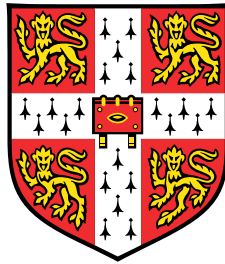


Variational Inference and Probabilistic Models for Parametric Partial Differential Equations



Arnaud Vadeboncoeur

Supervisor: Prof. Fehmi Cirak

Advisor: Prof. Mark Girolami

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

Declaration

This thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the Preface and specified in the text. I further state that no substantial part of my thesis has already been submitted, or, is being concurrently submitted for any such degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. It does not exceed the prescribed word limit for the relevant Degree Committee.

Arnaud Vadeboncoeur

July 2023

Abstract

Variational Inference and Probabilistic Models for Parametric Partial Differential Equations

Arnaud Vadeboncoeur

Parametric partial differential equations (PDEs) are of central importance to modern engineering sciences. They are the means for understanding the physical behaviour of systems for ranges of configurations and designs. The tools developed over the last few decades, such as finite elements, and finite volume methods are highly effective for single solution scenarios. However, these methods are ill-equipped in dealing with parametric problems as there is no information carry-over from one simulation to the next. This thesis is an attempt at adapting methods of probabilistic machine learning to create methodological advances in solving various problems relating to PDEs through variational inference and probabilistic models. The work is composed of three contributions. The first contribution lies in creating active learning surrogates for Bayesian inverse problems called *Active learning projected surrogates – SVGD*. Here we leverage Stein variational gradient descent methods to move clusters of particles through the information given by an active learning Gaussian process surrogate of the posterior surface posed by the classic Bayesian inverse problem framework. The second contribution comes from the development of a *physics driven deep latent variable model*. The developed variational inference framework leverages a virtual observable of a physics residual to inform the learning of jointly trained forward and inverse parametric PDE emulators. We implement our method with both finite element and physics informed neural network based residuals. The third contribution is the fundamental extension of the physics driven deep latent variable model into a function space view, inspired by neural processes we propose a variational inference framework leveraging *random grids and Gaussian processes for parametric PDEs*. The method treats partitions of PDE domains as random variables to be marginalized in training. Through this, we arrive at a random collocation method, shown to out-perform fixed grid methods. To implement this framework we also propose a new neural network architecture capable of effectively dealing with information given on random grids. We test our proposed methodologies on various PDEs relevant to engineering such as linear and nonlinear Poisson, nonlinear heat diffusion, thin-shells, wave, Burgers, and incompressible Navier-Stokes equations.

I dedicate this thesis to my loving parents and siblings.

Merci Louise, Louis, Claudia, Xavier et Anaïs.

Acknowledgements

I thank my supervisor, Professor Fehmi Cirak, for undertaking this long journey with me. He has tremendously helped my personal development as an independent researcher and critical thinker. Over the last few years he has pushed me to dig deeper, explore more, and refine ideas into useful research output. I also thank my advisor Professor Mark Girolami for his key advice and support in my work. I also thank them jointly for having created such a fruitful, dynamic, and inspiring environment in Cambridge. I owe special thanks to my colleagues and friends Ömer Deniz Akyildiz and Ieva Kazlauskaite, without whom my PhD experience would not have been complete. The collaborative work we have undertaken has shaped my PhD and will undoubtedly shape much of my academic path ahead. I also thank my friends, lab mates, and colleagues Connor, Ben, Alex, Justin, Yannis, Kim Jie, Tianyi, Eky, Oguzhan, Thomas, and Yanni. I would not have learned as much without you all.

My deepest thanks go to my parents Louise and Louis for having instilled in me a fascination for learning from early on. Their infinite support extends far beyond the confines of the PhD and I could never be too grateful for their love and help. I thank as well my siblings Claudia, Xavier, and Anaïs. From helping me move to the UK to discussing research ideas and general life advice, I've never doubted your support for a second.

I sincerely thank *La Fondation Baxter & Alma Ricard* for their invaluable financial support of my PhD. I would also like to thank *La Fondation Desjardins* for their financial help.

Table of contents

List of figures	xiii
List of tables	xv
1 Introduction	1
1.1 Parametric PDEs	2
1.2 Probabilistic Inference for PDEs	4
1.3 Aim, Contributions & Outline	7
2 Technical Background	9
2.1 Probabilistic Machine Learning	9
2.1.1 Probability	9
2.1.2 Methods of Bayesian Inference	13
2.2 Variational Inference	16
2.2.1 Variational Bayes	17
2.2.2 Stochastic Variational Inference	18
2.2.3 Jensen's Inequality	20
2.2.4 Neural Processes & Conditional Neural Processes	20
2.3 PDE Discretisation	21
2.3.1 Weighted Residual Methods	22
2.3.2 Finite Elements	23
2.3.3 Spectral Expansions	25
2.4 Machine Learning for PDEs	27
2.4.1 Physics Informed Neural Networks	27
2.4.2 Boundary Conditions	28
2.4.3 Operator Learning	29
3 Active Learning Surrogates: Accelerating Bayesian Inversion	33
3.1 Inverse Problems	33
3.2 Bayesian Inverse Problems	35

3.2.1	Sampling	37
3.2.2	Score Functions	42
3.2.3	Surrogate Modelling of Density Surfaces	45
3.3	Active Learning Projected Surrogates: ALPS-SVGD	46
3.4	Numerical Illustration	50
3.4.1	1D Poisson Problem	50
3.4.2	2D Poisson Problem	51
3.5	Discussion	53
4	Learning Parametric PDEs: Physics Driven Deep Latent Variable Models	55
4.1	Introduction	55
4.2	Contribution	56
4.3	Deep Probabilistic Models for Parametric PDEs	57
4.3.1	Finite Elements and Parametrisations	58
4.3.2	Probabilistic Model without Observed Data	60
4.3.3	Probabilistic Model with Observed Data	64
4.3.4	Algorithm and Implementation Details	66
4.4	Examples	67
4.4.1	1D Linear Poisson	68
4.4.2	1D Nonlinear Poisson Problem	73
4.4.3	Thin-Walled Flexible Shell	73
4.4.4	Nonlinear Heat Equation	79
4.4.5	Inhomogeneous Wave Equation	82
4.5	Discussion	84
5	Improved Learning of Parametric PDEs: Random Grids & Gaussian Processes	87
5.1	Introduction	87
5.2	Contribution	87
5.3	Reframing Φ -ML: Grid Invariance	88
5.3.1	Physics Informed ML for Forward Problems	89
5.3.2	Physics Informed ML for Inverse Problems	90
5.4	Model Derivation	90
5.4.1	Physics Informed Probabilistic Framework	90
5.4.2	Physics and Data Informed Model Derivation	93
5.4.3	Physics and Indirect Noisy Observations Informed Model	94
5.5	Low-Rank Covariance Matrix	97

5.6	Grid Invariant Inversion Networks	99
5.7	Experiments	101
5.7.1	The PDEs	101
5.7.2	Comparisons of Physics Informed Model	104
5.7.3	Comparison of Physics & Data Informed Model	104
5.7.4	Navier-Stokes Lid Driven Cavity Flow	107
5.8	Discussion	113
6	General Discussion	115
6.1	Aim	115
6.2	Summary of Contributions	115
6.3	Limitations and Future Work	117
6.4	Implications and Applications	118
	Appendix A PDDLVM	121
A.1	MNSE	121
A.2	Alternative Derivation of PDDLVM	121
A.3	Experimental Details	122
A.4	Additional Results for 1D Nonlinear Poisson	123
A.5	PINN-PDDLVM and Weighted Residuals	125
A.6	Additional Details	126
	Appendix B RGNP	129
B.1	Experimental Details	129
B.1.1	Nonlinear Poisson	129
B.1.2	Burgers	129
B.1.3	Navier-Stokes	130
B.2	Hardware	131
	References	133

List of figures

1.1	Forward and inverse problems	3
2.1	GP prior samples	16
2.2	First 5 Chebyshev polynomials	27
3.1	SVGD Gaussian kernel basis	40
3.2	1D log-density surrogate	46
3.3	ALPS 1D log-domain	46
3.4	1D sampling with ALPS	49
3.5	Evolution of gradients	49
3.6	ALPS samples 1D bar	51
3.7	ALPS results 2D diffusion field	52
3.8	ALPS 2D Poisson runtime information	52
4.1	Chebyshev and FEM function representations	58
4.2	Graphical models PDDLVM	60
4.3	Architectures FEM-PDLVM and PINN-PDDLVM	67
4.4	Linear Poisson results FEM-PDLVM	69
4.5	Parametric study of residual variance PDDLVM	70
4.6	Inverting the observation map	72
4.7	Nonlinear 1D Poisson problem FEM-PDLVM	74
4.8	Bunny mesh	75
4.9	Bunny results forward FEM-PDLVM	77
4.10	Bunny results inverse FEM-PDLVM	78
4.11	Nonlinear heat results FEM-PDLVM	81
4.12	Residual comparison PDDLVM	82
4.13	Inhomogeneous wave equation PINN-PDDLVM	83
5.1	GICNet visualisation	101
5.2	Nonlinear Poisson problem RGNP results	107
5.3	Burgers equation RGNP results	108

