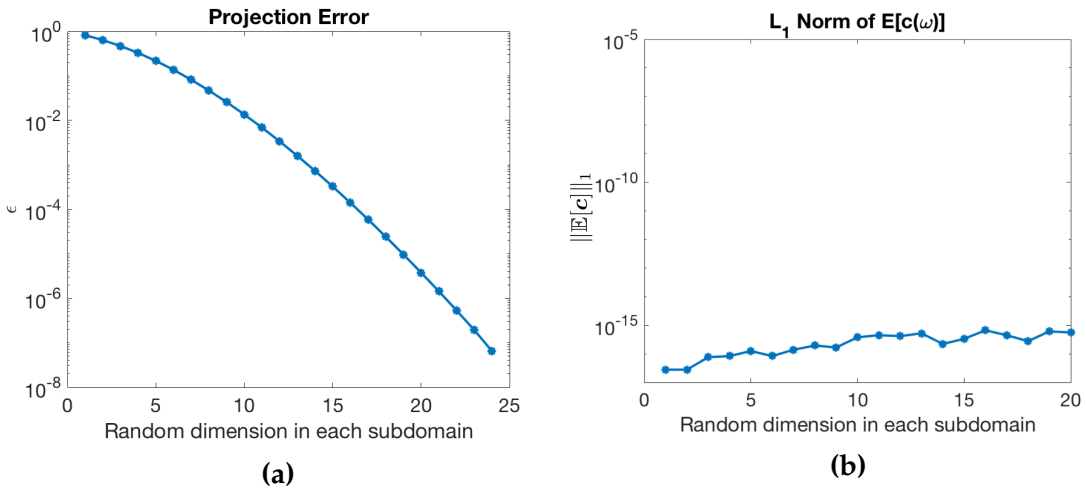


Figure 2.3: Accuracy of the local parametrization in two subdomains for the global random process of correlation length equal to 0.1, where $N_1 = N_2$: (a) the mean error of projecting $a(x;\omega)$ onto local bases $\{\phi_{i,k}\}$ decays exponentially when more random variables are used; (b) the largest value in $\{|\mathbb{E}[c_i]| \; | i = 1, 2, ..., N\}$ is close to the level of machine accuracy.

tiplying a lower-triangular matrix, getting from the Cholesky decomposition of the covariance matrix $\mathrm{Cov}(\boldsymbol{\xi_1}, \boldsymbol{\xi_2})$, to the sample vectors of independent normal distributed random variables [44]. Next, we write down the global KL expansion of $a(x;\omega)$,

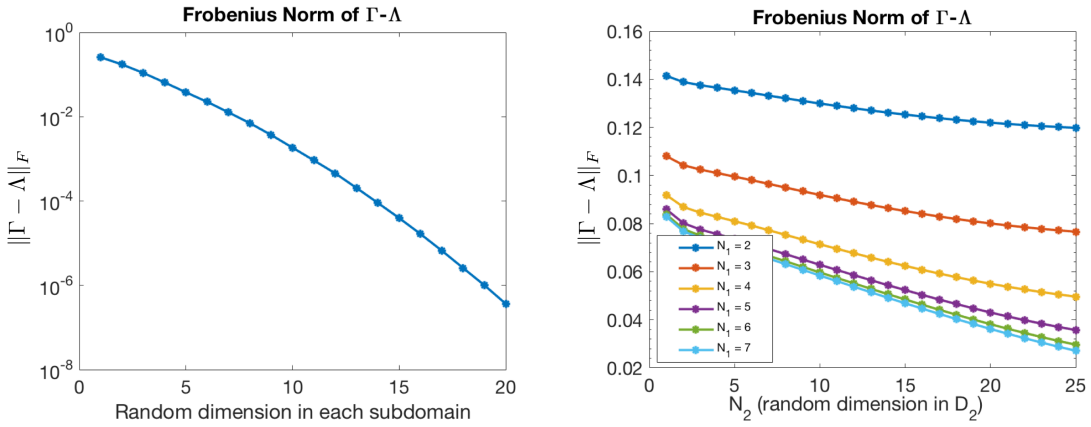
$$a(x;\omega) \approx \bar{a}(x) + \sum_{i=1}^{N} e_i(x)\sqrt{\lambda_k}c_i(\omega), \quad x \in \mathcal{D}. \tag{2.14}$$

We project $\tilde{a}(x;\omega)$ back to the global KL bases $\{e_i(x) | i = 1, 2, ...N\}$ in order to get the coefficients $\sqrt{\lambda_i}c_i$. Then we evaluate $\mathbb{E}[c_i]$ and $\Gamma_{N\times N}$, where

$$\Gamma_{i,j} = \mathrm{Cov}\left(\sqrt{\lambda_i}c_i, \sqrt{\lambda_j}c_j\right), \quad i, j \in \{1, 2, ..., N\}.$$

As the number of random variables in each subdomain increases, we expect $\mathbb{E}[c_i]$ to be 0, and Γ to converge to $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, ..., \lambda_N)$. Figure 2.3b indicates that the expectations of c_i is indeed 0. In Figure 2.4a, we plot the Frobenius norm of $\Gamma - \Lambda$, versus the number of random variables used in each subdomain. As we can see, when we approximate $a(x;\omega)$ in higher dimensional random spaces, $||\Gamma - \Lambda||_F$ decays exponentially, indicating spectral convergence of our approximation. Fig-

ure 2.4b shows that when we fix the random dimension of one subdomain, while increasing that of the other subdomain, we also get more accurate approximations for $a(x;\omega)$.



(a) **(b)**

Figure 2.4: $\|\Gamma-\Lambda\|_F$ versus subdomain random dimensions: (a) using a correlation length equal to 0.1 in both subdomains, and keeping $N_1 = N_2$, $\|\Gamma-\Lambda\|_F$ decays exponentially; (b) using a correlation length equal to 0.1 in $\mathcal{D}_1$, and 0.01 in $\mathcal{D}_2$, for a fixed $N_1 = 8$, increasing N_2 reduces $\|\Gamma-\Lambda\|_F$.

To summarize, the local randomness parametrization is conducted following these steps:

1. Obtain the local covariance function $K_{11}(x,y)$ and $K_{22}(x,y)$ from Eq. 2.6;
2. Perform local KL expansion for $K_{11}(x,y)$ and $K_{22}(x,y)$ to get $\boldsymbol{\xi_1}$ and $\boldsymbol{\xi_2}$;
3. Obtain $K_{12}(x,y)$ from Eq. 2.6 and calculate the cross-correlations between $\boldsymbol{\xi_1}$ and $\boldsymbol{\xi_2}$ from Eq. 2.12;
4. In each subdomain, sample $\boldsymbol{\xi_1}$ and $\boldsymbol{\xi_2}$ according to their covariance matrix.

2.4 Moment Minimizing Interface Condition

In this section we focus on building the subdomain interface conditions.

2.4.1 An Iterative Algorithm for Solving SPDEs

From the previous section, $a(x;\omega)$ can be parametrized locally as finite dimensional random vectors $\boldsymbol{\xi_1}$ and $\boldsymbol{\xi_2}$. Accordingly, the subdomain solutions are also functions of $\boldsymbol{\xi_1}$ and $\boldsymbol{\xi_2}$, and therefore can be written as

$$\begin{aligned} u_1(x,t;\omega) &\approx u_1(x,t;\boldsymbol{\xi_1},\boldsymbol{\xi_2}) = \sum_{i=0}^{+\infty} u_{1,i}(x,t)\Phi_i(\boldsymbol{\xi_1},\boldsymbol{\xi_2}), \quad x \in \mathcal{D}_1 \\ u_2(x,t;\omega) &\approx u_2(x,t;\boldsymbol{\xi_1},\boldsymbol{\xi_2}) = \sum_{j=0}^{+\infty} u_{2,j}(x,t)\Psi_j(\boldsymbol{\xi_1},\boldsymbol{\xi_2}), \quad x \in \mathcal{D}_2 \end{aligned} \tag{2.15}$$

where $u_{1,i}(x,t)$ and $u_{2,j}(x,t)$ are deterministic mode functions that only rely on x and t. Expansions given by Eq. 2.15 amount to a global stochastic and a local space representation of the random field $u(x,t;\omega)$. A widely used expansion is the generalized polynomial chaos (gPC) method [89]. Note that although $\boldsymbol{\xi_1}$ and $\boldsymbol{\xi_2}$ are not independent, they can be written as linear combinations of independent standard normal distributed random variables $\boldsymbol{\eta_1}$ and $\boldsymbol{\eta_2}$,

$$\begin{bmatrix} \boldsymbol{\xi_1} \\ \boldsymbol{\xi_2} \end{bmatrix} = A \begin{bmatrix} \boldsymbol{\eta_1} \\ \boldsymbol{\eta_2} \end{bmatrix},$$

where A results from the Cholesky decomposition of the covariance matrix of $\boldsymbol{\xi_1}$ and $\boldsymbol{\xi_2}$ and is invertible. Since we can always write $u_1(x,t;\omega)$ and $u_2(x,t;\omega)$ as gPC expansions of $\boldsymbol{\eta_1}$ and $\boldsymbol{\eta_2}$, by inverting A, they can also be written as polynomial expansions of $\boldsymbol{\xi_1}$ and $\boldsymbol{\xi_2}$. In practice, we truncate the expansion given by Eq. 2.15 and keep only the lower order terms. The local SPDE problem is then transformed to solving for $u_{1,i}(x,t)$ and $u_{2,j}(x,t)$. The evolution equation of the modes is obtained by performing standard stochastic Galerkin projection.

We are particularly interested in what conditions should be imposed at the

subdomain interfaces. Recall the Schwarz alternating method for deterministic equations. The global domain is decomposed into a *dominus* domain with a Dirichlet type interface condition, and a *servus* domain with a Neumann type interface condition [31, 5]. We propose a Schwarz type iterative algorithm for SPDEs: let $\mathcal{D}_1$ be the *dominus* domain with a Dirichlet type interface condition, and $\mathcal{D}_2$ be the *servus* domain with a Neumann type interface condition. Figure 2.5 shows an illustration for this domain decomposition.

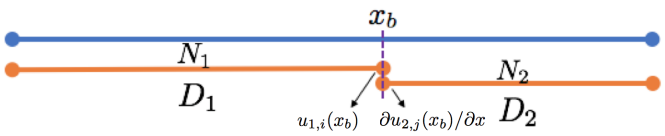


Figure 2.5: Domain decomposition approach for SDD-MM, where N_1 is the number of random variables in $\mathcal{D}_1$, and N_2 is the number of random variables in $\mathcal{D}_2$. The interface conditions are a Dirichlet boundary condition in $\mathcal{D}_1$, and a Neumann boundary condition in $\mathcal{D}_2$.

Consider the second-order statistical moment of the difference of local solutions at the subdomain interface x_b, that is

$$\mathcal{J}_1 = \mathbb{E}\left[\left(\sum_{i=0}^{M_1} u_{1,i}(x_b,t)\Phi_i(\boldsymbol{\xi_1},\boldsymbol{\xi_2}) - \sum_{j=0}^{M_2} u_{2,j}(x_b,t)\Psi_j(\boldsymbol{\xi_1},\boldsymbol{\xi_2})\right)^2\right], \tag{2.16}$$

where M_1 and M_2 are the numbers of expansion terms in Eq. 2.15. Ideally, $\mathcal{J}_1$ should be 0, due to the continuity of solution across subdomains. In practice, the interface conditions $u_{1,i}(x_b,t)$ is obtained by minimizing $\mathcal{J}_1$, with respect to $u_{1,i}(x_b,t)$.

Let $M(t;\boldsymbol{\xi_1},\boldsymbol{\xi_2}) = \sum_{j=0}^{M_2} u_{2,j}(x_b,t)\Psi_j(\boldsymbol{\xi_1},\boldsymbol{\xi_2})$. The first-order condition gives

$$
\begin{aligned}
0 =& \frac{\partial \mathcal{J}_1}{\partial u_{1,i}} \\
=& \frac{\partial \mathbb{E}[\sum_{j=0}^{M_1} u_{1,j}(x_b,t)\Phi_j(\boldsymbol{\xi_1},\boldsymbol{\xi_2}) - M(t;\boldsymbol{\xi_1},\boldsymbol{\xi_2})]^2}{\partial u_{1,i}} \\
=& \frac{\partial(\sum_{j=1}^{M_1} u_{1,j}^2 \mathbb{E}[\Phi_j^2] + \sum_{\substack{j,k=0 \\ j\neq k}}^{M_1} u_{1,j}u_{1,k}\mathbb{E}[\Phi_j\Phi_k] + \mathbb{E}[M^2] - 2\sum_{j=0}^{M_1} u_{1,j}\mathbb{E}[M\Phi_j])}{\partial u_{1,i}} \\
=& 2u_{1,i}\mathbb{E}[\Phi_i^2] + 2\sum_{\substack{j=0 \\ j\neq i}}^{M_1} u_{1,j}\mathbb{E}[\Phi_i\Phi_j] - 2\mathbb{E}[M\Phi_i] \\
=& 2u_{1,i}\mathbb{E}[\Phi_i^2] + 2\sum_{\substack{j=0 \\ j\neq i}}^{M_1} u_{1,j}\mathbb{E}[\Phi_i\Phi_j] - 2\sum_{j=0}^{M_2} u_{2,j}\mathbb{E}[\Phi_i\Psi_j].
\end{aligned}
\tag{2.17}
$$

Since $\mathcal{J}_1$ is a convex function, Eq. 2.17 is the minimizing condition, and therefore,

$$
u_{1,i}^{opt}(x_b,t) = \frac{\sum_{j=0}^{M_2} u_{2,j}(x_b,t)\mathbb{E}[\Phi_i\Psi_j] - \sum_{\substack{j=0 \\ j\neq i}}^{M_1} u_{1,j}(x_b,t)\mathbb{E}[\Phi_i\Phi_j]}{\mathbb{E}[\Phi_i^2]}. \tag{2.18}
$$

Note that in the above expression, $\sum_{j=0}^{M_2} u_{2,j}(x_b,t)\mathbb{E}[\Phi_i\Psi_j]/\mathbb{E}[\Phi_i^2]$ represents the projection of the stochastic boundary condition of $\mathcal{D}_2$ onto the stochastic bases of $\mathcal{D}_1$, and $\sum_{\substack{j=0 \\ j\neq i}}^{M_1} u_{1,j}(x_b,t)\mathbb{E}[\Phi_i\Phi_j]/\mathbb{E}[\Phi_i^2]$ subtracts the contribution of other $j \neq i$ stochastic bases from $\mathcal{D}_1$. An iterative scheme can be developed from Eq. 2.18. Let $u_{1,i}^k(x_b,t)$ be the interface condition of $u_{1,i}(x,t)$ in the k^{th} iteration. Similar to the deterministic Schwarz algorithm, we use a relaxed version of Dirichlet boundary conditions [16, 31, 57] for the *dominus* domain $\mathcal{D}_1$,

$$
u_{1,i}^{k+1}(x_b,t) = (1-\theta)u_{1,i}^k(x_b,t) + \theta u_{1,i}^{opt,k}(x_b,t), \quad \theta \in (0,1). \tag{2.19}
$$

For the *servus* domain $\mathcal{D}_2$, we consider the second moment of the difference of

flux across the interface x_b,

$$\mathcal{J}_2 = \mathbb{E}\left[\left(\sum_{i=0}^{M_1} u'_{1,i}(x_b,t)\Phi_i(\boldsymbol{\xi_1},\boldsymbol{\xi_2}) - \sum_{j=0}^{M_2} u'_{2,j}(x_b,t)\Psi_j(\boldsymbol{\xi_1},\boldsymbol{\xi_2})\right)^2\right]. \tag{2.20}$$

The interface conditions for $u_{2,j}(x,t)$ is obtained similiarly by minimizing $\mathcal{J}_2$ with respect to $u'_{2,j}(x_b,t)$, i.e., $\partial\mathcal{J}_2/\partial u'_{2,j}(x_b,t) = 0$, which results in

$$u'^{\,opt}_{2,j}(x_b,t) = \frac{\sum_{i=0}^{M_1} u'_{1,i}(x_b,t)\mathbb{E}[\Phi_i\Psi_j] - \sum_{\substack{i=0\\ i\neq j}}^{M_2} u'_{2,i}(x_b,t)\mathbb{E}[\Psi_i\Psi_j]}{\mathbb{E}[\Psi_j^2]}. \tag{2.21}$$

The Neumann boundary conditions for $u_{2,j}(x,t)$ will be

$$u'^{\,k+1}_{2,j}(x_b,t) = u'^{\,opt,k}_{2,j}(x_b,t). \tag{2.22}$$

The iterating process terminates when the change of boundary conditions between consecutive iterations is smaller than a prescribed tolerance ϵ_{tol}, which is taken as 10^{-7} in the numerical examples,

$$\sum_{i=1}^{M_1} |u^{k+1}_{1,i}(x_b) - u^k_{1,i}(x_b)| + \sum_{j=1}^{M_2} |u'^{\,k+1}_{2,j}(x_b) - u'^{\,k}_{2,j}(x_b)| \leq \epsilon_{tol}. \tag{2.23}$$

Algorithm 2: SDD-MM

Set $k = 0$;

Initialize $u^0_{1,i}(i = 1, 2, ..., M_1)$ and $u^0_{2,j}(j = 1, 2, ..., M_2)$ randomly;

while $\|u_1^{k+1}(x_b) - u_1^k(x_b)\| + \|{u'_2}^{k+1}(x_b) - {u'_2}^k(x_b)\| \leq \epsilon_{tol}$ **do**

from $u^k_{2,j}(x_b)$ calculate Dirichlet MMIC $u^{k+1}_{1,i}(x_b)$;

Solve for $u^{k+1}_{1,i}(x)$;

from ${u'_{1,i}}^{k+1}(x_b)$ calculate Neumann MMIC ${u'_{2,i}}^{k+1}(x_b)$;

Solve for $u^{k+1}_{2,i}(x)$;

$k = k + 1$;

end

Calculate subdomain random solutions from Eq. 2.15;

We name our method "stochastic domain decomposition via moment minimization" (SDD-MM), and the interface conditions are called "moment minimizing interface conditions" (MMIC). The workflow of the SDD-MM method is shown in Algorithm 2. We note that in expansions given by Eq. 2.15, both Φ_i and Ψ_i are taken to be generic stochastic bases, and as such the SDD-MM can be applied to a variety of methods. For example, the stochastic bases can be expanded to time-dependent bases as encountered in the Dynamically-Orthogonal decomposition [69, 11].

2.4.2 Computational Cost Analysis

We only consider the computational cost of repetitively solving local deterministic problems when a sparse grid stochastic collocation method is used. Let k be the Smolyak sparse grid level. The total number of sparse grid points can be estimated

by $2^k N^k/k!$ [22, 88], where N is the random space dimension. Let the computational cost for solving one single local deterministic PDE be one unit, and suppose we use N_1 random variables in $\mathcal{D}_1$, and N_2 random variables in $\mathcal{D}_2$. The computational cost is

$$\begin{aligned} C_{\text{SDD-MM}} &\sim n_{\text{it}} \times \left(\frac{2^k (N_1 + N_2)^k}{k!} + \frac{2^k (N_1 + N_2)^k}{k!} \right) \\ &\sim n_{\text{it}} \times \frac{2^{k+1} (N_1 + N_2)^k}{k!}, \end{aligned} \tag{2.24}$$

where n_{it} is the number of iterations needed for convergence.

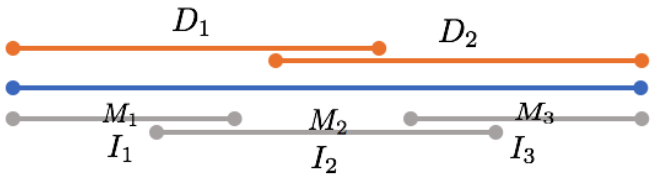


Figure 2.6: Domain decomposition approach for SDD-S, where M_1, M_2 and M_3 are the numbers of random variables in I_1, I_2 and I_3.

Our method is inspired and motivated by the precedent conditional moment interface method, developed by H. Cho et al. [9]. We compare Eq. 2.24 with the computational cost of the conditional moment interface method, and in particular, with the computational cost of the variance scaling boundary condition (SDD-S) (readers may refer to [9] for the details of SDD-S). Figure 2.6 is a schematic diagram of the domain decomposition for SDD-S. M_1, M_2, and M_3 are random dimensions for the interfacing domains I_1, I_2, and I_3. The computational cost for SDD-S is

$$\begin{aligned} C_{\text{SDD-S}} &\sim n'_{\text{it}} \times \frac{2^k M_2^k}{k!} \left(\frac{2^k M_1^k}{k!} + \frac{2^k M_3^k}{k!} \right) \\ &\sim n'_{\text{it}} \times \frac{2^{2k} ((M_1 M_2)^k + (M_3 M_2)^k)}{(k!)^2}, \end{aligned} \tag{2.25}$$

where n'_{it} is the number of iterations needed.

We can compare the computational cost of the two methods,

$$\begin{aligned}\frac{C_{\text{SDD-MM}}}{C_{\text{SDD-S}}} &\sim \frac{n_{\text{it}}}{n'_{\text{it}}} \times \frac{2k!(N_1+N_2)^k}{2^k((M_1M_2)^k+(M_3M_2)^k)} \\ &\sim \frac{n_{\text{it}}}{n'_{\text{it}}} \times \frac{2k!(2N_1)^k}{2^kM_2^k((M_1)^k+(M_3)^k)} \\ &\sim \frac{n_{\text{it}}k!}{n'_{\text{it}}(M_2M_1/N_1)^k} < 1.\end{aligned} \tag{2.26}$$

Generally speaking, $M_1 \approx M_3$, $M_1 \approx N_1$, $N_1 \approx N_2$, and k is smaller than 5. However, M_2 can be very large, especially for SPDEs with variable correlation length. This is because I_2 covers the overlapping part of $\mathcal{D}_1$ and $\mathcal{D}_2$, where change of correlation length happens. Our numerical tests show that n_{it} is usually less than 5, but n'_{it} varies depending on the initial guess. Therefore, for most cases, the SDD-MM method is computationally cheaper than the SDD-S method.

2.5 Numerical Results

We implement the SDD-MM method for solving a variety of SPDEs in both 1D and 2D domains, and SPDEs with strong nonlinearity. For demonstration purposes, all results are calculated using the gPC expansion.

2.5.1 Application to Stochastic Poisson's Equation

We start with a 1D stochastic Poisson's equation:

$$-\frac{\mathrm{d}}{\mathrm{d}x}\left(a(x;\omega)\frac{\mathrm{d}u(x;\omega)}{\mathrm{d}x}\right) = \sin(2\pi x), \quad x \in [0,1], \tag{2.27}$$

with homogeneous Dirichlet boundary conditions:

$$u(0;\omega) = u(1;\omega) = 0.$$

The randomness comes from the random coefficient $a(x;\omega)$, which is characterized as a Gaussian random process with Gaussian covariance function

$$\mathrm{Cov}\left(a(x;\omega), a(y;\omega)\right) = \sigma_a^2 \exp\left(-\frac{|x-y|^2}{l_c^2}\right), \tag{2.28}$$

where σ_a is the standard deviation, and l_c stands for the correlation length.

We use a spectral/*hp* element method [31] for spatial discretization. The global domain $\mathcal{D}$ is divided into 10 elements of equal lengths, and in each element, we approximate the solution with a 10^{th} order polynomial. We shall use the same spatial discretization in the following numerical examples, if not specifically mentioned.

First, we verify the convergence of the SDD-MM method. Set $\mathbb{E}[a(x;\omega)]$ to be 2, and σ_a to be 0.2. The global domain is decomposed in the middle into subdomains $\mathcal{D}_1$ and $\mathcal{D}_2$, each of which contains 5 elements. We generate random samples of $a_i(x;\omega)$ locally as discribed in Section 2.3, and use the stochastic collocation method for the local SPDE problems in each subdomain. The reference solution is obtained by performing the global KL expansion of $a(x;\omega)$, truncating to keep 99.99% energy (same for the other examples). A global SPDE solver is employed to get the reference solution. Mean and standard deviation of the solution $u(x;\omega)$ are calculated.

Two numerical tests are conducted. For the first test, we set the correlation length l_c as 0.3, fix the order of the gPC expansions and use an increasing number

of random variables to approximate $a_i(x;\omega)$ in both subdomains. In Figure 2.7a, spectral convergence is observed as we increase the number of random dimensions. This agrees with the result in Figure 2.4a. For the second test, we set the correlation length l_c to be 10, and fix the random dimensions in both subdomains. We look only at the effect of the gPC expansion order. Figure 2.7b shows that when higher order gPC expansions are adopted, the error decays exponentially.

These two numerical tests indicate that the error of the SDD-MM method comes from two sources:

1. the local finite order approximation of $a(x;\omega)$, which can be reduced by keeping more random variables, i.e. increasing the random dimensions in each subdomain;

2. the gPC approximation error, which can be reduced by increasing the order of the gPC bases to involve more modes.

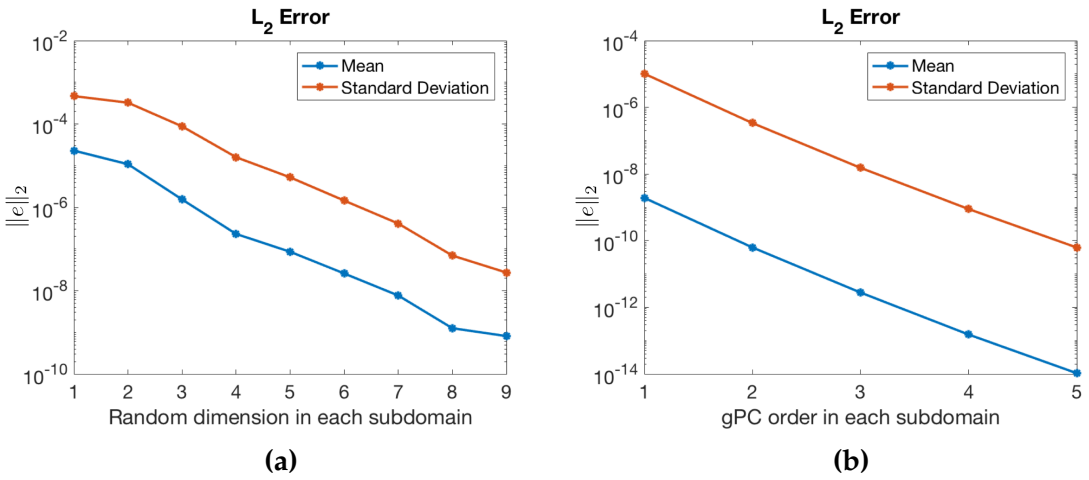


Figure 2.7: The exponential decay of the L_2 error of mean and standard deviation: (a) Fix the gPC expansion order to be 3, and in each subdomain approximate $a(x;\omega)$ with an increasing random dimensions ($N_1 = N_2$), in this case $l_c = 0.3$; (b) Fix the random dimensions in both subdomains to be 3, and increase the order of the gPC expansion in each subdomain, in this case $l_c = 10$.

Next, we compare the accuracy and the computational cost of the SDD-MM

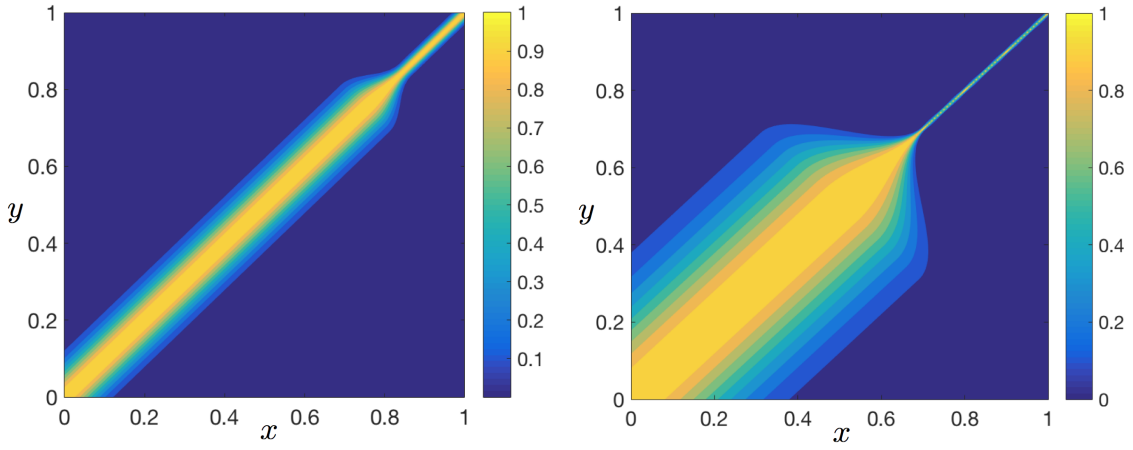


Figure 2.8: Covariance function in the $[0,1]$ domain with varying l_c: (a) $l_c = 0.08$ in $[0, 0.8]$, and $l_c = 0.02$ in $(0.8, 1]$; (b) $l_c = 0.25$ in $[0, 0.6]$, and $l_c = 0.005$ in $(0.6, 1]$.

method against the SDD-S method. We adopt the same settings as in [9]:

$$\mathbb{E}\left[a(x;\omega)\right] = 1, \quad \sigma_a = 0.2, \quad l_c = \begin{cases} 0.08, & \text{in } \mathcal{D}_1 \\ 0.02, & \text{in } \mathcal{D}_2 \end{cases}, \tag{2.29}$$

where $\mathcal{D}_1 = [0, 0.8]$, $\mathcal{D}_2 = (0.8, 1]$, and the covariance function of different correlation lengths is depicted in Figure 2.8a. The global domain is discretized by 10 spectral elements of order 20, and the local KL expansions are truncated to keep 95% energy. We employ Smolyak sparse grids of both level 1 and level 2. Table 2.1 compares the error and computing time of using different method and different level of sparse grid. It is evident that the SDD-MM method achieves better accuracy with the higher level sparse grid. This is because the higher level sparse grid allows us to implement the higher order of gPC expansion. Figure 2.9a shows the mean of the solution generated by both methods with the level 2 sparse grid. Both methods yield accurate mean solutions. However, as we can see in Figure 2.9b, the SDD-MM method outperforms the SDD-S method when computing the standard deviation.

Sparse Grid Level	Method	$\mathbb{E}[u]$ error	$\sigma(u)$ error	Time (s)
level 1	SDD-MM	$9.942e^{-5}, 0.53\%$	$2.670e^{-4}, 9.00\%$	0.34
	SDD-S	$9.254e^{-5}, 0.50\%$	$6.121e^{-4}, 20.63\%$	2.73
level 2	SDD-MM	$1.776e^{-6}, 0.01\%$	$2.524e^{-5}, 0.85\%$	7.24
	SDD-S	$7.239e^{-5}, 0.39\%$	$4.293e^{-4}, 14.31\%$	243.58

Table 2.1: Stochastic Poisson's equation: L_2 errors and relative L_2 errors for the solution mean and standard deviation, and the computing time.

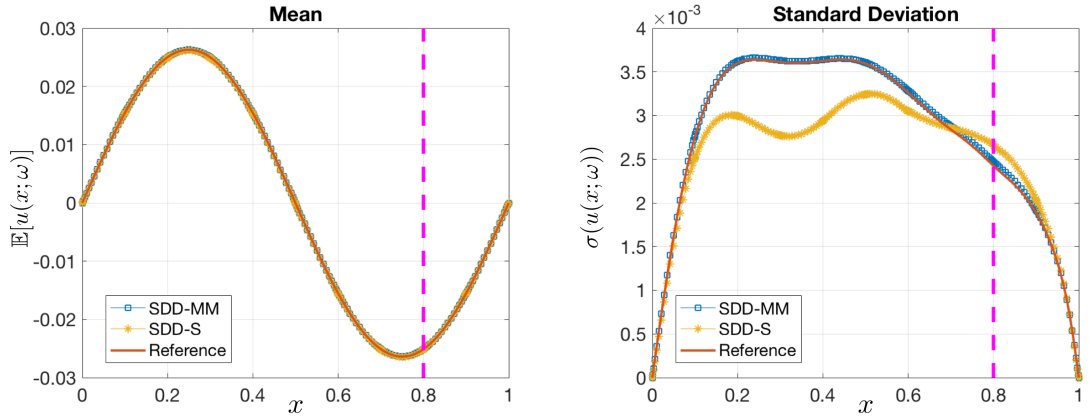


Figure 2.9: Stochastic Poisson's equation: a comparison of results obtained by SDD-MM and SDD-S for (a) the mean solution and (b) the standard deviation of solution. The pink dash line marks the subdomain interface.

We take a further look at the higher moments of the SPDE solution because they are sensitive and hard to capture. We use the level 3 sparse grid and compute the third central moment ($\mathbb{E}[(u(x;\omega) - \bar{u}(x))^3]$) and the fourth central moment ($\mathbb{E}[(u(x;\omega) - \bar{u}(x))^4]$) of the solution. In Figure 2.10a and Figure 2.10b, we compare the results for the second-order and the third-order gPC expansion. As we can see, higher order gPC expansions produces more accurate solutions, and the higher moments of solution are captured by the SDD-MM method within a 7% accuracy. We expect to improve the accuracy with gPC expansions of order four and above, but this would again require even higher levels of sparse grid.

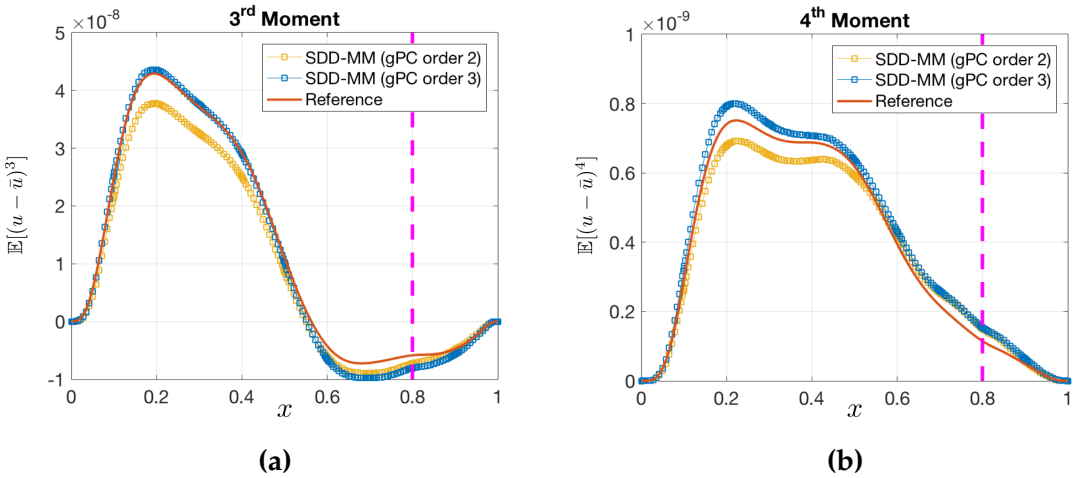


Figure 2.10: Stochastic Poisson's equation: a comparison of results obtained by SDD-MM for using second-order gPC expansion and third-order gPC expansion for (a) the third central moment and (b) the fourth central moment. The third-order gPC expansion generates more accurate solution than the second-order gPC expansion.

2.5.2 Application to Stochastic Advection Equation

As a second example, we consider the stochastic advection equation, given by:

$$\frac{\partial u(x,t;\omega)}{\partial t} = -a(x;\omega)\frac{\partial u(x,t;\omega)}{\partial x}, \quad x \in [0,1], \quad t \geq 0. \tag{2.30}$$

The advection coefficient $a(x;\omega)$ is a random process with the same variable correlation length l_c as in Eq. 2.29 and Figure 2.8a. The mean and standard deviation of $a(x;\omega)$ are

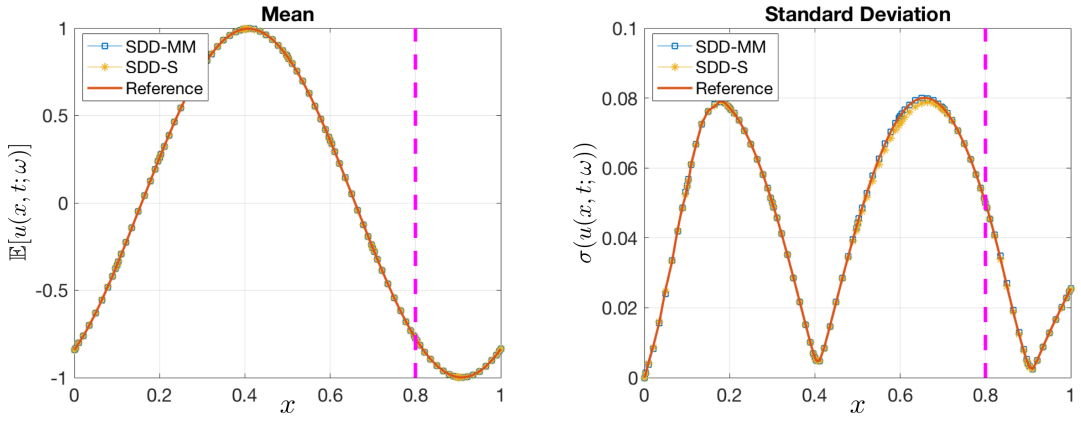
$$\mathbb{E}[a(x;\omega)] = \frac{1}{2\pi}, \quad \sigma_a = \frac{1}{20\pi}.$$

We set a Dirichlet boundary condition $u(0,t;\omega) = \sin(-t)$ for the left boundary. The initial condition is set to be $u(x,0;\omega) = \sin(2\pi x)$.

In this and the following numerical examples, level 2 sparse grid with the second-order gPC expansion are used, and the local KL expansions are truncated to keep 95% energy. We calculate the solution at $T = 1$ with time step $\Delta t = 10^{-4}$.

Method	$\mathbb{E}[u]$ error	$\sigma(u)$ error	Time (s)
SDD-MM	$5.042e^{-4}, 0.07\%$	$3.187e^{-4}, 0.60\%$	79.4
SDD-S	$2.315e^{-4}, 0.03\%$	$7.663e^{-4}, 1.45\%$	430.8

Table 2.2: Stochastic advection equation: L_2 errors and relative L_2 errors for the solution mean and standard deviation, and the computing time.



(a) **(b)**

Figure 2.11: Stochastic advection equation: a comparison of results obtained by SDD-MM and SDD-S for (a) the mean solution and (b) the standard deviation of solution.

For the advection equation, information flows in a single direction from the left domain $\mathcal{D}_1$ to the right domain $\mathcal{D}_2$, and therefore, there is no need to do the iteration process. One only has to decide the interface condition for $\mathcal{D}_2$ using the Dirichlet type moment minimizing interface condition Eq. 2.18. Figure 2.11a, Figure 2.11b and Table 2.2 indicate that both SDD-MM and SDD-S methods work well for this problem. Table 2.2 also shows the SDD-MM method takes less time for computing.