5.4	Observational data Poisson problem	111
5.5	Navier-Stokes RGNP results	112
A.1	MNSE parametric study FEM-PDDLVM	124
A.2	Convergence comparison residual variance PDDLVM	124

List of tables

4.1	MNSE 1D nonlinear Poisson PDDLVM	73
4.2	FEM against FEM-PDLVM results	77
4.3	Comparison PINN v.s. PINN-PDDLVM	79
4.4	Comparison FNO v.s. PINN-PDDLVM	84
5.1	Comparisons of physics informed models: MNSEs	105
5.2	Comparisons of physics informed models: UQ & Time	106
5.3	Comparisons of physics and data informed models: MNSEs	109
5.4	Comparisons of physics and data informed models: UQ & Time	109
5.5	Results for physics informed models on Navier-Stokes: MNSEs	110
5.6	Results for physics informed models on Navier-Stokes: UQ & Times	110
A.1	Basic information 1D Linear Poisson	126
A.2	Basic information 1D Linear Poisson Observable Map Inversion I	126
A.3	Basic information 1D Linear Poisson Observable Map Inversion II	126
A.4	Basic information 1D Nonlinear Poisson I	126
A.5	Basic information 1D Nonlinear Poisson II	126
A.6	Basic information Thin-Walled Flexible Shell	127
A.7	Basic information Nonlinear Heat Equation I	127
A.8	Basic information Nonlinear Heat Equation II	127
A.9	Basic information Inhomogeneous Wave Equation I	127
A.10	Basic information Inhomogeneous Wave Equation II	127

Chapter 1

Introduction

Partial differential equations are the fundamental way in which we are able to mathematically express the behaviour of physical systems. These equations are incomplete without a set of boundary conditions that ties the underlying dynamics to the physical setting in which they are expressed. Together, the PDE and boundary conditions form an enigma to be solved. The reward in solving this enigma is the ability to predict the behaviour of physical systems, a necessity for any engineering application. Some of the simpler equations may occasionally have closed form mathematical solutions, but in reality most equations pertinent to engineering are complicated, nonlinear, and relate to non-trivial geometries. Often the best we can achieve is a numerical approximation of the exact solution. Classical numerical methods such as finite elements methods (FEM) are the backbone of much of engineering disciplines because of their robustness in solving these equations. However, FEM is ill-equipped to deal with parametric physics scenarios. By parametric physics we mean settings where the dynamics, source terms, and boundary conditions may be subject to modulation by collections of parameters. This is due to the fact that these schemes cannot transfer information from the solution of a PDE for a given set of parameters to another. Everything must be re-solved again from scratch for each new set of parameters. On the other hand, inference methods in machine learning (ML) developed in the last few decades are specifically designed to infer behaviours of systems based on observations of the system under different conditions. An emerging idea is to leverage ML to enable the fast repeated solving of parametric PDEs. But of course this is a dangerous game, overconfidence and incorrect predictions have high consequences in engineering. When using data to predict the behaviour of systems based on observational data for engineering application, there is a duty to quantify the uncertainty these estimates carry.

First, in chapter 1 we describe the key concepts; then we introduce the technical background in chapter 2. Chapters 3, 4, 5 will contain our key novel contributions. Finally, chapter 6 will discuss the findings, implications, and future avenues of research.

1.1 Parametric PDEs

Parametric PDEs are PDEs which are modulated by sets of parameters that can be changed, adapted or calibrated. They arise in a number of contexts including optimisation for engineering design, uncertainty quantification (UQ), and inverse problems.

Optimisation for engineering design (Martins and Ning, 2021) is the task of finding the best design for a system according to some objective. This design will then be expressed in terms of parameters that can modulate material properties, geometry, dynamics, etc. The objective function might reflect the cost of an engineering structure, weight-to-strength ratio, or constructability and any combination thereof. Other examples of optimisation objectives include the lift to drag ratio of a wing or the maximal charge of a capacitor. These engineering objectives are often a carefully designed balance between desirable traits and various costs. In order to treat these problems quantitatively with effective optimisation algorithms we write out an optimisation objective as

$$\mathbf{z}^* = \underset{\mathbf{z}}{\operatorname{argmax}} J(\mathbf{z}), \quad \mathbf{z} \in \mathcal{Z}, \quad (1.1)$$

which can then be modified with additional constraints

$$h_i(\mathbf{z}) = 0 \quad i = 1, \dots, n, \quad (1.2)$$

$$g_j(\mathbf{z}) \leq 0 \quad j = 1, \dots, m. \quad (1.3)$$

Here, $\mathbf{z}$ might be a vector of continuous or discrete variables or a mixture of these (Pelamatti et al., 2018). In these engineering application there will be some physical system performance coming into the objective $J(\mathbf{z})$ which for every evaluation of this objective will require the solution to a set of differential equations. This setup would then call for the repeated solution of parametric PDEs.

UQ is a subfield of engineering preoccupied with understanding uncertainty in engineering contexts using computer models (Kennedy and O'Hagan, 2001). Forward UQ are problems where we want to understand the propagation of uncertainty of our

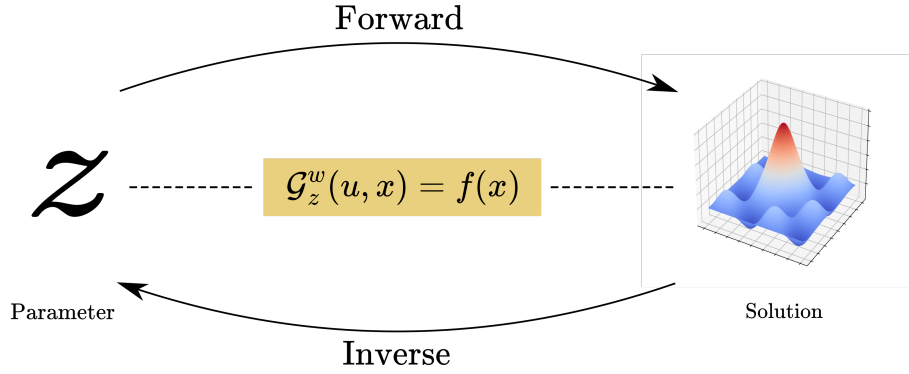


Fig. 1.1 A depiction of the relationship between the forward and inverse problems

model behaviour $p(G(\mathbf{z}))$ if we assume its parameters $\mathbf{z}$ are distributed according to some distribution $p(\mathbf{z})$. If our numerical model is then given by $G(\mathbf{z})$, one approach to quantify uncertainty is through Monte-Carlo sampling where we then obtain a distribution over a model behaviour by looking at the collection of evaluations $G(\mathbf{z}_i)$ for $\mathbf{z}_i \sim p(\mathbf{z})$. The model $G(\mathbf{z})$ would then be a parametric engineering model.

Inverse problems deal with the task of parameter identification. We want to identify what values of parameters are responsible for the observed behaviour of a parametrised system.

$$\text{Find } \mathbf{z}^* \text{ such that } \mathbf{y} = G(\mathbf{z}^*) + \mathbf{e},$$

for some stochastic error term $\mathbf{e}$ and a PDE model $G(\mathbf{z})$. In engineering application this is of central importance for tasks such as non-destructive testing (Cartz, 1995) where we investigate the internal qualities of a system without disturbing it. In order to do this in the inverse problem context we require a numerical model of the relevant dynamics that has parameters which can be changed to match the observed behaviour. In Fig 1.1 we show a depiction of the relationship between the forward and inverse problem.

Each of these fields revolve around the modelling of parametric PDE models of engineering systems. Unfortunately, the repeated evaluation of these models with classical numerical schemes can quickly become prohibitively computationally expensive. An important area of engineering is therefore devoted to the construction of sensible reduced order models (ROM) (Schilders et al., 2008) which are less-expensive to evaluate and yet capture the majority of the relevant dynamics. They present a trade-off between accuracy and efficiency. Techniques in this field include proper orthogonal decompositions (POD) (Astrid et al., 2008) and other surrogate modelling methods (Queipo et al., 2005, Sobester et al., 2008).

1.2 Probabilistic Inference for PDEs

Machine learning is a vast field and the words have become an umbrella term for most methods which algorithmically enhance predictions using data. Classes of method that fall into ML include supervised learning, which encompasses regression and classification; unsupervised learning, which includes generative models and clustering algorithms. There are also the subfields of self/semi-supervised learning which in the context of *physics-informed machine learning* (Φ -ML) usually means leveraging both data and *a priori* physics knowledge in the form of equations.

Probabilistic inference in the context of engineering has a long history. Some of the earliest efforts made use of Gaussian processes (GPs) to emulate computer code (Sacks et al., 1989, Diaconis, 1988, Gramacy, 2020). However, these methods may not be as scalable as other more modern approaches and may be limited in terms of richness of learned representations. Leveraging the most recent advances in ML is paramount to ameliorating methods for inference in the context of computational physics. This includes the use of neural networks (NN) and deep learning for function approximations (LeCun et al., 2015). These tools have proven to have immense potential for learning complex patterns from data. In our work, we will use neural networks as the work-horse at the centre of our proposed probabilistic frameworks.

The challenges associated with forward and inverse parametric PDE problems led to a proliferation of ML-based techniques that aim to aid the simulation of physical processes (Han et al., 2017, Sirignano and Spiliopoulos, 2018, Karniadakis et al., 2021). Although early approaches were primarily based on training ML regression models, such as random forests and deep NNs on observed or simulated physical data (Thuerey et al., 2021, Raveendran et al., 2014) and discovering governing physics laws from the data (Brunton et al., 2016), many recent methods focus on ML approaches where physics is an integral part of the model. This is achieved through physics-informed loss functions that combine the existing PDE-based parameterisation of physical processes with observed or simulated data (Kim et al., 2019), or through differentiable physics (Raissi et al., 2019, Seo and Liu, 2019, Rackauckas et al., 2021, Ardizzone et al., 2018, Zhao et al., 2022). The exceptional success of these methods brought tremendous attention to the emerging field of Φ -ML and physics informed neural networks (PINNs) (Karniadakis et al., 2021, Ghattas and Willcox, 2021). We summarise some of these advances and their relevance to our work below.

Modern Probabilistic Emulators: Probabilistic formulations of PDE-informed models vary from supervised approaches (Yang and Perdikaris, 2019b) and mixed

semi-supervised formulations (Yang and Perdikaris, 2019a), to uncertainty quantification procedures based on convolutional neural networks (CNN) (Winovich et al., 2019) and GPs combined with FEM (Cockayne et al., 2019, Girolami et al., 2021, Duffin et al., 2021, Akyildiz et al., 2022). The most relevant to our work are (Kaltenbach and Koutsourelakis, 2020, Rixner and Koutsourelakis, 2021) where the authors introduce the weighted residual method (which is a generalization of many classical numerical methods for PDEs (Strang and Freund, 1986, Chakraverty et al., 2019, Hatami, 2017, Finlayson, 2013, Lindgren, 2009, Gerald, 2004)), and treat the residual term as a pseudo-observation. This approach, however, differs from our framework in that we learn the direct mappings between the parameters and the solutions, while Rixner and Koutsourelakis (2021) introduce an intermediate low-dimensional embedding that does not have obvious physical interpretability.

Supervised and Semi-Supervised Operator Learning: Methods for learning differential operators are based either entirely or partially on *noiseless* PDE solution datasets created with classical numerical methods such as FEM or spectral methods. Such methods include Fourier Neural Operators (Li et al., 2020b, 2022, Fanaskov and Oseledets, 2022, Tripura and Chakraborty, 2022). Other methods which use a combination of pre-computed solutions from classical solvers and physics informed losses include Physics Informed Neural Operators and DeepONets (Li et al., 2021, Lu et al., 2021). Although these methods have been shown to be effective in certain scenarios, they fundamentally rely on classical solvers in order to learn, and so are bound to their natural limitations for learning new problems. The limitations include the need to recompute models from scratch for new parameter instances and rely on CPU-driven operations that may not parallelize in the same way neural networks can. Furthermore, these methods generally focus on learning the operator from one function space to another and are typically evaluated on fixed grids. This is restrictive in cases where the domain geometry, boundary, and initial conditions cannot be easily defined in a function space form. Instead, we focus our work on PDEs parametrised by sets of scalar coefficients (functions can easily be represented through a variety of spectral expansions). The parametric representation overcomes many of the limitations outlined before (Lu et al., 2022, Tang et al., 2023).

Physics Informed Variational Autoencoders: Several methods have been proposed to adapt the popular Variational Autoencoder (VAE) (Kingma and Welling, 2013) framework to solve various physics problems. Such methods include physics-informed VAEs (Zhong and Meidani, 2022, Zhu et al., 2019) and physics integrated VAEs (Takeishi and Kalousis, 2021). Some of these methods focus on the construction

of efficient and regularized inverse maps (O’Malley et al., 2019, Tait and Damoulas, 2020), while other methods construct additional latent spaces and relate them to physical observations (Qian et al., 2021, Zhong and Meidani, 2022). Related to this line of work, some seek to learn disentangled latent representations of physical processes (Jacobsen and Duraisamy, 2022). While these methods rely on variational inference (VI) like our approach, they aim at solving a fundamentally different problem as they relate the observable space to a *discovered* physical latent space. There are also techniques that use normalizing flows for solving stochastic differential equations in the inverse and forward direction (Guo et al., 2022).

Neural Processes: Neural processes are a new class of deep probabilistic regression tools (Garnelo et al., 2018b). Neural and conditional neural processes (Markou et al., 2022, Garnelo et al., 2018a) output uncertainty estimates around predictions similar to GPs, but they do not suffer from the same training scalability issues. These have partially been adapted to physics problems (Yang and Perdikaris, 2019b,a), however, in these works they are essentially data-driven rather than physics driven methods.

Methods related to single instances of solution fields are also pertinent (Long et al., 2022, Tronarp et al., 2022) as well as network parametrised physics informed GPs (Pang et al., 2019, Long et al., 2022, Zhang et al., 2022, Chen et al., 2021), but these are not adapted to parametric PDE scenarios (Bhattacharya et al., 2021). We also note work which uses invertible networks to solve inverse problems and the methods used for discovering dynamics (Ardizzone et al., 2018), as in Raissi and Karniadakis (2018), Brunton et al. (2016). However, these methods have different objectives than solving classic inverse problems and focus on hidden dynamics and discovering coordinate transformations.

To test our proposed inference frameworks we will use many comparison techniques from the field of ML, most notably in chapters 4 and 5. We will compare our method to alternatives in the literature. For cases requiring similar computational resources we will compare predictive errors with computed reference solutions. We will also compare computational resource requirements where pertinent and contrast the accuracy and reliability of the different methods and assess the quality of the proposed inference frameworks.

1.3 Aim, Contributions & Outline

This thesis aims to develop new methods that leverage the recent advances in probabilistic inference to extend the capabilities of classical numerical methods in the treatment of problems pertaining to parametric PDEs. There are three main novel contributions presented in this thesis.

1. The first contribution consists in developing an active learning surrogate methods for the efficient computation of PDE inverse problems. Here we tackle the challenge of efficiently sampling from a Bayesian posterior when the likelihood function includes the numerical solution of a parametrised PDE. We make use of projection techniques and sparse GPs to achieve computational speed-ups and easing of computational tasks by side-stepping the need for gradient computation of numerical models. We show our method on a series of numerical illustrations and outline the major limitations motivating the following works tackling similar issues. This work was presented at the UK association for computational mechanics (UKACM 2022) conference under the title of *Active Learning Projected Surrogates: ALPS-SVGD*.
2. The second contribution stems from the development of a novel VI framework for the simultaneous learning of probabilistic emulators for forward and inverse problems in parametric PDEs. We call this framework a physics driven deep latent variable model (PDDLVM). We adopt a latent variable perspective for this class of problems where the latent space has physical meaning. The framework makes use of modern deep learning frameworks to construct efficient and scalable forward and inverse emulators and use VI to output coherent uncertainty estimates of the predictions. We test our method on a series of PDEs of interest including thin shell equations, nonlinear heat dissipation and the inhomogeneous wave equation. This work is presented in the paper *Deep Probabilistic Models for Forward and Inverse Problems in Parametric PDEs*, Vadeboncoeur et al. (2022) (published in the *Journal of computational Physics*).
3. The third contribution of this thesis comes from building upon the work of PDDLVM and moving the inference scheme into a function space framework. Here, we directly parametrise our predictive distributions through probabilistic neural networks whose outputs are expressed as GPs. Doing this allows us to probabilistically treat the PDE domain and marginalize over input locations. We explore the meaning of this in chapter 5. To implement our method we propose a new neural network architecture called a grid invariant convolutional

network (GICNet) that can explicitly deal with functions evaluated at random locations. Furthermore, we demonstrate how to incorporate noisy observations of PDE solution fields into the inferential scheme. This work is presented in the paper *Random Grid Neural Processes for Parametric Partial Differential Equations*, Vadeboncoeur et al. (2023) (published at the *International Conference on Machine Learning* (ICML 2023)).