2.5.3 Application to Stochastic Advection-Reaction Equation

The SDD-MM method does not rely on the linearity of the problem, and we want to demonstrate the performance of our method for the time-dependent non-linear

stochastic advection-reaction equation with a stochastic reaction rate:

$$\frac{\partial u}{\partial t} = V(x)\frac{\partial u}{\partial x} + (k_0(x) + \sigma_k k_1(x;\omega))R(u), \quad x \in [0,1], \quad t \geq 0, \tag{2.31}$$

where

$$\begin{aligned} V(x) &= -\frac{1}{2}\left(1 + e^{(\sin(2\pi x)+\cos(2\pi x))/2} - \cos(2\pi x)\right), \\ k_0(x) &= 1 - \frac{2}{5}(e^{-\sin(2\pi x)/2} + \cos(2\pi x)), \\ \sigma_k &= 0.2, \quad R(u) = 1 - u^2. \end{aligned} \tag{2.32}$$

The initial condition is $u(x,0;\omega) = \sin(2\pi x)$, and we impose a periodic boundary condition. This problem was studied in [82] by using the Mori-Zwanzig projection operator method [78, 83], and then studied by using the conditional moment interface method in [9]. The perturbation in the reaction rate, $k_1(x;\omega)$, is modeled as a centered Gaussian random field with squared exponential covariance function and correlation length varying from 0.08 to 0.02 (refer to Eq. 2.29 and Figure 2.8a). The periodic boundary condition requires imposing the left boundary for $u_1(x,t;\omega)$ in $\mathcal{D}_1$, using the random field $u_2(x,t;\omega)$ in $\mathcal{D}_2$ at the right boundary. This is again handled with MMIC in Eq. 2.18.

Method	$\mathbb{E}[u]$ **error**	$\sigma(u)$ **error**	**Time (s)**
SDD-MM	$2.09e^{-6}, 0.0003\%$	$4.56e^{-5}, 0.27\%$	60.89
SDD-S	$4.66e^{-4}, 0.067\%$	$2.10e^{-3}, 12.87\%$	1492.17

Table 2.3: Stochastic advection-reaction equation: L_2 errors and relative L_2 errors for the solution mean and standard deviation, and the computing time.

We calculate the mean and standard deviation of $u(x,t;\omega)$ at $T = 0.2$, with time step $\Delta t = 10^{-4}$. In Figure 2.12a and Figure 2.12b, we compare the result of our method and the SDD-S method, and we list the L_2 error and relative L_2 error in Table 2.3. Clearly, the SDD-MM method is more accurate and efficient than the SDD-S method.

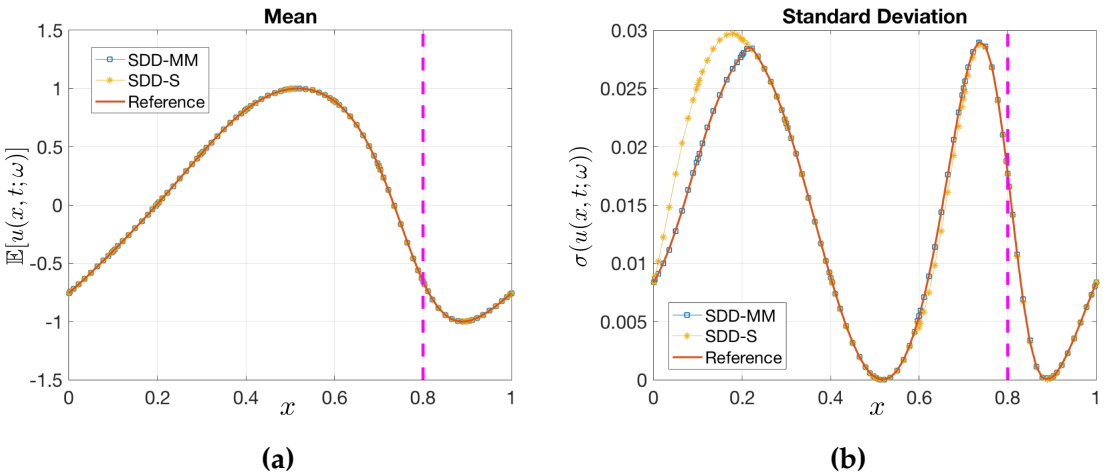


Figure 2.12: Stochastic advection-reaction equation: a comparison of results obtained by SDD-MM and SDD-S for (a) the mean solution and (b) the standard deviation of solution.

2.5.4 Application to Stochastic Fisher's Equation

In the previous cases, random processes with different correlation lengths have been considered, and in this example, we examine large variation in the correlation length by solving the Fisher's equation with stochastic nonlinear coefficient:

$$\frac{\partial u}{\partial t} = c(x)\frac{\partial^2 u}{\partial x^2} + a(x;\omega)u(1-u), \quad x \in [0,1], \quad t \geq 0. \tag{2.33}$$

The boundary conditions are $u(0) = u(1) = 0$, and the initial condition is $u(x,0;\omega) = \sin(\pi x)$. Set $c(x) = 0.1$, so the diffusivity is relatively small compared to the nonlinear effect. We set $a(x;\omega)$ to be a Gaussian random process with a large change of correlation length. The mean, standard deviation and correlation length of $a(x;\omega)$ are

$$\mathbb{E}\left[a(x;\omega)\right] = 1, \quad \sigma_a = 0.2, \quad l_c = \begin{cases} 0.25, & \text{in } \mathcal{D}_1 \\ 0.005, & \text{in } \mathcal{D}_2 \end{cases}, \tag{2.34}$$

where $\mathcal{D}_1 = [0, 0.6]$ and $\mathcal{D}_2 = (0.6, 1]$. Figure 2.8b displays the covariance function of this case. In applications, this happens when we have a multi-scale problem, where the uncertainty propagates across domains, which results in a multi-scale correlated structure. Numerical experiments show that the SDD-MM method performs well for this multi-scale problem.

Method	$\mathbb{E}[u]$ **error**	$\sigma(u)$ **error**	**Time (s)**
SDD-MM	$4.234e^{-6}, 0.0008\%$	$6.608e^{-5}, 0.82\%$	124.8
SDD-S	$1.964e^{-5}, 0.004\%$	$1.292e^{-3}, 15.96\%$	11921.28

Table 2.4: Stochastic Fisher's equation: L_2 errors and relative L_2 errors for the mean and standard deviation of the solution, and the computing time.

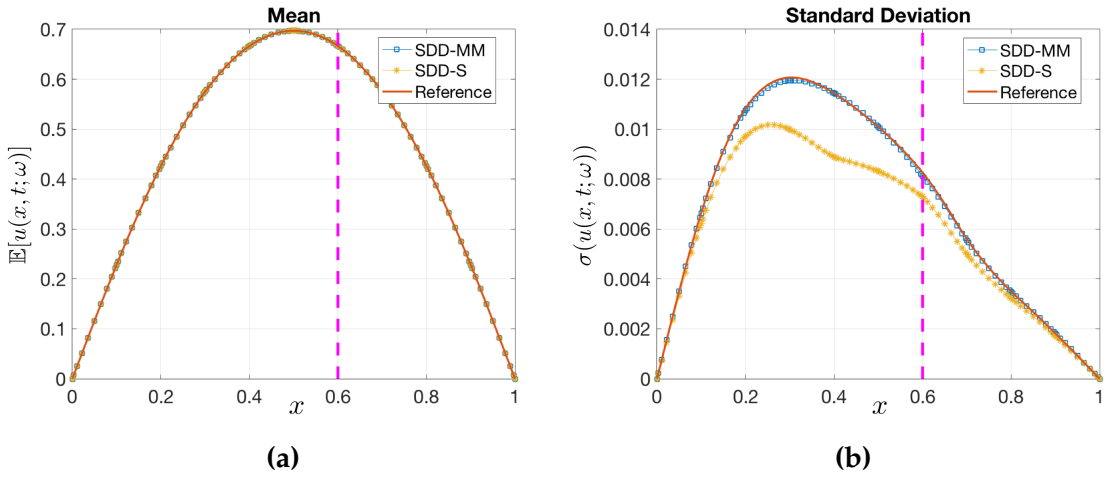


Figure 2.13: Stochastic Fisher's equation: a comparison of results obtained by SDD-MM and SDD-S for (a) the mean solution and (b) the standard deviation of solution.

We calculate the SPDE solution at $T = 0.5$, and use a time step $\Delta t = 0.001$. The correlation length l_c has an obvious impact on the standard deviation of the solution, breaking the symmetry and suppressing the standard deviation in the region of smaller correlation length (see Figure 2.13b). Both SDD-MM method and SDD-S method capture the solution mean, but the SDD-S method is not very accurate when calculating the standard deviation. Table 2.4 compares the accuracy and computing time of both methods. The SDD-MM method demonstrates better performance for both moments. Since there is a huge jump in correlation length,

M_2 in Eq. 2.26 needs to be very big to capture the random process in I_2, thus the computational cost for SDD-S method is much larger than that of the SDD-MM method.

2.5.5 Application to 2D Stochastic Allen-Cahn Equation

In the final example, we solve the 2D stochastic Allen-Cahn equation:

$$\frac{\partial u(x,y,t;\omega)}{\partial t} = \gamma\left(\Delta u - \frac{u(u^2-1)}{\epsilon_0^2}\right), \quad (x,y) \in [0,1]\times[0,1]. \tag{2.35}$$

This equation is defined in a square $\mathcal{D} = [0,1]\times[0,1]$, with homogeneous Neumann boundary conditions on all edges. The parameter γ is a scaling coefficient which is taken as 0.1, and ϵ_0 controls the thickness of the phase transition layer at steady state. We choose $\epsilon_0 = 0.05$, which will result in a very sharp phase transition layer. The randomness in this case comes from the random initial condition $u_0(x,y;\omega)$, modeled as a 2D Gaussian random field with covariance function:

$$\mathrm{Cov}\left(u_0(x_1,y_1;\omega), u_0(x_2,y_2;\omega)\right) = \sigma^2 \exp\left(-\frac{|x_1-x_2|^2}{l_{c,x}^2} - \frac{|y_1-y_2|^2}{l_{c,y}^2}\right), \tag{2.36}$$

where

$$\sigma = 0.02, \quad l_{c,x} = \begin{cases} 0.5, & x \in [0,0.5] \\ 0.05, & x \in (0.5,1] \end{cases}, \quad l_{c,y} = 0.5.$$

The initial condition mean is set to be $\mathbb{E}[u_0(x,y;\omega)] = 4\sin(2\pi x)\sin(\pi y)/5$.

Figure 2.14 shows the standard deviation of the truncated KL approximation of the initial condition. It is obtained by performing the global KL expansion of $u_0(x,y;\omega)$, and truncating at 95% energy level. The difference in correlation lengths

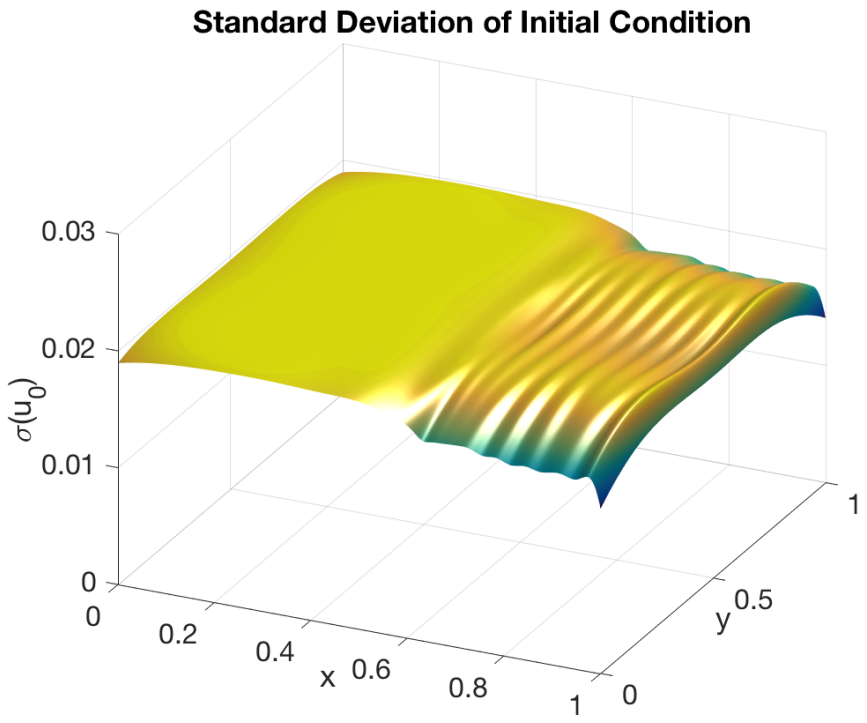


Figure 2.14: Standard deviation of the initial condition for the Allen-Cahn equation. Generated by performing the global KL expansion and truncating at 95% energy level.

can be clearly observed in this plot, where the left half domain is a smooth plane, but the right half domain is more wavy. Naturally, we divide the whole domain into two subdomains based on the different correlation lengths in the x direction: $\mathcal{D}_1 = [0, 0.5] \times [0, 1]$ and $\mathcal{D}_2 = (0.5, 1] \times [0, 1]$.

Error	$E[u]$	$\sigma(u)$
L_2 error	$2.948e^{-4}$	$2.767e^{-3}$
Relative L_2 error	0.03%	5.23%

Table 2.5: 2D Allen-Cahn equation: L_2 errors and relative L_2 errors for the mean and standard deviation of the solution.

We solve each subdomain SPDE using a direction splitting scheme [73], and the local solution is approximated by 32^{th} order polynomials in both x and y directions. The reference solution is obtained by using the global KL expansion to model $u_0(x, y; \omega)$, and solving the SPDE globally with the probabilistic collocation

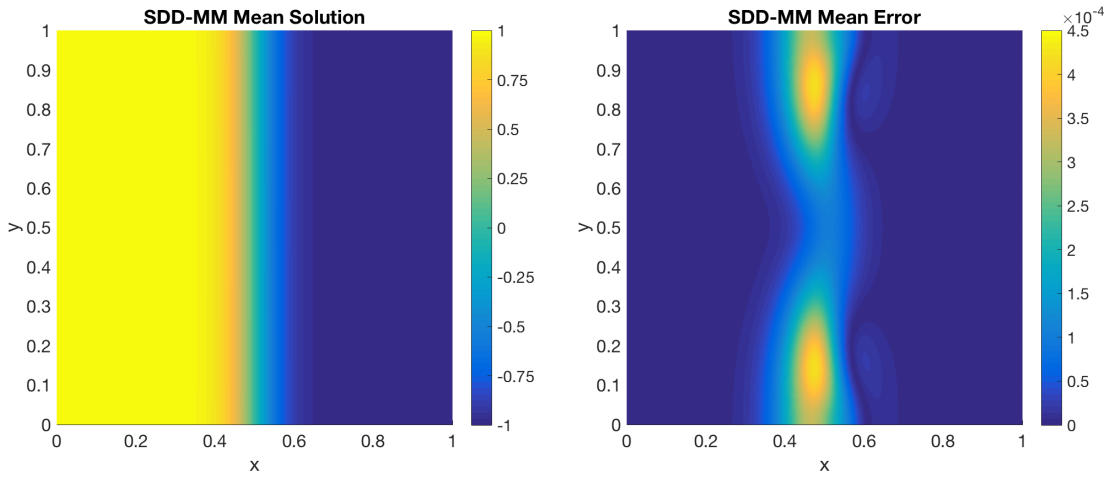


Figure 2.15: 2D Allen-Cahn equation: $\mathbb{E}[u(x, y, t; \omega)]$ obtained by SDD-MM and the error.

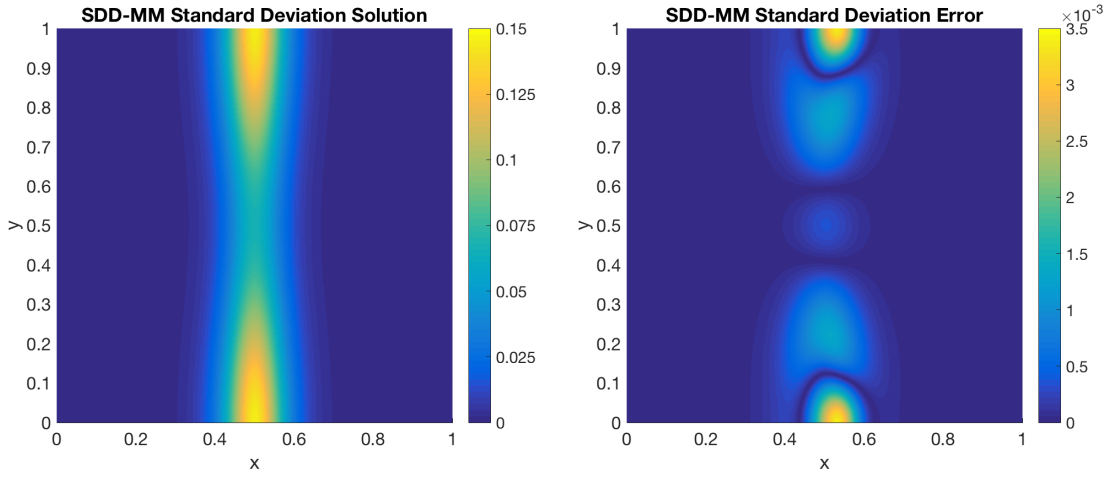


Figure 2.16: 2D Allen-Cahn equation: $\sigma(u(x, y, t; \omega))$ obtained by SDD-MM and the error.

method. The global reference solution is approximated by a 128^{th} order polynomial in x direction and a 32^{th} order polynomial in y direction. In Figure 2.15 and Figure 2.16, we show the mean and standard deviation of the domain decomposition solution, together with their error, when they reach a steady state. We can see in Figure 2.15 that the subdomain interface ($x = 0.5$) is right inside the phase transition layer, where the gradient of the solution is very large. Table 2.5 shows that our domain decomposition method is still accurate and reliable for this 2D problem with strong nonlinearity and sharp gradient.

2.6 Summary of Chapter

We presented the domain decomposition based on the moment minimization (SDD-MM) technique for solving stochastic partial differential equations. A new moment minimizing interface condition is proposed to match the solution modes at the interface, so the subdomain solutions can be glued together through a Schwarz type iterative procedure. Using the SDD-MM, we derive boundary conditions for subdomains with different random dimensions. This SDD-MM method is a generalized framework that relies on local random solvers and can be used to solve problems involving multi-scale phenomena and hybrid stochastic systems, without the need of global sampling.

Our numerical results show that the error in solution is due to two different sources: the local reduced order randomness parametrization, and the solution approximation using truncated gPC expansions. Errors from the first source can be decreased by choosing higher local random dimensions, while errors from the second source can be reduced by including more gPC expansion terms. For both types of error, exponential convergence is observed. Compared with the conditional moment interface method, the SDD-MM method displays higher accuracy, especially for capturing the standard deviation of solution, and better computation efficiency. The 2D numerical test shows the reliability of our method, even if the domain is decomposed exactly at the interface where the solution has a sharp gradient.

Chapter Three

Bi-directional Coupling of PDE- and Data-Domain

3.1 Introduction

There has been substantial progress in the last ten years on data inference models, but in many practical situations the data can only be collected in a restricted area from a number of scattered and heterogeneous sensors of variable fidelity. In subsurface applications, for example, we may have such measurements in one region, where we may or may not know the physics of the process, and this region may be adjacent to a more homogeneous region where we know the physical process and hence the mathematical model in the form of a PDE. We assume here that we have bi-directional coupling, such as the dispersion in porous media, between the region with data and the region where the PDE is defined. Currently, there is no scientific method to model this bi-directional propagation of information and the existing data inference techniques, e.g., for streaming data, are relying on very accurate high-fidelity input. Here we address, for the first time, such hybrid data-PDE problems and develop a method that propagates the information between the data and PDE regions. This new concept is schematically shown in Figure 3.1, where we allow a bi-directional coupling between the region with sensor data (right, yellow) and the region with governing PDEs (left, blue). In addition, we assume that we have available diverse sensors depicted by green and red to represent different fidelities of the data. In particular, we have only a few green data points that we fully trust whereas the majority of the data is represented by red crosses to denote less accurate or potentially even misleading information. This algorithm would naturally fall into the scope of domain decomposition methods, which have already been studied extensively within a deterministic problem setup [5, 17, 42, 72, 26, 31], based on the classical Schwarz alternating algorithm [71]. This domain coupling is also a natural extension of the previous work on propagation of stochastic solutions across domains [6, 41, 9].

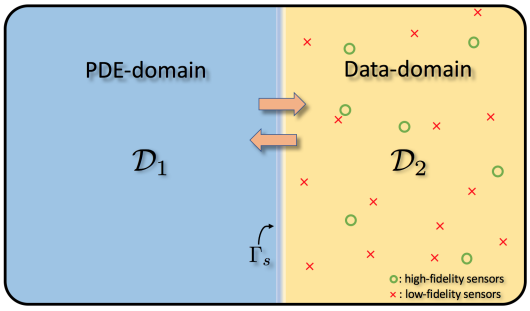


Figure 3.1: The physical domain $\mathcal{D}$ is decomposed into non-overlapping subdomains $\mathcal{D}_1$ and $\mathcal{D}_2$, where $\mathcal{D}_1$ is the PDE-domain and $\mathcal{D}_2$ is the Data-domain equipped with high-fidelity sensors (green circles) and low-fidelity sensors (red crosses). Information from the PDE-domain and the Data-domain propagates in both directions across the interface Γ_s.

Various methods [24, 70, 4] for solving a PDE based on machine learning tools have been developed recently, among which Gaussian process regression (GPR) [65] serves as an effective means for designing such data-driven algorithms. Raissi et al. [60] have proposed a GPR based algorithm for inferring solutions of differential equations from noisy multi-fidelity data, where the cheap, frequent but low-fidelity observations could help with the expensive, scarce and high-fidelity observations to obtain solutions with better accuracy. Inspired by such previous work in GPR and domain decomposition methods, we develop a new algorithm that reconstructs the solution from scattered sensors in the data subdomain coupled with an adjacent PDE subdomain, and we arrive at a converged global solution using an iterative method, similar to the Schwarz alternating method [71].

The organization of this chapter is as follows. In Section 3.2, we set up the domain decomposition problem and specify the two types of subdomains. In Section 3.3, we introduce the numerical GPR method for solving PDEs, using data of single- and multi-fidelity; the core algorithm of this chapter is also described in this section. In Section 3.4, we demonstrate the performance of the proposed algorithm in both 1D and 2D examples, and we conclude with a brief summary in Section 3.5.

3.2 Problem Setup

Suppose $\mathcal{D}$ is the entire physical domain and is divided into two non-overlapping subdomains $\mathcal{D}_1$ and $\mathcal{D}_2$ as depicted in Figure 3.1, where Γ_s represents the interface shared by those two subdomains. We refer to $\mathcal{D}_1$ as the *PDE-domain*, because the quantity of interest (QoI) $u(x)$ is governed by a known PDE in $\mathcal{D}_1$,

$$\begin{cases} \mathcal{L}_x u(x) = f(x), & x \in \mathcal{D}_1, \\ \mathcal{B}_x u(x) = g(x), & x \in \partial\mathcal{D}_1 \backslash \Gamma_s, \end{cases} \tag{3.1}$$

where $\mathcal{L}_x$ is the partial differential operator, $f(x)$ is the forcing term, $\mathcal{B}_x$ is a proper boundary condition operator acting on all boundaries of $\mathcal{D}_1$, except for Γ_s, and $g(x)$ is the prescribed boundary condition. We refer to $\mathcal{D}_2$ as the *Data-domain*, where we place sensors of the solution $u(x)$ at sparse locations, and collect data from the sensors. In another scenario, we may have some sparse measurements of the forcing (right-hand-side) term $f(x)$ of a known PDE, and hence we will subsequently solve the PDE using numerical GPR [60] instead of the classical discretization method as in domain $\mathcal{D}_1$. Moreover, these sensors provide information of variable fidelity.

Since a physically admissible solution $u(x)$ should satisfy some continuity conditions at the interface Γ_s, such as the continuity of $u(x)$ and $\nabla u(x)$, the value of $u(x)$ on Γ_s, which may be undetermined at first, depends on $u(x)$ in both $\mathcal{D}_1$ and $\mathcal{D}_2$. Our goal is to design a domain decomposition algorithm that imposes the aforementioned continuity constraints and will make use of both the PDE in $\mathcal{D}_1$ and the sparse multi-fidelity sensors data in $\mathcal{D}_2$ to solve for $u(x)$ in the entire domain $\mathcal{D}$.

3.3 Methodology

3.3.1 Numerical Gaussian Process Regression

The general form of a linear PDE is,

$$\begin{cases} \mathcal{L}_x u(x) = f(x), & x \in \Omega, \\ u(x) = p(x), & x \in \Gamma_D, \\ \frac{\partial}{\partial \boldsymbol{n}} u(x) = q(x), & x \in \Gamma_N, \end{cases} \tag{3.2}$$

where $\mathcal{L}_{(\cdot)}$ is a linear operator (here $\mathcal{L}_x$ means that the operator acts on the variable x), $f(x)$ is the external forcing term, $p(x)$ and $q(x)$ are the Dirichlet and Neumann boundary conditions on their respective boundaries Γ_D and Γ_N, and $\boldsymbol{n}$ is the unit outer normal. Assuming that the PDE solution $u(x)$ belongs to $C^1(\Omega)$, the solution space can be approximated using Gaussian random processes with a covariance kernel characterized by tunable hyper-parameters θ, while the actual solution is one of the possible Gaussian process trajectories. We shall infer the actual solution using a machine learning strategy, given the sensors' data of $f(x)$ and $u(x)$ at sparse locations and possibly other boundary information. The hyper-parameters θ in the covariance kernel can be estimated using a maximum likelihood estimation, and after that the solution $u(x)$ for any provided x shall be predicted via the Bayesian estimation. In practice, the sensors' data might be polluted by random measurement error, which could be modeled by independent zero-mean Gaussian random variables with standard deviation σ_u and σ_f that are either known constants, or could be learned together with θ from the data.

To be more specific, let $\boldsymbol{x} = (x_1, x_2, ..., x_{N_0})$ be the locations where we place our

$u(x)$ sensors. Assuming a Gaussian process prior of the solution $u(x)$, i.e.,

$$u(x) \sim \mathcal{GP}(0, g(x, x'; \theta)), \tag{3.3}$$

where $g(x, x'; \theta)$ is the covariance kernel that takes the form of, for example, the squared exponential kernel

$$g(x, x'; \theta) = \sigma_l^2 \exp\left(-\frac{1}{2}\sum_{d=1}^{D}\frac{(x^{(d)} - x'^{(d)})^2}{l_d^2}\right). \tag{3.4}$$

Non-stationary kernels, such as the neural network covariance function [65, 64, 52], can also be applied, and the main algorithm to be proposed does not restrict itself to stationary kernels either. For demonstrative purpose we will use the squared exponential kernel described here for our numerical tests. In Eq. 3.4), $x^{(d)}$ is the dth dimensional coordinate of x and D is the dimension of the Data-domain, and the hyper-parameters to be optimized are $\theta = (\sigma_l, l_1, l_2, ..., l_D)$. The Dirichlet boundary conditions can be viewed as noiseless sensors (sensors that return the exact value) on its respective boundary and they will be handled in the same way as the other $u(x)$ sensors inside the domain. Thus we only need to consider the situation when we have Neumann boundary conditions. Let $\boldsymbol{z} = (z_1, z_2, ..., z_{N_1})$ be the locations where we place $f(x)$ sensors and $\boldsymbol{y} = (y_1, y_2, ..., y_M)$ be the locations of sampling points on the Neumann boundary. Since the derivatives and linear combinations of Gaussian processes are still Gaussian processes, we have

$$\begin{aligned} u'(\boldsymbol{y}) &\sim \mathcal{GP}(0, k(\boldsymbol{y}, \boldsymbol{y}'; \theta)), \\ f(\boldsymbol{z}) &\sim \mathcal{GP}(0, h(\boldsymbol{z}, \boldsymbol{z}'; \theta)), \end{aligned} \tag{3.5}$$

where

$$k(\boldsymbol{y}, \boldsymbol{y}'; \theta) = \frac{\partial}{\partial \boldsymbol{n_1}} \frac{\partial}{\partial \boldsymbol{n_2}} g(\boldsymbol{y}, \boldsymbol{y}'; \theta),$$
$$h(\boldsymbol{z}, \boldsymbol{z}'; \theta) = \mathcal{L}_z \mathcal{L}_{z'} g(\boldsymbol{z}, \boldsymbol{z}'; \theta). \tag{3.6}$$

In Eq. 3.6, $\boldsymbol{n_1}$ and $\boldsymbol{n_2}$ are the unit outer normal of the first and second entries of the covariance kernel function. Moreover, the joint distribution of $u(\boldsymbol{x})$, $u'(\boldsymbol{y})$ and $f(\boldsymbol{z})$ is

$$U = \begin{bmatrix} u(\boldsymbol{x}) \\ u'(\boldsymbol{y}) \\ f(\boldsymbol{z}) \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K_{00} & K_{01} & K_{02} \\ K_{10} & K_{11} & K_{12} \\ K_{20} & K_{21} & K_{22} \end{bmatrix}\right), \tag{3.7}$$

where

$$\begin{aligned} &K_{00} = g(\boldsymbol{x}, \boldsymbol{x}; \theta) + \sigma_u^2 \boldsymbol{I}, \quad K_{01} = \frac{\partial}{\partial \boldsymbol{n_2}} g(\boldsymbol{x}, \boldsymbol{y}; \theta), \quad K_{02} = \mathcal{L}_z g(\boldsymbol{x}, \boldsymbol{z}; \theta), \\ &K_{10} = K_{01}^T, \qquad\qquad K_{11} = k(\boldsymbol{y}, \boldsymbol{y}; \theta), \qquad K_{12} = \frac{\partial}{\partial \boldsymbol{n_1}} \mathcal{L}_z g(\boldsymbol{y}, \boldsymbol{z}; \theta), \\ &K_{20} = K_{02}^T, \qquad\qquad K_{21} = K_{12}^T, \qquad K_{22} = h(\boldsymbol{z}, \boldsymbol{z}; \theta) + \sigma_f^2 \boldsymbol{I}. \end{aligned} \tag{3.8}$$

In Eq. 3.8, the additional variances due to the measurement error are included in K_{00} and K_{22} as $\sigma_u^2 I$ and $\sigma_f^2 I$.

The sensors' data and sample points on domain boundaries serve all together as the training data set. The covariance kernel hyper-parameter θ (including σ_u and σ_f, if they are not given) will be estimated by minimizing the negative log marginal likelihood defined by

$$\mathcal{NLML} := -\log p(U|\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}; \theta, \sigma_u, \sigma_f), \tag{3.9}$$

which can be explicitly written as

$$\mathcal{NLML} = \frac{1}{2} \boldsymbol{Y}^T K^{-1} \boldsymbol{Y} + \frac{1}{2} \log |K| + \frac{n}{2} \log(2\pi), \tag{3.10}$$

where $\boldsymbol{Y} := [u(\boldsymbol{x}), u'(\boldsymbol{y}), f(\boldsymbol{z})]^T$, and n is the total number of training points [65].

Given the assumption that the solution is a Gaussian process, the value $u(x_0)$ at $x_0 \in \Omega$ and all the training data follow a multi-variant Gaussian distribution,

$$\begin{bmatrix} u(x_0) \\ \boldsymbol{Y} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} g(x_0, x_0; \sigma_u, \sigma_f) & \boldsymbol{a} \\ \boldsymbol{a^T} & K \end{bmatrix}\right), \tag{3.11}$$

where $\boldsymbol{a} = [g(x_0, \boldsymbol{x}; \theta), \frac{\partial}{\partial \boldsymbol{n_2}} g(x_0, \boldsymbol{y}; \theta), \mathcal{L}_z g(x_0, \boldsymbol{z}; \theta)]$. The posterior distribution of $u(x_0)$ given all the training data is then calculated from the Bayesian formula,

$$p(u(x_0)|\boldsymbol{Y}) = \mathcal{N}(\boldsymbol{a}K^{-1}\boldsymbol{Y}, g(x_0, x_0; \theta) - \boldsymbol{a}K^{-1}\boldsymbol{a^T}). \tag{3.12}$$

Therefore, the maximum a posteriori estimation of $u(x_0)$ is $\boldsymbol{a}K^{-1}\boldsymbol{Y}$, with the prediction variance $g(x_0, x_0; \theta) - \boldsymbol{a}K^{-1}\boldsymbol{a^T}$.

3.3.2 Inferring Solutions of PDEs from Multi-fidelity Data

Here we consider a realistic situation where the information gathered by the sensors is of variable fidelity due to primarily different sensor qualities, for example, different resolutions. Typically, the availability of high-fidelity data is quite limited, while the low-fidelity data is much easier to collect. In general, low-fidelity data could come from inexpensive sensors or uncalibrated measurements, e.g., satellite data, or even from computations, e.g., using inexpensive reduced-order models. We denote the high-fidelity model of $u(x)$ by $u^h(x)$, and the low-fidelity model of $u(x)$ by $u^l(x)$. The auto-regressive model [49, 60, 55] reads

$$u^h(x) = \rho u^l(x) + \delta(x), \tag{3.13}$$

where $u^l(x)$ and $\delta(x)$ are two independent Gaussian processes with

$$u^l(x) \sim \mathcal{GP}(0, g_1(x, x'; \theta_1)), \quad \delta(x) \sim \mathcal{GP}(0, g_2(x, x'; \theta_2)). \tag{3.14}$$

Here, $g_1(x, x'; \theta_1)$ and $g_2(x, x'; \theta_2)$ are covariance functions, θ_1, θ_2 denote their hyper-parameters and ρ is the cross-correlation parameter to be learned from the data. Therefore, from Eq. 3.13 we can get

$$u^h(x) \sim \mathcal{GP}(0, g(x, x'; \theta_1, \theta_2)), \tag{3.15}$$

where

$$g(x, x'; \theta_1, \theta_2) = \rho^2 g_1(x, x'; \theta_1) + g_2(x, x'; \theta_2). \tag{3.16}$$

Given that the operator $\mathcal{L}_x$ is linear, we arrive at the auto-regressive structure $f^h(x) = \rho f^l(x) + \gamma(x)$ on the forcing, where $\gamma(x) = \mathcal{L}_x \delta(x)$ and $f^l(x) = \mathcal{L}_x u^l(x)$ are two independent Gaussian processes. We note that the low-fidelity model $u^l(x)$ and $f^l(x)$ satisfies the same equation as the high-fidelity model.

Suppose we have two types of $f(x)$ sensors, where the highly accurate but expensive sensors are placed at location $\boldsymbol{z}^h$, while the sensors low accuracy are placed at $\boldsymbol{z}^l$. Thus, the joint distribution of all training data is

$$\begin{bmatrix} u^h(\boldsymbol{x}) \\ u'(\boldsymbol{y}) \\ f^h(\boldsymbol{z}^h) \\ f^l(\boldsymbol{z}^l) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K_{00} & K_{01} & K_{02} & K_{03} \\ K_{10} & K_{11} & K_{12} & K_{13} \\ K_{20} & K_{21} & K_{22} & K_{23} \\ K_{30} & K_{31} & K_{32} & K_{33} \end{bmatrix} \right). \tag{3.17}$$

In Eq. 3.17, term K_{00}, K_{01}, K_{02}, K_{10}, K_{11}, K_{12}, K_{20}, K_{21}, K_{22} are defined in the same way as in Eq. 3.8, with $g(x, x'; \theta)$ replaced by Eq. 3.16. Due to the independence

assumption of $u^l(x)$ and $\delta(x)$, we can write down the rest of the matrix explicitly,

$$\begin{aligned} K_{03} &= \rho \mathcal{L}_{z^l} g_1(\boldsymbol{x}, \boldsymbol{z}^l; \theta_1), & K_{13} &= \rho \frac{\partial}{\partial \boldsymbol{n_1}} \mathcal{L}_{z^l} g_1(\boldsymbol{y}, \boldsymbol{z}^l; \theta_1), \\ K_{23} &= \rho \mathcal{L}_{z^h} \mathcal{L}_{z^l} g_1(\boldsymbol{z}^h, \boldsymbol{z}^l; \theta_1), & K_{33} &= \mathcal{L}_{z^l} \mathcal{L}_{z^l} g_1(\boldsymbol{z}^h, \boldsymbol{z}^l; \theta_1) + \sigma_{f^l}^2 \boldsymbol{I}, \end{aligned} \tag{3.18}$$

and $K_{30} = K_{03}^T$, $K_{31} = K_{13}^T$, $K_{32} = K_{23}^T$. The training and predicting procedures resemble those in the single-fidelity situation. We note that if possible, it is desirable to use nested sensors so that the high-fidelity sensors are at the same location as the low-fidelity sensors to reduce the computational cost [25], however, it is appreciated that may not possible in a real situation.

3.3.3 Domain Decomposition Algorithm with Gaussian Process Regression

Algorithm 3: GPDD Algorithm

Initializing $u_m^0(x_b)$ with 0;
Set $k = 0$;
while $\|u_m^{k-1}(x_b) - u_m^k(x_b)\| \leq \epsilon_{tol}$ **do**
    Solve for $u_m^k(x)$ with the given solver in $\mathcal{D}_m$;
    Calculate ${u'_m}^k(x_b)$ as the Neumann boundary condition in $\mathcal{D}_s$;
    Predict $u_s^k(x)$ with GPR in $\mathcal{D}_s$;
    Set $u_m^{k+1}(x_b) = u_s^k(x_b)$;
    Set $k = k + 1$;
end

As the core of this work, we introduce the hybrid domain decomposition algorithm with Gaussian process regression (GPDD). Suppose we are provided

with a PDE solver in the *dominus* domain $\mathcal{D}_m$, which we call the PDE-domain, while in the *servus* domain $\mathcal{D}_s$, also referred to as the Data-domain, we have access to sensors data. The GPDD algorithm serves to couple the data together with the PDE solver via the Dirichlet-Neumann type Schwarz iterative method. We note that a reversed version of interface conditions could also be applied, i.e., imposing a Dirichlet boundary condition to the Data-domain and a Neumann boundary condition to the PDE-domain. A comparison between both versions in terms of the solution accuracy is displayed in Section 3.4. For simplicity, we set the PDE-domain to be the *dominus* domain and the Data-domain to be the *servus* domain unless it is specified otherwise. Depending on the type of sensors we have in $\mathcal{D}_s$, this GPDD algorithm can be applied in the following two cases:

Case 1: We have a PDE solver in the PDE-domain, while in the Data-domain we have access to the sensors' data of u. We assume that we do *not* know the PDE in the Data-domain.

Case 2: We have a PDE solver in the PDE-domain, while in the Data-domain we have access to the sensors' data of the forcing term f, which could be of variable fidelity. In this case, we know the exact form of the PDE left-hand-side operator $\mathcal{L}_x$ in the Data-domain. No other information about u, except for the boundary conditions, is needed in the Data-domain.

3.4 Numerical Results

3.4.1 Application to 1D Helmholtz Equation

We solve the 1D Helmholtz equation

$$-u_{xx} + \lambda^2 u = f(x), \quad x \in [0,1], \quad u(0) = u(1) = 0, \tag{3.19}$$

where $\lambda = 1$ is a constant, $f(x)$ is the forcing term, and $u(x)$ is the QoI. The entire domain $\mathcal{D} : [0,1]$ is divided into two non-overlapping subdomains: the PDE-domain $\mathcal{D}_1 : [0, 0.6]$, and the Data-domain $\mathcal{D}_2 : [0.6, 1.0]$. The numerical PDE solver in $\mathcal{D}_1$ is implemented using a spectral/*hp* element method [31, 37] with 6 spectral elements and the 10th order polynomial but will only be accessed as a black box. We manufactured the reference solution so that it displays two different length scales in different subdomains:

$$u(x) = \begin{cases} 3\sin\left(\frac{10\pi x}{3}\right), & x \in \mathcal{D}_1 \\ \sin\left(\frac{45\pi}{2}\left(x - \frac{3}{5}\right)\right) - 5\sin\left(\frac{5\pi}{2}\left(x - \frac{3}{5}\right)\right), & x \in \mathcal{D}_2. \end{cases} \tag{3.20}$$

The resulting forcing term $f(x)$ is

$$f(x) = \begin{cases} \left(-\frac{100\pi^2}{3} + 3\right)\sin\left(\frac{10\pi x}{3}\right), & x \in \mathcal{D}_1, \\ \left(-\frac{2025\pi^2}{4} + 1\right)\sin\left(\frac{45\pi}{2}\left(x - \frac{3}{5}\right)\right) & \\ \quad + \frac{5}{2}(25\pi^2 - 1)\sin\left(5\pi(x - \frac{3}{5})\right), & x \in \mathcal{D}_2. \end{cases} \tag{3.21}$$

The GPDD solution is denoted by $\tilde{u}(x)$. Both cases in Section 3.3.3 will be considered, and we set the Schwarz iteration terminating threshold ϵ in Eq. 1.11 to be

10^{-7}. We test our GPDD algorithm performance using both noiseless and noisy sensor data.