This thesis is organised as follows. In chapter 2 we go over relevant technical background for the proposed methods of the core chapters. Starting with general probabilistic ML, we summarise some basic, yet fundamental, probability concepts and Bayesian inference, which ties into a discussion on VI. From there we delve into the field of PDE discretisation through the lens of the weighted residual framework. We articulate these subjects together in our discussion of ML for physics. In chapter 3, after introducing a further few technical concepts related to Bayesian inverse problems, we develop our proposed method of active surrogate modelling for SVGD sampling of posterior distributions. We demonstrate the use of our method on a few numerical illustrations and outline the more pressing limitations of such approaches, thereby motivating the work in the following chapters. In chapter 4, we derive our physics driven deep latent variable model derived through weighted residuals and VI. We then test our method on a varied selection of PDE problems from simple parametric linear Poisson problems to shell equations and nonlinear heat dissipation and more. We conclude that chapter with some notes on limitations and future directions of work addressed in the next chapter. In chapter 5, we extend the PDDLVM framework into a function space view and incorporate observational data in a mixed inferential scheme thereby addressing many of the previous limitations of PDDLVM. To implement this proposed VI framework we develop a new neural network architecture. We test our methods in a data-rich and data-less setting for a nonlinear Poisson problem, a Burgers equation, and full incompressible Navier-Stokes simulations. Finally, in chapter 6 we conclude the work by outlining the novel contributions, implications, limitations and future direction of work.

Chapter 2

Technical Background

This chapter introduces the fundamental technical concepts underpinning the work presented in this thesis. We explore relevant concepts and go over notation as well as describe the uses for the presented methods. First, we cover probabilistic ML, with a brief overview of concepts in probability and Bayesian inference. We then discuss various pertinent methods of inference such as GPs and VI. Following this, we discuss PDE discretisations and the topic of weighted residuals, as the overarching framework for most numerical solving schemes. We then focus on FEM, followed by spectral parametrisation. This then brings us to combining methods in computational mechanics with ML. Finally, we discuss residual based inference methods, treatment of boundary value conditions, and operator learning.

2.1 Probabilistic Machine Learning

Probabilistic ML refers to the collection of methods that attempt to predict the behaviour of a system from data and quantify the uncertainty these predictions carry. The degree of correctness in the quantification of uncertainty can vary widely for different methods. From giving simple error bars of predictions to complete non-parametric estimates of probability densities over ranges of states and events. The appropriate degree of correctness in the method of uncertainty estimation and the corresponding trade-off in computational cost is chosen by the practitioner.

2.1.1 Probability

Probability is the mathematical treatment of randomness and attempts to understand and quantify uncertainty in variables of interest. In this sense probabilities can be

used to quantify our confidence about the state of a variable in a rigorous and principled manner. It allows one to relate uncertainties between variables describing different aspects of a system. Three fundamental properties of probability are: (1) the probability of an event is a non-negative real number, (2) a certain event has probability 1, (3) the sum of probabilities of mutually exclusive events is equal to the probability of their union (Kolmogorov and Bharucha-Reid, 2018, Papoulis and Unnikrishna Pillai, 2002).

Probability Rules

In order to manipulate the probabilistic relationship between variables we define two rules of probability. These are the marginal rule and the product rule. The marginal rule can be written as (Bishop and Nasrabadi, 2006)

$$p(y) = \int_{\mathcal{Z}} p(z, y) dz, \quad (2.1)$$

where z, y are random events and $\mathcal{Z}$ is the space of possible events of z . This rule allows us to recover probability distributions over selected variables from their joint distribution. If y is data, then (2.1) is the *marginal likelihood*. As we will see later, this plays a crucial role when constructing objective functions for ML. The second rule, the product rule, can be written as

$$p(z, y) = p(y|z)p(z) = p(z|y)p(y). \quad (2.2)$$

This rule relates joint distributions with their conditionals. It is important to note that conditioning a model in one direction or another is mathematically just as valid, but these conditional distributions will then describe different relationships. Using these simple rules, most models for probabilistic ML can be derived.

Taking the relationship in (2.2) we can rearrange to obtain

$$p(z|y) = \frac{p(y|z)p(z)}{p(y)}, \quad (2.3)$$

which is the famous Bayes' theorem. We will expand on it in chapter 3. If the value of $p(y)$ is not known, it is an intractable distribution.

Now, with two fundamental manipulation rules we can discuss statistical moments of distributions. The first four moments are the mean, variance, skewness, and kurtosis (Papoulis and Unnikrishna Pillai, 2002). In general, we are mostly interested

in the first and second moments. We define the expectation $f(z)$ with $z \sim p(z)$, as

$$\mathbb{E}_{z \sim p(z)}[f(z)] = \int_{\mathcal{Z}} p(z) f(z) \, dz. \quad (2.4)$$

We obtain the mean of $p(z)$, μ , if $f(z) = z$. Expectations and marginalization are very related as (2.1) can be expressed as $p(y) = \mathbb{E}_{z \sim p(z)}[p(y|z)]$. This property explains why expectations come up so often in learning objectives when maximising the marginal probability of an observed variable. The second moment of $p(z)$, also known as the variance of z , is obtained if $f(z) = (z - \mu)^2$ and it can be computed as

$$\text{var}[z] = \mathbb{E}[(z - \mu)^2] = \mathbb{E}[z^2] - \mathbb{E}[z]^2. \quad (2.5)$$

From this, are defined covariances in the case of the cross-variance of multiple variables.

Gaussians

The mean and variance are summary statistics to a few canonical distributions. The most prominent of these distributions is the normal distribution

$$\mathcal{N}(z; \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma^2} (z - \mu)^2\right), \quad (2.6)$$

where z is the variable distributed according to (2.6), μ is the mean of the distribution, and σ^2 is the variance of the distribution. The normal distribution comes up very often in ML because of its ease of manipulation and its natural appearance in sums of independent random events (McElreath, 2020).

The multivariate normal distribution (MVN) is defined as

$$\mathcal{N}(\mathbf{z}; \mathbf{m}, \mathbf{\Sigma}) = \frac{1}{\sqrt{(2\pi)^d |\mathbf{\Sigma}|}} \exp\left(-\frac{1}{2}(\mathbf{z} - \mathbf{m})^\top \mathbf{\Sigma}^{-1}(\mathbf{z} - \mathbf{m})\right), \quad (2.7)$$

where $\mathbf{\Sigma}$ must be positive semi-definite. We write certain identities for manipulating MVNs, the first of which is the conditioning property (Petersen et al., 2008). For some joint MVN with $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{m}, \mathbf{\Sigma})$ and

$$\mathbf{z} = \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \end{bmatrix}, \quad \mathbf{m} = \begin{bmatrix} \mathbf{m}_1 \\ \mathbf{m}_2 \end{bmatrix}, \quad \mathbf{\Sigma} = \begin{bmatrix} \mathbf{\Sigma}_1 & \mathbf{\Sigma}_3 \\ \mathbf{\Sigma}_3^\top & \mathbf{\Sigma}_2 \end{bmatrix}, \quad (2.8)$$

then the conditional distribution $p(\mathbf{z}_1|\mathbf{z}_2) = \mathcal{N}(\mathbf{z}_1; \bar{\mathbf{m}}, \bar{\Sigma})$, where

$$\bar{\mathbf{m}} = \mathbf{m}_1 + \Sigma_3 \Sigma_2^{-1} (\mathbf{z}_2 - \mathbf{m}_2), \quad (2.9)$$

$$\bar{\Sigma} = \Sigma_1 + \Sigma_3 \Sigma_2^{-1} \Sigma_3^\top. \quad (2.10)$$

The second MVN identity of importance to us, is the marginalization property. Simply put, if one is interested in the marginal distribution of a subset $\mathbf{z}_1$ of entries in the vector $\mathbf{z}$, we only need to ignore all other entries not in this subset $\mathbf{z}_1$. The marginal distribution of $\mathbf{z}_1$ can then be written as

$$\int p(\mathbf{z}_1, \mathbf{z}_2) d\mathbf{z}_2 = p(\mathbf{z}_1) = \mathcal{N}(\mathbf{z}_1; \mathbf{m}_1, \Sigma_1). \quad (2.11)$$

This property is of crucial importance in MVNs as marginalization is usually a non-trivial task for non-Gaussian joint distributions.

We discuss four types of covariance structure parametrisations, each with varying computational burdens: the full, low-rank, sparse, and diagonal parametrisations. The full covariance parametrisation is the most complete version and is expressed as $\Sigma_\theta = \mathbf{L}\mathbf{L}^\top$, where $\mathbf{L}$ is a lower triangle matrix. The low rank approximation is constructed as the product of a rectangular matrix as $\Sigma = \mathbf{A}\mathbf{A}^\top$. The sparse approximation takes advantage of spatial correlations (Povala et al., 2022) through a Laplacian matrix of spatial points. Finally, the diagonal approximation, $\Sigma = \text{diag}(\mathbf{a})$, parametrises only the diagonal elements of the matrix and assumes independence of random variables in different dimensions. It is also known as the mean-field approximation (Murphy, 2012). Kernel parametrisations for GPs are discussed in 2.1.2.

Dimensionality

We finish this section with a short discussion on the *curse of dimensionality*. If we are interested in computing an integral of the form given by (2.4) using a classic Riemann sum, then one would grid the interval of interest on real line with n rectangles. If we then treat a multivariate problem we would grid the n -dimensional interval with n^d intervals to obtain the same refinement as the 1D case. We can see how such an integral then becomes computationally intractable for anything more than a few dimensions. In problems pertinent to probabilistic ML we often deal with space of tens or hundreds or thousands of dimensions, rendering direct naive integration impractical. There are many other issues that arise when dealing with high-dimensional space and this will be a recurring theme throughout this work. In

general the *curse of dimensionality* occurs when we have a property that depends exponentially on the dimension of the problem. This comes up in regression tasks and in many Bayesian inference settings.

2.1.2 Methods of Bayesian Inference

Bayesian inference is a useful framework by which probabilistic ML methods can be derived. It relates prior distributions and predictive distributions by conditioning on data. The method of Bayesian inference can be described in 3 major steps (Gelman et al., 1995). Firstly, one postulates a joint probability model over all variables of interest. Secondly, we condition on observational data. Thirdly, we evaluate model fit. In probabilistic ML, to predict states for unseen data we can usually add another step: maximise the marginal likelihood with respect to model parameters. We can see how these steps relate to the basic rules of probability. Firstly, we write a joint model in terms of conditional distributions that capture the nature of the relationship between the variables. Secondly, we use the product rule to condition our model on data. Thirdly, we use marginalization to sum over all other variables and obtain the marginal likelihood. In the rest of this section we will go over practical methods of inference and algorithmic tools used in probabilistic inference.

Monte-Carlo Integration

This method stochastically computes integrals like the one in (2.4) (Robert et al., 1999). We write our Monte-Carlo integration as

$$I = \int_{\mathcal{Z}} p(z) f(z) \, dz, \quad (2.12)$$

$$I \approx \frac{1}{N} \sum_{i=1}^N f(z_i); \quad z_i \sim p(z), \quad (2.13)$$

which naturally follows from (2.4). The most notable consequence of this approach is that the error reduces given the number of samples N . It can be shown (Caflisch, 1998) that the error of the estimate of the integrand I converges at a rate of $\frac{1}{\sqrt{N}}$. This convergence rate is not a function of the dimensionality of the space $\mathcal{Z}$ as opposed to deterministic integration methods mentioned in Sec. 2.1.1 which depend exponentially on dimension.

If we can then express objective functions in terms of integrals, which often arises in probabilistic ML because of the desire to maximise marginal likelihoods through a Monte-Carlo sum. Furthermore, we can control the degree of approximation of this

integral by choosing the number of samples N . The method as described above works if we can sample from the distribution $p(z)$. If $p(z)$ is uniform, Gaussian, or any number of tractable distributions, this can be approximately done efficiently in most modern computing languages (Box and Muller, 1958, Matsumoto and Nishimura, 1998). If we cannot directly sample from $p(z)$ we resort to Markov Chain Monte Carlo methods which we will examine in Sec. 3.2.1.

Probabilistic Regression and Gaussian Processes

A Gaussian process (GP) is an infinite dimensional stochastic process in which any finite collection of samples from it are distributed according to a MVN distribution. GPs have the *consistency* property by which the treatment of a set of points is not affected by the treatment of a superset of points (Williams and Rasmussen, 2006). Furthermore, a GP is typically expressed through a mean and covariance function as

$$g(x) \sim \mathcal{GP}(m(x), k(x, x')), \quad (2.14)$$

where $m(x)$ is the mean function and $k(x, x')$ is a positive semi-definite kernel function measuring a kernelised distance between the points x and x' , where x denotes a point in a spatial domain $\mathcal{X}$. A GP expresses the probability of a function value g_i at spatial locations x_i as a Gaussian distribution. If we write this out for collection of g_i at multiple x locations we can write

$$p(\mathbf{g} | \mathbf{X}) = \mathcal{N}(\mathbf{g}; \mathbf{m}(\mathbf{X}), \mathbf{K}(\mathbf{X}, \mathbf{X})), \quad (2.15)$$

where $\mathbf{K}(\mathbf{X}, \mathbf{X})$ is the Gram matrix of the kernel function. The evaluation of (2.15) can be computed according to (2.7). Using Bayes rule (2.3) we can compute the posterior GP over $\mathbf{g}^*$ given some observational data $\mathbf{y}$. We start by posing a joint distribution over the value of the test locations $\mathbf{X}^*$ associated with the predicted function value $\mathbf{g}^*$ and the data locations $\mathbf{X}$ and data observations $\mathbf{y}$ where we assume (Rasmussen and Williams, 2004)

$$y = g(x) + e, \quad e \sim \mathcal{N}(0, \sigma_n^2). \quad (2.16)$$

then we have

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{g}^* \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} \mathbf{m}(\mathbf{X}) \\ \mathbf{m}(\mathbf{X}^*) \end{bmatrix}, \begin{bmatrix} \mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I} & \mathbf{K}(\mathbf{X}, \mathbf{X}^*) \\ \mathbf{K}(\mathbf{X}^*, \mathbf{X}) & \mathbf{K}(\mathbf{X}^*, \mathbf{X}^*) \end{bmatrix} \right). \quad (2.17)$$

Using the conditional rules given by (2.9) we can write down the predictive distribution $p(\mathbf{g}^*|\mathbf{y}) = \mathcal{N}(\mathbf{g}^*; \bar{\mathbf{m}}, \bar{\mathbf{K}})$

$$\bar{\mathbf{m}} = \mathbf{m}(\mathbf{X}^*) + \mathbf{K}(\mathbf{X}^*, \mathbf{X}) \left(\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I} \right)^{-1} (\mathbf{y} - \mathbf{m}(\mathbf{X})) \quad (2.18)$$

$$\bar{\mathbf{K}} = \mathbf{K}(\mathbf{X}^*, \mathbf{X}^*) - \mathbf{K}(\mathbf{X}^*, \mathbf{X}) \left(\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I} \right)^{-1} \mathbf{K}(\mathbf{X}, \mathbf{X}^*). \quad (2.19)$$

We can fit this model to better capture the data trends by tuning the mean and kernel function parameters to maximising the marginal likelihood (equivalent to maximising the log marginal likelihood) to maximise the probability of the data given the model. The marginal likelihood can be computed exactly as in Rasmussen and Williams (2004) (Williams and Rasmussen, 1995).

$$p(\mathbf{y}|\mathbf{X}) = \int p(\mathbf{y}|\mathbf{g}, \mathbf{X}) p(\mathbf{g}|\mathbf{X}) d\mathbf{g}, \quad (2.20)$$

which, when simplified and taking the log, yields

$$\log p(\mathbf{y}|\mathbf{X}) = -\frac{1}{2} \mathbf{y}^\top (\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I})^{-1} \mathbf{y} - \frac{1}{2} \log |\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I}| - \frac{n}{2} \log(2\pi). \quad (2.21)$$

The computational cost of using GPs is dominated by two distinct operations that carry most of the computational costs. These are the solution of the linear system of equations $\mathbf{K}(\mathbf{X}, \mathbf{X})^{-1} \mathbf{a}$ where $\mathbf{a} = (\mathbf{g} - \mathbf{m}(\mathbf{X}))$ and the evaluation of the log-determinant $\log |\mathbf{K}(\mathbf{X}, \mathbf{X})|$. Both operations scale $\mathcal{O}(n^3)$ for $\mathbf{K}(\mathbf{X}, \mathbf{X}) \in \mathbb{R}^{n \times n}$. We can incorporate prior information into the inference in three main ways: mean functions, kernels, and parameter priors.

It has been shown that if we are only interested in interpolation, we can forgo the mean function and use a simple zero-mean GP and centre and scale the data for most regression tasks. However, mean functions allow us to characterise how we expect the inputs and outputs to be related in the absence of data. That is to say, we can characterise the expected behaviour of our regression function in the out of data distribution regime. We will see in Chapter 3 how this can be a very useful tool in surrogate modelling.