Case 1

For demonstration purpose, we distribute the $u(x)$ sensors uniformly in the Data-domain $\mathcal{D}_2$. To generate noisy sensors data, we deliberately add independent Gaussian noise of a fixed standard deviation σ_u to each sensor. In Figure 3.2a and Figure 3.2b, we compare the GPDD solution $\tilde{u}(x)$, calculated with both noiseless and noisy sensors, and the reference solution $u(x)$. As we can see, even if the exact solution displays a multi-scale property, for both cases, the GPDD solution $\tilde{u}(x)$ can approximate $u(x)$ very well. Although the problem to be solved is deterministic, predicting the solution in $\mathcal{D}_2$ is indeed the progress of maximum likelihood estimation, and therefore, for every $x \in \mathcal{D}_2$ there exists a Gaussian distribution associated with the predicted solution $\tilde{u}(x)$. Therefore, we can feed the PDE-domain $\mathcal{D}_1$ with a Dirichlet interface condition of a Gaussian distribution, instead of a single value. Due to the linearity of the PDE, solving the equation in $\mathcal{D}_1$ generates a Gaussian distributed Neumann boundary condition at the interface, which shall be passed to the numerical GPR solver in $\mathcal{D}_2$ naturally as a "noisy" boundary condition measurement. By repeatedly solving for $\tilde{u}(x)$ in both subdomains with Gaussian distributed boundary conditions, we manage to propagate the uncertainty back and forth between these two subdomains. We can observe that in Figure 3.2a the exact solution falls into the 95% confidence interval around the predicted solution generated by noiseless sensors.

Table 3.1 shows the number of iterations needed for the Schwarz iterative scheme to converge when an "optimal" relaxation parameter $\eta = 0.58$ (calculated

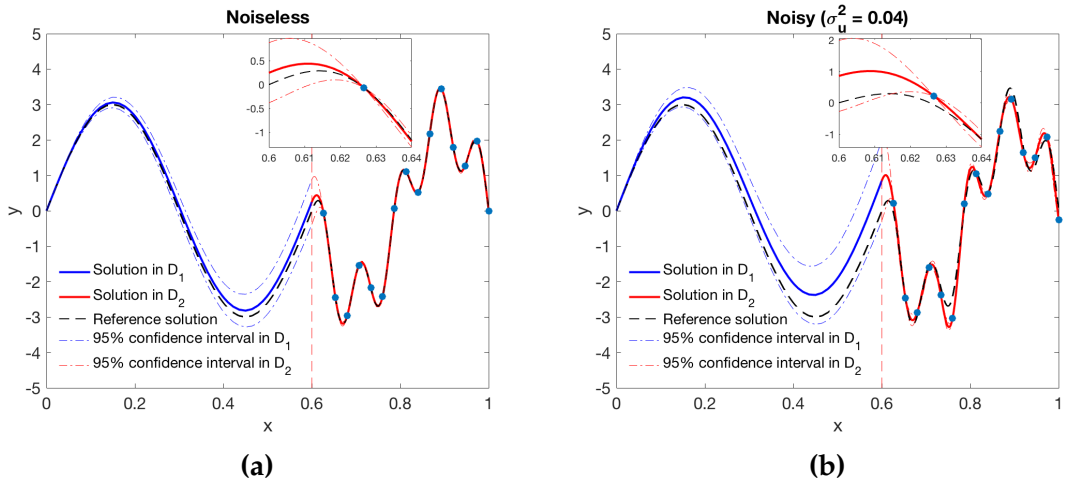


Figure 3.2: Case 1: The GPDD solutions of the 1D Helmholtz equation using (a) noiseless sensors data, and (b) noisy sensors data polluted by the Gaussian noise of standard deviation $\sigma_u = 0.2$. The left-hand side is the PDE-domain and the right-hand side is the Data-domain. The interface between them is denoted by the vertical dashed red line. The Data-domain contains 15 $u(x)$ sensors, marked by the blue dots. The dashed lines mark the 95% confidence interval of the prediction.

from the formula derived in [31]) is employed. Ideally, if both subdomains are equipped with traditional PDE solvers, the serial iterations should converge after two steps, but when we have a GPR solver in the Data-domain, we observe that more iterations are needed to converge, due to the fact that the estimated solution cannot be simply characterized by a fixed PDE during iterations, especially when the sensor data are polluted by the random measurement error (noise). Nevertheless, we can observe the trend that given more sensors, less iterations are needed for the GPDD algorithm to converge at least for noiseless data.

Number of Sensors	20	25	30	35	40
Noiseless ($\sigma = 0$)	17	15	15	14	13
Noisy ($\sigma = 0.2$)	18	17	17	17	16

Table 3.1: Case 1: Number of iterations needed for the Schwarz iterations to converge, using $\epsilon = 10^{-7}$ threshold.

The accuracy of the numerical solution $\tilde{u}$ is measured by the relative L_2 error

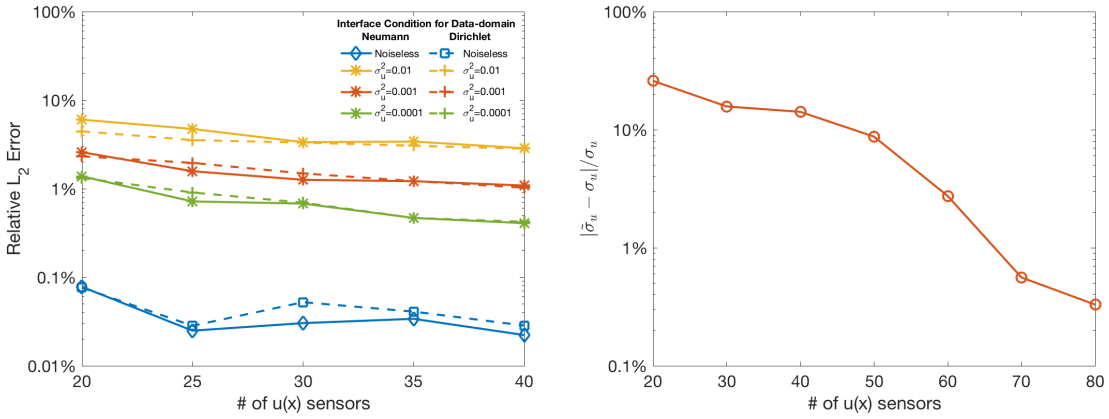


Figure 3.3: Case 1: (a) The relative L_2 error in the entire domain of the predicted solution versus the number of $u(x)$ sensors in the Data-domain $\mathcal{D}_2$ for different levels of sensor's noise. Both the proposed GPDD algorithm and the algorithm with reversed interface conditions are implemented here. (b) The relative error of the predicted magnitude of noise $\tilde{\sigma}_u$ (from the GPDD algorithm), with respect to the input (nominal) noise σ_u.

ϵ_r, defined by

$$\epsilon_r = \frac{\|\tilde{u} - u\|}{\|u\|}, \tag{3.22}$$

where $\|\cdot\|$ denotes the L_2 norm taken in the entire domain. Figure 3.3a compares the relative error of the solutions obtained from the proposed GPDD algorithm and the algorithm of reversed boundary conditions, i.e., feeding the PDE-domain with a Neumann boundary condition and the Data-domain with a Dirichlet boundary condition. Both algorithms produce solutions of similar accuracy, and obviously, we get the most accurate solutions with the noiseless sensors. If we put more $u(x)$ sensors in $\mathcal{D}_2$, we would generally obtain solutions with slightly better accuracy. This makes sense because the more data we collect, the better knowledge we have about the pattern of the solution. Also, as demonstrated in Figure 3.3b, the standard deviation of sensors' noise, σ_u, is learned with the GPDD algorithm, and by increasing the number of sensors we learn the sensors' noise better.

Case 2

In this case, the sensors for $f(x)$ are placed in the Data-domain $\mathcal{D}_2$, and the linear operator $\mathcal{L}_x$ is known a priori:

$$\mathcal{L}_x := -\frac{d^2}{dx^2} + I. \tag{3.23}$$

We test the GPDD algorithm in both situations using either noiseless or noisy sensors. Similarly, the noisy f sensor are manufactured by adding independent Gaussian random noise of standard deviation σ_f to each sensor.

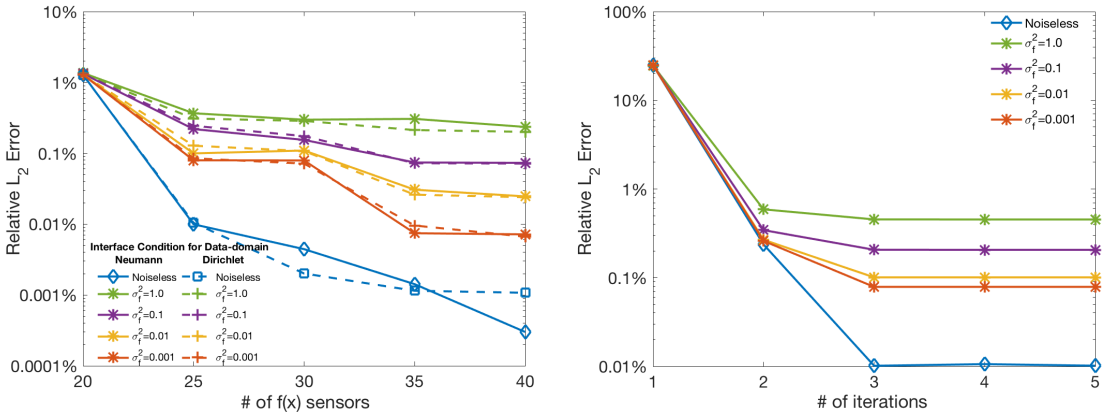


Figure 3.4: Case 2: (a) The relative L_2 error in the entire domain of the predicted solution versus the number of $f(x)$ sensors in the Data-domain $\mathcal{D}_2$ for different levels of sensor's noise. Both the proposed GPDD algorithm and the algorithm with reversed interface conditions are implemented here. (b) For different levels of sensor noise, the iterative process of the GPDD algorithm converges within 5 iterations. In this example, 25 sensors are used in the Data-domain.

In Figure 3.4a we compare the accuracy of the GPDD algorithm and the algorithm of reversed boundary conditions for different numbers of $f(x)$ sensors and different levels of sensors' noise. Again, both algorithms display similar accuracy. It is evident that the relative L_2 error decreases when we place a greater larger number of less noisy sensors in the Data-domain. By using the noiseless sensors, we achieve a very accurate solution with the relative L_2 error less than 0.01%.

Even when we use noisy sensors of standard deviation $\sigma_f = 1.0$, we still obtain solutions with relative error less than 1%. Figure 3.4b shows the convergence of the GPDD algorithm after a few iterations when 25 $f(x)$ sensors are uniformly placed in the Data-domain and an optimal $\eta = 0.58$ is adopted. The relative error of the GPDD solution decays to reach a stable level within 5 iterations, indicating very fast convergence of the GPDD algorithm for this case.

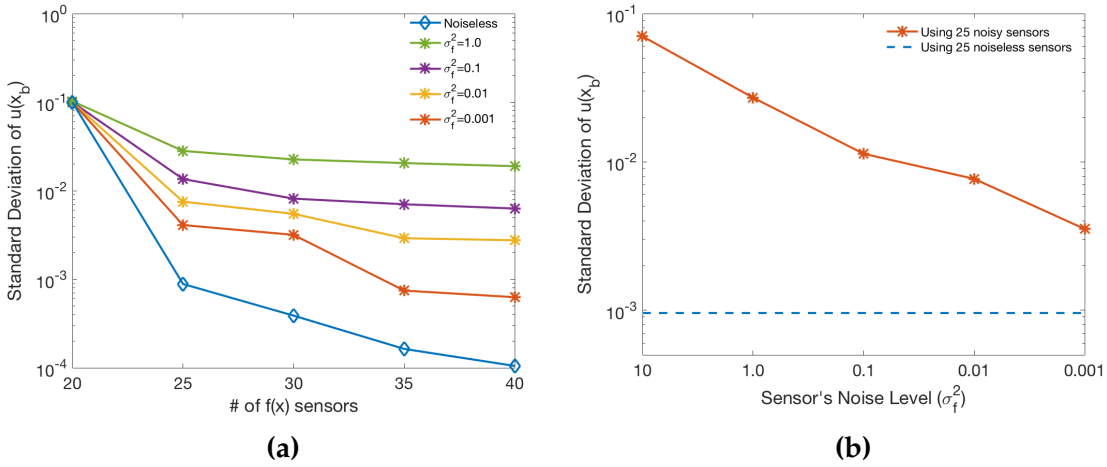


Figure 3.5: Case 2: (a) The standard deviation of $u(x_b)$ is reduced when more $f(x)$ sensors are used. (b) The standard deviation of $u(x_b)$ is reduced when less noisy $f(x)$ sensors are being used.

We are also interested in the uncertainty associated with our GPDD solution. Figure 3.5a shows that when more sensors are placed in $\mathcal{D}_2$, the standard deviation of the predicted $\tilde{u}(x_b)$ decays, indicating that we are more confident with our solution. Figure 3.5b also confirms that less noisy sensors would result in more confident prediction of solution. In Figure 3.6 we plot the 95% confidence interval of the predicted solution $\tilde{u}(x)$, obtained with 25 noisy sensors ($\sigma_f = 1.0$) in $\mathcal{D}_2$. Here we intentionally choose the boundary relaxation parameter $\eta = 0.3$ to suppress the convergence rate. The confidence interval shrinks rapidly within the first few iterations, indicating that we are gaining confidence of our predictions through this Schwarz type iteration.

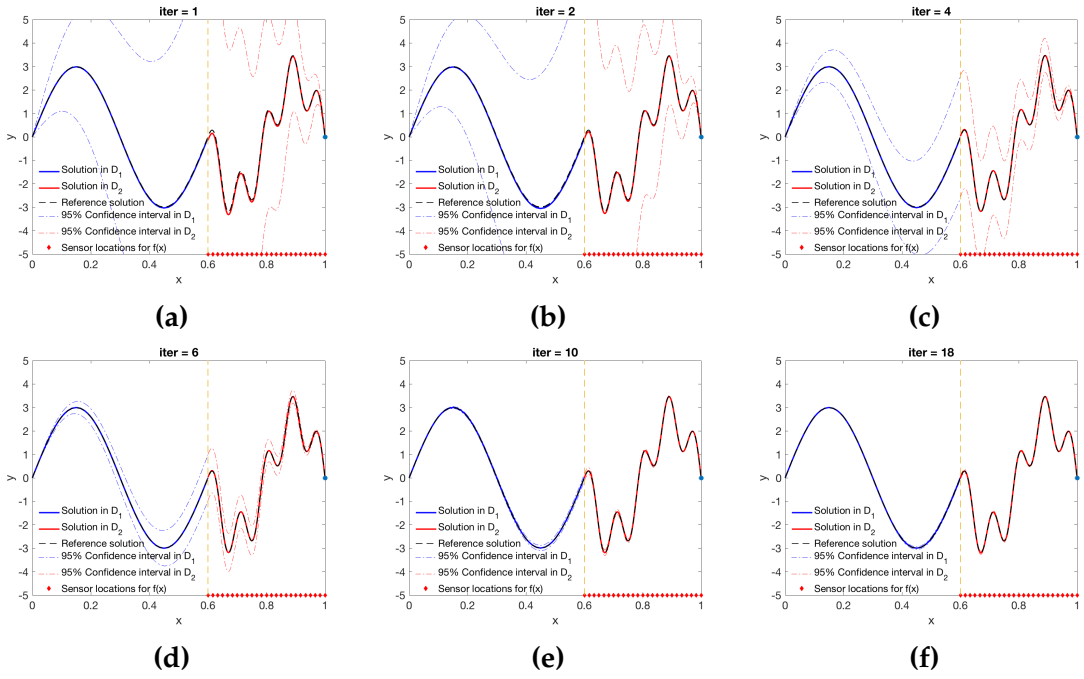


Figure 3.6: Case 2: The solution profile of the 1D Helmholtz equation ($\lambda = 1$), using 25 noisy sensors ($\sigma_f = 1.0$) in $\mathcal{D}_2$. The 95% confidence interval of predicted solution is reduced quickly in less than 10 Schwarz iterations.

3.4.2 Application to 2D Helmholtz Equation

We extend our demonstration example into the 2D physical space, and solve the 2D Helmholtz equation:

$$-\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) u + \lambda^2 u = f(x, y), \quad (x, y) \in [0, 1.5] \times [0, 1]. \tag{3.24}$$

In this section, we also demonstrate the useful information that can be extracted from a network of *multi-fidelity* sensors.

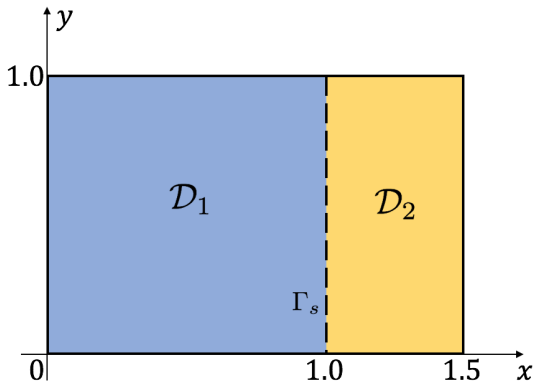


Figure 3.7: The 2D physical domain and its decomposition for case 1: *dominus* domain: $\mathcal{D}_1 := [0,1] \times [0,1]$, and *servus* domain: $\mathcal{D}_2 := [1,1.5] \times [0,1]$.

Case 1

We enforce a homogeneous Dirichlet boundary condition for the global domain $\mathcal{D}$:

$$u(1.5, y) = u(0, y) = u(x, 0) = u(x, 1) = 0. \tag{3.25}$$

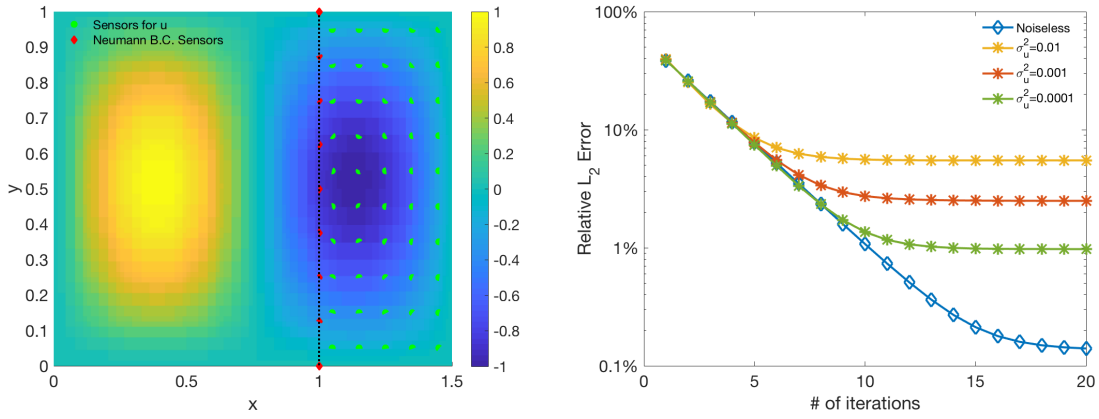
The global domain $\mathcal{D}$ is divided into two non-overlapping subdomains as depicted in Figure 3.7, while the PDE-domain $\mathcal{D}_1$ serves as the *dominus* domain, equipped with a Dirichlet interface condition at the interface Γ_s, and the Data-domain $\mathcal{D}_2$ is the *servus* domain with a Neumann interface condition. The manufactured reference solution is:

$$u(x, y) = \sin\left(\frac{4}{3}\pi x\right)\sin(\pi y), \tag{3.26}$$

and by choosing $\lambda = 1$,

$$f(x, y) = (\lambda^2 + \frac{25}{9}\pi^2)\sin\left(\frac{4}{3}\pi x\right)\sin(\pi y). \tag{3.27}$$

As shown in Figure 3.8a, the u sensors (green dots) are uniformly distributed in the Data-domain $\mathcal{D}_2$ to form a 10×5 lattice grid and the Neumann interface



(a) **(b)**

Figure 3.8: Case 1: (a) The GPDD solution of the 2D Helmholtz equation using fifty u sensors (marked in green). The sample points for the Neumann boundary conditions are marked in red. The black dotted line indicates the subdomain interface. (b) The relative L_2 error of solution in the entire domain versus the number of iterations.

condition is sampled at the red diamonds. We use a relaxation parameter $\eta = 0.3$ and a stopping threshold $\epsilon = 10^{-5}$ for the Schwarz iterations. A visualization of the GPDD solution generated by noiseless sensors is displayed in Figure 3.8a. Figure 3.8b shows that the relative L_2 error decays after a few iterations, where less noisy sensors lead to more accurate predicted solutions.

Case 2 with Multi-fidelity Data

We test our GPDD algorithm under the situation where we have two sources of f sensors data of different fidelity, using a new 2D domain decomposition paradigm displayed in Figure 3.9. For this test we enforce a homogeneous Neumann boundary condition on the global domain $\mathcal{D} := [-1, 1] \times [-1, 1]$, i.e.

$$u_x(1.5, y) = u_x(0, y) = u_y(x, 0) = u_y(x, 1) = 0. \tag{3.28}$$

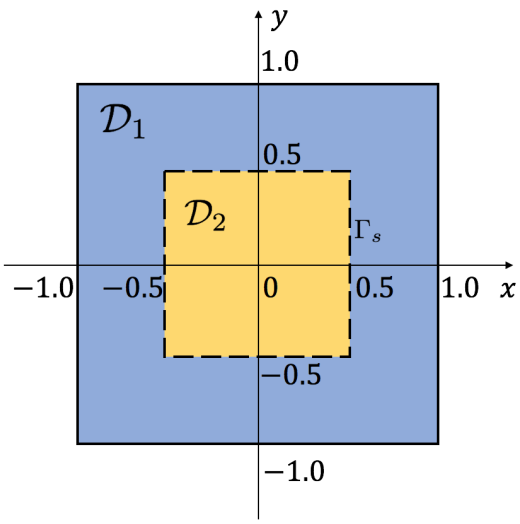


Figure 3.9: The 2D physical domain and its decomposition for case 2: *servus* domain: $\mathcal{D}_2 := [-0.5, 0.5] \times [-0.5, 0.5]$, *dominus* domain: $\mathcal{D}_1 := [-1, 1] \times [-1, 1] \setminus \mathcal{D}_2$.

The manufactured reference solution is:

$$u(x, y) = \sin\left(\frac{3\pi x}{2}\right)(2 + \cos(\pi y)). \tag{3.29}$$

As for the forcing term, the high-fidelity function is

$$f^h(x, y) = \left(\left(1 + \frac{9\pi^2}{4}\right)(2 + \cos(\pi y)) + \pi^2 \cos(\pi y)\right)\sin\left(\frac{3\pi x}{2}\right), \tag{3.30}$$

while the low-fidelity function is chosen to be a scaling of the high-fidelity function combined with a non-trivial noise:

$$f^l(x, y) = \frac{4}{5} f^h(x, y) + \frac{2\pi x}{15}\sin(\pi y). \tag{3.31}$$

In order to evaluate the effectiveness of our method with multi-fidelity data, we compare the error of the GPDD solutions when different numbers of low-fidelity and high-fidelity f sensors are placed in the Data-domain. For the first experiment, we fix the number and position of the high-fidelity sensors and investigate the

effect of using an increase number of low-fidelity sensors. Positions of the sensors are illustrated in Figure 3.10, where we intentionally choose a nested setup of the low-fidelity sensors so that the information from the previous tests is kept intact in the succeeding tests. Since the Data-domain is fed only with information of the derivatives of u, the predicted solution can vary by any constant. Therefore in addition to the sensors for f, we place 4 anchor points for u in the Data-domain to pin the solution. The relaxation parameter η is set to be 0.2. To avoid the influence of the initialization of hyper-parameters, in practice, we conduct 50 independent runs for each setup with randomly initialized hyper-parameters, and the errors are calculated using an average of the 15 runs with minimum $\mathcal{NLML}$s.

Figure 3.11b indicates the decay of relative L_2 error in our predicted solution with different sensor setups. We compare in Figure 3.11c the error of the numerical solutions after the iteration converges, as the number of low-fidelity sensors increases, the error shows a decreasing trend at first. However, when we continue adding more low-fidelity sensors, the error starts increasing; this is an indication that we reached the point of "diminishing return" from the low-fidelity sensors. Instead, for the second experiment, we put just another high-fidelity sensor (Figure 3.11a), the error could be further decreased (indicated by the yellow triangle in Figure 3.11c. The general strategies of selecting a proper number and the positions of high-fidelity and low-fidelity sensors are important issues and are related to active learning so we plan to investigate it systematically in the future work.

3.4.3 Computational Cost Analysis

Here we provide a rough estimation of the computational cost in the Data-domain. Note that in each iteration, $\mathcal{NLML}$ is minimized and a prediction is conducted

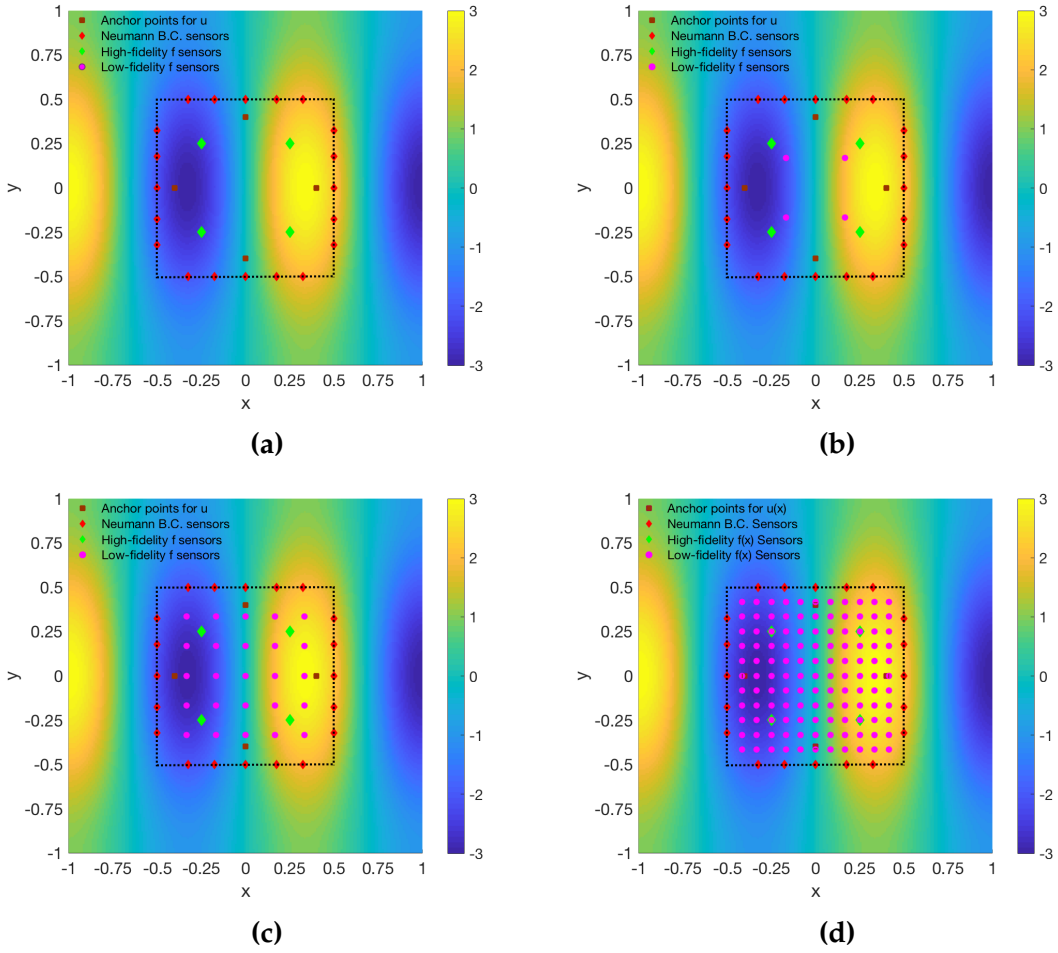


Figure 3.10: Positions of the high-fidelity and low-fidelity sensors. The number of high-fidelity sensors is fixed at 4 while the number of low-fidelity sensors varies: 0, 4, 25, 121. The black dotted line indicates the subdomain interface.

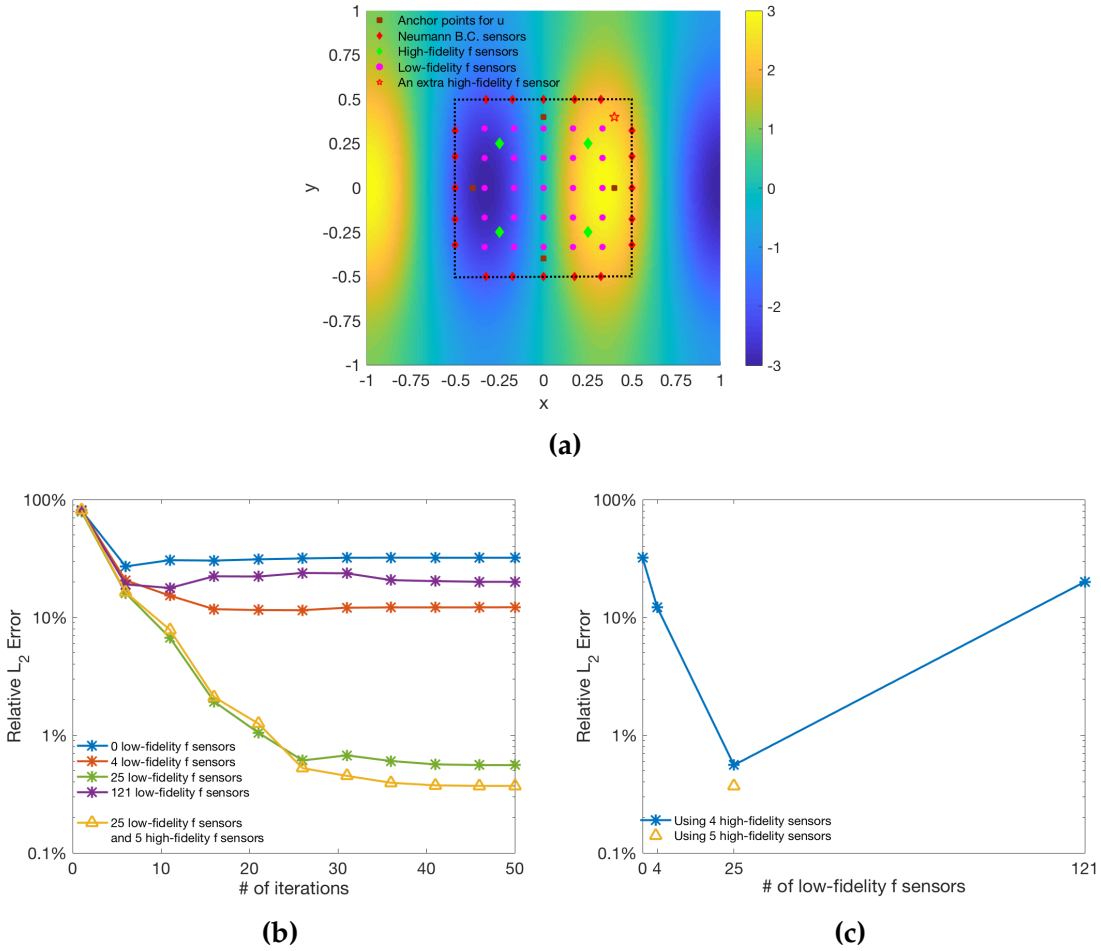


Figure 3.11: (a) Illustration of adding an extra high-fidelity sensor (5 high-fidelity sensors in total) to the 25 low-fidelity sensors setup. (b) The relative L_2 error of solution decays after a few Schwarz iterations and remains at a stable level, showing convergence. (c) Error of the numerical solution at the end of the iteration versus the number of low-fidelity sensors. Using an extra high-fidelity sensor improves the accuracy.

in the Data-domain. During each minimization, $\mathcal{NLML}$ is repeatedly evaluated, and we denote the number of evaluations as E. Assume N to be the total number of sensors and boundary condition points in the Data-domain, then the size of the covariance matrix used for computing $\mathcal{NLML}$ will be N by N. As a consequence, during each evaluation of $\mathcal{NLML}$, the computational cost of generating the covariance matrix is $O(N^2)$, and the computational cost of calculating $\mathcal{NLML}$ with the covariance matrix is $O(N^3)$ if we use Cholesky decomposition to invert the covariance matrix. The covariance matrix and its inverse could be reused in the prediction stage. The total computational cost for each $\mathcal{NLML}$ evaluation is $O(N^3)$, and hence the total computational cost for each minimization is $O(EN^3)$.

Suppose that we need to make predictions at M points in the Data-domain. During the iterations, we only need to make predictions on the Neumann boundary condition points for the sake of information fusion. The computational cost of generating covariance matrices, i.e., g and $\boldsymbol{a}$ in Eq. 3.11, is $O(MN) + O(M^2)$, and the computational cost of making predictions with the matrices is $O(MN^2) + O(M^2N)$. Therefore, the cost of making predictions is $O(MN^2) + O(M^2N)$. We conclude that, roughly the computational cost in the Data-domain in each iteration is $O(EN^3) + O(MN^2) + O(M^2N)$.

3.5 Summary of Chapter

In this chapter we address the issue of coupling a solution in two types of domains, one with a traditional PDE solver, and the other one with sparse sensor data. We proposed a GPDD algorithm where the PDE-domain and the Data-domain are synchronized by the Schwarz type iterative method that can propagate informa-

tion across the subdomain interface in both directions. The uncertainty in the GP prediction is spread and results in a distribution of the predicted global solution. The PDE-domain acts as the *dominus* domain where we impose a Dirichlet boundary condition at the interface, while the Data-domain acts as the *servus* domain where numerical GPR is used to infer the solution subject to the Neumann interface condition. The sensor data in the Data-domain can be either noiseless (exact) or noisy (with measurement error). Two specific situations were considered:

1. We have sensor data of the QoI in the Data-domain but we do not have a governing PDE. In this situation, GPR is performed in the Data-domain.

2. We have sensor data of the forcing term and we also know the governing PDE in the Data-domain. In this situation, we build a joint distribution of the solution and the forcing term. The solution in the Data-domain can be inferred using the numerical GPR.

The GPDD algorithm is proved to be reliable for solving linear equations in both 1D and 2D physical domains. The iterative process helps with the training of GP, as the error in solution and the variance of prediction decays fast after just a few iterations, which is a non-trivial result. We also observed that by using noiseless sensors and by using larger amount of sensors, we obtain more accurate and more trustworthy results. Moreover, multi-fidelity sensors could be incorporated with the GPDD framework. A combination of cheap low-fidelity sensors and expensive high-fidelity sensors can contribute to better solutions, which is of great significance in practice, especially because in most applications one has to operate at limited budget and resources.

There are several open questions and challenges related to the GPDD algo-

rithm. For example, measurements in practice could be collected at a scale distant from that of the PDE model, and the small-scale behavior of the QoI would be extremely difficult to resolve and would be easily attributed to the sensor's noise without careful treatment. To deal with this situation, a sufficiently large number of sensors should be employed and we should also use the non-stationary GP kernels because they are more adapted to the locally small-scale changes. Moreover, this data-driven domain decomposition method could be integrated with the non-linear information fusion algorithm [54] and time-dependent non-linear GPR algorithm [64] to learn the complex space and time dependent cross-correlations in multi-fidelity data sets, and to safeguard our computations against erroneous data or the low-fidelity models that may provide wrong trends.

Chapter Four

Solving Data-driven Forward and Inverse Stochastic Problems with Physics-Informed Neural Networks

4.1 Introduction

Data-driven modeling is widely applied as a powerful tool for physical and biological systems. For example, in geophysics, researchers have been using the remote sensing data collected from multi-spectral satellites and the top-of-atmospheric reflectance model as a calibration of the data to study the soil salinization [15], or estimating the Earth heat loss based on the heat flow measurements and a model of the hydrothermal circulation in the oceanic crust [56]. Data can be used to provide closures in nonlinear models or to estimate parameters or functions in mathematical models. Moreover, mathematical models can be used as additional knowledge to formulate "informative priors" in statistical estimation methods or be encoded in specially designed machine learning tools so that a smaller amount of data is required for inference of system identification. There has been recent progress for both forward (inference) and inverse (identification) problems using different methods. For example, for the forward problem, some of the popular choices of machine learning tools are Gaussian process [24, 70, 4, 64, 52, 91] and deep neural networks (DNNs) [38, 39, 34, 62, 47]. For inverse problems, similar methods have been advanced, e.g., Bayesian estimation [75] and variational Bayes inference [93], and have been proposed for a wide variety of objectives, from parameter estimation [61] to discovering partial differential equations [68, 67, 63, 59] to learning constitutive relationships [77].

In this work we focus on the DNNs, and in particular the *Physics-Informed Neural Networks* (PINNs) for forward and inverse problems, first introduced in [62, 63]. However, in those works the mathematical models were deterministic differential equations, so here we consider stochastic differential equations that model either random micro-structure in a medium or the lack of complete knowledge ("un-

certainty"), e.g., of the material property. There have only been very few works published on solving stochastic differential equations using DNNs, e.g., [14, 58], for forward problems. Here we study the special and perhaps the most complex case where some of the physics is known, namely via the stochastic differential equations, and the parameter in the equation is represented as a stochastic process, introducing *parametric uncertainty*. First, we solve the forward stochastic Poisson's equation, where there is uncertainty associated with the driving force. Subsequently, we consider the inverse stochastic elliptic equation, where the diffusivity is modeled as a random process. In the latter case, we have only partial information of the diffusivity from scattered sensors but we have much more data available for the solution, and we aim to infer the stochastic processes of not only the solution but also the diffusivity, and quantify their uncertainties given the randomness in the data. An additional uncertainty is due to the DNN approximation, which we will refer to as the *approximation uncertainty*. Taken together, we refer to the parametric uncertainty and the approximation uncertainty as the *total* uncertainty. To the best of our knowledge, the current work is the first to address total uncertainty in solving stochastic forward and inverse problems using DNNs.

In particular, in this chapter we combine the *arbitrary polynomial chaos* (aPC) with PINNs for both the forward and the inverse stochastic problems. One of the most popular methods for uncertainty quantification studies is the *polynomial chaos* [23, 89] because it has been very effective in representing correlated stochastic fields. However, aPC [92, 84, 50, 40, 86] is more suitable for building the orthogonal basis from arbitrary random space, without the need of any assumption on the distribution of the data. Therefore, in the current work, we employ the aPC to develop a combined method that we call NN-aPC, where we use the DNNs to learn each individual mode of the aPC expansion. More importantly, after training, the

proposed method can be used to predict new realizations of the solution based only on very few measurements.

Treatment of the DNN approximation uncertainty has been addressed using different methods in the past. The traditional way to estimate uncertainty in DNNs is using the Bayes' theorem, e.g., the Bayesian neural networks (BNNs) [45, 48]. BNNs are standard DNNs with prior probability distributions placed over their weights, and given observed data inference is then performed on weights. Because the inference is not tractable in general, variational inference is often used to approximate the inference [30, 51, 36, 66, 79, 28]. However, these models have very high additional computational cost because they require more parameters for the same network size and more time for the DNN parameters to converge. Recently, Gal et al. developed a new way to quantify uncertainty in DNNs by using *dropout* [18, 19, 20], which is largely used as a regularization technique [27, 74] to address the problem of over-fitting. Gal et al. [18] showed that a DNN with dropout is mathematically equivalent to approximating a probabilistic deep Gaussian process [13], no matter what network architecture and non-linearities are used. Moreover, dropout does not induce much computation overhead and thus has been used as a practical tool to obtain uncertainty estimation effectively in real applications including language modelling [19], computer vision [32, 33] and medical applications [2, 90]. In this chapter, dropout is used to to estimate the uncertainty in approximating each aPC mode. Based on the magnitude of this uncertainty we set up an active learning strategy and deploy additional sensors to obtain more measurements of the quantity of interest (QoI), in order to improve the predictability of PINNs.

The organization of this chapter is as follows. In Section 4.2, we set up the data-driven forward and inverse problems. In Section 4.3, we introduce our main

algorithm, the NN-aPC, followed by the method of dropout for uncertainty. In Section 4.4, we provide a detailed study of the accuracy and performance of the NN-aPC method for solving both the forward and inverse stochastic diffusion equation and demonstrate the effectiveness of active learning via dropout-induced uncertainty. Finally, we conclude with a brief discussion in Section 4.5.

4.2 Problem Setup

Suppose we have a stochastic differential equation:

$$\begin{aligned} \mathcal{N}_x[u(x;\omega);k(x;\omega)] = 0, \quad x \in \mathcal{D}, \quad \omega \in \Omega, \\ \text{B.C.:} \qquad \mathcal{B}_x[u(x;\omega)] = 0, \quad x \in \Gamma, \end{aligned} \tag{4.1}$$

where $\mathcal{N}_x$ is the general form of a differential operator that could be nonlinear, $\mathcal{D}$ is a d-dimensional physical domain in $\mathbb{R}^d$, Ω is the random space, and $u(x;\omega)$ is the solution to this equation. The boundary condition is imposed through the generalized boundary condition operator $\mathcal{B}_x$ at the domain boundary Γ. The random parameter $k(x;\omega)$ is the source of parametric uncertainty, which could be represented by either a few random variables or by an infinite dimensional (in the random space) random process.

We consider two types of problems here: first, a *forward* problem, where we know exactly the distribution of $k(x;\omega)$ everywhere in the domain $\mathcal{D}$ and $u(x;\omega)$ is our QoI; and second, an *inverse* problem, where we assume that we have incomplete information on $k(x;\omega)$ but some extra knowledge on $u(x;\omega)$, and we are interested in inferring the full stochastic profile of $k(x;\omega)$. In practice, both problems are data-driven, since the information usually comes from data collected via

sensor measurements. Here, we summarize the different scenarios of the sensors placement for each type of the problems:

- *Forward problem*: The u-sensors are placed only at the boundary Γ to provide boundary condition, while the k-sensors are virtual (since we know the distribution of k), thus we can have as many k-sensors as we want and they can be placed anywhere in $\mathcal{D}$.
- *Inverse problem*: In addition to having u-sensors at the boundary Γ, we have a limited number of extra u-sensors that can be placed in the domain $\mathcal{D}$, whereas we only have a limited number of k-sensors.

In this chapter, we address both types of problems but we will focus more on solving the inverse problem.

4.3 Methodology

4.3.1 NN-aPC

We generalize the PINN method to solve stochastic differential equations for both forward and inverse problems, i.e., we aim to infer continuous random processes. Assume that a sensor will generate a sequence of measurements after being installed, and when the data is recorded, all sensors are read simultaneously. We denote the measurements from all the sensors at the same instant by a *snapshot* of the sensor data. Although the measurement results change from one measurement to the next due to randomness, it is reasonable to believe that every snapshot

of sensor data corresponds to the same random event in the random space. We also assume that when the number of snapshots is big enough, the empirical distribution approximates the true distribution.

Let us consider Eq. 4.1. Suppose we have N_k sensors for $k(x;\omega)$ placed at $\{x_k^{(i)}\}_{i=1}^{N_k}$, N_u sensors for $u(x;\omega)$ placed at $\{x_u^{(i)}\}_{i=1}^{N_u}$, and N_f collocation points at $\{x_f^{(i)}\}_{i=1}^{N_f}$ that are used to calculate the residual of Eq. 4.1. A total number of N snapshots of measurements are made from all these sensors. Let $k_s^{(i)}$ and $u_s^{(i)}$ $(s = 1, 2, ..., N)$ be the s-th measurement of k and u at location $x_k^{(i)}$ and $x_u^{(i)}$ respectively, and ω_s is the random instance at the s-th measurement, i.e., $k_s^{(i)} = k(x_k^{(i)};\omega_s)$ and $u_s^{(i)} = u(x_u^{(i)};\omega_s)$. The training data set can be represented by

$$\mathcal{S}_t = \left\{ \left\{ (x_k^{(i)}, k_s^{(i)}) \right\}_{i=1}^{N_k}, \left\{ (x_u^{(i)}, u_s^{(i)}) \right\}_{i=1}^{N_u}, \left\{ (x_f^{(i)}, 0) \right\}_{i=1}^{N_f} \right\}_{s=1}^{N}. \tag{4.2}$$

The proposed NN-aPC method consists of the following steps:

1. dimension reduction;
2. constructing the aPC basis;
3. building the NN-aPC as a surrogate model of aPC modes and train the network for each mode.

The trained NN-aPC can then be used to calculate the statistics of our QoI and to predict new instances of the continuous trajectories of the QoI, with newly collected sensor data. We will explain each of three steps and the prediction procedure below.

Dimension Reduction with Principal Component Analysis

As the first step, we find a lower dimensional random space spanned by a set of hidden random variables for the dimension reduction of our QoI. The most convenient way to do this is via the principal component analysis (PCA). Naturally, we would analyze the data of k, which is the source of randomness in Eq. 4.1. Let K be the $N_k \times N_k$ covariance matrix for the sensor measurements on k, i.e.,

$$K_{i,j} = \mathrm{Cov}(k^{(i)}, k^{(j)}). \tag{4.3}$$

Let λ_l and ϕ_l be the l-th largest eigenvalue and its associated normalized eigenvector of K. Therefore, PCA yields

$$K = \mathbf{\Phi}^T \mathbf{\Lambda} \mathbf{\Phi}, \tag{4.4}$$

where $\mathbf{\Phi} = [\phi_1, \phi_2, ..., \phi_{N_k}]$ is an orthonormal matrix and $\mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \lambda_2, ...\lambda_{N_k})$ is a diagonal matrix. Let $\boldsymbol{k}_s = [k_s^{(1)}, k_s^{(2)}, ..., k_s^{(N_k)}]^T$ be the results of the k measurements of the s-th snapshot, then

$$\boldsymbol{\xi}_s = \mathbf{\Phi}^T \sqrt{\mathbf{\Lambda}}^{-1} \boldsymbol{k}_s \tag{4.5}$$

is an uncorrelated random vector, and hence $\boldsymbol{k}_s$ can be rewritten as a reduced dimensional expansion

$$\boldsymbol{k}_s \approx \boldsymbol{k}_0 + \sqrt{\mathbf{\Lambda}^M} \mathbf{\Phi}^M \boldsymbol{\xi}_s^M, \quad M < N_k, \tag{4.6}$$

where $\boldsymbol{k}_0 = \mathbb{E}[\boldsymbol{k}]$ is the mean of each sensor's measurements. The choice of M depends on how much energy should be maintained in the low-dimensional random space. For reasons of simplicity, we shall always use the reduced dimensional representation and omit the superscript M. We note that Eq. 4.6 can also be written

in the form of the Karhunen-Loève expansion:

$$k(x_k^{(i)};\omega_s) \approx k_0(x_k^{(i)}) + \sum_{l=1}^{M} \sqrt{\lambda_l} k_l(x_k^{(i)}) \xi_{s,l}, \quad M < N_k, \tag{4.7}$$

where $k_0(x_k^{(i)})$ is the mean of k measurements at $x_k^{(i)}$, $k_l(x)$ is the l-th mode function of $k(x;\omega)$ whose value at $x_k^{(i)}$ coincides with the i-th entry of the eigenvector ϕ_l, and $\xi_{s,l}$ is the l-th entry of the random vector $\boldsymbol{\xi}_s$. We want to extend the range of k_l to the entire domain to approximate the continuous samples of $k(x;\omega)$.

Arbitrary Polynomial Chaos

Assume that we have a set of M-dimensional samples of random vectors

$$S := \{\boldsymbol{\xi}_s\}_{s=1}^{N}$$

with hidden probability measure $\rho(\boldsymbol{\xi})$. Given a sufficiently large number of snapshots, we can approximate the underlying probability measure $\rho(\boldsymbol{\xi})$ by the discrete measure $\nu_S(\boldsymbol{\xi})$,

$$\rho(\boldsymbol{\xi}) \approx \nu_S(\boldsymbol{\xi}) = \frac{1}{N} \sum_{\boldsymbol{\xi}_s \in S} \delta_{\boldsymbol{\xi}_s}(\boldsymbol{\xi}), \tag{4.8}$$

where $\delta_{\boldsymbol{\xi}_s}$ is the Dirac measure. Then, a set of multivariate orthonormal polynomial basis $\{\psi_\alpha(\boldsymbol{\xi})\}_{\alpha=0}^{P}$ can be constructed via the Gram-Schmidt orthogonalization process following [86, 40]. The subscript α is the graded lexicographic multi-index and the number of basis, $P+1$, depends on the highest allowed polynomial order r in $\psi_\alpha(\boldsymbol{\xi})$, following the formula

$$P + 1 = \frac{(r+M)!}{r!\,M!}. \tag{4.9}$$

Specifically, the basis $\{\psi_\alpha(\boldsymbol{\xi})\}_{\alpha=0}^{P}$ are constructed using the recursive algorithm

$$\psi_\alpha(\boldsymbol{\xi}) = w_\alpha^\alpha \psi_\alpha^*(\boldsymbol{\xi}) - \sum_{\beta \prec \alpha} w_\beta^\alpha \psi_\beta(\boldsymbol{\xi}), \tag{4.10}$$

where $\psi_\alpha^*(\boldsymbol{\xi}) \coloneqq \prod_{i=1}^{M} \boldsymbol{\xi}_i^{\alpha_i}$ represents the multivariate monomial basis function; the coefficients w_β^α are determined by imposing the orthonormal condition with respect to the discrete measure ν_S, i.e.,

$$\begin{aligned} \int \psi_\alpha(\boldsymbol{\xi})\psi_\beta(\boldsymbol{\xi})\mathrm{d}\rho(\boldsymbol{\xi}) &\approx \int \psi_\alpha(\boldsymbol{\xi})\psi_\beta(\boldsymbol{\xi})\mathrm{d}\nu_S(\boldsymbol{\xi}) \\ &= \frac{1}{N}\sum_{s=1}^{N} \psi_\alpha(\boldsymbol{\xi}_s)\psi_\beta(\boldsymbol{\xi}_s) \\ &\equiv \delta_{\alpha,\beta}, \quad \beta \preceq \alpha. \end{aligned} \tag{4.11}$$

With the polynomial basis $\{\psi_\alpha(\boldsymbol{\xi})\}$ that are automatically adapted to the distribution of $\boldsymbol{\xi}$, we can write any function $g(x;\boldsymbol{\xi})$ in the form of the aPC expansion,

$$g(x;\boldsymbol{\xi}) = \sum_{\alpha=0}^{P} g_\alpha(x)\psi_\alpha(\boldsymbol{\xi}), \tag{4.12}$$

where the functions $g_\alpha(x)$ are called the aPC modes of g and can be calculated by

$$g_\alpha(x) = \frac{1}{N}\sum_{s=1}^{N} \psi_\alpha(\boldsymbol{\xi}_s) g(x;\boldsymbol{\xi}_s). \tag{4.13}$$

Learning Stochastic Modes

The key to our method is to train DNNs that predict the stochastic modes of our QoI. In this section, we focus on the inverse problem where we have to learn both the modes of u and k. (Solving a forward problem is similar and more

straightforward, and will be briefly discussed in Section 4.4.1.) Two disjoint DNNs are constructed, i.e., the network $\widehat{u_\alpha}$, which takes the coordinate x as the input and outputs a $(P+1)\times 1$ vector of the aPC modes of u evaluated at x, and the network $\widehat{k_i}$ that also takes the coordinate x as the input and outputs a $(M+1)\times 1$ vector of the k modes (we take k_0 in Eq. 4.7 as the 0-th mode of k). Then, we can approximate k and u at the s-th snapshot by

$$\tilde{k}(x;\omega_s) = \widehat{k_0}(x) + \sum_{i=1}^{M} \sqrt{\lambda_i}\widehat{k_i}(x)\xi_{s,i}, \tag{4.14}$$

and

$$\tilde{u}(x;\omega_s) = \sum_{\alpha=0}^{P} \widehat{u_\alpha}(x)\psi_\alpha(\boldsymbol{\xi}_s). \tag{4.15}$$

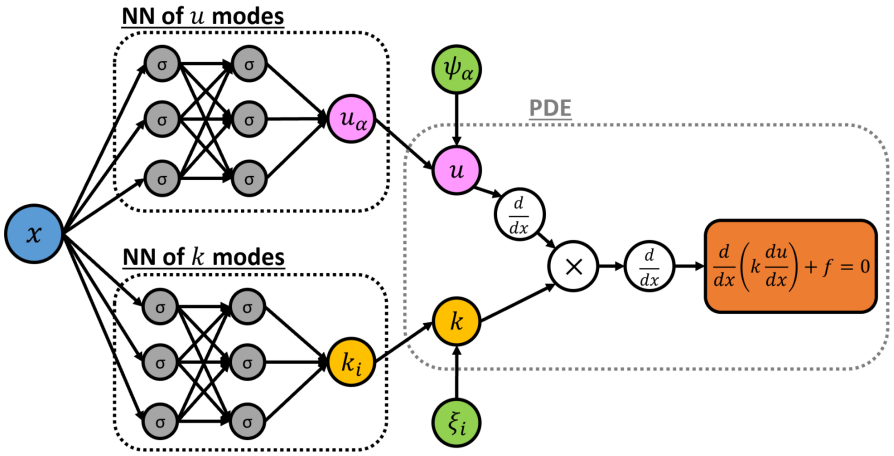


Figure 4.1: Schematic of the NN-aPC for solving the stochastic elliptic equation $-\frac{\mathrm{d}}{\mathrm{d}x}\left(k(x;\omega)\frac{\mathrm{d}}{\mathrm{d}x}u\right) = f$.

Similar to the PINN method, we construct the residual network via automatic differentiation and arithmetic operations of DNNs by substituting $u(x;\omega)$ and $k(x;\omega)$ in Eq. 4.1 with $\tilde{u}(x;\omega_s)$ and $\tilde{k}(x;\omega_s)$. This residual network is aPC-informed, because instead of being an uninterpretable black-box this network is designed to reflect the essence of the aPC expansion (see Figure 4.1 for a schematic of the NN-aPC). In practice, for $\widehat{k_i}$, we separate the mean from the rest of the modes and

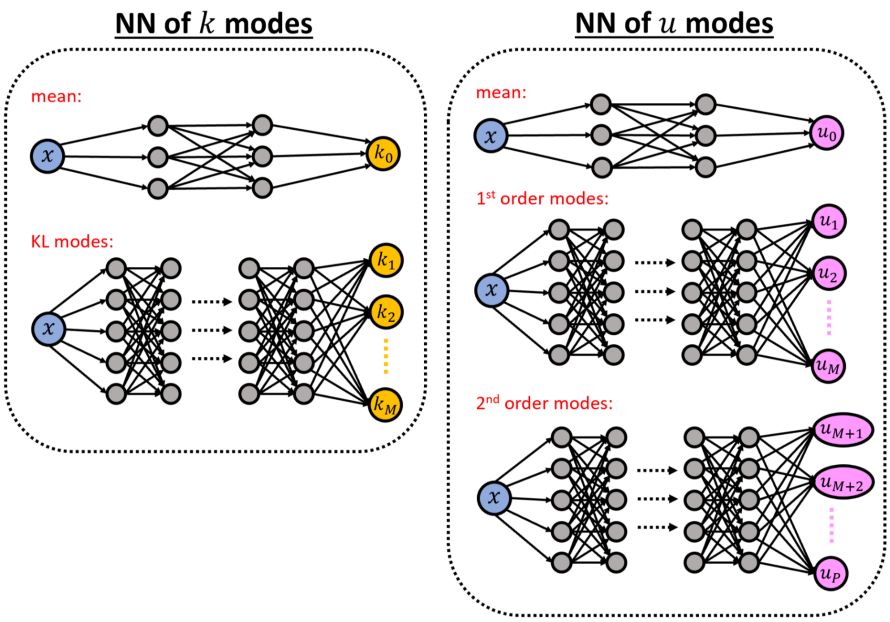


Figure 4.2: Schematic of the DNNs used for learning the stochastic modes of k (left-hand side plot) and u (right-hand side plot). The mean functions are modeled separately using small scale DNNs. For k, all its rest modal functions are modeled using one DNN. For u, the modes that correspond to the same order of aPC expansion are grouped together and are modeled with a single DNN.

learn it with a small scale DNN, and for $\widehat{u_\alpha}$, we group the modes corresponding to the same order of aPC expansion together and learn each group of modes with a separate DNN, as depicted in Figure 4.2. This is due to fact that the mean and the modes of different orders often correspond to vastly different scales.

The loss function is defined as a sum of the mean squared errors (MSE):

$$\mathcal{L}(\mathcal{S}_t) = MSE_u + MSE_k + MSE_f, \tag{4.16}$$

where

$$MSE_u = \frac{1}{NN_u} \sum_{s=1}^{N} \sum_{i=1}^{N_u} \left[\left(\tilde{u}(x_u^{(i)}; \omega_s) - u(x_u^{(i)}; \omega_s) \right)^2 \right], \tag{4.17}$$

$$MSE_k = \frac{1}{NN_k} \sum_{s=1}^{N} \sum_{i=1}^{N_k} \left[\left(\tilde{k}(x_k^{(i)}; \omega_s) - k(x_k^{(i)}; \omega_s) \right)^2 \right], \tag{4.18}$$

and

$$MSE_f = \frac{1}{NN_f}\sum_{s=1}^{N}\sum_{i=1}^{N_f}\left[\left(\mathcal{N}_x[\tilde{u}(x_f^{(i)};\omega_s);\tilde{k}(x_f^{(i)};\omega_s)]\right)^2\right]. \tag{4.19}$$

So far, we have specified the training data, constructed the DNNs and formalized the loss function, and we now ready to train the DNNs.

Predicting Stochastic Realizations

In real applications, the training set could be generated from historical data, and the training process should be performed at the offline stage. The fine-tuned model shall be used to predict new random instances provided with new snapshots of sensor data, at the online stage. Suppose we have a snapshot of sensor data:

$$\left\{\left\{(x_k^{(i)}, k_{\text{new}}^{(i)})\right\}_{i=1}^{N_k}, \left\{(x_u^{(i)}, u_{\text{new}}^{(i)})\right\}_{i=1}^{N_u}\right\}.$$

The first step is to extract the hidden random variables $\boldsymbol{\xi}_{\text{new}}$ from $\{k_{\text{new}}^{(i)}\}_{i=1}^{N_k}$ using Eq. 4.5. Then, for any assigned location x, we can predict the modal functions for both $k(x;\omega)$ and $u(x;\omega)$ from the trained DNNs $\widehat{k_i}$ and $\widehat{u_\alpha}$. Finally, the prediction of k and u for the new random instance can be made via Eq. 4.14 and Eq. 4.15, respectively.

4.3.2 Dropout for Uncertainty

Although DNNs can be used to approximate any measurable function accurately, standard DNNs do not capture model uncertainty. Dropout is one convenient way to quantify the approximation uncertainty in DNNs. The key idea of dropout is to drop units from the DNN independently and randomly with a pre-selected

probability $p \in (0, 1)$. In the original work, dropout was only used during training, while no units were dropped at test time, i.e., the prediction of the DNN for the unknown data was deterministic. In the dropout for uncertainty, the units are also dropped at test time, resulting in stochastic predictions each time. The loss in the dropout inference is the summation of the original loss and a l_2 regularization term over the DNN parameters:

$$\mathcal{L} = \frac{1}{N_t} \sum_{i=1}^{N_t} l(\widehat{y_i}, y_i) + \lambda \sum_i \theta_i^2, \tag{4.20}$$

where N_t is the number of training points, $\widehat{y_i}$ is the prediction, y_i is the true value, $l(\cdot, \cdot)$ is the loss for a single prediction, θ_i is any weight and bias, and λ is the l_2 regularization rate. During prediction, the mean of the output is directly estimated by the Monte Carlo (MC) method,

$$\mathbb{E}(y) \approx \frac{1}{T} \sum_{t=1}^{T} \mathcal{NN}_t(x), \tag{4.21}$$

where $\mathcal{NN}_t$ is the dropped neural network at the t-th prediction. The output variance is also estimated from these MC outputs.

Figure 4.3 shows an example of using the dropout DNN for regression. We plan to use the dropout strategy in the NN-aPC method to estimate the uncertainty of our DNN model and as a guidance for active learning.

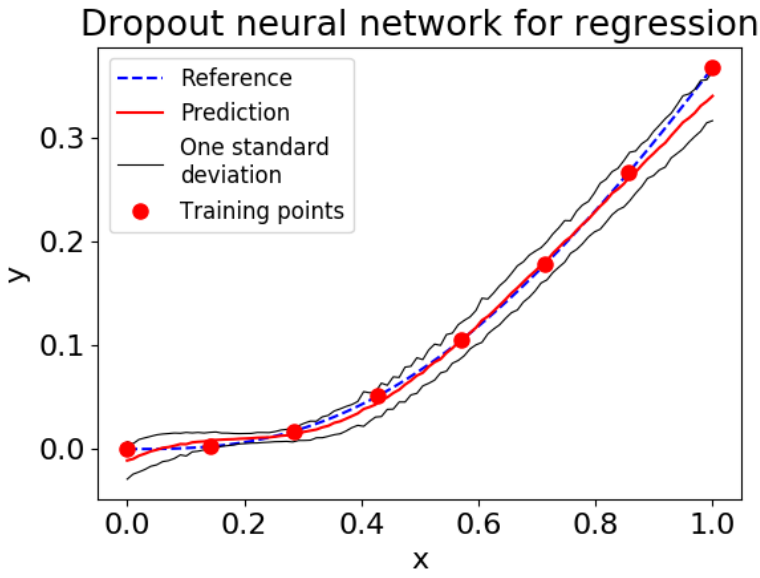


Figure 4.3: Dropout for uncertainty: An example of using the dropout in DNN to approximate the function $y = x^3 e^{-x}$ in the domain $[0, 1]$, where we use 4 hidden layers and 20 neurons per hidden layer, and we choose $p = 0.01, \lambda = 10^{-6}$. The mean and standard deviation are calculated from 1000 MC samples.

4.4 Numerical Examples

4.4.1 Solving Stochastic Differential Equations

We first demonstrate the effectiveness of solving stochastic differential equations with the NN-aPC method for the forward and inverse problems.

Forward Problem: Stochastic Poisson's Equation

Consider the following one-dimensional stochastic Poisson's equation with homogeneous boundary conditions:

$$\begin{aligned} -\frac{\mathrm{d}^2}{\mathrm{d}x^2}u = f(x;\omega), \quad x \in [-1,1] \text{ and } \omega \in \Omega, \\ u(-1) = u(1) = 0. \end{aligned} \tag{4.22}$$

Here Ω is the random space, the forcing term $f(x;\omega) \sim \mathcal{GP}(f_0(x), \mathrm{Cov}(x,x'))$ is a Gaussian random process with mean $f_0(x) = 10\sin(\pi x)$ and a squared exponential covariance function

$$\mathrm{Cov}(x,x') = \sigma^2 \exp\left(-\frac{(x-x')^2}{l_c^2}\right), \tag{4.23}$$

where the standard deviation $\sigma = 1.0$ and the correlation length $l_c = 0.5$.

In this and the following examples, all data are generated by the MC sampling method. Specifically, we sample $N = 1000$ snapshots of continuous $f(x;\omega)$ trajectories $\{f_s = f(x;\omega_s)\}_{s=1}^N$ and extract from $\{f_s\}_{s=1}^N$ the values where the N_f (virtual) sensors are located. For every $f(x;\omega)$ trajectory, we solve for its corresponding solution trajectories $u(x;\omega)$ using the finite difference method, and will use the statistics of these u trajectories as our reference. To evaluate the performance of the trained model, we collect another $N_s = 500$ snapshots of continuous $f(x;\omega)$ and $u(x;\omega)$ trajectories independently from the training data. The N_s pairs of (f,u) trajectories form our test sample set. Similarly, we extract the f-sensor data from every snapshot in the test set as the input at the predicting stage. We shall use the same N and N_s in the following tests, if not explicitly mentioned.

We place $N_f = 13$ sensors of $f(x;\omega)$ in the $[-1,1]$ domain (the sensors are equidistant) and keep 6 principal random variables corresponding to 99% stochas-

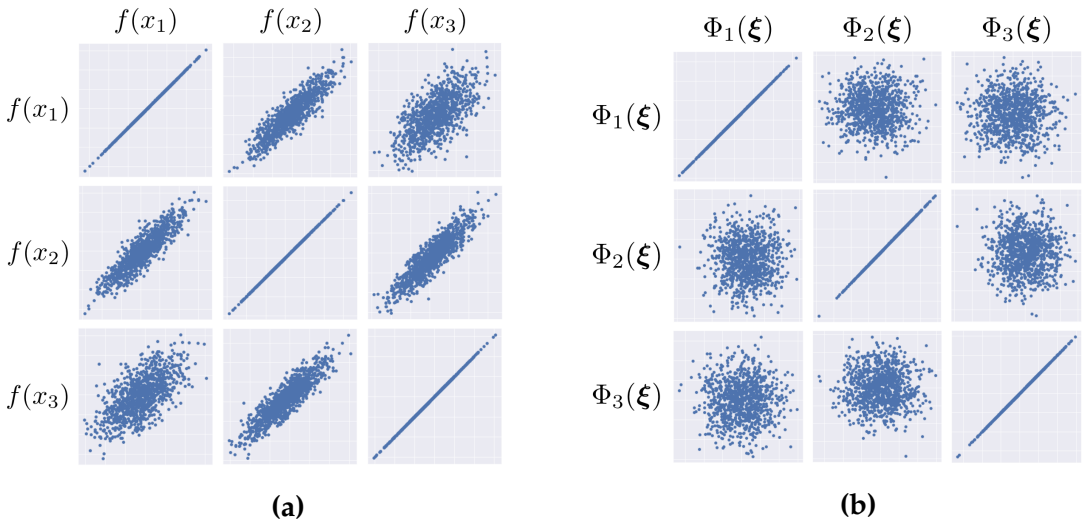


Figure 4.4: Correlation structure: (a) Scattered plots of data collected from the first three f-sensors. The correlation between different sensors is significant, and the closer the sensors are, the more correlated measurements they produces. (b) Scattered plots of the first three aPC basis evaluations. There is no correlation between different aPC basis functions.

tic energy after performing PCA. The solution $u(x;\omega)$ is approximated with a first-order aPC expansion. Figure 4.4 shows the scattered plots of the measurements from the first three f-sensors and the first three arbitrary polynomial basis. It is evident that the raw data from the measurements are correlated while their induced polynomials are not, thus the induced polynomials would serve as a valid set of basis in the random space.

The DNNs used to approximate the modes of u are constructed as in Figure 4.2, where we use an isolated small scale DNN of 2 hidden layers with 4 neurons per hidden layer to approximate the mean profile, and a DNN of 4 hidden layers with 32 neurons per hidden layer to model the modes. The *tanh* function is selected as the default activation function due to it is second order differentiable. Then, a DNN for the residual can be constructed via auto-differentiation and arithmetic operations. The training set $\mathcal{S}_t$ is

$$\mathcal{S}_t = \left\{\{(-1,0),(1,0)\},\{(x_f^{(i)},0\}_{i=1}^{N_f}\right\}_{s=1}^{N},$$

where the u data is collected only at the boundaries to provide boundary conditions. The loss function is slightly modified based on Eq. 4.16 to add a l_2 regularization term. At the training stage, we choose the l_2 regularization rate $\lambda = 0.001$, and use the Adam [35] optimizer with learning rate 0.001 to train our model for 20000 epochs. Figure 4.5 shows the predicted mean and standard deviation of the solution u versus the reference. Figure 4.6 shows our DNN prediction of three u modes where the reference modes are calculated by Eq. 4.13. We can see that the NN-aPC method makes accurate predictions of the mean and standard deviation of the solution $u(x;\omega)$ and learns the arbitrary polynomial chaos modes.

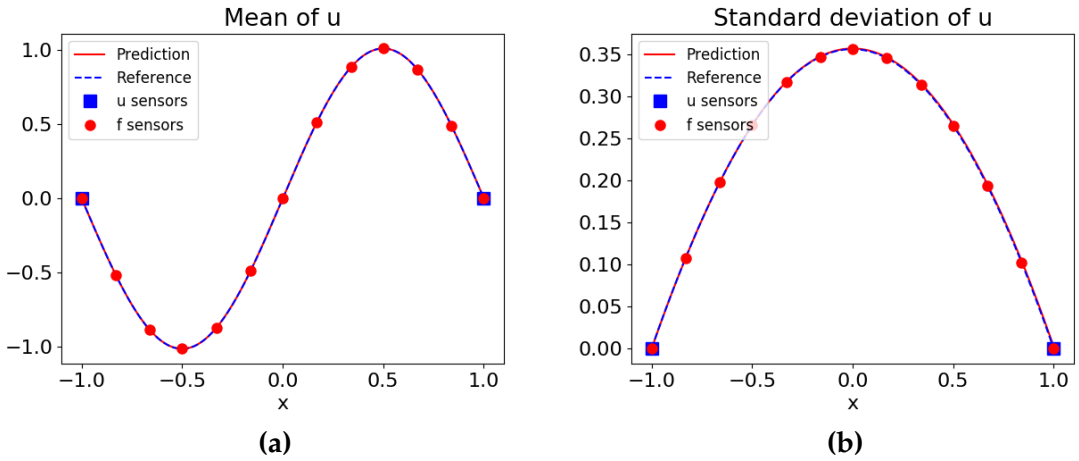


Figure 4.5: Forward problem: The mean function (a) and the standard deviation (b) of u predicted using the trained DNN. The reference is calculated from the 1000 snapshots of continuous u samples. In this case, 13 f-sensors are employed, while only 2 u-sensors are placed at the domain boundaries.

The trained model is then used to predict the QoI at any location x_o given new snapshots of sensor data, with only one forward evaluation of the DNN with x_o as the input, as described in Section 4.3.1. Figure 4.7a illustrates the prediction of solution for three different snapshots of the sensor data in the test samples; our prediction recovers the true solution very well. In Figure 4.7b, we use an increasing value of N_f to study the effect of the number of f-sensors on the accuracy of the predicted solutions; we observe that when more f-sensors are deployed, we can

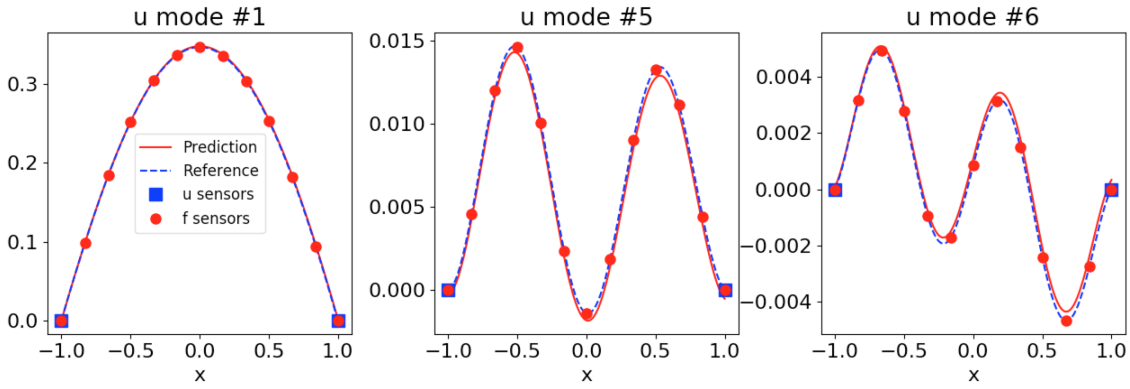


Figure 4.6: Forward problem: Plots of three (out of six) aPC modes of u versus the reference, which is calculated from Eq. 4.13 using the 1000 continuous u samples. The predicted modes match the true modes closely.

achieve better accuracy.

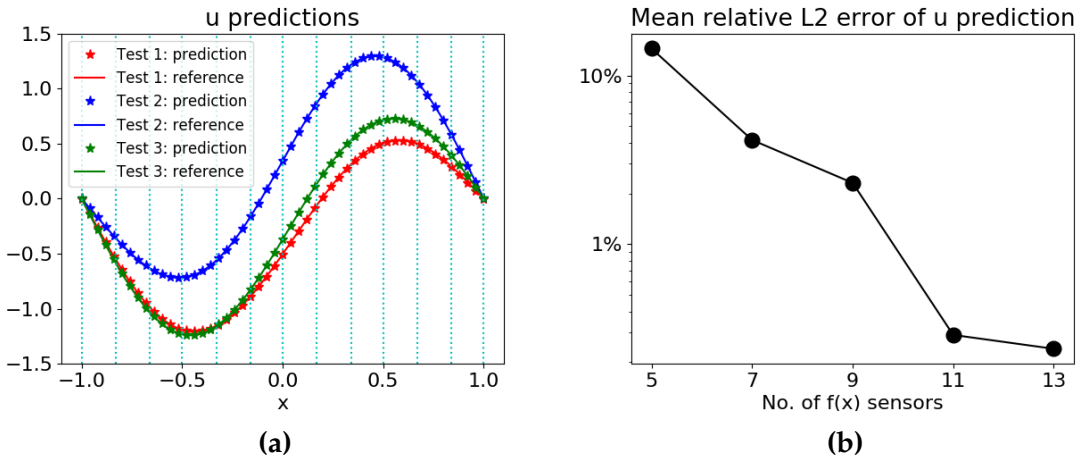


Figure 4.7: Making predictions for the forward problem using the NN-aPC: (a) For three different snapshots in the test data set, we compare the predicted solution u, calculated using the measurements of 13 f-sensors whose locations are marked with the dashed lines, versus the true u. (b) Relative L_2 error of the predicted u, averaged for all snapshots in the test data set, versus the number of f-sensors placed in the domain.

Inverse Problem: Stochastic Elliptic Equation

We solve the one-dimensional stochastic elliptic equation as an inverse problem, where we have some extra information on the solution $u(x;\omega)$ but incomplete

information of the diffusion coefficient $k(x;\omega)$. The equation reads

$$\begin{aligned} -\frac{\mathrm{d}}{\mathrm{d}x}\left(k(x;\omega)\frac{\mathrm{d}}{\mathrm{d}x}u\right) = f(x), \quad x \in [-1,1] \text{ and } \omega \in \Omega, \\ u(-1) = u(1) = 0. \end{aligned} \tag{4.24}$$

In this example, we use a constant forcing term $f(x) = 10$. The randomness comes from the diffusion coefficient $k(x;\omega)$, for which we only have limited information at the locations where we place the k-sensors. Here in this example, k is modeled and sampled from a non-Gaussian random process such that

$$\log(k(x;\omega)) \sim \mathcal{GP}(k_0(x), \mathrm{Cov}(x,x')), \tag{4.25}$$

where the mean $k_0(x) = -\cos(3\pi x/2)/5$ and the covariance function has the same form as in Eq. 4.23, where we set the standard deviation $\sigma = 0.1$ and correlation length $l_c = 1.0$. We use the same strategy as in Section 4.4.1 to generate the training and testing samples, and the training set is constructed as in Eq. 4.2.

For this case, two groups of DNNs, i.e., $\widehat{k_i}$ and $\widehat{u_\alpha}$ are built to calculate the modes for k and u. Again, we use the Adam optimizer with learning rate 0.001 to train the DNNs for 50000 epochs. In Figure 4.8a, we use the 1st-order aPC expansion and we study the impact of using different l_2 regularization rate λ and different shapes of DNNs. We only change the DNNs that learn the stochastic modes, while the DNNs that learn the mean profiles are fixed to have 2 hidden layers and 4 neurons per hidden layer; this is the default setting for the future examples as well. In these plots, we compare the averaged relative L_2 error of predicting k and u in the test set. The results indicate that a moderate choice of λ (0.0005) gives us the most accurate predictions, since a too small/large choice of λ causes over-/under- fitting. A suitable choice of DNN shape (4 hidden layers with

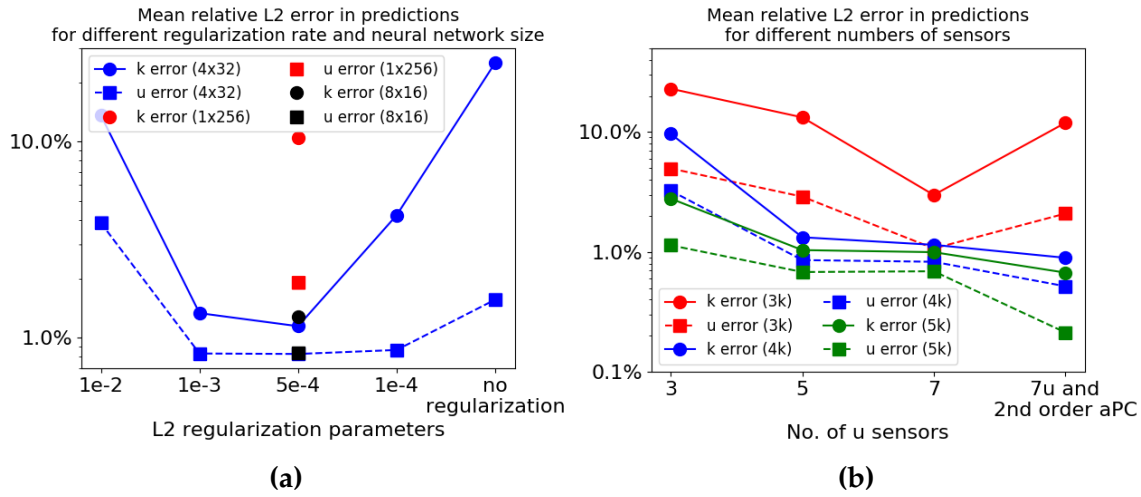


Figure 4.8: Comparing prediction accuracy for the inverse problem using the NN-aPC: (a) The mean of relative L_2 error in predicting u and k trajectories in the test set when using different sizes of DNNs and different l_2 regularization rate. In this case, we use 1st-order aPC expansion 4 k-sensors and 7 u-sensors are deployed and both DNNs have the same size. (b) The mean relative L_2 error in predicting u- and k-trajectories versus the number of sensors deployed. In this case we use the 1st-order aPC expansion, choose $\lambda = 0.0005$, and the DNNs have 4 hidden layers with 32 neurons per hidden layer.