Kernels can be chosen to characterise the nature of the correlation between data points (Genton, 2001, Stein, 1999). Through them, we can control the dynamics of the regression function itself. With the Matérn class of functions (Matérn, 2013, Whittle, 1954) we can control the smoothness of the regression function. The Matérn_{1/2} kernel yields C^0 non-differentiable functions; Matérn_{3/2} functions are once differentiable,

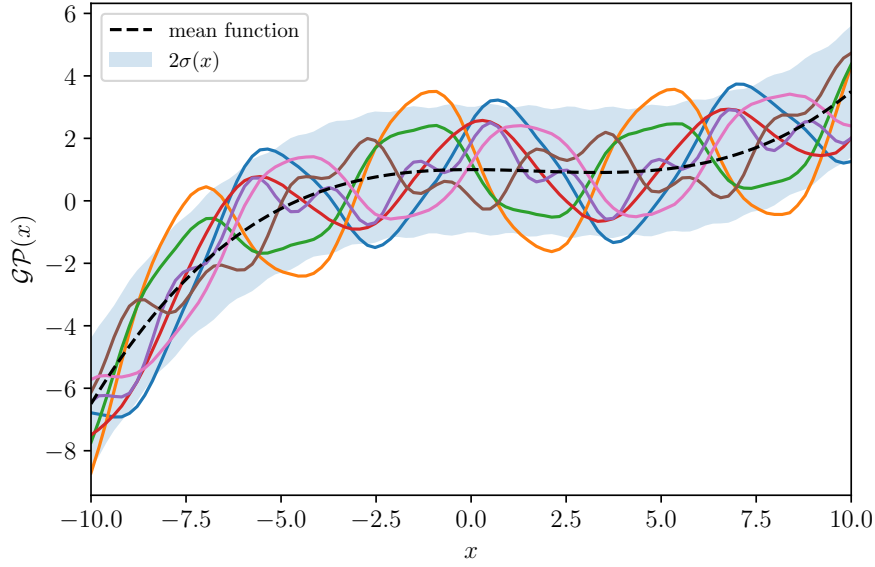


Fig. 2.1 GP prior samples (solid colored lines) and marginal $2\sigma(x)$ uncertainty regions (shaded blue).

and so on, until convergence to the infinitely smooth exponential quadratic kernel. In Fig 2.1 we see samples from a GP with a specified mean function and where the covariance function is the inner product between three trigonometric basis functions.

2.2 Variational Inference

To approximate distributions of interest with tractable parametrised distributions, many methods exist such as the Laplace approximation, also known as the *saddle point approximation* (Kass, 1990). This method relies on a truncation of a Taylor expansion at the mode of the distribution of interest and uses Hessian information of the true unnormalized distribution to construct the covariance matrix of a Gaussian approximation. Many issues arise when using this approximation (Rogers and Girolami, 2016) when the true distributions are even mildly non-Gaussian.

A highly successful approach to approximating intractable distributions comes from VI (MacKay and Mac Kay, 2003). We start by constructing a measure of dissimilarity, called $D(q||p)$, between the intractable target distribution $p(\mathbf{z})$, and the approximation $q(\mathbf{z})$. The important properties of $D(q||p)$ is $D(q||p) = 0$ iff. $q = p$ and that $D(q||p) \geq 0$. We restrict the set of approximating distributions to a family of distributions $\mathcal{Q}(\mathcal{Z}) \subset \mathcal{P}(\mathcal{Z})$, where $\mathcal{P}(\mathcal{Z})$ is the space of probability measures (Knoblauch et al., 2019) and $\mathbf{z} \in \mathbb{R}^{d_z}$ is the variable subject to inference.

We pose the problem of approximating the intractable distribution $p(\mathbf{z}) \in \mathcal{P}(\mathcal{Z})$ with $q(\mathbf{z}) \in \mathcal{Q}(\mathcal{Z})$ as (Blei et al., 2017)

$$q^* = \operatorname{argmin}_{q \in \mathcal{Q}} D(q||p). \quad (2.22)$$

The most popular $D(\cdot||\cdot)$ in probabilistic ML is the Kullback–Leibler divergence (Kullback and Leibler, 1951, Joyce, 2011) (KL-divergence) denoted as $D_{\text{KL}}(\cdot||\cdot)$, and defined as

$$D_{\text{KL}}(q||p) = \int_{\mathcal{Z}} \log \frac{q(\mathbf{z})}{p(\mathbf{z})} q(\mathbf{z}) \, d\mathbf{z}. \quad (2.23)$$

This divergence is non-symmetric, meaning the forward KL and backward KL do not measure the same thing. Of course the direction of the measure of dissimilarity ($D(q||p)$ v.s. $D(p||q)$ denoted as the backward and forward KL-divergence) can be changed depending on the application. The $D_{\text{KL}}(q||p)$ term can occasionally be evaluated analytically depending on the choice of prior and variational approximation. For example, the KL divergence has a closed form for measuring the divergence between two Gaussian distributions.

2.2.1 Variational Bayes

To derive the variational Bayes inference method, we write the KL divergence between the true posterior distribution of the parameters of interest given by $p(\mathbf{z}|\mathbf{y})$ and the variational approximation $q(\mathbf{z})$ (Blei et al., 2017),

$$D_{\text{KL}}(q(\mathbf{z})||p(\mathbf{z}|\mathbf{y})) = \int_{\mathcal{Z}} \log \frac{q(\mathbf{z})}{p(\mathbf{z}|\mathbf{y})} q(\mathbf{z}) \, d\mathbf{z}. \quad (2.24)$$

Remembering that $p(\mathbf{z}|\mathbf{y}) = p(\mathbf{y}|\mathbf{z})p(\mathbf{z})/p(\mathbf{y})$ we can write this as

$$D_{\text{KL}}(q(\mathbf{z})||p(\mathbf{z}|\mathbf{y})) = \int_{\mathcal{Z}} \log \frac{q(\mathbf{z})p(\mathbf{y})}{p(\mathbf{y}|\mathbf{z})p(\mathbf{z})} q(\mathbf{z}) \, d\mathbf{z}. \quad (2.25)$$

Through some manipulation we arrive at

$$\log p(\mathbf{y}) - D_{\text{KL}}(q(\mathbf{z})||p(\mathbf{z}|\mathbf{y})) = \int_{\mathcal{Z}} \log p(\mathbf{y}|\mathbf{z}) q(\mathbf{z}) \, d\mathbf{z} - \int_{\mathcal{Z}} \log \frac{q(\mathbf{z})}{p(\mathbf{z})} q(\mathbf{z}) \, d\mathbf{z}, \quad (2.26)$$

and as the KL divergence is by definition positive we have that

$$\log p(\mathbf{y}) \geq \mathbb{E}_{\mathbf{z} \sim q}[\log p(\mathbf{y}|\mathbf{z})] - D_{\text{KL}}(q(\mathbf{z})||p(\mathbf{z})). \quad (2.27)$$

This lower bound on the log marginal likelihood $\log p(\mathbf{y})$ is called the evidence lower-bound (ELBO). Maximising the ELBO acts as a proxy for maximising the log marginal likelihood.

In implementation, we need to choose a parametrised variational approximation $q_\theta(\mathbf{z})$, where θ represent the set of learnable parameters. The better this approximation can capture the variance of the posterior measure the better the inference will be, however it is not always possible to a priori know which approximation is best. Common choices of variational families include multivariate normal distribution with a optimised mean vector and covariance entries.

Of course many other VI schemes can be proposed depending on the chosen distance D and the variational family $\mathcal{Q}$ (Knoblauch et al., 2019).

2.2.2 Stochastic Variational Inference

As we saw before, the KL divergence consists of an integral over the space of random variables. These integrals found in the ELBO often do not have a closed form solution, and in high dimensions cannot be reasonably approximated through classical integration methods. Hence, the need to use Monte-Carlo integration. This results in a first source of stochasticity in the training objective. However, likelihood evaluations over the entire dataset may still be prohibitively expensive. Stochastic variational inference (SVI) improves the practicality and scalability of inference frameworks.

When maximising the ELBO, optimisation routines repeatedly evaluate the ELBO and its gradients. In doing this, much cost is associated with the likelihood term as it requires as many model evaluations as data points in the dataset. To approximate this efficiently, we can use a weighted average approach as

$$\log p(\mathbf{y}|\mathbf{z}) \approx \frac{N}{M} \sum_{i \in M} \log p(\mathbf{y}_i|\mathbf{z}), \quad (2.28)$$

for a mini-batch set M which is a random subset of the N data points $\mathbf{y}$. Mini-batching results in a second source of stochasticity in the estimate of the likelihood. This is known as a doubly-stochastic objective function and it is far more efficient to evaluate than the exact likelihood term which requires N model evaluations. This is the method of SVI (Hoffman et al., 2013).

Reparametrisation Trick

The minimization problem arising in VI contains a term which is the expectation of the likelihood function under the variational approximation $q_\theta(\mathbf{z})$ parametrised by θ . To optimise the ELBO, the derivate of this term with respect to the parameters θ is

$$\frac{\partial}{\partial \theta} \mathbb{E}_{\mathbf{z} \sim q_\theta(\mathbf{z})} [\log p(\mathbf{y}|\mathbf{z})]. \quad (2.29)$$

This poses an important question; How do we compute gradients of losses with respect to variables that act on the loss though stochastic samples? The answer to this question is to use the reparametrisation trick (Kingma and Welling, 2013). The idea is to insert an auxiliary variable $\boldsymbol{\varepsilon}$ that captures all stochasticity. This auxiliary sample is then deterministically transformed through $h(\boldsymbol{\varepsilon}, \theta)$ to obtain samples from $q_\theta(\mathbf{z})$. The gradients can then easily be computed as

$$\frac{\partial}{\partial \theta} \mathbb{E}_{\boldsymbol{\varepsilon} \sim p(\boldsymbol{\varepsilon})} [\log p(\mathbf{y}|h(\boldsymbol{\varepsilon}, \theta))] = \mathbb{E}_{\boldsymbol{\varepsilon} \sim p(\boldsymbol{\varepsilon})} \left[\frac{\partial}{\partial \theta} \log p(\mathbf{y}|h(\boldsymbol{\varepsilon}, \theta)) \right]. \quad (2.30)$$

If we choose $q(\mathbf{z})$ to be a MVN then

$$h(\boldsymbol{\varepsilon}, \theta) = \mathbf{m}_\theta + \mathbf{L}_\theta \boldsymbol{\varepsilon}, \quad (2.31)$$

where $\mathbf{m}_\theta$ is the parametrised mean vector and $\mathbf{L}_\theta$ is the parametrised Cholesky factorisation of $\boldsymbol{\Sigma}_\theta$. These operations are further simplified with the use of automatic differentiation linear algebra frameworks such as TensorFlow (Abadi et al., 2016), Jax (Bradbury et al., 2018) or Pytorch (Paszke et al., 2019).