32 neurons per hidden layer) produces the best trained model. For the rest of this numerical example, we shall adopt these optimal DNN setting.

In Figure 4.8b, we use the 1st-order aPC expansion and compare the averaged relative L_2 error in predictions when different numbers of k- and u-sensors are deployed to collect the training data. In general, the proposed method makes more accurate predictions after training with data collected from more k- and u-sensors. One reason is that a larger number of sensors supports a greater variety of input x in the training data, thus feeding more information to the model to reduce the probability of over-fitting. Also, more k-sensors allows for a higher effective random dimension of the aPC expansion for better approximation. Figure 4.8b also shows that a 2nd-order aPC expansion helps to improve predictions. We note that this is not the case when we use only three k-sensors. The bottleneck here is insufficient random dimension, so without enough training information, adopting the 2nd-order aPC expansion doubles the number of $\widehat{u_\alpha}$ net outputs and would

only increase the risk of over-fitting.

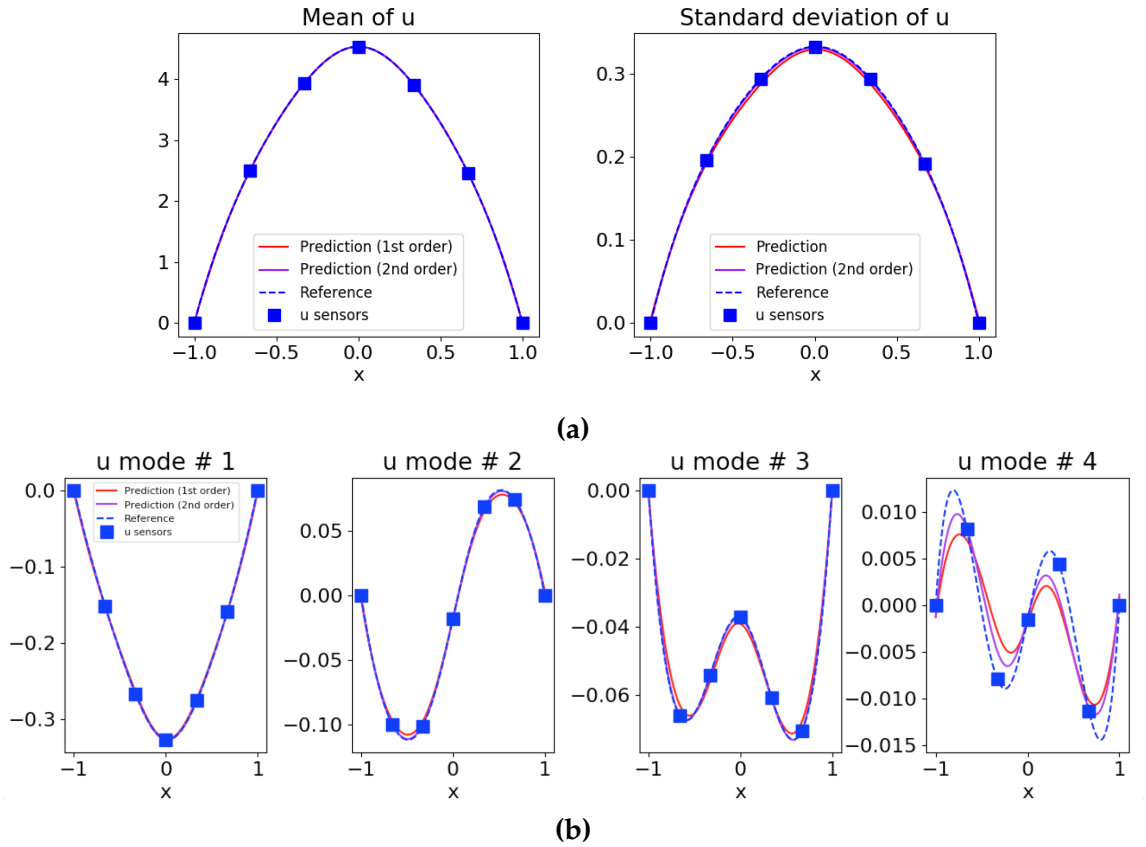


Figure 4.9: Testing 2nd-order aPC expansions: (a) The predicted mean/standard deviation of u calculated with a 1st- and 2nd-order aPC expansions versus the reference. (b) The first 4 modes of u calculated by the 1st- and 2nd-order aPC expansion. The reference solutions in all plots are calculated with the continuous trajectories that produced the training data. The results are generated with 7 u-sensors (blue squares) and 4 k-sensors (red dots in Figure 4.10).

We use a combination of 7 u-sensors and 4 k-sensors. Figure 4.9a and Figure 4.10a compare the mean and standard deviation of u and k calculated by the trained DNNs when we use the 1st- and 2nd-order aPC expansions. Figure 4.9b shows the first four common aPC modes of u for both expansions, where we can see that the 2nd-order aPC expansion helps to improve the accuracy of the predicted lower-order modes. The 2nd-order aPC expansion would also help with learning the k modes, as depicted in Figure 4.10b, even if the $\widehat{k_i}$ network is disjoint with the $\widehat{u_\alpha}$ network. Table 4.1 lists the relative L_2 error of the trained model in calculating the mean, standard deviation and all the common modes of k and u.

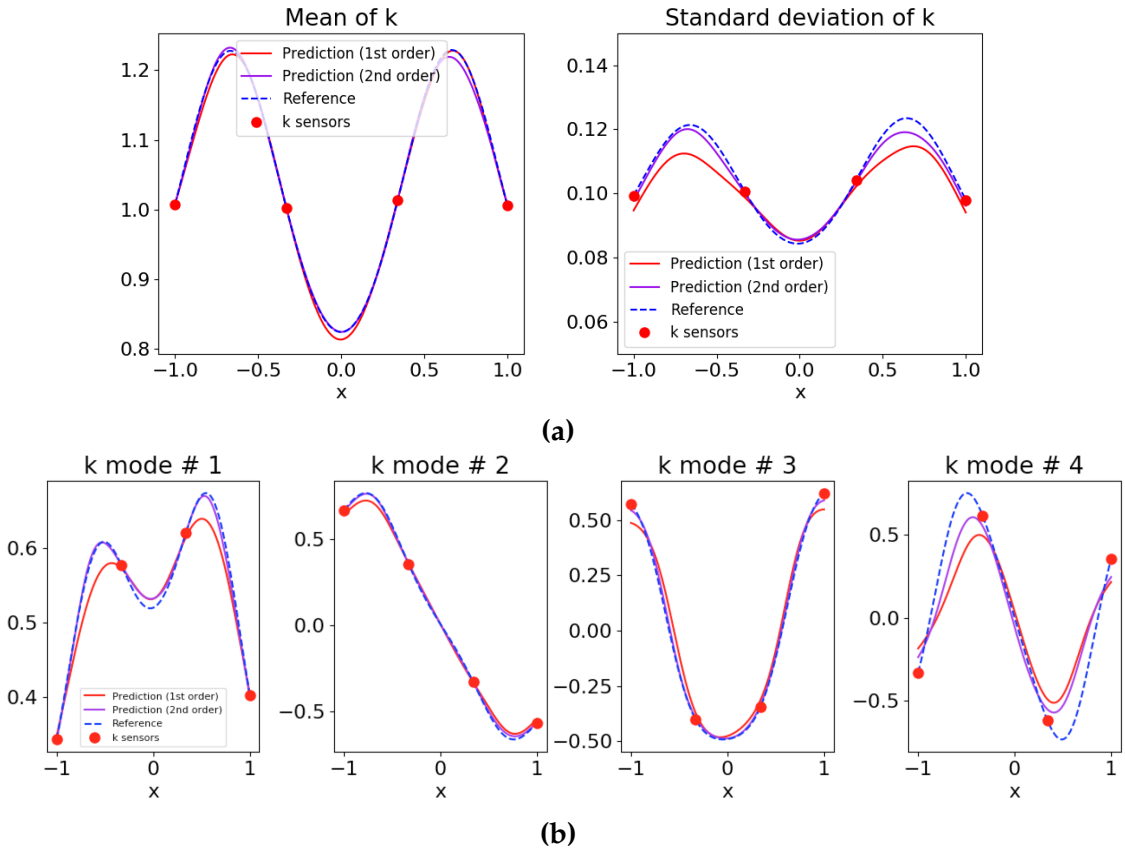


Figure 4.10: Testing 2nd-order aPC expansions: (a) The predicted mean/standard deviation of k calculated with a 1st- and 2nd-order aPC expansions versus the reference. (b) The first 4 modes of k calculated by the 1st- and 2nd-order aPC expansion. The reference in all plots are calculated with the continuous trajectories that produced the training data. The results are generated with the same setup of sensors as that of Figure 4.9.

		mean	std	mode 1	mode 2	mode 3	mode 4
k	**1st-order**	0.54%	5.26%	4.15%	5.39%	12.03%	42.81%
	2nd-order	0.45%	1.87%	1.28%	1.95%	3.67%	29.95%
u	**1st-order**	0.14%	1.83%	0.98%	3.60%	4.34%	45.56%
	2nd-order	0.14%	0.51%	0.08%	0.80%	1.04%	32.16%

Table 4.1: Comparing the relative L_2 error when using the 1st- and 2nd-order aPC expansion, and using data from 4 k-sensors and 7 u-sensors for training.

We can see that using a 2nd-order aPC significantly improves the accuracy. We also note that in Figure 4.10a, due to the placement of k-sensors, the measurements from each sensor will yield almost the same mean (1.0) and standard deviation (0.1), but the trained k net would reveal the non-trivial wavy structure of its mean and standard deviation in the entire domain, containing information more than just the training data could provide. This indicates that there is information fusion of all three types of training data and the stochastic differential equation, rather than a simple interpolation. Figure 4.11 shows that the higher-order aPC modes can also be captured even if they exist in a much smaller scale compared to the lower-order more energetic modes.

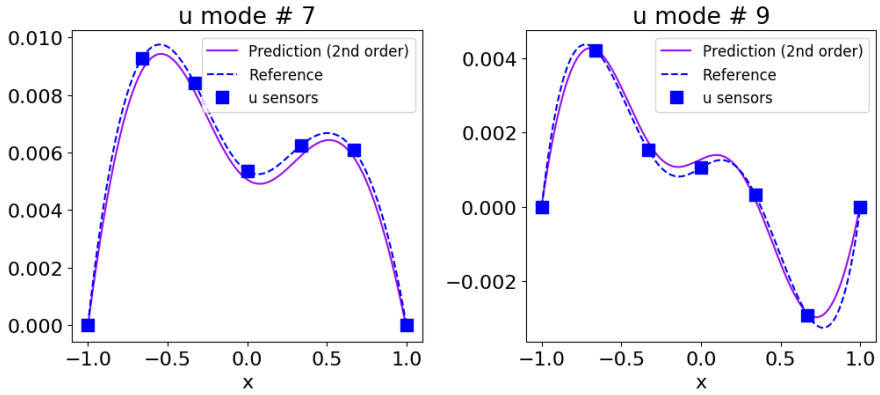


Figure 4.11: Testing 2nd-order aPC expansions: Two u-modes of higher order calculated with the 2nd-order aPC expansion. The magnitude of the higher-order modes is smaller compared to the lower-order modes, but the NN-aPC method is still able to capture the small magnitude modes. The results are generated with the same setup of sensors as that of Figure 4.9.

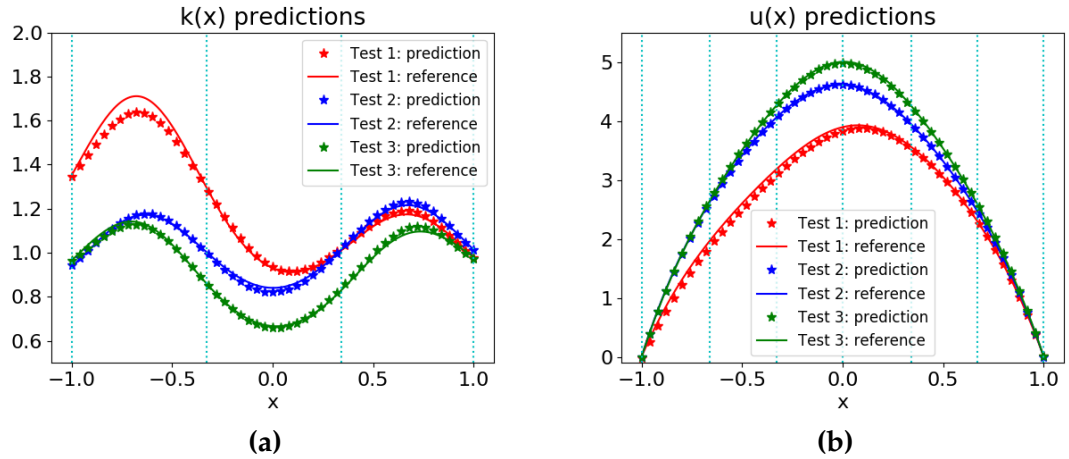


Figure 4.12: Making predictions for the inverse problem using the NN-aPC: For three different snapshots in the test data set, we compare the predicted solution k (in plot (a)) and u (in plot (b)), calculated using the measurements of 7 u-sensors and 4 k-sensors (denoted by the dashed lines).

Figure 4.12 shows the prediction of k and u for three arbitrary snapshots in the test sample set. Every pair of predictions are made based on a single time measurement from 4 k- and 7 u-sensors, and they agree with the true reference value. Again, the predicting stage takes little computational time since only one forward evaluation of the well-trained DNNs is needed for every input x, no matter how many snapshots of predictions are to be made.

4.4.2 Combining PINNs and Dropout

In this part, we first show that dropout reduces over-fitting in solving differential equations. Then, more importantly, we show that the dropout-induced uncertainty serves as useful guidance for active learning.

Reducing Over-fitting

To show how dropout reduces over-fitting, we implement the dropout neural networks to solve both a forward Poisson's equation and an inverse elliptic equation. They are the deterministic version of Eq. 4.22 and Eq. 4.24. For the forward Poisson's equation, we choose $f(x) = 9\pi^2 \sin(3\pi x/2)/4$ as the forcing term. A dropout neural network with 6 hidden layers and 100 neurons per hidden layer is constructed to model the solution $u(x)$. The training data consists of 2 u measurements at both boundaries and 6 f-sensors in the domain. For the inverse elliptic equation, we choose, as before, $f(x) = 10$ and $k(x) = \exp(\sin(3\pi x/2)/5)$ as the hidden diffusion coefficient, and we use 5 k-sensors and 7 u-sensors. Two separate DNNs are constructed: a small scale regular DNN that has 2 hidden layers with 4 neurons per hidden layer to model the function $u(x)$, and a large scale dropout neural network with 6 hidden layers and 100 neurons per hidden layer to model the function $k(x)$. The dropout rate in both examples is fixed at 0.01.

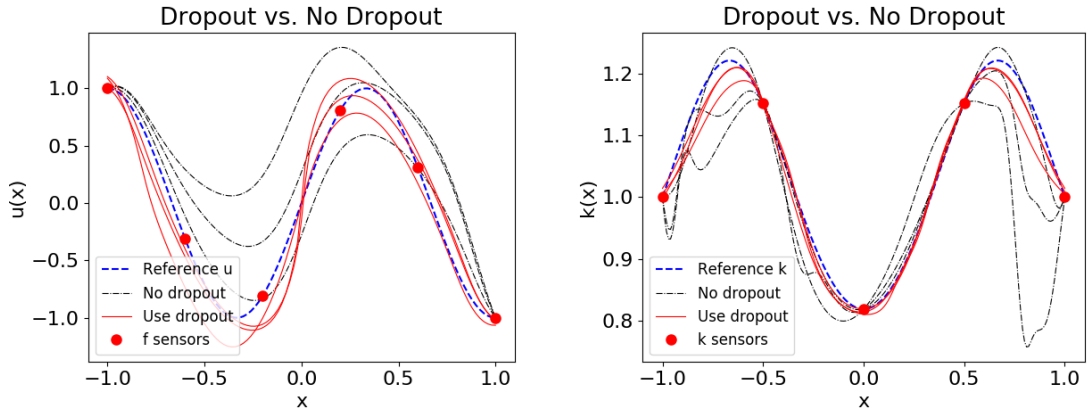


Figure 4.13: Dropout to reduce over-fitting: A comparison of the predicted QoI using the PINN/dropout and the regular PINN. For each case, three independent runs are conducted. (a) Forward problem: we solve the Poisson's equation with 6 f-sensors (red dots) and 2 u-sensors (at the domain boundary, not shown in the plot). (b) Inverse problem: we solve an elliptic equation with 5 k-sensors (red dots) and 7 u-sensors (equidistantly placed in the domain, not shown in the plot).

For both the forward and inverse problem, we train the dropout DNNs, using an Adam optimizer with learning rate 0.001 for 30000 epochs. Due to the lack of sufficient number of sensors, there is a big chance that over-fitting occurs at the training stage if we use just regular DNNs. Figure 4.13 shows a comparison of the results when we train the networks with and without using dropout. As we can see in the plots, the results from a regular DNN are very different from each other, showing irregular jumps of large amplitudes, while the results from dropout DNNs are similar to each other, and they are closer to the truth. This shows that dropout works as an effective means of reducing over-fitting.

Estimating the Approximation Uncertainty of DNNs

More importantly, the uncertainty introduced by dropout serves as a useful guidance for active learning. Again, we solve the same inverse elliptic equation but this time we are provided with additional k-sensors. At the training stage, we add an l_2 regularization term with $\lambda = 10^{-6}$ to the loss function. At the predicting stage, we evaluate the dropout network for 10000 times to estimate the mean and standard deviation. Next, a new k-sensor is placed where the standard deviation reaches its maximum. If it happens that the location to add the new sensor is close to an existing k-sensor by a threshold distance of ρ, we do not add the new sensor, but instead, we count the nearest existing sensor twice as if we have added a virtual sensor at the same location of the existing one. In practice, here we choose $\rho = 0.03$. We have designed an automatic iterative procedure for active learning that reduces the cost of adding new sensors by efficiently re-using the old ones.

Figure 4.14 shows the initial prediction of k and the prediction after iterating the above algorithm for 15 steps. In Figure 4.14b, the sensors are clustered where

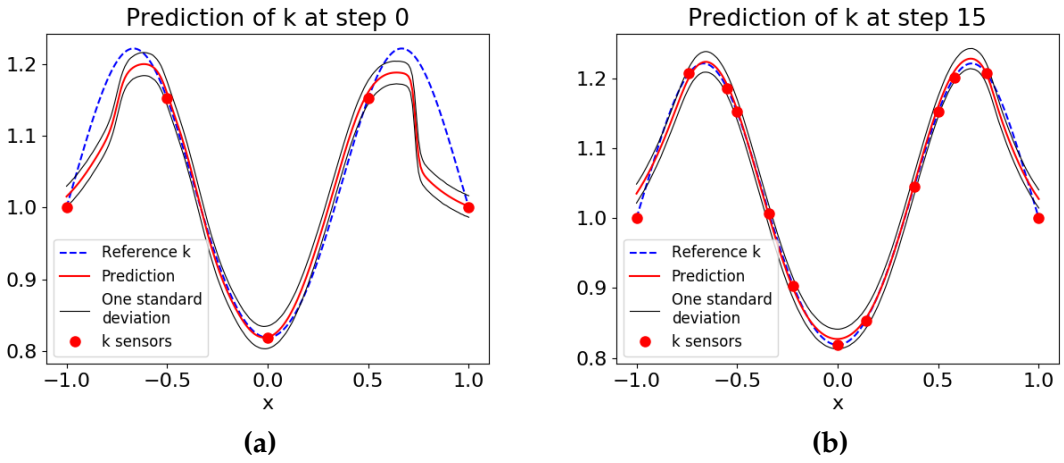


Figure 4.14: Active learning using dropout approximation uncertainty: (a) The prediction of k and its dropout-induced uncertainty of the starting step with 5 k-sensors. (b) The prediction of k and its dropout-induced uncertainty of the last step with 13 k-sensors.

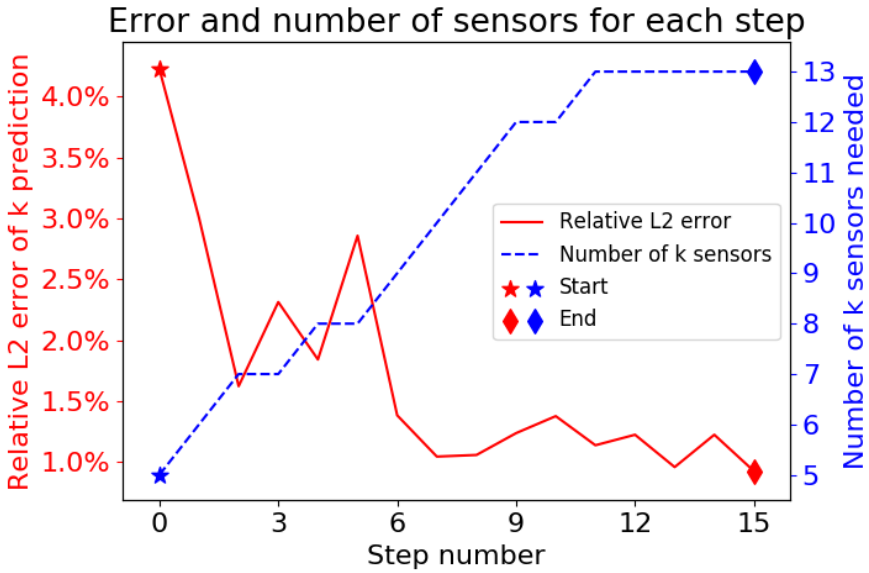


Figure 4.15: Effectiveness of active learning: The red solid line shows the relative L_2 error of the predicted k revealing a decaying trend. The blue dashed line shows the number of k-sensors deployed in each step.

the curvature of k is big, which is consistent with our intuition that we should put more sensors where the function changes rapidly. In Figure 4.15, we see that during these 15 iterations, only 8 new sensors are deployed while the relative error of k predictions reduces from more than 5% to less than 1%.

4.4.3 Active Learning for Inverse Stochastic Elliptic Problems

We consider the inverse stochastic elliptic problem as described in Eq. 4.24 for active learning, but in this example, $\log(k(x;\omega))$ is modeled by a Gaussian random process with correlation length $l_c = 0.5$. To start with, we have 1000 snapshots of data from 3 k-sensors, 7 u-sensors and 21 f-sensors that are equidistantly distributed in the physical domain, and our goal is to infer $k(x;\omega)$ in the entire domain. Suppose we are then provided with additional sensors of k; we shall allocate them according to the uncertainty induced by the dropout neural networks. In practice, we model the modes of $k(x;\omega)$ with a dropout neural network that has 4 hidden layers with 128 neurons per hidden layer, and a dropout rate 0.01. The solution $u(x;\omega)$ is expanded with the 1st-order aPC expansion, while the modes are modeled by a regular DNN with 4 layers and 32 neurons per hidden layer. The reasons for implementing dropout only on $\widehat{k_i}$ are as follows:

1. Dropout can be used to efficiently reduce over-fitting, and the over-fitting issue of $k(x,\omega)$ is worse than that of $u(x,\omega)$.

2. As an inverse problem, our main goal is to identify $k(x,\omega)$ and then identify where we should add more k-sensors to enhance the accuracy of prediction.

We use an Adam optimizer with learning rate 0.0005 to train the networks for 50000

epochs. The mean and standard deviation of the k modes model are evaluated from 10000 independent evaluations of the dropout neural network. We place the additional k-sensor where the standard deviation of the first mode attains its maximum. This is because the first mode is associated with the largest eigenvalue, therefore bringing the largest impact to the stochastic structure, thus it is most important to learn the first mode accurately.

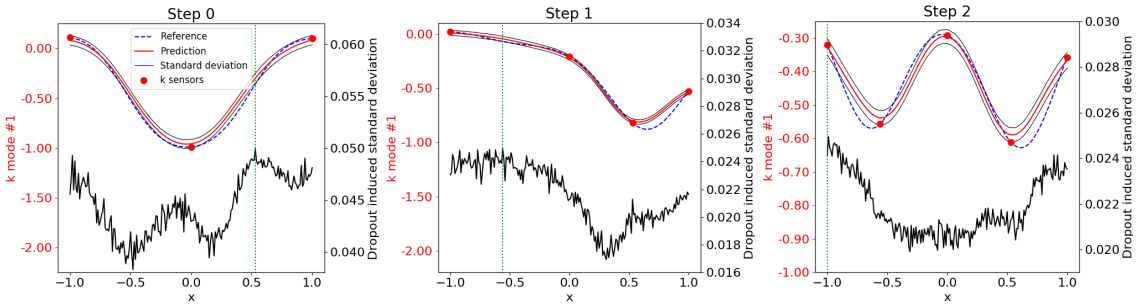


Figure 4.16: Active learning for stochastic inverse problems: The first modes of k in the three steps and their associated standard deviation induced from the dropout neural network. The green dashed line indicates the location where the standard deviation reaches its maximum, and where the new sensor will be added in the next step.

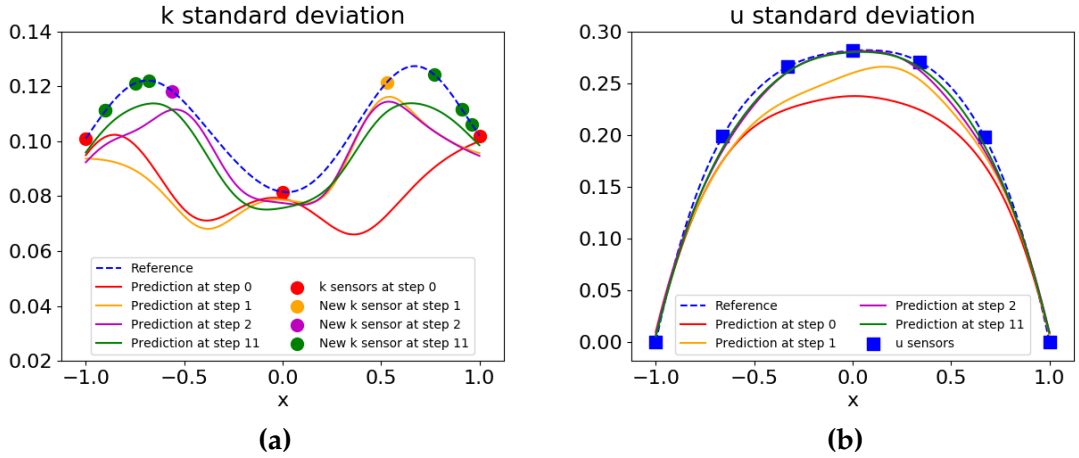


Figure 4.17: Active learning for stochastic inverse problems: The predicted standard deviation of k (in plot (a)) and u (in plot (b)) in the first three and the last steps. In plot (a) the k-sensors at step 0 are colored in red and the newly added k-sensors are denoted in different colors. The u-sensors (as depicted in plot(b)) are kept the same. The reference solutions are calculated using the continuous trajectories that generate the training data.

We carry out the active learning steps until the k prediction error does not decrease after a new sensor is added. To better illustrate the process of adding

new sensors, in Figure 4.16a–4.16c we show the learned first mode of k and its associated standard deviation from dropout uncertainty. Three steps of active learning are displayed. Note that the shapes of the first modes do not stay the same due to the fact that every time a new k-sensor is added, the principal components of K in Eq. 4.4 are therefore changing. Figure 4.17a and Figure 4.17b show the comparison of the predicted standard deviation of k and u in the first three steps and the last step, respectively. It is evident that adding extra k-sensors automatically according to the dropout-induced uncertainty will improve the accuracy of standard deviation prediction for both k and u. Finally, the trained model is used to predict continuous trajectories of k and u in the test samples. We can conclude from Table 4.2 that adding extra k-sensors based on active learning with the dropout helps us to make better predictions.

	k **mean**	k **std**	k **prediction**	u **mean**	u **std**	u **prediction**
Step 0	0.87%	26.57%	6.07%	0.18%	15.23%	3.06%
Step 1	2.79%	20.19%	5.29%	0.24%	9.33%	2.41%
Step 2	1.48%	10.21%	2.58%	0.14%	3.89%	1.08%
Step 11	0.46%	8.49%	2.01%	0.04%	2.93%	0.67%

Table 4.2: Comparison of the relative L_2 error at different steps

4.5 Summary of Chapter

We have presented a new approach to quantify the parametric uncertainty in the Physics-Informed Neural Networks (PINNs) by employing the arbitrary polynomial chaos (aPC) expansion to represent the stochastic solution. We use the data collected from sensor measurements to build a set of arbitrary polynomial basis and learn the modal functions of the aPC expansion through the PINNs, i.e., DNNs

that encode the underlying stochastic differential equation. The proposed data-driven method can be used to solve forward problems, but more importantly, it deals with stochastic *inverse* problems. In the classical inverse problem, typically all the information available is for the solution, and we aim to identify the parameters. Here we selected to solve the inverse problems, where we have partial information available both for the solution and the parameter, which is a stochastic process. Once the model is trained with existing sensor data, i.e., historical data, it can be used to predict new instances of trajectories of the quantity of interest (solution or parameter) at very small additional computational cost.

We aim at quantifying two different types of uncertainties, i.e., the *parametric* uncertainty due to the stochastic equation, as well as the *approximation* uncertainty of the PINN. The latter represents how well the PINN is trained and how robust it is at the predicting stage. To this end, we adopt the dropout strategy for estimating the approximation uncertainty. Dropout is typically used to deal with over-fitting problems but it can also be exploited to quantify the approximation uncertainty at no additional cost. In our examples, we use DNNs to learn the modal functions of the stochastic parametric modes and the dropout strategy to quantify their associated uncertainty. Based on this, we propose an iterative method of actively learning where to place new sensors to enhance the approximation accuracy of PINNs. The numerical results exhibit the effectiveness of such an *active learning* strategy, not only in placing new sensors but also in making better use of the existing ones.

There are other possible methods of quantifying parametric and approximation uncertainties. For example, we can abandon aPC and use directly the stochastic data as the input. One possible approach is to consider the random space together with the physical space. Hence, standard DNNs, and in particular the determinis-

tic PINNs developed in [62, 63] can be directly used to solve stochastic differential equations. However, we did not choose this approach for two reasons: first, it does not lead to explicit expressions for stochasticity of the quantity of interest (QoI); and second, different from sampling in the physical space where we can always mark the location by their coordinates, marking random instances with random variables is much harder. Moreover, the dimension of the random space is usually much higher than that of the physical space, requiring a proper dimension reduction procedure to be carried out before actually solving the problem. Although weaker, the high dimensionality is also a limitation of the aPC approach because it requires the high-dimensional DNN outputs, which could make the training process harder.

Chapter Five

Solving Time-dependent Stochastic Problems with Physics-Informed Neural Networks

5.1 Introduction

In the past decades, there has been an increasing demand in quantifying uncertainty propagation in complex systems, among which a more interesting and challenging quest is to model the uncertainty propagation and evolution as time grows. Such problems can be described by time-dependent stochastic partial differential equations (SPDEs) with random inputs, which is usually modeled by including randomness in physical parameters, initial and/or boundary conditions, random excitation, etc.. Recently, the Dynamically Orthogonal (DO) [69] was proposed to depict the dynamics of the low-dimensional random structure in stochastic systems. The DO method has been successfully applied in several applications including fluid mechanics [80, 76]. A rigorous and sharp error bounds of DO method was first given by Zhou, et al. in [46], where it was shown the DO modes can capture the effective directions. In 2013, Cheng and et al. developed the Bi-Orthogonal (BO) [7, 8] method to reach the same goal. Both the DO and BO methods are created from the same starting point, i.e. a generalized Karhunen-Loève expansion, but they utilized different constraints to remove the time redundancy.

The DO and BO formulations are mathematically equivalent [12], but they exhibit computationally complimentary properties. Specifically, the BO formulation may fail due to crossing of the eigenvalues of the covariance matrix [12], while both BO and DO become unstable when there is a high condition number of the covariance matrix or zero eigenvalues. The authors in [80] suggested to use a pseudo-inverse technique to invert the singular covariance matrix of the DO and BO formulation for SPDEs with deterministic initial conditions. It was shown in [12] that numerically BO performs better than DO when it is absent

of eigenvalue crossing, thus a robust hybrid pseudo-inverse BO/DO method was proposed in [3] for stochastic flows.

The recently developed Physics-Informed Neural Networks (PINNs) have shown their power to efficiently solve both the deterministic differential equations and inverse problems [38, 39, 62, 63]. In the previous chapter, we combined the arbitrary polynomial chaos and PINNs for solving both forward and inverse time-independent SDEs. In this chapter, we introduce a new strategy to solve the time-dependent SPDEs by employing PINNs within the DO/BO framework. Concretely, we use DNNs to learn both the spatial and stochastic bases, where the loss function are formulated in the weak form of the corresponding SPDEs. The merits of the proposed methods are:

- There is no assumption about the covariance matrix of the random coefficients. The proposed method can be readily used for solving SPDEs with deterministic initial conditions;
- This method works equally well when eigenvalue crossing exists in the given time domain;
- It shall be seamlessly applied to solving time-dependent stochastic *inverse* problems.

The organization of this chapter is as follows. In Section 5.2, we set up the time-dependent stochastic problems. In Section 5.3, we give an brief review of the DO and BO methods. In Section 5.4, we formulate our NN-DO/BO framework for solving time-dependent SPDEs. In Section 5.5, we provide a detailed study of the accuracy and performance of the NN-DO/BO approach with two benchmark cases that are specially designed to have exact solution for the DO and BO

representations. Finally, we conclude with a brief discussion in Section 5.6.

5.2 Problem Setup

Let $(\Omega, \mathcal{F}, P)$ be a probability space, where Ω is the sample space, $\mathcal{F}$ is the σ-algebra of subsets of Ω, and P is a probability measure. Let $\mathcal{D}$ be a bounded domain in $\mathbb{R}^d$ ($d = 1, 2$, or 3) whose boundary is denoted by $\partial\mathcal{D}$, and $[0, T]$ be the time domain of interest. We consider the following time-dependent SPDE:

$$\frac{\partial u}{\partial t} = \mathcal{N}_x[u(x,t;\omega)], \quad x \in \mathcal{D},\ t \in [0,T],\ \omega \in \Omega, \tag{5.1}$$

with initial and boundary conditions:

$$u(x,t;\omega) = u_0(x;\omega), \qquad t = t_0, \tag{5.2}$$

$$\mathcal{B}_x[u(x,t;\omega)] = h(x,t;\omega), \qquad x \in \partial\mathcal{D}. \tag{5.3}$$

where $\mathcal{N}_x$ is a differential operator and $\mathcal{B}_x$ is a linear differential operator acting on the domain boundary. Assuming that our quantity of interest, $u(x,t;\omega)$, is a second-order random field. The initial and boundary conditions for Eq. 5.1 are denoted by $u_0(x;\omega)$ and $h(t,x;\omega)$. Our target is to solve Eq. 5.1, and specifically, evaluating the mean and standard deviation of the solution $u(x,t;\omega)$.

5.3 An Overview of the DO and BO Approaches

For random fields $u(x,t;\omega)$ that involves the time evolution, the Karhunen-Loève (KL) expansion at a given time t writes

$$u(x,t;\omega) = \bar{u}(x,t) + \sum_{i=1}^{\infty} \sqrt{\lambda_i}\phi_i(x,t)\xi_i(t;\omega), \quad \omega \in \Omega, \tag{5.4}$$

where $\bar{u}$ is the mean, λ_i and ϕ_i are the i-th largest eigenvalue and the corresponding eigenfunction of the covariance kernel, i,e., they solve the following eigen problem:

$$\int_{\mathcal{D}} C_{u(x_1,t)u(x_2,t)}\phi_i(x_2,t)\mathrm{d}x_2 = \lambda_i\phi_i(x_1,t). \tag{5.5}$$

Here $C_{u(x_1,t)u(x_2,t)} = \mathbb{E}\left[(u(x_1,t;\omega) - \bar{u}(x_1,t))(u(x_2,t;\omega) - \bar{u}(x_2,t))\right]$ is the covariance kernel of u.

Let's consider a generalized expansion:

$$u(x,t;\omega) = \bar{u}(x,t) + \sum_{i=1}^{\infty} u_i(x,t)Y_i(t;\omega), \quad \omega \in \Omega. \tag{5.6}$$

Similar to the KL expansion, the random field $u(x,t;\omega)$ is decomposed into two parts: (i) the deterministic mean field function $\bar{u}(x,t)$, and (ii) the random fluctuation part consists of an infinite summation of deterministic orthogonal fields $u_i(x,t)$ with 0-mean stochastic coefficients $Y_i(t;\omega)$. Formally, we have

$$\bar{u}(x,t) = \mathbb{E}[u(x,t;\omega)] = \int_{\Omega} u(x,t;\omega)\,\mathrm{d}P(\omega)\,, \tag{5.7}$$

$$\langle u_i, u_j\rangle = 0, \quad \text{for } i \neq j \text{ and } i,j = 1,2,\ldots, \tag{5.8}$$

and

$$\mathbb{E}[Y_i] = 0, \quad \text{for } i = 1, 2, \ldots. \tag{5.9}$$

We define the linear subspace $V_S = \text{span}\,\{u_i(x,t)\}_{i=1}^N$ as the linear space spanned by the first N deterministic bases. For now lets assume $Y_i(t;\omega)$ are linearly independent and $\Omega_S = \text{span}\,\{Y_i(t;\omega)\}_{i=1}^N$ is the linear subspace in Ω spanned by the first N stochastic coefficients. The truncated expansion $u_N(x,t;\omega)$, defined by

$$u_N(x,t;\omega) = \bar{u}(x,t) + \sum_{i=1}^{N} u_i(x,t) Y_i(t;\omega), \tag{5.10}$$

is the projection of $u(x,t;\omega)$ to the subspace $V_S \times \Omega_S$. Without making any assumptions on their form, the governing equations Eq. 5.1 and Eq. 5.7–5.9 are the only information can be utilized to derive the evolution equations of u_i and Y_i. Note that both the stochastic coefficients $Y_i(t;\omega)$ and the orthogonal bases $u_i(x,t)$ are time-dependent (and they are evolving according to the system dynamics), unlike the standard polynomial chaos where the stochastic coefficients are time-independent. There exists some redundancy in the Eq. 5.10, and therefore, additional constraints need to be imposed in order to formulate a well posed problem for the unknown quantities. Here we review the DO and BO approaches respectively, which have different assumptions on the constraints.