Sparse Gaussian Processes with Variational Inference

There exist many variations of GPs with different kinds of approximations to the covariance matrix designed to alleviate the cubic in data complexity of using GPs. These variations include Nyström approximations (Williams and Seeger, 2000), sparse inducing points (Snelson and Ghahramani, 2005, Snelson, 2008, Rossi et al., 2021), variational methods for learning inducing point locations (Titsias, 2009, Matthews et al., 2016), and stochastic VI for GPs (Hensman et al., 2013). The methods of interest to this work are the ones related to inducing points methods. These methods use some reduced form of the covariance matrix by using an extra set of $\mathbf{S}$ of inducing points. These points can be chosen from the set of data locations $\mathbf{X}$ randomly or though some greedy optimisation scheme. They can also be outright optimised to maximise the marginal likelihood, but this tends to be computationally expensive.

2.2.3 Jensen's Inequality

Estimating marginal likelihoods is of central importance in ML and one of the primary ways of computing objective functions. These will usually have the form

$$p(y) = \int_{\mathcal{Z}} p_{\theta}(y, z) \, dz \quad (2.32)$$

where y is the data, z is the parameter, and θ is some set of parameter to optimise the likelihood and prior. We can approximate the integral through Monte-Carlo sampling. However, directly evaluating $p_{\theta}(y, z)$ is treacherous as it can often lead to numerical over or under flow as explained in Sec. 2.2.2. The question becomes, how can we only deal with log densities instead. We derive a lower bound on the integral for optimisation based on Jensen's inequality.

Jensen's inequality describes a mathematical property of concave and convex transformations of expectations (Royden and Fitzpatrick, 1988). For some concave function $\varphi(z)$ we can write Jensen's inequality as

$$\varphi(\mathbb{E}[z]) \geq \mathbb{E}[\varphi(z)], \quad (2.33)$$

and in the integral expression of expectations as

$$\varphi\left(\int_{\mathcal{Z}} f(z) \, dz\right) \geq \int_{\mathcal{Z}} \varphi \circ f(z) \, dz. \quad (2.34)$$

The composition of the log inside the integral applies only to the parts of the function subject to averaging, not to the parts of the function that characterise the distribution of the integration variables. We obtain

$$\log p(y) \geq \int_{\mathcal{Z}} \log [p_{\theta}(y|z)] p(z) \, dz, \quad (2.35)$$

$$\log p(y) \geq \mathbb{E}_{z \sim p(z)} [\log p_{\theta}(y|z)], \quad (2.36)$$

which is a lower bound on the log marginal likelihood. As it is a lower bound, we can then maximise it, giving us a proxy for maximising the marginal likelihood itself. Jensen's inequality will serve to obtain loss functions, as will be seen in chapters 4 and 5.

2.2.4 Neural Processes & Conditional Neural Processes

We now briefly discuss neural processes and conditional neural processes (Garnelo et al., 2018a,b, Markou et al., 2022). The models consist of neural network based

variational approximations to conditional distributions that express their approximations in terms of GPs. But differently to the previously discussed variational sparse GPs, these models do not rely on traditional Gram matrices but rather use NNs to directly control the covariance of random fields. These models are informed by data made available during training through the ELBO. Conditional neural processes do this for single instances of functions and act as probabilistic regression functions, whereas neural processes associate a particular instance of a function with a latent representation $\mathbf{z}$ and thus infer ensembles of functions. In Singh et al. (2019) they introduce sequential neural processes to capture temporal relationships explicitly. Other related models output GPs through neural networks Louizos et al. (2019), Lee et al. (2020), Dutordoir et al. (2022).

2.3 PDE Discretisation

Numerical methods for PDEs consists in the study and implementation of discretisation schemes for the solution of partial differential equations. These problems range from nonlinear solid mechanics, to fluid dynamics, and magnetics. The PDEs describing these problems often escape all methods for analytical solutions, and in some cases uniqueness of solutions may not even be guaranteed (Fefferman, 2000). Even for simple PDEs such as the Laplace equation which reads

$$\nabla^2 u(x) = f(x), \quad \text{for } x \in \Omega, \quad (2.37)$$

$$u(x) = 0, \quad \text{for } x \in \partial\Omega, \quad (2.38)$$

we need to resort to numerical methods if the domain Ω is complicated as is the case for many engineering applications. Even if solving these problems is very challenging, the predictive capabilities these models provide is well worth the effort needed to solve these equations.

As such, the PDE on a domain Ω , a bounded subset of $\mathbb{R}^d$ with boundary $\partial\Omega$, is expressed as

$$\mathcal{G}_z^w(u, x) = f(x), \quad \text{for } x \in \Omega, \quad (2.39)$$

$$\mathcal{B}_w(u, x) = 0, \quad \text{for } x \in \partial\Omega, \quad (2.40)$$

where $d \in \{1, 2, 3\}$, $\mathcal{G}_z^w$ and $\mathcal{B}_z$ are (possibly) nonlinear operators that describe the PDE and its boundary conditions respectively. The function $u(x)$ is the solution field, $f(x)$ is the source field, and $z(x)$ is a parameter of the PDE such as the

diffusivity field or a constant like the wave speed while w may represent parametrised initial conditions in (2.40) or inhomogeneous terms in (2.39). A general way to derive different numerical solution schemes for solving PDEs is given by the weighted residual methods.

2.3.1 Weighted Residual Methods

The method of weighted residuals provides a convenient framework for the derivation of many well-known PDE discretisation techniques (Strang and Freund, 1986, Finlayson, 2013, Lindgren, 2009, Gerald, 2004). Given the PDE of the form (2.39), the domain residual is defined as

$$\mathcal{R}(u, z, f, x) := \mathcal{G}_z^w(u, x) - f(x). \quad (2.41)$$

For simplicity of exposition the boundary term (2.40) is omitted. The treatment of boundary terms depends on the specific method derived as we will see later. We obtain a weighted residual functional equation by multiplying both sides by some arbitrary test function $v(x)$ and integrating over the domain

$$\int_{\Omega} v(x) (\mathcal{G}_z^w(u, x) - f(x)) \, dx = \int_{\Omega} v(x) \mathcal{R}(u, z, f, x) \, dx. \quad (2.42)$$

From there we choose a set of test functions $\{v_i(x)\}_{i=0}^n$ against which to integrate the residual. We further pose that $\hat{u}(x)$ is the *approximant* of the solution field $u(x)$ and that $\hat{u}(x)$ is parametrised in terms of the vector $\mathbf{u}$. The form of the relationship between the function $\hat{u}(x)$ defined everywhere on Ω and the vector $\mathbf{u}$ describing $\hat{u}(x)$ depend on the approximation technique chosen. Similarly, we pose that the PDE parameter field $z(x)$ and forcing field $f(x)$ are represented by approximants $\hat{z}(x)$ and $\hat{f}(x)$ respectively parametrised by $\mathbf{z}$ and $\mathbf{f}$. We can assemble the resulting discrete system of equations into a vectorized form

$$\mathbf{A}_{\mathbf{z}}(\mathbf{u}) = \left[\int_{\Omega} v_i(x) \mathcal{G}_{\hat{z}}^w(\hat{u}, x) \, dx \right]_{i=1:n}, \quad (2.43)$$

$$\mathbf{f}^{\text{WR}} = \left[\int_{\Omega} v_i(x) \hat{f}(x) \, dx \right]_{i=1:n}, \quad (2.44)$$

Here, $\mathbf{A}_{\mathbf{z}} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is a discretised nonlinear mapping of the differential operator. It allows us to express our discrete residual compactly as

$$\mathbf{r} := \mathbf{A}_{\mathbf{z}}(\mathbf{u}) - \mathbf{f}^{\text{WR}}, \quad (2.45)$$

with $\mathbf{r}, \mathbf{u}, \mathbf{f}^{\text{wr}} \in \mathbb{R}^n$. This residual description is agnostic to the particular method of discretisation used in practice. We stress the generality of using the weighted residual formulation, as noted by (Rixner and Koutsourelakis, 2021), as different choices for the test function $v(x)$ results in different well-known discretisation methods. Choosing $v_i(x) = \delta(x - x_i)$ results in a collocation method (Strikwerda, 2004), and for $v_i(x) = \delta_{ij}$ for $x \in \Omega_j$ we obtain a subdomain (finite volume) method (Eymard et al., 2000). For $v_i(x) = \partial \mathcal{R}(\hat{u}, \dots) / \partial u_j$ we have a least-squares method (Finlayson and Scriven, 1966), and for $v_i(x) = x^{m-1}$ a moment method (used in, *e.g.*, electromagnetism (Gibson, 2021)). For $v_i(x) = \phi_i(x)$ we obtain the family of Galerkin methods (Ern and Guermond, 2004). This provides a spectral method if $v_i(x)$ is globally defined over the entire domain Ω ; it can also be FEM if $v_i(x) \neq 0$ only for $x \in \Omega_j \subset \Omega$, which are locally defined basis functions.