5.3.1 Dynamically Orthogonal Representation

As first proposed in [69], a natural constraint to overcome redundancy is that the evolution of the bases $\{u_i(x,t)\}_{i=1}^N$ be orthogonal to the space V_S; this can be

expressed through the following Dynamically Orthogonal (DO) condition:

$$\frac{dV_S}{dt} \perp V_S \Longleftrightarrow \left\langle \frac{\partial u_i(x,t)}{\partial t}, u_j(x,t) \right\rangle = 0 \quad i,j = 1,\dots,N. \tag{5.11}$$

Comparing Eq. 5.10 with the classical KL expansion Eq. 5.4, in the DO representation, we set u_i to have unit length and Y_i carry the scaling coefficient as the result of the eigenvalues. Note that the DO condition preserves the orthonormality and the length of the bases $\{u_i(x,t)\}_{i=1}^N$ since

$$\frac{\partial}{\partial t}\left\langle u_i(\cdot,t), u_j(\cdot,t)\right\rangle = \left\langle \frac{\partial u_i(\cdot,t)}{\partial t}, u_j(\cdot,t)\right\rangle + \left\langle u_i(\cdot,t), \frac{\partial u_j(\cdot,t)}{\partial t}\right\rangle = 0, \quad i,j = 1,\dots,N. \tag{5.12}$$

It is proved in [69] that the DO condition leads to a set of independent and explicit evolution equations for all the unknown quantities. Here we state the DO evolution equations without proof:

Theorem 5.3.1 (see [69])**.** *Under the assumptions of the DO representation, the original SPDE (Eq. 5.1) is reduced to the following system of equations:*

$$\begin{aligned}
&\frac{\partial \bar{u}(t,x)}{\partial t} = \mathbb{E}[\mathcal{N}_x[u(\cdot,t;\omega)]],\\
&\frac{\mathrm{d}Y_i(t;\omega)}{\mathrm{d}t} = \langle \mathcal{N}_x[u(\cdot,t;\omega)] - \mathbb{E}[\mathcal{N}_x[u(\cdot,t;\omega)], u_i(\cdot,t)\rangle, \quad i = 1,\dots,N,\\
&\sum_{i=1}^{N} C_{Y_i(t)Y_j(t)} \frac{\partial u_i(t,x)}{\partial t} = \prod_{V_S^\perp} \mathbb{E}\left[\mathcal{N}_x\left[u(\cdot,t;\omega)\right] Y_j\right], \qquad j = 1,\dots,N.
\end{aligned} \tag{5.13}$$

The projection in the orthogonal complement of the linear subspace V_S is defined as $\prod_{V_S^\perp} F(x) = F(x) - \prod_{V_s} F(x) = F(x) - \sum_{k=1}^{N} \langle F(\cdot), u_k(\cdot,t)\rangle\, u_k(\cdot,t)$, and the covariance of the stochastic coefficients is $C_{Y_i(t)Y_j(t)} = E\left[Y_i(t;\omega)Y_j(t;\omega)\right]$. The associated

boundary conditions are determined by

$$\begin{aligned} \mathcal{B}_x\left[\bar{u}(x,t;\omega)\right]|_{x\in\partial\mathcal{D}} &= \mathbb{E}[h(x,t;\omega)], \\ \mathcal{B}_x\left[u_i(x,t)\right]|_{x\in\partial\mathcal{D}} &= \mathbb{E}\left[Y_j(t;\omega)h(x,t;\omega)\right]C^{-1}_{Y_i(t)Y_j(t)}, \quad \text{for } i=1,2,\ldots,N, \end{aligned} \tag{5.14}$$

and the initial conditions at $t = t_0$ for the DO components are given by

$$\begin{aligned} \bar{u}\left(x,t_0\right) &= \mathbb{E}\left[u_0(x;\omega)\right], \\ Y_i\left(t_0;\omega\right) &= \langle u_0(\cdot,\omega) - \bar{u}\left(x,t_0\right), v_i(\cdot)\rangle, \\ u_i\left(x,t_0\right) &= v_i(x), \end{aligned} \tag{5.15}$$

for all $i = 1,\ldots,N$, where $v_i(x)$ is the eigenfields of the classical KL expansion of $u(x,t_0;\omega)$.

It is shown in [69] that by imposing suitable restrictions on the DO representation, the equations for methods such as Polynomial Chaos (PC) or Proper Orthogonal Decomposition (POD) can be recovered from the DO evolution equations. For example, PC can be recovered by setting $Y_i(t;\omega) = \Psi_i(\xi(\omega))$, where $\Psi_i(\xi)$ is an orthogonal polynomial in terms of ξ. Besides, it is shown in [10] that there exists an one-to-one correspondence between the eigenvalues of the KL expansion for $u(x,t;\omega)$ and the eigenvalues of the covariance matrix $C_{Y_i(t)Y_j(t)}$ in the DO representation given any fixed time t, and thus the stochastic coefficients Y_i together with the modes u_i provide the necessary information to describe both the shape and the magnitude of the uncertainty that characterizes a stochastic field but also the principle directions in which the stochasticity is distributed.

The moments of $u(x,y;\omega)$ can be readily computed from the DO representation. For example, the first moment, i.e., the mean, can be trivially obtained from the

first term $\bar{u}(x,t)$, and the variance can be calculated as follows:

$$\mathrm{Var}[u] = \mathbb{E}\left[(u-\bar{u})^2\right] = \mathbb{E}\left[\left(\sum_{i=1}^{N} u_i Y_i\right)^2\right] = \sum_{i,j=1}^{N} u_i \mathbb{E}[Y_i Y_j] u_j. \tag{5.16}$$

5.3.2 Bi-Orthogonal Representation

In order to overcome the aforementioned redundancy, the BO condition imposes the static constraint, which is the Bi-Orthogonality of the bases and stochastic coefficients in time [8]. In other words, we have the following conditions:

$$\langle u_i(\cdot,t), u_j(\cdot,t)\rangle = \lambda_i(t)\delta_{ij}, \quad \mathbb{E}[Y_i Y_j] = \delta_{ij}, \quad i,j = 1,\ldots,N, \tag{5.17}$$

where the λ_is are eigenvalues of the solution. There is a slight difference between the DO and BO representation; the stochastic bases carry the eigenvalues of the covariance operator in the DO representation while the spatial bases carry the eigenvalues of the covariance operator in the BO representation.

Define matrix S and M whose entries are

$$S_{ij} = \left\langle u_i, \frac{\partial u_j}{\partial t}\right\rangle, \quad M_{ij} = \mathbb{E}\left[Y_i \frac{\mathrm{d}Y_j}{\mathrm{d}t}\right]. \tag{5.18}$$

Then by taking time derivative of Eq. 5.17, we have

$$\begin{aligned}
&S_{ij} + S_{ji} = \left\langle u_i, \frac{\partial u_j}{\partial t}\right\rangle + \left\langle \frac{\partial u_i}{\partial t}, u_j\right\rangle = 0, && \text{for } i \neq j,\\
&S_{ij} = \frac{1}{2}\frac{\mathrm{d}\lambda_i(t)}{\mathrm{d}t}, && \text{for } i = j,\\
&M_{ij} + M_{ji} = \mathbb{E}\left[Y_i \frac{\mathrm{d}Y_j}{\mathrm{d}t}\right] + \mathbb{E}\left[\frac{\mathrm{d}Y_i}{\mathrm{d}t} Y_j\right] = 0.
\end{aligned} \tag{5.19}$$

Here we state the BO evolution equations without proof:

Theorem 5.3.2 (see [7]). *We assume that the bases and stochastic coefficients satisfy the BO condition. Then the original SPDE (Eq. 5.1) is reduced to the following system of equations:*

$$\begin{aligned}
&\frac{\partial \bar{u}(t,x)}{\partial t} = \mathbb{E}[\mathcal{N}_x[u(\cdot,t;\omega)]], \\
&\lambda_i \frac{\mathrm{d}Y_i(t;\omega)}{\mathrm{d}t} = -\sum_{j=1}^{N} S_{ij} Y_j + \langle \mathcal{N}_x[u] - \mathbb{E}\left[\mathcal{N}_x[u]\right], u_i(\cdot,t)\rangle, \quad i = 1,\ldots,N, \\
&\frac{\partial u_i(t,x)}{\partial t} = -\sum_{j=1}^{N} M_{ij} u_j + \mathbb{E}\left[\mathcal{N}_x[u] Y_i\right], \qquad i = 1,\ldots,N.
\end{aligned} \tag{5.20}$$

Moreover, if $\lambda_i \neq \lambda_j$ for $i \neq j$, $i, j, = 1, 2, \ldots, N$, the N-by-N matrices S and M have closed form expression:

$$\begin{aligned}
M_{ij} &= \begin{cases} \frac{G_{ij}+G_{ji}}{-\lambda_i+\lambda_j}, & \text{if } i \neq j \\ 0, & \text{if } i = j \end{cases}, \\
S_{ij} &= \begin{cases} G_{ij} + \lambda_i M_{ij}, & \text{if } i \neq j \\ G_{ii}, & \text{if } i = j \end{cases},
\end{aligned} \tag{5.21}$$

where the matrix G_{ij} is defined as $\left\langle \mathbb{E}\left[\mathcal{N}_x[u] Y_j\right], u_i \right\rangle$.

Similar to the DO method, the boundary condition is given by

$$\begin{aligned}
\mathcal{B}_x\left[\bar{u}(x,t;\omega)\right]|_{x\in\partial\mathcal{D}} &= \mathbb{E}[h(x,t;\omega)], \\
\mathcal{B}_x\left[u_i(x,t)\right]|_{x\in\partial\mathcal{D}} &= \mathbb{E}\left[Y_i(t;\omega) h(x,t;\omega)\right], \quad \text{for } i = 1,2,\ldots,N,
\end{aligned} \tag{5.22}$$

and the initial condition is generated from the KL expansion of $u(x, t_0; \omega)$.

5.3.3 A Brief Summary of Both Methods

Note the difference between the DO and BO condition: the spatial bases, u_i, under the DO condition evolve to the direction which is normal to the space V_S they expand (orthogonality is automatically maintained), while under the BO condition, only orthogonality is required and there is no restriction to the direction of their evolution. As a compensation to the lack of constraints in spatial bases, the BO condition puts an additional orthogonality restriction on the random coefficients Y_i. As a remark, Choi and et al. [12] prove theoretically the equivalence between both the DO and the BO methods, in a sense that one method is an exact reformulation of the other.

However, either method applies to a limited range of problems since the evolution equations are valid only if certain assumptions are satisfied. For the DO method, it is assumed that the covariance matrix of random coefficients, $C_{Y_iY_j}$, is invertible. Therefore, it fails when applied to some benchmark problems such as a stochastic PDE with deterministic initial condition. This is because all coefficients Y_i is equal to 0 at the initial state and their covariance matrix is undefined. For the BO method, it is assumed that there is no eigenvalue crossing in the given time domain in order to calculate the explicit expression of M_{ij} and S_{ij}. Researchers have found strategies to get around those pit holes, e.g., the hybrid gPC-DO method [11] and the psedo-inverse hybrid BO-DO method [3] are developed to address the above limitations respectively.

Inspired by the DO and BO methods, we introduce a new procedure of solving the time-dependent stochastic PDEs within the framework of Physics-Informed Neural Networks (PINNs). The introduced method inherits the similarities be-

tween the DO and the BO methods and can be implemented with either a DO or a BO flavor that are free from the aforementioned restrictions.

5.4 Methodology

5.4.1 Another Interpretation of the DO/BO Constrarints

The derivation of equations for both the DO and the BO methods can be summarized into four steps as follows:

1. Act operator $\mathbb{E}[\cdot]$ on both side of the SPDE and replace u by the finite expansion in Eq. 5.10. Notice that $\mathbb{E}[Y_i] = 0$. This leads to the first equation in Eq. 5.13 and Eq. 5.19:

$$\mathbb{E}\left[\frac{\partial u}{\partial t}\right] = \frac{\partial \bar{u}}{\partial t} = \mathbb{E}\left[\mathcal{N}_x[u(x,t;\omega)]\right] \tag{5.23}$$

2. Acting operator $\langle \cdot, u_i \rangle$ on both sides of the SPDE:

$$\left\langle \frac{\partial u}{\partial t}, u_i \right\rangle = \langle \mathcal{N}_x[u(x,t;\omega)], u_i \rangle \tag{5.24}$$

3. Acting operator $\mathbb{E}[\cdot Y_i]$ on both sides of the SPDE:

$$\mathbb{E}\left[\frac{\partial u}{\partial t} Y_i\right] = \mathbb{E}\left[\mathcal{N}_x[u(x,t;\omega)] Y_i\right] \tag{5.25}$$

4. Substitute u by the truncated expansion in Eq. 5.10, and use the DO and BO constraints (Eq. 5.11 and Eq. 5.17) to simplify Eq. 5.24 and Eq. 5.25.

Due to the orthogonality of $u_i(x,t)$, they form a valid set of basis in the physical space $\mathcal{D}$. The random coefficients $Y_i(t;\omega)$ are also linearly independent as they are orthogonal under the BO representation, and the DO representation is equivalent to the BO representation so Y_i will not degenerate in the DO expansion either. Therefore, the random coefficients $Y_i(t;\omega)$ form a valid set of basis in the probability space Ω. Consequently, Eq. 5.23–5.25 are the weak formulation of the original SPDE in the physical space and the probability space (note that Eq. 5.23 is the inner product of both side of the original SPDE on constant 1, which can be regarded as the 0th basis in the probability space), and they provide all the necessary information to find the solution u_N in $V_S \times \Omega_S$.

5.4.2 NN-DO/BO Approach

In this section we formalize the algorithm of solving time-dependent stochastic PDEs using PINNs. First, we rewrite Eq. 5.10 as

$$u_N(x,t;\omega) = \bar{u}(x,t) + \sum_{i=1}^{N} a_i(t)u_i(x,t)Y_i(t;\omega), \tag{5.26}$$

while enforcing $\langle u_i, u_i \rangle = 1$ and $\mathbb{E}[Y_i^2] = 1$. The time-dependent coefficients $a_i(t)$ are scaling factors and play the role of $\sqrt{\lambda_i}$ when we compare Eq. 5.26 with the classsical KL expansion Eq. 5.4. Suppose the original SPDE is parameterized into a PDE that involves a finite set of random variables $\xi(\omega)$, then $u(x,t;\omega)$ can be written as $u(x,t;\xi)$ and $Y_i(t;\omega)$ can be written as $Y_i(t;\xi)$. Four separate neural networks are constructed:

1. The neural net $\bar{u}_{nn}(x,t)$ that takes x and t as the input and outputs $\mathbb{E}[u(x,t;\omega)]$;

2. The neural net $A_{nn}(t)$ that takes t as the input and outputs a N-dimensional vector representing $a_i(t)$, for $i = 1, 2, \ldots, N$;

3. The neural net $U_{nn}(x, t)$ that takes x and t as the input and outputs a N-dimensional vector representing $u_i(x, t)$, for $i = 1, 2, \ldots, N$;

4. The neural net $Y_{nn}(\xi, t)$ that takes ξ and t as the input and outputs a N-dimensional vector representing $Y_i(t; \xi)$, for $i = 1, 2, \ldots, N$.

A surrogate neural net for the solution $u_N(x, t; \omega)$ can be constructed from those four neural nets by substituting them into Eq. 5.26, yielding

$$u_{nn}(x, t; \xi) = \bar{u}_{nn} + \sum_{i=1}^{N} A_{nn,i} U_{nn,i} Y_{nn,i}. \tag{5.27}$$

Since the weak formulation of SPDE involves integration in both the physical and the probability spaces, the neural nets are evaluated at the physical collocation points $\{x_c^k\}_{k=1}^{n_x}$ and the probabilistic collocation points $\{\xi_c^l\}_{l=1}^{n_\xi}$, where n_x and n_ξ are the numbers of collocation points. In the time domain $[0, T]$ we uniformly sample n_t random points $\{t_c^s\}_{s=1}^{n_t}$. Once we have constructed the computation graph, the derivatives of the quantity of interest with respect to time t and space coordinate x can be easily obtained via the auto-differentiation algorithm, and the integration terms can be evaluated by using a numerical quadrature rule.

The loss function is a weighted summation of four components: the weak formulation of SPDE, initial/boundary conditions, constraints on U_{nn} and Y_{nn}, and the additional regularization terms. The loss function in each part consists of mean squared errors (MSEs) associated with the prescribed constraints, calculated from the sampled training points. Next, we will illustrate each of these four components of loss function and write down their explicit expressions.

Loss Function for the Weak Formulation of SPDE

The weak form of the SPDE, i.e., Eq. 5.23–Eq. 5.25 can be rewritten as

$$\epsilon_1^{ks} := \mathbb{E}\left[\frac{\partial u_{nn}}{\partial t}(x_c^k, t_c^s; \xi) - \mathcal{N}_x[u_{nn}(x_c^k, t_c^s; \xi)]\right] = 0, \tag{5.28}$$

$$\epsilon_2^{sl} := \left\langle \frac{\partial u_{nn}}{\partial t}(x, t_c^s; \xi_c^l) - \mathcal{N}_x[u_{nn}(x, t_c^s; \xi_c^l)], U_{nn,i}(x, t_c^s) \right\rangle = 0, \tag{5.29}$$

$$\epsilon_3^{ks} := \mathbb{E}\left[\left(\frac{\partial u_{nn}}{\partial t}(x_c^k, t_c^s; \xi) - \mathcal{N}_x[u_{nn}(x_c^k, t_c^s; \xi)]\right) Y_{nn,i}(t_c^s; \xi)\right] = 0, \tag{5.30}$$

where the integration in the physical space and the probability space shall be evaluated by using a numerical quadrature rule. The first part of the loss function is calculated by

$$\mathrm{MSE_w} = \frac{1}{n_x n_t}\sum_{k,s}\left(\epsilon_1^{ks}\right)^2 + \frac{1}{n_t n_\xi}\sum_{s,l}\left(\epsilon_2^{sl}\right)^2 + \frac{1}{n_x n_t}\sum_{k,s}\left(\epsilon_3^{ks}\right)^2. \tag{5.31}$$

Loss Function for Initial and Boundary Conditions

Let t_0 be the initial time of computation. The initial condition for the representation in Eq. 5.26 is similar to Eq. 5.15. The only difference is that Y_i are normalized to have unit variance, and the standard deviation of $\langle u_0(x;\xi) - \bar{u}(x,t_0), v_i(x)\rangle$ is assigned to be the initial value for a_i. Here $v_i(x)$ are the normalized KL modes for $u(x, t_0; \omega)$,

and they are the initial value for u_i. That is,

$$\begin{aligned}
&\bar{u}(x,t_0) = \mathbb{E}\left[u(x,t_0;\xi)\right], \\
&u_i(x,t_0) = v_i(x), \\
&a_i(t_0) = \sqrt{\mathbb{E}\left[\langle u(x,t_0;\xi) - \mathbb{E}[u(x,t_0;\xi)], v_i\rangle^2\right]}, \\
&Y_i(t_0;\xi) = \frac{1}{a_i(t_0)}\langle u(x,t_0;\xi) - \mathbb{E}[u(x,t_0;\xi)], v_i\rangle .
\end{aligned} \tag{5.32}$$

For deterministic initial condition, $u_i(x,t_0)$ are set to be orthonormal bases satisfying the boundary condition, $Y_i(t_0;\xi)$ are set to be the gPC bases of ξ with unit variance, and $a_i(t_0)$ is set to be 0. The initial condition shall be imposed to the neural network by adding an extra penalty term $\mathrm{MSE}_{\mathrm{ic}}$, and it is calculated as follows:

$$\begin{aligned}
\mathrm{MSE}_{\mathrm{ic}} =& \frac{1}{n_x}\sum_{k=1}^{n_x}\left(\bar{u}_{nn}(x_c^k,t_0) - \bar{u}(x_c^k,t_0)\right)^2 + \frac{1}{Nn_x}\sum_{i=1}^{N}\sum_{k=1}^{n_x}\left(U_{nn,i}(x_c^k,t_0) - u_i(x_c^k,t_0)\right)^2 \\
&+ \frac{1}{N}\sum_{i=1}^{N}\left(A_{nn,i}(t_0) - a_i(t_0)\right)^2 + \frac{1}{Nn_\xi}\sum_{i=1}^{N}\sum_{l=1}^{n_\xi}\left(Y_{nn,i}(t_0;\xi_c^l) - Y_i(t_0;\xi_c^l)\right)^2 .
\end{aligned} \tag{5.33}$$

The boundary condition is imposed by taking the weak formulation of Eq. 5.3 in the random space, i.e.,

$$\begin{aligned}
&\mathbb{E}\left[\mathcal{B}_x[u(x_b,t;\xi)]\right] = \mathbb{E}[h(x_b,t;\xi)] \Rightarrow \mathcal{B}_x\left[\bar{u}(x_b,t)\right] = \mathbb{E}[h(x_b,t;\xi)], \\
&\mathbb{E}\left[\mathcal{B}_x[u(x_b,t;\xi)]Y_i(t;\xi)\right] = \mathbb{E}[h(x_b,t;\xi)Y_i(t;\xi)] \\
&\qquad\qquad \Rightarrow \sum_{j=1}^{N} C_{Y_iY_j}a_j(t)\mathcal{B}_x\left[u_j(x_b,t)\right] = \mathbb{E}[h(x_b,t;\xi)Y_i(t;\xi)].
\end{aligned} \tag{5.34}$$

Thus, the loss associated with the boundary condition is

$$\begin{aligned}\mathrm{MSE}_{\mathrm{bc}} =& \frac{1}{n_t}\sum_{s=1}^{n_t}\left(\mathcal{B}_x[\bar{u}_{nn}(x_b,t_c^s)] - \mathbb{E}\left[h(x_b,t;\xi)\right]\right)^2 \\ &+ \frac{1}{Nn_t}\sum_{i=1}^{N}\sum_{s=1}^{n_t}\left(\sum_{j=1}^{N} C_{Y_iY_j}(t_c^s)A_{nn,j}(t_c^s)\mathcal{B}_x\left[U_{nn,j}(x_b,t_c^s)\right] - \mathbb{E}\left[h(x_b,t_c^s;\xi)Y_{nn,i}(t_c^s;\xi)\right]\right)^2,\end{aligned} \tag{5.35}$$

where the expectations and covariance matrix shall be evaluated by using a numerical quadrature rule.

Note that the periodic boundary condition can be strictly imposed by modifying the neural nets $\bar{u}_{nn}$ and U_{nn} by replacing the input x with the combination of $\sin(2\pi x/L)$ and $\cos(2\pi x/L)$, where L is the length of domain $\mathcal{D}$. This is because any continuous 2π-periodic function can be written as a nonlinear function of $\sin(x)$ and $\cos(x)$. This modification simplifies the loss function by removing the loss due to the periodic boundary condition.

Loss Function for the Constraints on U_{nn} and Y_{nn}

This is the part where we can have different implementations in favor of the DO or the BO method. Both DO and BO representations require that $\mathbb{E}[Y_i] = 0$, and thus the loss functions in both implementations should involve the term $\frac{1}{n_t}\sum_{s=1}^{n_t}\left(\mathbb{E}[Y_i(t_c^s;\xi)]\right)^2$. For the DO constraint, Eq. 5.11 should be satisfied. In addition, we require that

$$\mathbb{E}\left[Y_i(t;\xi)\frac{\mathrm{d}Y_i(t;\xi)}{\mathrm{d}t}\right] = 0, \quad \forall t \text{ and } i = 1,2,\ldots,N,$$

so that Y_i stay normalized with unit variance. The loss function for DO writes

$$\begin{aligned}\mathrm{MSE}_{\mathrm{DO}} =& \frac{1}{Nn_t}\sum_{i=1}^{N}\sum_{s=1}^{n_t}\left(\mathbb{E}[Y_i(t_c^s;\xi)]\right)^2 \\ &+ \frac{1}{N^2n_t}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{s=1}^{n_t}\left\langle\frac{\mathrm{d}U_{nn,i}(x,t_c^s)}{\mathrm{d}t}, U_{nn,j}(x,t_c^s)\right\rangle^2 \\ &+ \frac{1}{Nn_t}\sum_{i=1}^{N}\sum_{s=1}^{n_t}\mathbb{E}\left[Y_{nn,i}(t_c^s;\xi)\frac{\mathrm{d}Y_{nn,i}(t_c^s;\xi)}{\mathrm{d}t}\right]^2.\end{aligned} \tag{5.36}$$

For the BO constraints, Eq. 5.19 generates the following loss function:

$$\begin{aligned}\mathrm{MSE}_{\mathrm{BO}} =& \frac{1}{Nn_t}\sum_{i=1}^{N}\sum_{s=1}^{n_t}\left(\mathbb{E}[Y_i(t_c^s;\xi)]\right)^2 \\ &+ \frac{1}{N^2n_t}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{s=1}^{n_t}\left(\left\langle\frac{\mathrm{d}U_{nn,i}(x,t_c^s)}{\mathrm{d}t}, U_{nn,j}(x,t_c^s)\right\rangle + \left\langle\frac{\mathrm{d}U_{nn,j}(x,t_c^s)}{\mathrm{d}t}, U_{nn,i}(x,t_c^s)\right\rangle\right)^2 \\ &+ \frac{1}{Nn_t}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{s=1}^{n_t}\left(\mathbb{E}\left[Y_{nn,i}(t_c^s;\xi)\frac{\mathrm{d}Y_{nn,j}(t_c^s;\xi)}{\mathrm{d}t}\right] + \mathbb{E}\left[Y_{nn,j}(t_c^s;\xi)\frac{\mathrm{d}Y_{nn,i}(t_c^s;\xi)}{\mathrm{d}t}\right]\right)^2.\end{aligned} \tag{5.37}$$

Since we put all the scaling factor to $a(t)$ and keep $u_i(x,t)$ normalized, $S_{ij} + S_{ji} = 0$ still holds true when i is equal to j.

Loss Function for Additional Regularization

Additional regularization terms shall be added to the loss function to reduce the risk of overfitting. Here we remark that it is helpful to add a penalty term from the original equation (Eq. 5.1) to speed up the training. The loss from the original equation writes

$$\mathrm{MSE}_0 = \frac{1}{n_x n_t n_\xi}\sum_{k=1}^{n_x}\sum_{s=1}^{n_t}\sum_{l=1}^{n_\xi}\left(\frac{\partial u_{nn}}{\partial t} - \mathcal{N}_x\left[u_{nn}(x_c^k, t_c^s;\xi_c^l)\right]\right)^2. \tag{5.38}$$

Putting the Loss Functions Together

In all, the loss function used for training the PINNs is the weighted summation of the aforementioned MSEs. In practice we put a large weight on the loss for the BO/DO constraints and the initial/boundary conditions, and put a small weight on the loss for the original SPDE. Intuitively, there would be too much redundancy in the expansion (Eq. 5.26) if the BO/DO constraints were not strictly enforced, and the training process would be meaningless. We put large weights in the loss for DO/BO constraints and initial/boundary conditions to kill the redundancy in the first place, and we put a small weight for the regularization term since it is only used to help speedup the training process, and is not essential. Nevertheless, the distribution of weights is still an open question for future research. In the numerical tests we train our neural nets by minimizing the following loss function:

$$\mathcal{LOSS} = \mathrm{MSE}_{\mathrm{w}} + 100 \times (\mathrm{MSE}_{\mathrm{ic}} + \mathrm{MSE}_{\mathrm{bc}} + \mathrm{MSE}_{\mathrm{BO/DO}}) + 0.1 \times \mathrm{MSE}_{0}. \tag{5.39}$$

This proposed algorithm can be implemented with the DO or the BO constraints, and we are name them the NN-DO or NN-BO method respectively. The

proposed algorithm is summarized as follows:

Algorithm 4: NN-DO/BO

Step 1: Build the neural networks for $\bar{u}_{nn}(x,t)$, $A_{nn}(t)$, $U_{nn}(x,t)$ and $Y_{nn}(t;\xi)$;

Step 2: Select n_x collocation points in the physical domain $\mathcal{D}$, n_ξ collocation points in the stochastic space Ω. Randomly pick n_t points in the time domain $[0,T]$ from a uniform distribution;

Step 3: Specify the method to use (DO or BO) and calculate the loss function in Eq. 5.39;

Step 4: Train the neural networks by minimizing the loss function with your favorite optimization algorithm (e.g. Adam [35]);

Step 5: Reconstruct the SPDE solution using Eq. 5.27.

We remark that the bottle neck of the original DO/BO method is to generate an explicit expression for the temporal derivatives of the bases (Step 4 in Section 5.4.1). For the DO method, it involves calculating the inverse of a covariance matrix which could be singular, and for the BO method, to obtain explicit expression for matrices M and S, one has to assume no eigenvalue cross. In the proposed PINN-DO/BO algorithm, there is no need to derive explicit expressions from constraints, instead we only need to write the constraints into the loss functions as they are.

5.5 Numerical Examples

We test our NN-DO/BO method with two benchmark cases that are specially designed to have exact solution for the DO and BO representations. For all test cases we use deep feed-forward neural networks for $\bar{u}_{nn}(x,t)$, $A_{nn}(t)$, $U_{nn}(x,t)$ and $Y_{nn}(t;\xi)$. The loss functions are defined in Eq. 5.39, and a Adam optimizer with

learning rate 0.001 is used to train the networks for 100000 epochs.

5.5.1 Application to Linear Stochastic Problem

In this section we demonstrate a pedagogical example by solving the linear stochastic advection equation using the NN-DO/BO methods. The stochastic advection equation with a random advection coefficient has the form

$$\begin{aligned} &\frac{\partial u(x,t;\xi)}{\partial t} + \xi \frac{\partial u(x,t;\xi)}{\partial x} = 0, \quad \forall (x,t) \in \mathcal{D} \times [0,T], \\ &u(x,0;\xi) = -\sin(x), \quad \forall x \in \mathcal{D}, \end{aligned} \tag{5.40}$$

where the physical domain $\mathcal{D}$ is $[-\pi, \pi]$ and we calculate the solution till $T = \pi$. Periodic boundary condition is considered, such that $u(-\pi, t) = u(\pi, t)$, $\forall t \in [0, T]$. The randomness comes from the advection velocity, which is considered as a Gaussian random variable $\xi \sim N(0, \sigma^2)$ where we set σ to be 0.8.

The exact solutions for the mean and variance of the stochastic advection equation, Eq. 5.40, can be calculated, and the closed form formulas of the DO and BO expansion components u_i and Y_i, $i = 1, 2, \ldots, N$ can be derived. Here we write down the exact solution and the expansion components without giving details of the derivation:

- Exact solutions:

$$\begin{aligned} u(x,t;\xi) &= -\sin(x - \xi t) \\ \mathbb{E}[u](x,t) &= -\sin(x) \exp\left(-\frac{\sigma^2 t^2}{2}\right) \\ \mathrm{Var}[u](x,t) &= \frac{1}{2}\left[1 - \cos(2x)\exp\left(-2\sigma^2 t^2\right)\right] - \mathbb{E}[u]^2 \end{aligned} \tag{5.41}$$

- DO components:

$$u(x,t;\xi) = \mathbb{E}[u](x,t) + u_1^{DO}(x,t)Y_1^{DO}(t;\xi) + u_2^{DO}(x,t)Y_2^{DO}(t;\xi), \tag{5.42}$$

where

$$\begin{aligned} &u_1^{DO}(x,t) = -\frac{1}{\sqrt{\pi}}\cos(x), \quad u_2^{DO}(x,t) = -\frac{1}{\sqrt{\pi}}\sin(x), \\ &Y_1^{DO}(t;\xi) = -\sqrt{\pi}\sin(\xi t), \quad Y_2^{DO}(t;\xi) = \sqrt{\pi}\left(\cos(\xi t) - \exp\left(-\frac{\sigma^2 t^2}{2}\right)\right). \end{aligned} \tag{5.43}$$

- BO components:

$$u(x,t;\xi) = \mathbb{E}[u](x,t) + u_1^{BO}(x,t)Y_1^{BO}(t;\xi) + u_2^{BO}(x,t)Y_2^{BO}(t;\xi), \tag{5.44}$$

where

$$\begin{aligned} &u_1^{BO}(x,t) = -\frac{\alpha_1(t)}{\sqrt{\pi}}\cos(x), \quad u_2^{BO}(x,t) = -\frac{\alpha_2(t)}{\sqrt{\pi}}\sin(x), \\ &Y_1^{BO}(t;\xi) = -\frac{\sqrt{\pi}}{\alpha_1(t)}\sin(\xi t), \quad Y_2^{BO}(t;\xi) = \frac{\sqrt{\pi}}{\alpha_2(t)}\left(\cos(\xi t) - \exp\left(-\frac{\sigma^2 t^2}{2}\right)\right), \end{aligned} \tag{5.45}$$

and the normalizing factors

$$\alpha_1(t) = \sqrt{\pi\mathbb{E}\left[sin^2(\xi t)\right]}, \quad \alpha_2(t) = \sqrt{\pi\mathbb{E}\left[\left(\cos^2(\xi t) - \exp\left(-\frac{\sigma^2 t^2}{2}\right)\right)^2\right]}.$$

We set n_t, n_x and n_ξ all to be 50. The data points in time domain $\{t_c^s\}_{s=1}^{n_t}$ are sampled from a uniform distribution. The collocation points $\{x_c^k\}_{k=1}^{n_x}$ are equidistantly distributed in $[-\pi, \pi]$. For the collocation points in the stochastic space, instead of using the Gauss-Hermite quadrature rule, we generate $\{\xi_c^l\}_{l=1}^{n_\xi}$ by applying the inverse cumulative distribution function of the standard normal distribution to the Gauss-Legendre quadrature points in $[0, 1]$, because the generated ξs will be

more concentrated near the origin, making it easier to train the neural networks.

Case 1: NN-DO Method

The classical DO method cannot be directly applied to this SPDE with deterministic initial condition. However, by applying the NN-DO method and setting proper initial conditions, we obtain good results. Considering Eq. 5.26, the initial conditions are

$$\begin{aligned} &\bar{u}(x,0) = u(x,0), && a_1(0) = a_2(0) = 0, \\ &u_1(x,0) = -\frac{1}{\pi}\cos(x), && u_2(0) = -\frac{1}{\pi}\sin(x), \\ &Y_1(0;\xi) = -\xi, && Y_2(0;\xi) = -\frac{\sqrt{2}}{2}(\xi^2 - 1), \end{aligned} \tag{5.46}$$

where we use the periodic orthonormal bases in the $[-\pi, \pi]$ interval as the initial conditions for u_1 and u_2, and we use normalized Hermite polynomials for the initial conditions of Y_1 and Y_2. The neural network $\bar{u}_{nn}(x,t)$ and $U_{nn}(x,t)$ have three hidden layers with 32 neurons per hidden layer, the network $A_{nn}(t)$ has three hidden layers with 16 neurons per hidden layer, and the network $Y_{nn}(t;\xi)$ has four hidden layers with 64 neurons in each hidden layer. The references for the solution mean, variance, and u_i are taken directly from Eq. 5.41 and Eq. 5.43. The references for the normalizing factors, a_i, are the standard deviations of Y_i^{DO} in Eq. 5.43, and the references for Y_i are calculated by Y_i^{DO}/a_i.

We compare the results obtained from the NN-DO method with the exact solutions. Figure 5.1a shows the development of the scaling factors a_i $(i = 1, 2)$ with time. As time evolves, the scaling factors increase monotonically and converge at $T = \pi$, indicating the randomness in the system grows from none to fully

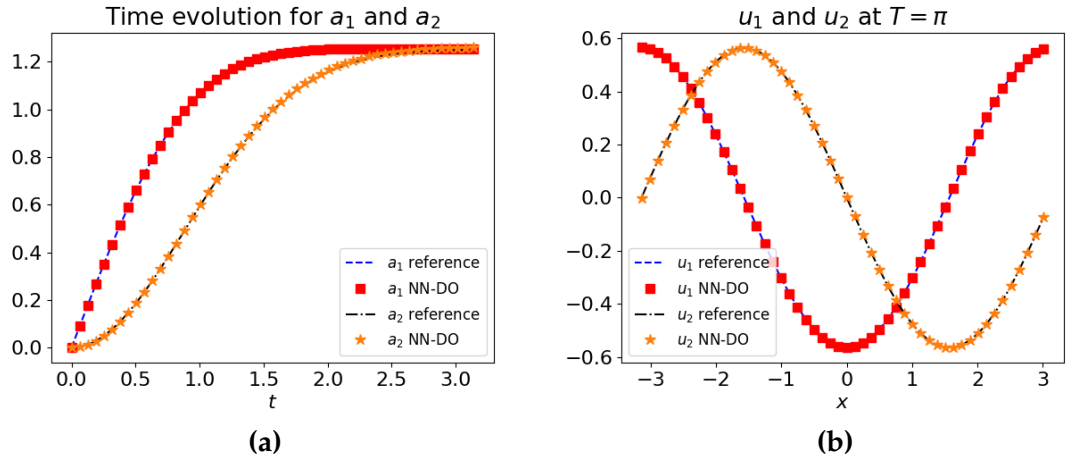


Figure 5.1: Stochastic advection equation (NN-DO): On the left-hand side we plot the evolution of the scaling factors a_i. They start from zero because of the deterministic initial condition, and increase with time, indicating the randomness in the SPDE solution accumulates as time grows. On the right-hand side we plot the bases u_i at the final time $T = \pi$ versus the exact solutions. The scattered points for u_i indicate the collocation points in the physical space.

developed during the time period $t \in [0, \pi]$, as a result of the stochastic advection coefficient. Figure 5.1b shows the comparison of the DO bases obtained from the NN-DO method and the exact bases at $T = \pi$. The DO bases generated by the neural networks coincide with the reference solutions very well. Figure 5.2 shows the comparison of the stochastic coefficients Y_i $(i = 1, 2)$ versus the normalized exact DO coefficients Y_i^{DO} at four different times $t = 0, \pi/3, 2\pi/3$ and π. The random coefficients as functions of the random variable ξ evolve with time to develop a subtle wavy structure, while preserving the orthogonality. The NN-DO method uncovers the evolution behavior of Y_i. Figure 5.3 shows the mean and variance of the NN-DO solution, versus the exact ones, at $t = 2\pi/3$ and $t = \pi$. Apparently, the scale of variance is huge compared to the scale of mean, indicating that the random fluctuation dominates the averaged solution profile. We report the errors of the NN-DO method in Figure 5.4 and Table 5.1 where we analyze the L_2 error (defined by $\|f_{NN} - f_{exact}\|_2$ for any function f) and the relative L_2 error (defined by $\|f_{NN} - f_{exact}\|_2 / \|f_{exact}\|_2$) of the NN-DO results versus the exact solutions. As we can see in Figure 5.4, the relative L_2 error of the estimated variance stays

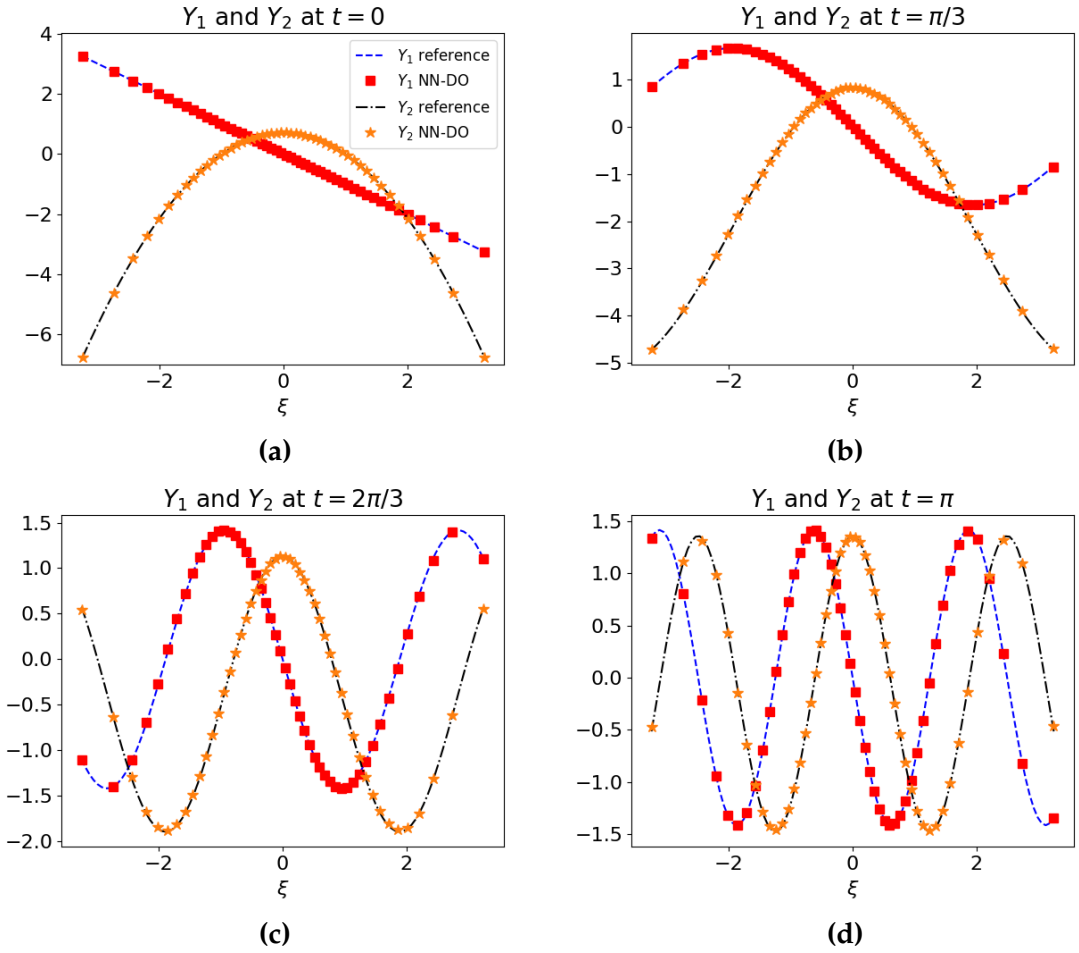


Figure 5.2: Stochastic advection equation (NN-DO): Solutions for the random coefficients Y_1 and Y_2 at four different times $t = 0, \pi/3, 2\pi/3$ and π. Both of them coincide with the exact solution. The scattered points indicate the collocation points in the probabilistic space.

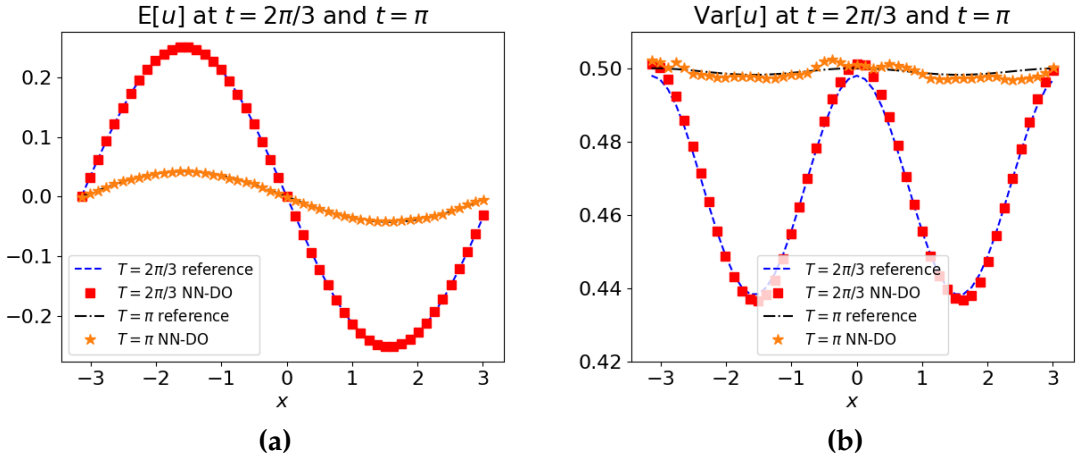


Figure 5.3: Stochastic advection equation (NN-DO): Mean and variance of the solution at time $T = 2\pi/3$ and $T = \pi$. The scattered points indicate the collocation points in the physical space.

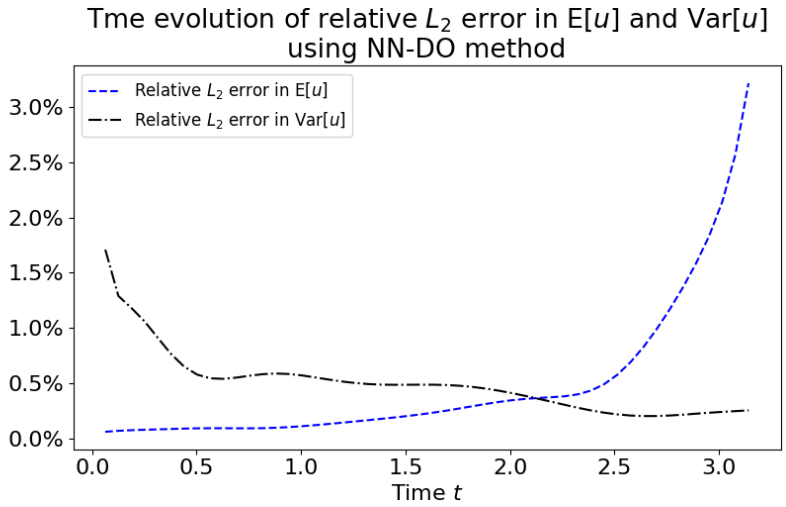


Figure 5.4: Stochastic advection equation (NN-DO): Relative L_2 error in the mean and variance versus time. The Relative error in mean grows because the L_2 norm of the exact solution mean shrinks exponentially with time.

below 1% after $t = 0.5$. The relative L_2 error in mean is small at the beginning and grows with time, because of the exponential shrinkage in the L_2 norm of the exact solution mean. Table 5.1 summarizes the errors at the final time $T = \pi$, indicating the good performance of the NN-DO method.

	$\mathbb{E}[u]$	$\mathbf{Var}[u]$	a_1	a_2
L_2 **error**	0.0010	0.0013	0.0018	0.0063
Relative L_2 **error**	3.22%	0.25%	0.17%	0.68%
	u_1	u_2	Y_1	Y_2
L_2 **error**	0.0007	0.0005	0.0013	0.0019
Relative L_2 **error**	0.18%	0.13%	0.89%	1.36%

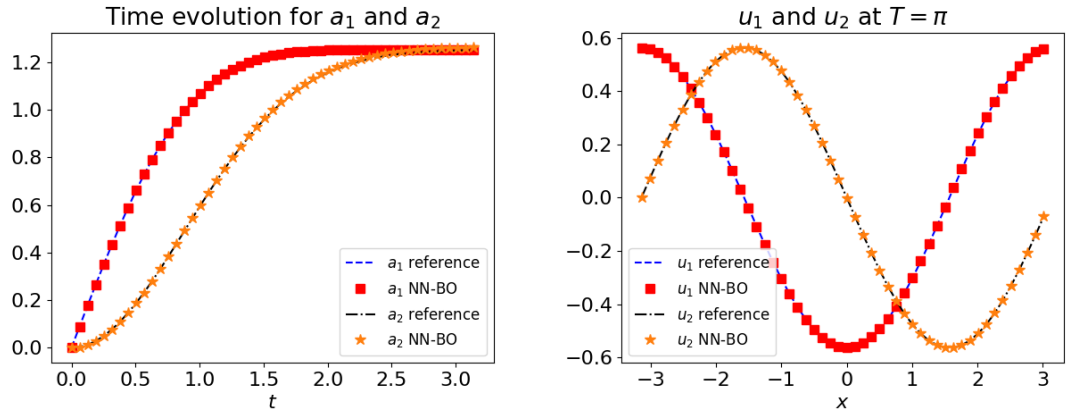
Table 5.1: Stochastic advection equation (NN-DO): The L_2 and relative L_2 errors of NN-DO solutions versus the exact solutions at the final time $T = \pi$.

Case 2: NN-BO Method

We solve the same problem (Eq. 5.40) again, but this time we use the BO constraints by including Eq. 5.37 as part of the loss function. The initial conditions and

reference solutions for the BO components, i.e., a_i, u_i and Y_i, are the same as that of the previous case, and neural networks used to approximate the BO components have the same size as with the networks used in the previous case.

Similarly, we compare the results obtained using the NN-BO method with the exact solutions. Figure 5.5a and Figure 5.5b display the scaling factors a_i $(i = 1, 2)$ at $t \in [0, \pi]$ and the BO bases u_i $(i = 1, 2)$ at $t = \pi$, respectively. Figure 5.6 shows the stochastic coefficients Y_i $(i = 1, 2)$ versus the normalized exact BO coefficients Y_i^{BO} at four different times: $t = 0, \pi/3, 2\pi/3$ and π. Figure 5.7 shows the mean and variance calculated by the NN-BO method at $t = 2\pi/3$ and $t = \pi$. They all show significant agreement of the BO solutions with the exact reference solutions. The L_2 and relative L_2 errors of the solution mean and variance are pictured in Figure 5.8, and Table 5.2 summarizes the errors of the BO components at the final time $T = \pi$. The NN-BO method demonstrates very good performance as with the NN-DO method, and this is no surprise since the BO constraints are equivalent to the DO constraints.



(a) **(b)**

Figure 5.5: Stochastic advection equation (NN-BO): On the left-hand side we plot the evolution of the scaling factors a_i. On the right-hand side we plot the bases u_i at the final time $T = \pi$ versus the exact solutions. The scattered points for u_i indicate the collocation points in the physical space.

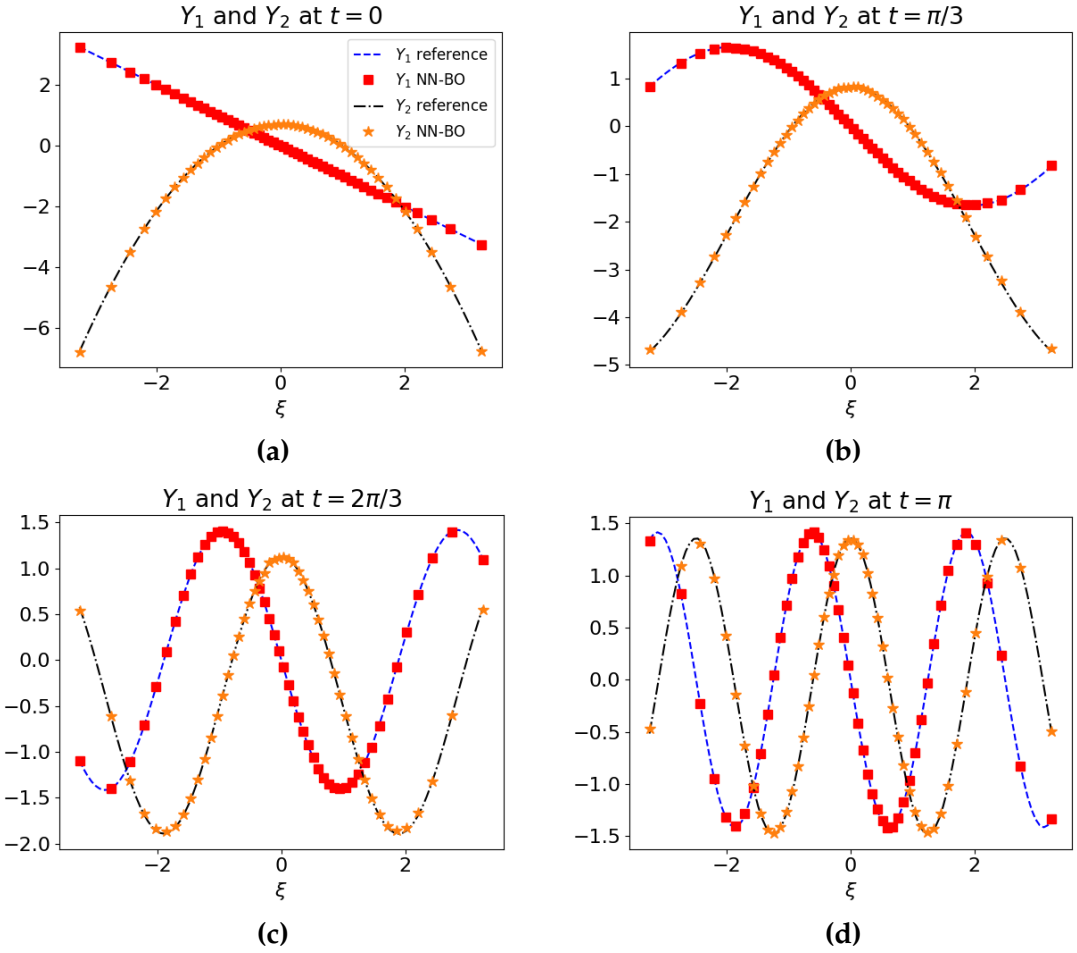


Figure 5.6: Stochastic advection equation (NN-BO): Solutions for the random coefficients Y_1 and Y_2 at four different times $t = 0, \pi/3, 2\pi/3$ and π. Both of them coincide with the exact solutions. The scattered points indicate the collocation points in the probabilistic space.

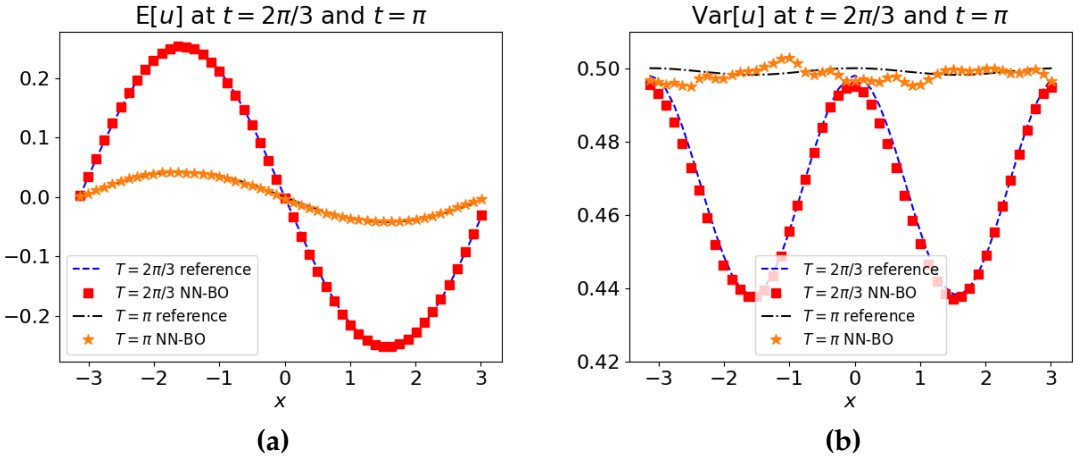


Figure 5.7: Stochastic advection equation (NN-BO): Mean and variance of the solution at time $T = 2\pi/3$ and $T = \pi$. The scattered points indicate the collocation points in the physical space.

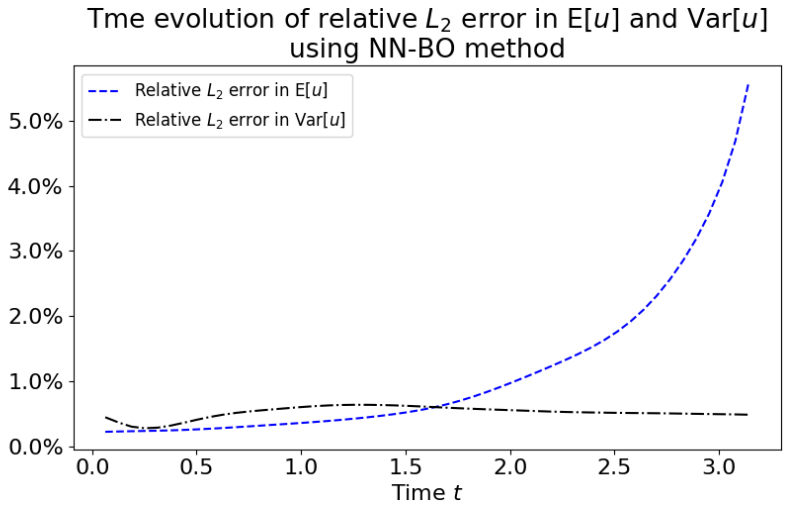


Figure 5.8: Stochastic advection equation (NN-BO): Relative L_2 error in the mean and variance versus time.

	$\mathbb{E}[u]$	$\mathbf{Var}[u]$	a_1	a_2
L_2 **error**	0.0017	0.0024	0.0015	0.0068
Relative L_2 **error**	5.58%	0.48%	0.14%	0.74%
	u_1	u_2	Y_1	Y_2
L_2 **error**	0.0019	0.0008	0.0013	0.0017
Relative L_2 **error**	0.46%	0.21%	0.89%	1.18%

Table 5.2: Stochastic advection equation (NN-BO): The L_2 and relative L_2 errors of NN-BO solutions versus the exact solutions at the final time $T = \pi$.

5.5.2 Application to Long-term Nonlinear Stochastic Problem

In this section, we apply the NN-DO/BO methods to solving non-linear stochastic problems, by considering the following stochastic Burgers' equation:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} + f(x,t;\omega), \quad \forall t \in [0,T] \quad \text{and} \quad x \in \mathcal{D}, \tag{5.47}$$

where the physical domain $\mathcal{D}$ is $[-\pi,\pi]$, and $\nu = 0.1$ is the viscosity coefficient. Suppose the random forcing term $f(x,t;\omega)$ is parameterized by two identically independent uniformly distributed random variables in $[0,1]$, denoted by $\xi_1(\omega)$

and $\xi_2(\omega)$, thus the stochastic behavior of solution $u(x,t;\omega)$ shall be fully described by ξ_1 and ξ_2, too. In this example, we create a manufactured solution $u(x,t;\xi_1,\xi_2)$ such that the exact DO and BO components can be calculated explicitly. The manufactured solution is

$$\begin{aligned} u(x,t;\xi_1,\xi_2) = &-\sin(x-t) - \sqrt{3}(1.5+\sin(t))\cos(x-t)(2\xi_1-1) \\ &+ \sqrt{3}(1.5+\cos(3t))\cos(2x-3t)(2\xi_2-1). \end{aligned} \tag{5.48}$$

The random forcing term $f(x,t;\omega)$ can be uniquely calculated given the manufactured solution. Due to its lengthy expression, here we omit writing down the explicit formula for $f(x,t;\omega)$. Without going into too much detail, Eq. 5.48 can be rewritten as either a DO expansion or a BO expansion, given by:

- DO components:

$$u(x,t;\xi_1,\xi_2) = \mathbb{E}[u](x,t) + u_1^{DO}(x,t)Y_1^{DO}(t;\xi_1,\xi_2) + u_2^{DO}(x,t)Y_2^{DO}(t;\xi_1,\xi_2), \tag{5.49}$$

  where

$$\begin{aligned} u_1^{DO}(x,t) = -\frac{1}{\sqrt{\pi}}\cos(x-t),\ u_2^{DO}(x,t) = \frac{1}{\sqrt{\pi}}\cos(2x-3t), \\ Y_1^{DO}(t;\xi_1,\xi_2) = \sqrt{3\pi}(1.5+\sin(t))(2\xi_1-1), \\ Y_2^{DO}(t;\xi_1,\xi_2) = \sqrt{3\pi}(1.5+\cos(3t))(2\xi_2-1); \end{aligned} \tag{5.50}$$

- BO components:

$$u(x,t;\xi_1,\xi_2) = \mathbb{E}[u](x,t) + u_1^{BO}(x,t)Y_1^{BO}(t;\xi_1,\xi_2) + u_2^{BO}(x,t)Y_2^{BO}(t;\xi_1,\xi_2), \tag{5.51}$$

where

$$
\begin{aligned}
u_1^{BO}(x,t) &= -(1.5+\sin(t))\cos(x-t),\\
u_2^{BO}(x,t) &= (1.5+\cos(3t))\cos(2x-3t),\\
Y_1^{BO}(t;\xi_1,\xi_2) &= \sqrt{3}(2\xi_1-1),\ Y_2^{BO}(t;\xi_1,\xi_2) = \sqrt{3}(2\xi_2-1).
\end{aligned}
\tag{5.52}
$$

If we normalize the bases and the random coefficients, and write the above expansions in the form of Eq.5.26, both the DO expansion and the BO expansion yield the same expression:

$$
\begin{aligned}
u_1(x,t) &= -\frac{1}{\sqrt{\pi}}\cos(x-t), \quad u_2(x,t) = \frac{1}{\sqrt{\pi}}\cos(2x-3t),\\
a_1(t) &= \sqrt{\pi}(1.5+\sin(t)), \qquad a_2(t) = \sqrt{\pi}(1.5+\cos(3t)),\\
Y_1(t;\xi_1,\xi_2) &= 2\xi_1-1, \qquad Y_2(t;\xi_1,\xi_2) = 2\xi_2-1.
\end{aligned}
\tag{5.53}
$$

We calculate the solution till $T = 10\pi$ to demonstrate the long-term performance of the NN-DO/BO method. In practice, we divide the time domain into ten non-overlapping subdomains (chunks) of equal length, each of which has the length π. In each chunk the components of Eq. 5.26 are approximated by an independent set of feed-forward neural networks. We train the time domains one-after-another and use the results from the previous chunk at the end time as the initial conditions for the next chunk. This domain decomposition strategy circumvents the difficulty of approximating functions of massive fluctuations with a single neural network, and thus will make the training process easier. We use an equal number of collocation points for all time subdomains, and set $n_t = 30$ and $n_x = 50$. Again, the samples of $\{t_c^s\}_{s=1}^{n_t}$ are drawn from a uniform distribution, and the spatial collocation points $\{x_c^k\}_{k=1}^{n_x}$ are equidistantly distributed in $[-\pi,\pi]$. For the collocation points in the stochastic space, we use eighth order Gauss-Legendre

quadrature rule for both ξ_1 and ξ_2, generating 64 collocation points in the probabilistic space. The same neural network setups are implememted for the following two test cases: the $\bar{u}_{nn}$, A_{nn} and Y_{nn} networks all have three hidden layers, each of which has 32 neurons, and the U_{nn} network is constructed with 3 hidden layers and 64 neurons per hidden layer. We only change the loss function in favor of either the DO or the BO condition.

Case 1: NN-DO Method

First, we test the NN-DO method. The initial conditions are taken directly from Eq. 5.50.

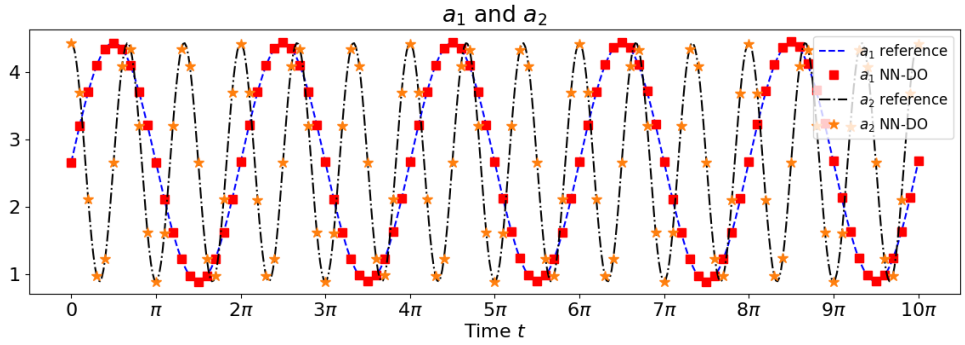


Figure 5.9: Stochastic Burgers' equation (NN-DO): A comparison of neural network approximations and exact solutions of the scaling factors a_i ($i = 1, 2$), as functions of t ($t \in [0, 10\pi]$).

Figure 5.9 shows the evolution of a_i ($i = 1, 2$) as time grows, where the low frequency component, a_1, and the high frequency component, a_2, co-exist at the same amplitude. They do not decay with time, indicating that the stochasticity in the system has already fully developed. In Figure 5.10, we compare the bases u_i ($i = 1, 2$) at $t = 10\pi$ obtained from the NN-DO method to the exact solutions, and in Figure 5.11, we plot the NN-DO solution mean and variance at two times, $t = 10\pi$ and $t = 5\pi$, versus the exact values. It is evident that the NN-DO solutions agree with the exact reference solutions very well. From Figure 5.11b we can observe

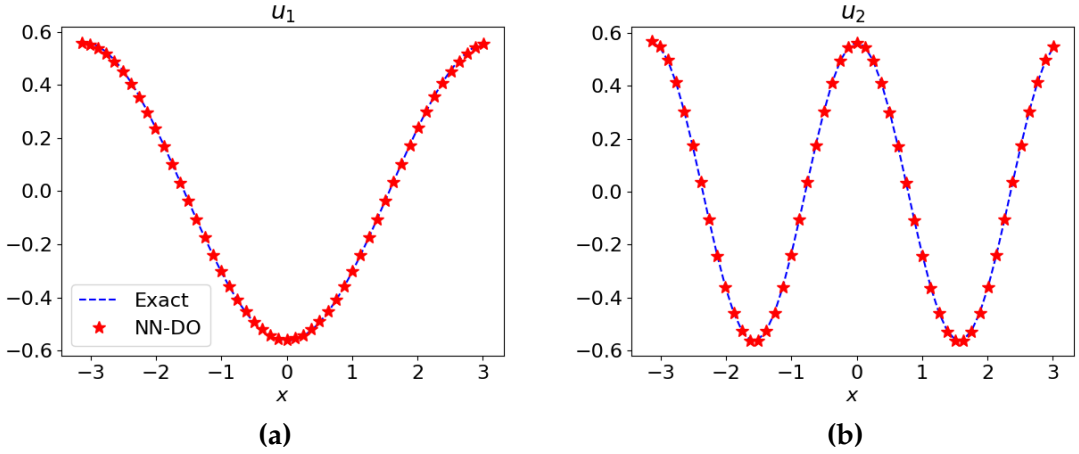


Figure 5.10: Stochastic Burgers' equation (NN-DO): A comparison of neural network approximations and exact solutions of the bases u_i ($i = 1, 2$) at the final time $t = 10\pi$. The red stars indicate the collocation points in the physical space.

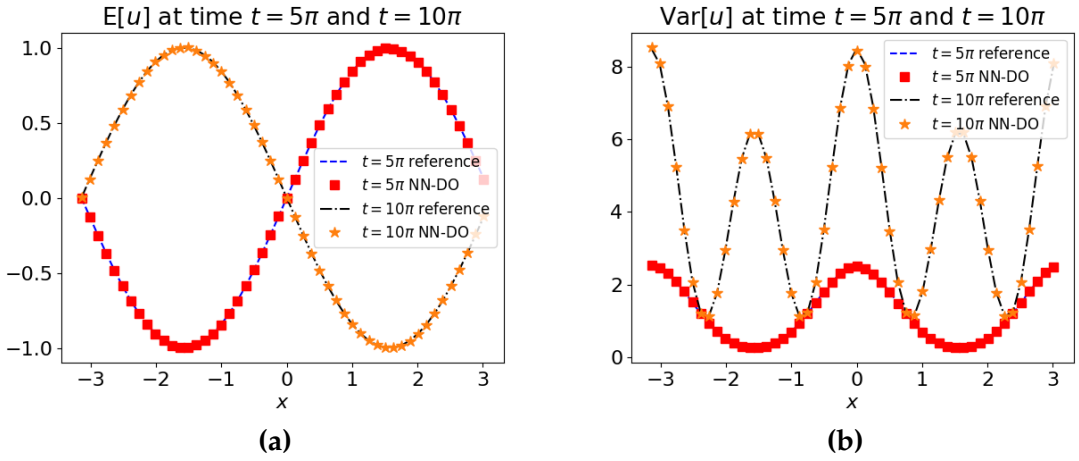


Figure 5.11: Stochastic Burgers' equation (NN-DO): Mean and variance of the solution at time $t = 5\pi$ and $t = 10\pi$, calculated using the NN-DO method. Both of them show good agreement with the reference exact value. The scattered points indicate the collocation points in the physical space.

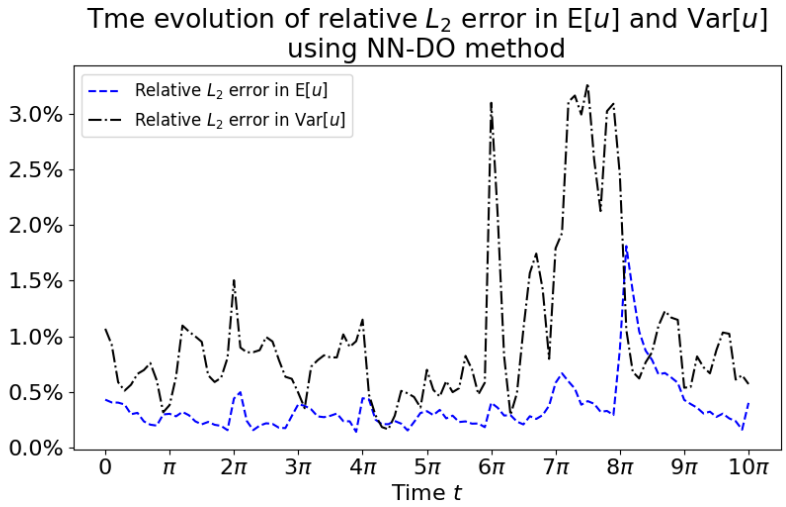


Figure 5.12: Stochastic Burgers' equation (NN-DO): Relative L_2 errors in the mean and variance calculated by the NN-DO method. The relative error in variance is slightly higher than the relative error in mean, and both errors are below 1% for most of the time.

that the solution variance evolves from $t = 5\pi$ to $t = 10\pi$ to develop a greater scale and a wavier shape, and the NN-DO method precisely captures this progress. Figure 5.12 shows the relative L_2 errors of the solution mean and variance, and in Table 5.3 we report both errors for all the DO components at the final time $T = 10\pi$. All the relative L_2 errors are around or less than 1%, indicating excellent performance of the proposed NN-DO method.

	$\mathbb{E}[u]$	$\mathbf{Var}[u]$	a_1	a_2
L_2 **error**	0.0029	0.0278	0.0104	0.0084
Relative L_2 **error**	0.40%	0.57%	0.35%	0.28%
	u_1	u_2	Y_1	Y_2
L_2 **error**	0.0042	0.0021	0.0008	0.0004
Relative L_2 **error**	1.04%	0.53%	0.62%	0.34%

Table 5.3: Stochastic Burgers' equation (NN-DO): The L_2 and relative L_2 errors of NN-DO solutions versus the exact solutions at the final time $T = 10\pi$.

Case 2: NN-BO Method

In this section we use the BO constraints to train the neural networks. Similar to the NN-DO counterpart, here we provide all the figures (Figure 5.13–Figure 5.16) showing a comparison between the NN-BO results and the reference exact solutions. To avoid redundancy, we refer readers to read the captions below the figures and will skip explaining each of them one-by-one. However, we would like to address that in Figure 5.13, the scaling factors a_i correspond to the eigenvalues in the classical BO method, and there is a significant amount of eigenvalue crossings during the whole time evolution, and also within each time chunk. In this situation, the classical BO method would fail due to the lack of explicit formulas for matrices M and S in Eq. 5.18. The proposed NN-BO method doesn't suffer from this restriction. In Table 5.3 we report that both the L_2 and relative L_2 errors for all the BO components the final time $T = 10\pi$. As with the NN-DO method, all relative L_2 errors are less than 1%, indicating excellent performance of the NN-BO method.

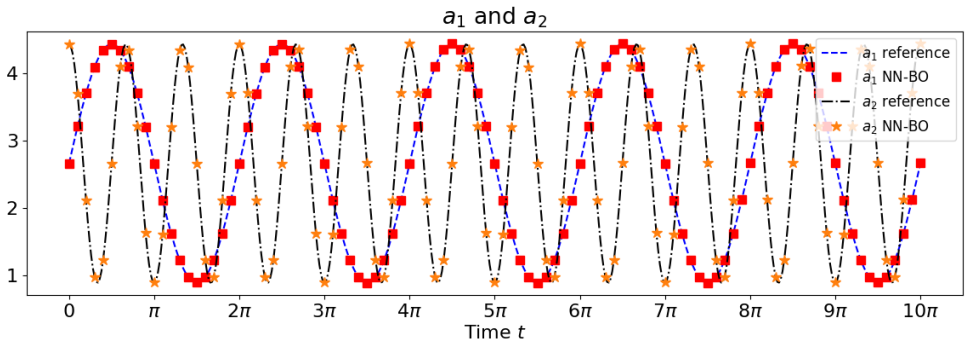


Figure 5.13: Stochastic Burgers' equation (NN-BO): A comparison of neural network approximations and exact solutions of the scaling factors a_i ($i = 1, 2$), as functions of t ($t \in [0, 10\pi]$). They corresponds to the eigenvalues in the classical BO method. As we can see, there is a significant amount of eigenvalue crossings during the whole time evolution and also within each chunk. Therefore, the classical BO method cannot be directly applied to this problem.

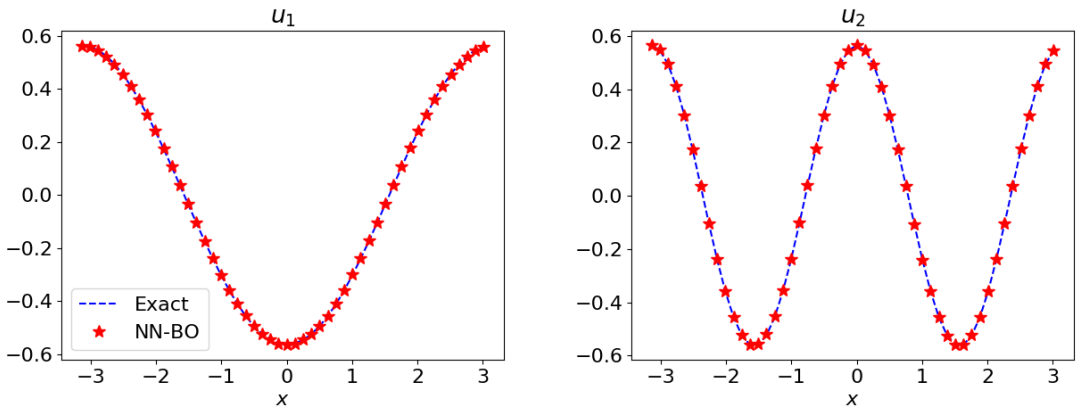


Figure 5.14: Stochastic Burgers' equation (NN-BO): A comparison of neural network approximations and exact solutions of the bases u_i ($i = 1, 2$) at the final time $t = 10\pi$. The red stars indicate the collocation points in the physical space.

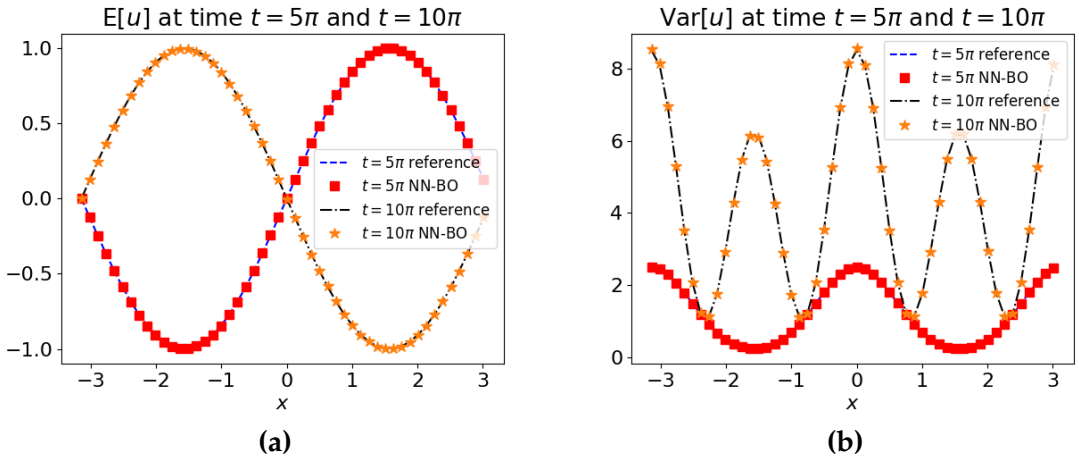


Figure 5.15: Stochastic Burgers' equation (NN-BO): Mean and variance of the solutions at time $t = 5\pi$ and $t = 10\pi$, calculated using the NN-BO method. Both of them show good agreement with the reference exact values.

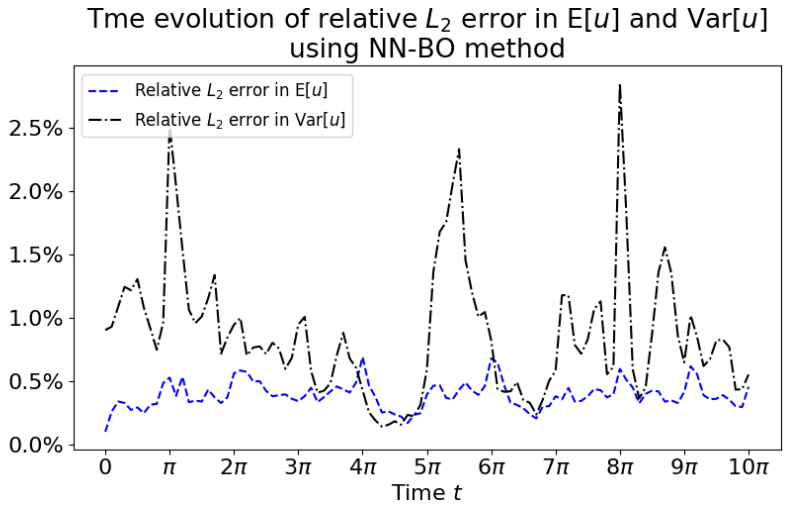


Figure 5.16: Stochastic Burgers' equation (NN-BO): Relative L_2 errors in the mean and variance calculated by the NN-BO method. The relative error in variance is slightly higher than the relative error in mean, and both errors are below 1% for most of the time.

	$\mathbb{E}[u]$	$\mathbf{Var}[u]$	a_1	a_2
L_2 **error**	0.0032	0.0267	0.0055	0.0073
Relative L_2 **error**	0.45%	0.55%	0.19%	0.25%
	u_1	u_2	Y_1	Y_2
L_2 **error**	0.0018	0.0020	0.0007	0.0005
Relative L_2 **error**	0.45%	0.49%	0.59%	0.39%

Table 5.4: Stochastic advection equation (NN-BO): The L_2 and relative L_2 errors of NN-BO solutions versus the exact solutions at the final time $T = 10\pi$.

5.5.3 Application to Nonlinear Reaction Diffusion Equation

Consider the following reaction diffusion equation with a nonlinear source term:

$$\frac{\partial u}{\partial t} = a u_{xx} + b u^2 + f(x;\omega), \quad \forall t \in [0,1] \text{ and } x \in [-1,1], \tag{5.54}$$

where the random force $f(x;\omega) = (1-x^2)g(x;\omega)$ is the source of randomness, while a and b are time-independent diffusion and reaction coefficients, respectively. The random process $g(x;\omega)$ is modeled as a Gaussian random field, i.e., $g(x;\omega) \sim$

$\mathcal{GP}(1, C(x_1, x_2))$, where $C(x_1, x_2)$ is a squared exponential kernel with standard deviation σ_g and correlation length l_c:

$$C(x_1, x_2) = \sigma_g^2 \exp\left(-\frac{(x_1 - x_2)^2}{l_c^2}\right). \tag{5.55}$$

The equation satisfies the Dirichlet boundary conditions, $u(-1, t; \omega) = u(1, t; \omega) = 0$, and the deterministic initial condition $u(x, 0; \omega) = -\sin(\pi x)$. Two different scenarios are considered here:

- Forward problem: coefficients a and b are given, and we solve for $u(x, t; \omega)$.
- Inverse problem: coefficients a and b are unknown but additional information for $u(x, t; \omega)$ is given, we solve for $u(x, t; \omega)$ while identifying a and b.

For the purpose of simplicity and demonstration, here we only show the results obtained from the NN-BO method as the NN-DO method exhibits a similar performance.

Forward Problem

In this case, we set the diffusion coefficient $a = 0.1$, and set the reaction coefficient $b = 0.5$. For the random force $f(x; \omega)$, we set $\sigma_g = 1$ and $l_c = 0.1$, making it a high-dimensional random process and thus 19 KL modes are needed to capture at least 98% stochastic energy of $f(x; \omega)$. The neural networks used in the NN-BO method are built as follows: $\bar{u}_{nn}$ has three hidden layers with 32 neurons per layer, U_{nn} and Y_{nn} have three hidden layers with 64 neurons per layer, and A_{nn} is composed of N independent neural networks (N is the number of BO

expansion terms), each of with has three hidden layers and four neurons per layer, approximating one single scaling factor a_i. This is because we expect that a_i will manifest in vastly different scales during the time evolution. We use $n_x = 51$ equidistantly distributed collocation points $\{x_c^k\}_{k=1}^{n_x}$ in space, $n_t = 50$ uniformly distributed collocation points $\{t_c^s\}_{k=1}^{n_t}$ in the time domain, and $n_l = 1000$ random samples $\{\xi_c^l\}_{l=1}^{n_\xi}$ in the 19-dimensional random space. The neural networks are trained with an Adam optimizer (learning rate 0.001) for 300000 epochs.

First, we investigate the performance of NN-BO method using six BO expansion terms. To obtain the reference solution for the BO decomposition, we numerically solving the original BO equations with the finite difference scheme in space and a 3rd-level Adam-Bashforth scheme in time. Due to the deterministic initial condition, in practice we start with a Monte Carlo method until $t = 0.01$, and then switch to solving the BO equations. To obtain the reference for the solution statistics we solve the SPDE using a Monte Carlo method with 1000 samples.

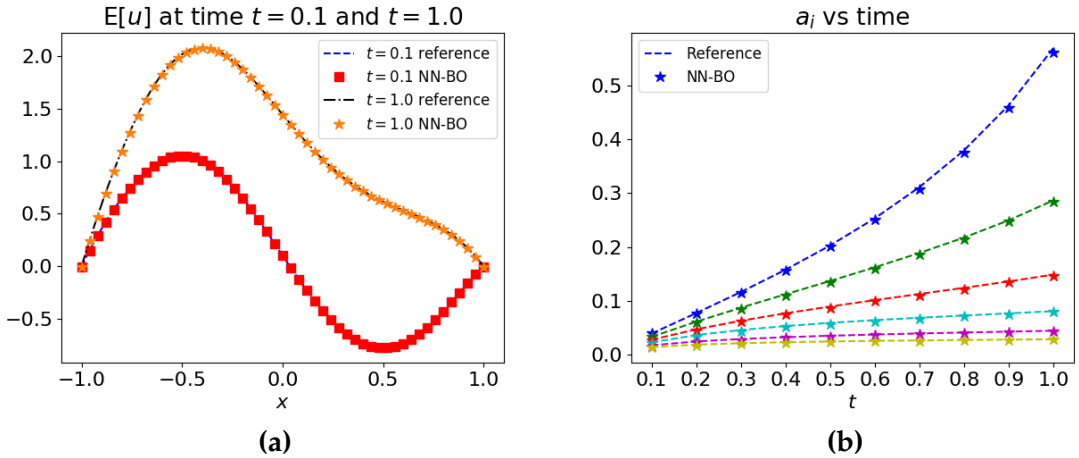


Figure 5.17: Stochastic reaction diffusion equation (forward): (a) solution mean at $t = 0.1$ and $t = 1.0$, while the reference mean is calculated from a Monte Carlo simulation. (b) scaling factors a_i at different time steps, while the reference a_i is calculated using the classical numerical BO method.

Figure 5.17a shows the NN-BO solution mean at $t = 0.1$ and $t = 1.0$, and Figure 5.17b shows the evolution of the scaling factors a_i, where the first four

BO modes gradually pick up energy as the result of the nonlinear source term, while the energy in the fifth and sixth modes is relatively stable. This illustrates the efficiency of the BO representation, i.e., only a small number of modes are necessary to capture most of the stochasticity in this high-dimensional SPDE. Figure 5.18 compares of the modal functions learned from the NN-BO method with the reference, and Table 5.5 displays the root mean squared error of the random coefficients Y_i. The proposed NN-BO method generates accurate predictions at both the early stage of the solution ($t = 0.1$) and the end time ($t = 1.0$).

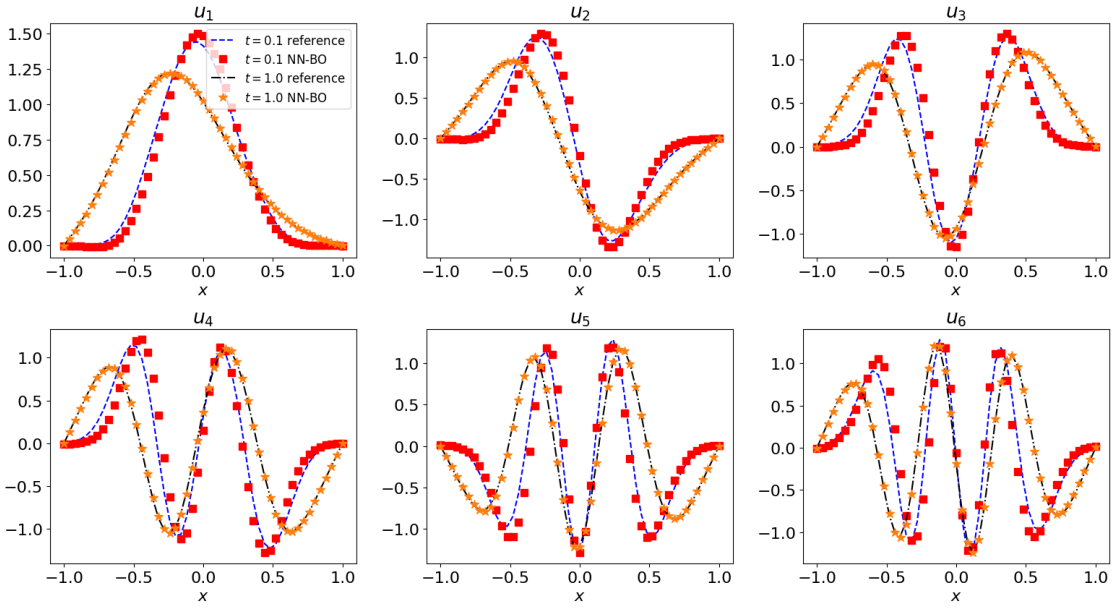


Figure 5.18: Stochastic reaction diffusion equation (forward): the BO modes u_i at $t = 0.1$ and $t = 1.0$, while the reference u_i are calculated using the classical numerical BO method.

RMSE	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6
$t = 0.1$	0.098	0.175	0.225	0.292	0.237	0.275
$t = 1.0$	0.042	0.039	0.045	0.050	0.061	0.057

Table 5.5: Stochastic reaction diffusion equation (forward): root mean squared error of the random coefficients Y_i calculated using the NN-BO method at $t = 0.1$ and $t = 1.0$, while the reference Y_i are calculated using the classical numerical BO method.

Next, we analyze the effect of the number of BO expansion modes by comparing the variances of solution calculated using five, six and seven BO modes.

Figure 5.19a shows the predicted variance at time $t = 1.0$ versus the reference Monte Carlo solution. The NN-BO method slightly underestimates the variance due to the truncated expansion. Figure 5.19b compares the relative L_2 error of solution variance obtained using three different methods: NN-BO, gPC and classical BO. The gPC method generates the largest error as it fails to capture the evolution of the system's stochastic structure due to the non-linearity, therefore, to achieve the same accuracy, one has to include a larger number of modes using the gPC method than using the BO method. Again, we can observe that a better accuracy can be achieved when more modes are included. The NN-BO method is less accurate than the classical numerical BO method, however, it circumvents the need of generate artificial stochastic initial conditions. Another advantage of the NN-BO method over the classical BO method is that it enables solving an time-dependent nonlinear *inverse* stochastic problem.

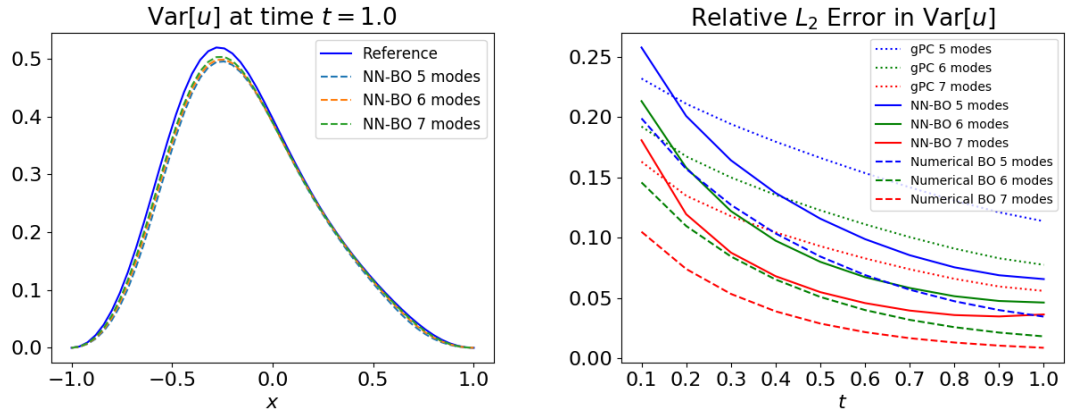


Figure 5.19: Stochastic reaction diffusion equation (forward): (a) variance of the NN-BO solution calculated using 5, 6 and 7 modes, while the reference variance is calculated from the Monte Carlo simulation. (b) comparing the L_2 errors of solution variance calculated by the NN-BO method, classical numerical BO method and the gPC method. The gPC method generates the largest error since it fails to capture the dynamic evolution of stochastic basis for nonlinear problems.

Inverse Problem

Again, we solve Eq. 5.54 but this time we pretend not knowing the exact diffusion and reaction coefficients a and b. Some extra information about $u(x,t;\omega)$ is provided to help us infer these two coefficients. In this example, the extra information is the mean value of $u(x,t;\omega)$ evaluated at three locations $x = -0.5, 0, 0.5$ and at two times $t = 0.1, 0.9$, i.e., a total of six measurements of $\mathbb{E}[u]$. We set $\sigma_g = 1$ and $l_c = 0.4$, the hidden values of a and b are selected to be 0.5 and 0.3, respectively. To solve this inverse problem, we use a BO representation with four modes, and adopt the same setup of the neural networks and collocation points as used in the forward problem. When setting up the PINNs, a and b are coded as "variables" instead of as "constants" so that they will be tuned at the training stage. Meanwhile, we include an additional term in the loss function that calculates the MSE of the predicted $\bar{u}_{nn}(x,t)$ versus the measurement data, so that the loss function will make use of the extra information to infer the coefficients. Without loss of generality, we choose both the initial values of a and b to be 1.0, and in practice these values could be chosen based on reasonable guesses. The neural networks are trained with the Adam optimizer (learning rate 0.001) for 300000 epochs. Same as the previous case, the reference solution statistics are calculated with Monte Carlo simulation and the reference BO components are generated by numerically solving the BO equations.

RMSE	Y_1	Y_2	Y_3	Y_4
$t = 0.1$	0.061	0.088	0.075	0.084
$t = 1.0$	0.019	0.041	0.039	0.060

Table 5.6: Stochastic reaction diffusion equation (inverse): root mean squared error of the random coefficients Y_i calculated using the NN-BO method at $t = 0.1$ and $t = 1.0$, while the reference Y_i are calculated from the forward problem using the classical numerical BO method.

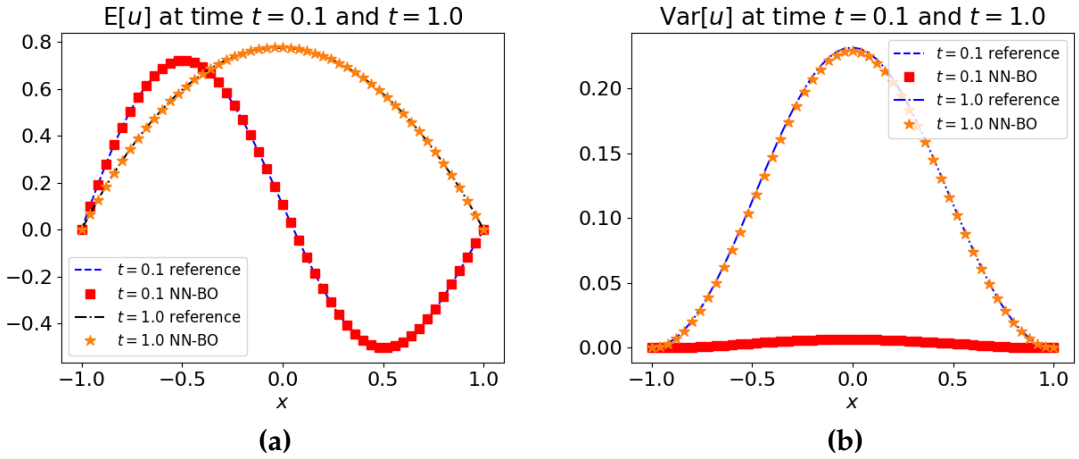


Figure 5.20: Stochastic reaction diffusion equation (inverse): mean (a) and variance (b) of the NN-BO solution at $t = 0.1$ and $t = 1.0$, while the reference solutions are calculated from the forward problem using the Monte Carlo method.

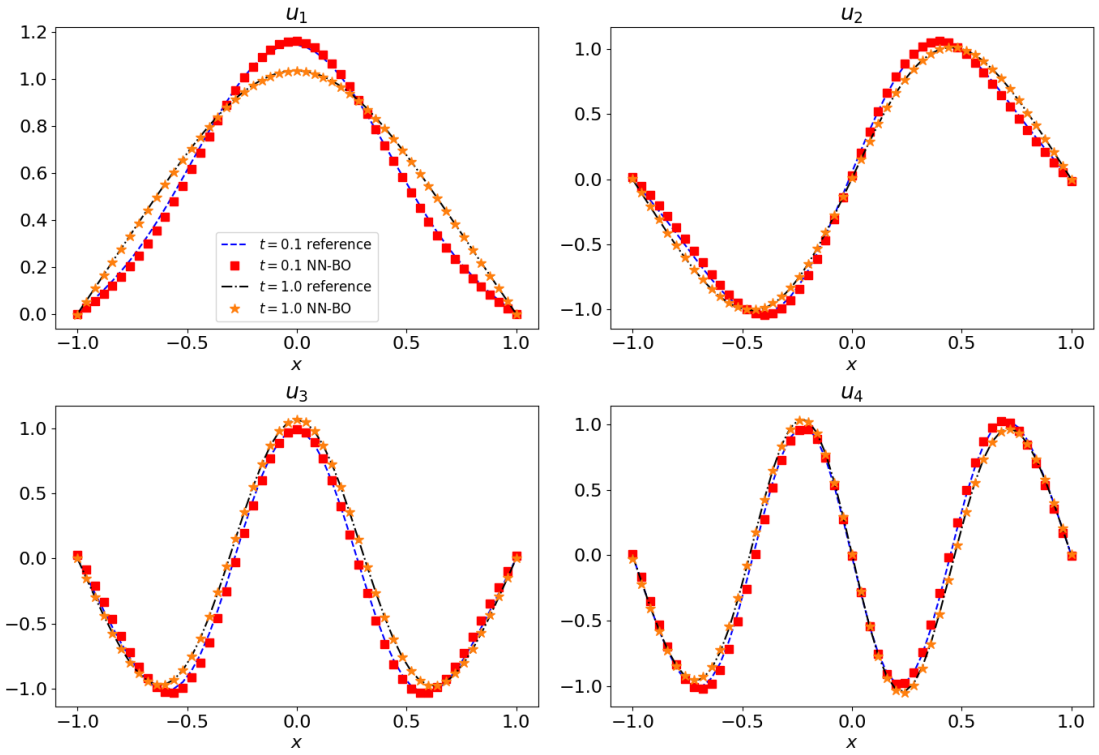


Figure 5.21: Stochastic reaction diffusion equation (inverse): the BO modes u_i at $t = 0.1$ and $t = 1.0$, while the reference u_i are calculated from the forward problem using the classical numerical BO method.

Figure 5.20a and Figure 5.20b shows the predicted solution mean and variance, respectively. Figure 5.21 and Figure 5.22a shows the predicted BO modes u_i and the scaling factors a_i. Table 5.6 displays the root mean squared errors of the random coefficients Y_i. It is evident that when compared to the references, the NN-BO method is still accurate at solving the inverse problem. Last but not least, we display the convergence property of the predicted a and b in Figure 5.22b and we can observe that the inferred values converge to the true values after less than 100000 training epochs.

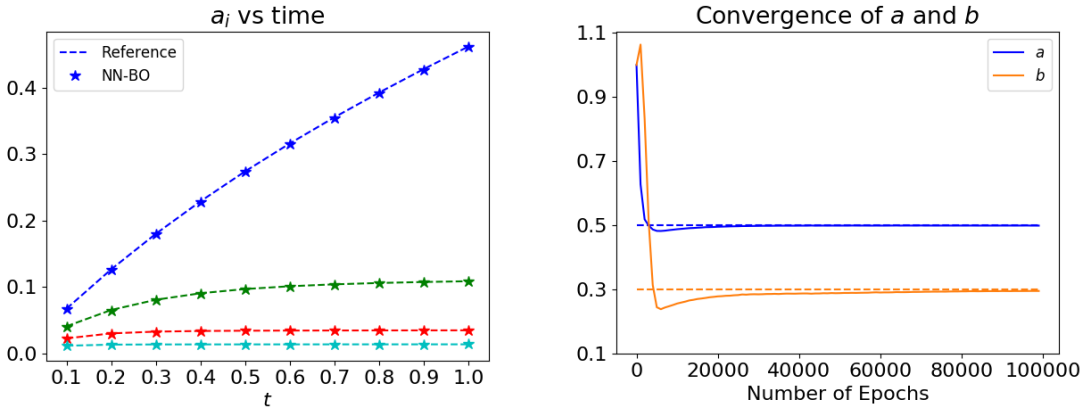


Figure 5.22: Stochastic reaction diffusion equation (inverse): (a) the evolution of scaling factors a_i by NN-BO, compared with the reference a_i calculated using the classical numerical BO method for a forward problem. (b) the convergence of predicted a and b to the true hidden values during the training process.

5.6 Summary of Chapter

To summarize, in this chapter we presented two numerical methods for solving time dependent stochastic partial differential equations (SPDEs), i.e. the NN-DO method and the NN-BO method. They both make use of the expressiveness of Physics-Informed Neural Networks (PINNs). Similar to the classical Dynamically Orthogonal (DO) and Bi-Orthogonal (BO) methods, the proposed methods use

either dynamical constraints on the spatial bases (NN-DO), or static constraints on both the spatial and the stochastic bases (NN-BO) to remove the time redundancy of the generalized Karhunen-Lòeve expansion. Since the loss functions of neural networks can be directly established from an implicit form of the DO/BO constraints, the proposed methods are free from the assumptions needed for deriving the classical DO and BO equations, and thus they shall be applied to a broader range of UQ problems. We demonstrate the performance of the NN-DO/BO methods with two artificially designed benchmark cases where exact DO/BO solutions exist, and we apply the NN-BO method to solve a time-dependent nonlinear diffusion reaction equation. Our numerical results show that the proposed NN-DO/BO methods are accurate for SPDEs with deterministic initial conditions and frequent eigenvalue crossings, and are reliable for long-time integration and high-dimensional random input. Moreover, as another advantage over the classical BO/DO method, the proposed method seamlessly solves the time-dependent stochastic inverse problems by encoding the extra information in the loss function while tuning the hidden parameters at the training stage. This was demonstrated in the last numerical example, and it exhibits huge potential of the NN-DO/BO method when applied to real physics/engineering applications.

Chapter Six

Conclusion

6.1 Summary

Thanks to the evolutionary development in electrical engineering and computer science, huge amount of data that were untouchable in the past can be cheaply stored and quickly processed nowadays. Research in machine learning, optimization and data science are booming, so does the need for making use of the valuable information from data as additional guidance to characterize stochastic complex systems. Motivated by the demand of modeling data-driven stochastic complex systems, in this dissertation, I developed four distinct strategies to addressed the challenges in developing effective data-driven stochastic complex systems models.

To address the difficulty of modeling the information fusion in heterogeneous data-driven stochastic systems, two methods are developed within the framework of domain decomposition. The stochastic domain decomposition via moment minimization (SDD-MM) method is developed to facilitate the uncertainty propagation between stochastic solvers that only exist in partial domains, and we derive boundary conditions for subdomains with different random dimensions. This method serves as a general framework that takes local random solvers as black boxes and can be used to solve problems involving multi-scale phenomena and hybrid stochastic systems, without the need of sampling continuous global trajectories. The Domain Decomposition with Gaussian Process Regression (GPDD) method is created to address the issue of incorporating information from both the classical physical laws and the data. Specifically, it is a domain decomposition algorithm designed to couple solutions in two types of domains, one with a classical partial differential equation (PDE) solver, named the PDE-domain, and the other one with sparse sensor data of multi-fidelities, i.e., the Data-domain. A numerical Gaussian Process Regression model is applied to the Data-domain to

infer the quantity of interest and to quantify the uncertainty in the predictions. The PDE-domain and the Data-domain are synchronized by the Schwarz alternating method, and the uncertainty in the prediction is spread to the global domain, resulting in a distribution of the predicted global solution. A combination of cheap low-fidelity sensors and expensive high-fidelity sensors contributes to better solutions, which is of great significance in practice, because in most applications one has to operate at limited budgets and resources.

To quantify the uncertainty of data-driven stochastic systems, two method are developed based on the Physics-Informed Neural Networks (PINNs). The NN-aPC method is designed for solving both stochastic forward (model inference) and inverse (model identification) problems. We use data collected from sensor measurements to build a set of arbitrary polynomial basis and learn the modal functions of the polynomial expansion via PINNs, i.e., DNNs that encode the underlying stochastic differential equation. Two different types of uncertainties are quantified, i.e., the parametric uncertainty due to the stochastic differential equation, as well as the approximation uncertainty of the PINNs, and the latter is measured by the dropout strategy. The approximation uncertainty represents how well the PINNs are trained and how robust it is at the predicting stage, thus can be served as effective guidance for active learning. And finally, the proposed NN-DO/BO methods focus on time-dependent stochastic problems, where the bases in both the physical space and the stochastic space evolve as a consequence of the development in the system's stochastic structure. The Dynamically Orthogonal/Bi-Orthogonal conditions shall be imposed seamlessly to the PINNs by making the best use of the flexiblility in designing the loss functions, thus avoiding making addition assumptions on the stochastic behavior of the solution. The NN-DO/BO method shall be readily applied to time-dependent stochastic inverse problems

once provided with extra information of the solution.

6.2 Future Work

The proposed methods shall be furthur developed and generalized. For example, the GPDD method is powerful at measuring the approximation uncertainty and it may be extended to quantify the measurement uncertianty or the parametric uncertainty, too. As another prospective research direction, the proposed methods are mutually compatible and thus could be combined to deal with problems where domain decomposition, information fusion and uncertainty quantification co-exist. For example, we may implement the NN-aPC method within the SDD-MM framework to analyze stochastic systems where a concrete stochastic partial differential equation (SPDE) only exists in some part of the domain, while in the rest of the domain we have collected historical sensor data, additionally, there are some unknown parameters to be estimated. We aim at reconstructing the full stochastic solution profile in the global domain while inferring the unknown SPDE parameters. Moreover, there is huge potential to be fullfiled by extending the proposed algorithms to real physics/engineering applications.

The future work will also focus on analyzing the PINNs. Great progresses have been made by using PINNs to model data-driven systems, however, as with most other popular machine learning tools, plenty of unknown properties of PINNs are yet to be discovered. For example, the distribution of weights for the components in a loss function is decided based largely on researchers' physical intuition, but there might be a systematic procedure of choosing the weights that will make training PINNs easier.

Bibliography

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (2016)*, pages 265–283, 2016.

[2] C. Angermueller, H. J. Lee, W. Reik, and O. Stegle. DeepCpG: Accurate prediction of single-cell DNA methylation states using deep learning. *Genome Biology*, 18(1):67, 2017.

[3] H. Babaee, M. Choi, T. P. Sapsis, and G. E. Karniadakis. A robust Bi-Orthogonal/Dynamically-Orthogonal method using the covariance pseudo-inverse with application to stochastic flow problems. *Journal of Computational Physics*, 344:303–319, Sept. 2017.

[4] I. Bilionis. Probabilistic solvers for partial differential equations. *arXiv e-prints*, page 160703526, 2016.

[5] C. Canuto and D. Funaro. The Schwarz algorithm for spectral methods. *SIAM journal on numerical analysis*, 25(1):24–40, 1988.

[6] Y. Chen, J. Jakeman, C. Gittelson, and D. Xiu. Local polynomial chaos expansion for linear differential equations with high dimensional random inputs. *SIAM Journal on Scientific Computing*, 37(1):A79–A102, Jan. 2015.

[7] M. Cheng, T. Y. Hou, and Z. Zhang. A dynamically Bi-Orthogonal method for time-dependent stochastic partial differential equations I: Derivation and algorithms. *Journal of Computational Physics*, 242:843–868, 2013.

[8] M. Cheng, T. Y. Hou, and Z. Zhang. A dynamically Bi-Orthogonal method for time-dependent stochastic partial differential equations II: Adaptivity and generalizations. *Journal of Computational Physics*, 242:753–776, 2013.

[9] H. Cho, X. Yang, D. Venturi, and G. E. Karniadakis. Algorithms for propagating uncertainty across heterogeneous domains. *SIAM Journal on Scientific Computing*, 37(6):A3030–A3054, 2015.

[10] M. Choi. *Time-dependent Karhunen-Loève type decomposition methods for SPDEs*. PhD thesis, Brown University, 2014.

[11] M. Choi, T. P. Sapsis, and G. E. Karniadakis. A convergence study for SPDEs using combined polynomial chaos and dynamically-orthogonal schemes. *Journal of Computational Physics*, 245:281–301, 2013.

[12] M. Choi, T. P. Sapsis, and G. E. Karniadakis. On the equivalence of dynamically orthogonal and bi-orthogonal methods: Theory and numerical simulations. *Journal of Computational Physics*, 270:1–20, 2014.

[13] A. Damianou and N. Lawrence. Deep Gaussian processes. In *Artificial Intelligence and Statistics (2013)*, pages 207–215, 2013.

[14] W. E, J. Han, and A. Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *arXiv e-prints*, page 170604702, June 2017.

[15] X. Fan, Y. Liu, J. Tao, and Y. Weng. Soil salinity retrieval from advanced multi-spectral sensor with partial least square regression. *Remote Sensing*, 7(1):488–511, 2015.

[16] D. Funaro and D. Gottlieb. Convergence results for pseudospectral approximations of hyperbolic systems by a penalty-type boundary treatment. *Mathematics of Computation*, 57(196):585–596, 1991.

[17] D. Funaro, A. Quarteroni, and P. Zanolli. An iterative procedure with interface relaxation for domain decomposition methods. *SIAM Journal on Numerical Analysis*, 25(6):1213–1236, 1988.

[18] Y. Gal and Z. Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning (2016)*, pages 1050–1059, 2016.

[19] Y. Gal and Z. Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in Neural Information Processing Systems (2016)*, pages 1019–1027, 2016.

[20] Y. Gal, J. Hron, and A. Kendall. Concrete dropout. In *Advances in Neural Information Processing Systems*, pages 3584–3593, 2017.

[21] M. Gander. Optimized Schwarz methods. *SIAM Journal on Numerical Analysis*, 44(2):699–731, Jan. 2006.

[22] T. Gerstner and M. Griebel. Numerical integration using sparse grids. *Numerical Algorithms*, 18(3-4):209–232, 1998.

[23] R. Ghanem and P. D. Spanos. Polynomial chaos in stochastic finite elements. *Journal of Applied Mechanics*, 57(1):197–202, 1990.

[24] T. Graepel. Solving noisy linear operator equations by Gaussian processes: Application to ordinary and partial differential equations. In *International Conference on Machine Learning (2003)*, pages 234–241, Washington, DC, USA, 2003. AAAI Press.

[25] L. L. Gratiet and J. Garnier. Recursive co-kriging model for design of computer experiments with multiple levels of fidelity. *International Journal for Uncertainty Quantification*, 4(5):365–386, 2014.

[26] R. Henderson and G. E. Karniadakis. Hybrid spectral-element-low-order methods for incompressible flows. *Journal of Scientific Computing*, 6(2):79–100, 1991.

[27] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv e-prints*, page 12070580, 2012.

[28] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

[29] S. P. Huang, S. T. Quek, and K. K. Phoon. Convergence study of the truncated Karhunen–Loéve expansion for simulation of stochastic processes. *International Journal for Numerical Methods in Engineering*, 52(9):1029–1043, Nov. 2001.

[30] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. In *Learning in graphical models*, pages 105–161. Springer, (1998).

[31] G. E. Karniadakis and S. Sherwin. *Spectral/hp element methods for computational fluid dynamics*. Oxford University Press, 2013.

[32] A. Kendall, V. Badrinarayanan, and R. Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *arXiv e-prints*, page 151102680, 2015.

[33] A. Kendall and Y. Gal. What uncertainties do we need in Bayesian deep learning for computer vision? In *Advances in Neural Information Processing Systems (2017)*, pages 5580–5590, 2017.

[34] Y. Khoo, J. Lu, and L. Ying. Solving parametric PDE problems with artificial neural networks. *arXiv e-prints*, page 170703351, 2017.

[35] D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. *arXiv e-prints*, page 14126980, Dec. 2014.

[36] D. P. Kingma and M. Welling. Auto-encoding variational Bayes. *arXiv e-prints*, page 13126114, 2013.

[37] D. A. Kopriva. *Spectral Element Methods*, pages 293–354. Springer Netherlands, Dordrecht, 2009.

[38] I. E. Lagaris, A. C. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, Sept. 1998.

[39] I. E. Lagaris, A. C. Likas, and D. G. Papageorgiou. Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, 11(5):1041–1049, Sept. 2000.

[40] H. Lei, J. Li, P. Gao, P. Stinis, and N. Baker. Data-driven approach of quantifying uncertainty in complex systems with arbitrary randomness. *arXiv e-prints*, page 180408609, Apr. 2018.

[41] Q. Liao and K. Willcox. A domain decomposition approach for uncertainty analysis. *SIAM Journal on Scientific Computing*, 37(1):A103–A133, 2015.

[42] P.-L. Lions. On the Schwarz alternating method. I. In *First international symposium on domain decomposition methods for partial differential equations*, pages 1–42. Paris, France, SIAM, 1988.

[43] P.-L. Lions. On the Schwarz alternating method. III: a variant for nonoverlapping subdomains. In *Third international symposium on domain decomposition methods for partial differential equations*, volume 6, pages 202–223. SIAM Philadelphia, PA, 1990.

[44] P. M. Lurie and M. S. Goldberg. An approximate method for sampling correlated random variables from partially-specified distributions. *Management science*, 44(2):203–218, 1998.

[45] D. J. C. MacKay. A practical Bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

[46] E. Musharbash, F. Nobile, and T. Zhou. Error analysis of the dynamically orthogonal approximation of time dependent random PDEs. *SIAM Journal on Scientific Computing*, 37(2):A776–A810, Jan. 2015.

[47] M. A. Nabian and H. Meidani. A deep neural network surrogate for high-dimensional random partial differential equations. *arXiv e-prints*, page 180602957, 2018.

[48] R. M. Neal. *Bayesian learning for neural networks*, volume 118 of *Lecture Notes in Statistics*. Springer, 1996.

[49] A. O'Hagan and M. Kennedy. Predicting the output from a complex computer code when fast approximations are available. *Biometrika*, 87(1):1–13, Mar. 2000.

[50] S. Oladyshkin and W. Nowak. Data-driven uncertainty quantification using the arbitrary polynomial chaos expansion. *Reliability Engineering and System Safety*, 106:179–190, 2012.

[51] J. Paisley, D. Blei, and M. Jordan. Variational Bayesian inference with stochastic search. *arXiv e-prints*, page 12066430, 2012.

[52] G. Pang, L. Yang, and G. E. Karniadakis. Neural-net-induced Gaussian process regression for function approximation and PDE solution. *arXiv e-prints*, page 180611187, June 2018.

[53] A. Papoulis and S. U. Pillai. *Probability, random variables and stochastic processes fourth edition*. McGraw Hill, 2002.

[54] P. Perdikaris, M. Raissi, A. Damianou, N. D. Lawrence, and G. E. Karniadakis. Nonlinear information fusion algorithms for data-efficient multi-fidelity modelling. *Proceedings of the Royal Society A*, 473(2198):20160751, Feb. 2017.

[55] P. Perdikaris, D. Venturi, and G. E. Karniadakis. Multifidelity Information Fusion Algorithms for High-Dimensional Systems and Massive Data sets. *SIAM Journal on Scientific Computing*, 473(2198):20160751, Jan. 2016.

[56] H. N. Pollack, S. J. Hurter, and J. R. Johnson. Heat flow from the Earth's interior: Analysis of the global data set. *Reviews of Geophysics*, 31(3):267–280, Aug. 1993.

[57] A. Quarteroni and A. Valli. *Domain decomposition methods for partial differential equations*. Oxford University Press, Jan. 1999.

[58] M. Raissi. Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations. *arXiv e-prints*, page 180407010, Apr. 2018.

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

[60] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Inferring solutions of differential equations using noisy multi-fidelity data. *Journal of Computational Physics*, 335:736–746, 2017.

[61] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Machine learning of linear differential equations using Gaussian processes. *Journal of Computational Physics*, 348:683–693, 2017.

[62] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics informed deep learning (part I): Data-driven solutions of nonlinear partial differential equations. *arXiv e-prints*, page 171110561, Nov. 2017.

[63] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics informed deep learning (part II): Data-driven discovery of nonlinear partial differential equations. *arXiv e-prints*, page 171110566, Nov. 2017.

[64] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Numerical Gaussian processes for time-dependent and nonlinear partial differential equations. *SIAM Journal on Scientific Computing*, 40(1):A172–A198, 2018.

[65] C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*, volume 1. MIT Press Cambridge, 2006.

[66] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv e-prints*, page 14014082, 2014.

[67] S. Rudy, A. Alla, S. L. Brunton, and J. N. Kutz. Data-driven identification of parametric partial differential equations. *arXiv e-prints*, page 180600732, 2018.

[68] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 3(4), 2017.

[69] T. P. Sapsis and P. Lermusiaux. Dynamically orthogonal field equations for continuous stochastic dynamical systems. *Physica D: Nonlinear Phenomena*, 238(23):2347–2360, 2009.

[70] S. Särkkä. Linear operators and stochastic partial differential equations in Gaussian process regression. In *International Conference on Artificial Neural Networks*, pages 151–158. Springer, 2011.

[71] H. A. Schwarz. Uber einige abbildungsaufgaben. *Journal für die reine und angewandte Mathematik*, 70:105–120, 1869.

[72] B. Smith, P. Bjorstad, W. D. Gropp, and W. Gropp. *Domain decomposition: parallel multilevel methods for elliptic partial differential equations*. Cambridge University Press, 2004.

[73] F. Song, C. Xu, and G. E. Karniadakis. A fractional phase-field model for two-phase flows with tunable sharpness: Algorithms and simulations. *Computer Methods in Applied Mechanics and Engineering*, 305:376–404, 2016.

[74] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, Jan. 2014.

[75] A. M. Stuart. Inverse problems: A Bayesian perspective. *Acta Numerica*, 19:451–559, 2010.

[76] D. N. Subramani and P. F. J. Lermusiaux. Energy-optimal path planning by stochastic dynamically orthogonal level-set optimization. *Ocean Modeling*, 100:57–77, Apr. 2016.

[77] A. M. Tartakovsky, C. O. Marrero, P. Perdikaris, G. D. Tartakovsky, and D. Barajas-Solano. Learning parameters and constitutive relationships with physics informed deep neural networks. *arXiv e-prints*, page 1808.03398v2, 2018.

[78] D. M. Tartakovsky and S. Broyda. PDF equations for advective–reactive transport in heterogeneous porous media with uncertain properties. *Journal of Contaminant Hydrology*, 120:129–140, 2011.

[79] M. Titsias and M. Lázaro-Gredilla. Doubly stochastic variational Bayes for non-conjugate inference. In *International Conference on Machine Learning*, pages 1971–1979, 2014.

[80] M. P. Ueckermann, P. F. J. Lermusiaux, and T. P. Sapsis. Numerical schemes for dynamically orthogonal equations of stochastic fluid and ocean flows. *Journal of Computational Physics*, 233:272–294, 2013.

[81] D. Venturi. A fully symmetric nonlinear biorthogonal decomposition theory for random fields. *Physica D: Nonlinear Phenomena*, 240(4):415–425, 2011.

[82] D. Venturi and G. E. Karniadakis. Convolutionless Nakajima–Zwanzig equations for stochastic analysis in nonlinear dynamical systems. In *Proc. R. Soc. A*, volume 470. The Royal Society, 2014.

[83] D. Venturi, D. M. Tartakovsky, A. M. Tartakovsky, and G. E. Karniadakis. Exact PDF equations and closure approximations for advective-reactive transport. *Journal of Computational Physics*, 243:323–343, 2013.

[84] X. Wan and G. E. Karniadakis. Multi-element generalized polynomial chaos for arbitrary probability measures. *SIAM Journal on Scientific Computing*, 28:901–928, 2006.

[85] N. Wiener. The homogeneous chaos. *American Journal of Mathematics*, 60(4):897, Oct. 1938.

[86] J. A. S. Witteveen and H. Bijl. Modeling arbitrary uncertainties using Gram-Schmidt polynomial chaos. In *44^{th} AIAA aerospace sciences meeting and exhibit (2006)*, page 896, 2006.

[87] D. Xiu. *Numerical methods for stochastic computations: A spectral method approach*. Princeton University Press, 2010.

[88] D. Xiu and J. S. Hesthaven. High-order collocation methods for differential equations with random inputs. *SIAM Journal on Scientific Computing*, 27(3):1118–1139, 2005.

[89] D. Xiu and G. E. Karniadakis. The Wiener–Askey polynomial chaos for stochastic differential equations. *SIAM Journal on Scientific Computing*, 24(2):619–644, 2002.

[90] X. Yang, R. Kwitt, and M. Niethammer. Fast predictive image registration. In *Deep Learning and Data Labeling for Medical Applications*, pages 48–57. Springer, 2016.

[91] X. Yang, G. Tartakovsky, and A. Tartakovsky. Physics-informed kriging: A physics-informed Gaussian process regression method for data-model convergence. *arXiv e-prints*, page 180903461, Sept. 2018.

[92] M. Zheng, X. Wan, and G. E. Karniadakis. Adaptive multi-element polynomial chaos with discrete measure: Algorithms and application to SPDEs. *Applied Numerical Mathematics*, 90:91–110, 2015.

[93] Y. Zhu and N. Zabaras. Bayesian deep convolutional encoder-decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415–447, 2018.