2.3.2 Finite Elements

Finite elements is one to the most popular numerical solving schemes. Numerical implementation of this method can be easily done in simple case, but complex geometry (Mohd-Yusof, 1997, Logg et al., 2012), nonlinear PDEs (Belytschko et al., 2014), and isogeometric methods (Cottrell et al., 2009) make self-implementation of this type of code extremely challenging. We will first discuss the relationship between FEM and the weighted residual framework, then the practical considerations in using FEM.

Looking at the Sec 2.3.1 the *approximant* of $u(x)$ is given by

$$\hat{u}(x) = \sum_i u_i \phi_i(x), \quad (2.46)$$

where $\phi_i(x)$ are the FEM basis functions defined on a mesh. We pose our test functions $v_i(x) = \phi_i(x)$. The coefficients u_i on the boundaries are chosen to respect the (Dirichlet) boundary conditions. The mesh allows to both define the basis functions and describe the geometry of the domain Ω . Integration by parts provides the *weak form* of (2.42) prior for finite element discretisation (Brenner et al., 2008). The weak form reduces the order of differentiation of the original PDE and transfers some of these order of differentiation to the test function $v_i(x)$. This reduced differentiability requirement on the basis functions is one of the main strengths of using the weak form and it generally makes numerical schemes more stable and efficient. Once discretised, this weak form leads to a general system of equations whose dimensionality depends on the number of mesh nodes $\mathbf{u}$. Using (2.43) and (2.44) we

have the vectors $\mathbf{u}, \mathbf{f}^{\text{WR}} \in \mathbb{R}^n$, and the discrete system of equations

$$\mathbf{A}_{\mathbf{z}}(\mathbf{u}) = \mathbf{f}^{\text{WR}}, \quad (2.47)$$

where $\mathbf{A}_{\mathbf{z}} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is a possibly nonlinear map. Finding FEM coefficients $\mathbf{u}$ for fixed $\mathbf{z}$ is the forward problem. If nonlinear, the system of equations (2.47) is typically solved approximately using a fixed-point iteration technique, like the Newton-Raphson (Gil et al., 2007), and requires the repeated solution of sparse system of equations (Wriggers, 2008). If the system is linear we can use fast numerical solvers that scale well due to the sparse nature of FEM matrices. Some of these fast linear solvers include sparse LU decomposition (Bunch and Rose, 2014). In all cases, (2.47) is satisfied approximately by the solution coefficients $\mathbf{u}$, and we then say $\hat{u}(x)$ is a numerical solution to the PDE in question.

Poisson Equation

We show below an example derivation of the FEM form for a 1D Poisson problem (Reddy, 2005). For this problem in 1D we have

$$0 = \frac{d}{dx} \left(\kappa(x) \frac{du(x)}{dx} \right) + f(x), \quad (2.48)$$

for $\Omega = [0, 1]$ and boundary conditions $u(0) = u(1) = 0$. Here $\kappa(x)$ is the diffusion field in the Poisson equation and corresponds to $z(x)$ in Sec. 2.3.1. We multiply both sides by $v(x)$ and integrate over the domain

$$\int_0^1 v(x) \mathcal{R}(u, f, x) dx = \int_0^1 v(x) \frac{d}{dx} \left(\kappa(x) \frac{du(x)}{dx} \right) dx + \int_0^1 v(x) f(x) dx. \quad (2.49)$$

Doing integration by parts and setting the residual to zero we obtain

$$0 = v(x) \kappa(x) \frac{du(x)}{dx} \Big|_0^1 - \int_0^1 \kappa(x) \frac{dv(x)}{dx} \frac{du(x)}{dx} dx + \int_0^1 v(x) f(x) dx. \quad (2.50)$$

With Dirichlet boundary conditions, we choose to impose test functions with zero value at the boundaries. We select as the test functions $v_i(x)$ to be Lagrange polynomial hat functions and similarly the arbitrary test functions to be the same

basis as the trial solution. Hence, a Galerkin method;

$$\hat{u}(x) = \sum_i^N \phi_i(x) u_i, \quad (2.51)$$

$$\hat{v}(x) = \sum_j^N \phi_j(x) v_j, \quad (2.52)$$

where u_i and v_j are scalar coefficients in the Galerkin expansion. We obtain

$$\int_0^1 \kappa(x) \sum_j^N \sum_i^N u_i v_j \frac{d\phi_j(x)}{dx} \frac{d\phi_i(x)}{dx} dx = \int_0^1 \sum_j^N v_j \phi_j(x) f(x) dx. \quad (2.53)$$

Since $v(x)$ is an arbitrary test function, we can choose an instance k of $v_{1:N}$ such that $v_{j=k} = 1$ and $v_{j \neq k} = 0$. This yields $k = 1, \dots, N$ equations

$$\sum_i^N u_i \int_0^1 \kappa(x) \frac{d\phi_j(x)}{dx} \frac{d\phi_i(x)}{dx} dx = \int_0^1 \phi_j(x) f(x) dx. \quad (2.54)$$

We then collect the terms into matrix form with

$$\mathbf{A} \mathbf{u} = \mathbf{f}^{\text{WR}}, \quad (2.55)$$

$$\mathbf{A}_{ij} = \int_0^1 \kappa(x) \frac{d\phi_j}{dx} \frac{d\phi_i}{dx}(x) dx, \quad (2.56)$$

$$\mathbf{f}_j^{\text{WR}} = \int_0^1 \phi_j(x) f(x) dx. \quad (2.57)$$

Because of the choice of local basis function, most of the entries in $\mathbf{A}$ will be zero because their integrand will be zero. We can in practice use some Gauss quadrature to efficiently compute (2.56) (this can be exact if we integrate polynomials of a given degree with a specific number of integration points). These systems of linear equations can be solved using fast sparse linear algebra methods such as sparse LU decomposition. Advanced methods can be used to make computations of stiffness matrices $\mathbf{A}$ more efficient, such as isoparametric analysis (Cook, 1994).

2.3.3 Spectral Expansions

Spectral parametrisations are of great interest in computational mechanics. They are a way to describe functions through discrete sets of coefficients efficiently using globally defined basis functions. Spectral parametrisations can be expressed as

$$\hat{a}(x) = \sum_i a_i \phi_i(x). \quad (2.58)$$

Many various basis functions can be used in these expansions. Among them are Fourier expansions and Chebyshev polynomials. Fourier expansions can be expressed in different ways, these include the sine-cosine form and the exponential form. In the exponential form the basis functions are expressed as (Asmar, 2016)

$$\phi_j(x) = \operatorname{Re} \left(e^{i2\pi jx} \right), \quad (2.59)$$

where i is the imaginary unit. This is then multiplied with $a_j \in \mathbb{C}$. Such expressions are of particular use in periodic boundary value problems do to the ease with which we can compute derivatives in the Fourier domain. These expansions are defined over the entire real line.

Another well know expansion basis are the Chebyshev polynomials. The Chebyshev polynomials of the first kind are defined as

$$T_n(x) = \cos(n \arccos(x)) \quad (2.60)$$

and are defined for $-1 \leq x \leq 1$. These functions form an orthogonal set with respect to a weight function $\frac{1}{\sqrt{1-x^2}}$, that is to say

$$\langle T_i, T_j \rangle = \int_{-1}^1 \frac{T_i(x)T_j(x)}{\sqrt{1-x^2}} dx = \delta_{ij}, \quad (2.61)$$

they are also linearly independent.

In Fig. 2.2 we plot the first 5 polynomials. From any one dimensional expansion of N terms we can create d dimensional expansions using a tensor product of bases as

$$\hat{a}(x_{(1)}, \dots, x_{(d)}) = \sum_{i_1, \dots, i_d} \mathbf{U}_{1, \dots, d} \phi_1(x_{(1)}) \dots \phi_d(x_{(d)}) \quad (2.62)$$

where $\mathbf{U} \in \mathbb{R}^{N_1, \dots, N_d}$ is a d -dimensional tensor of expansion coefficients.

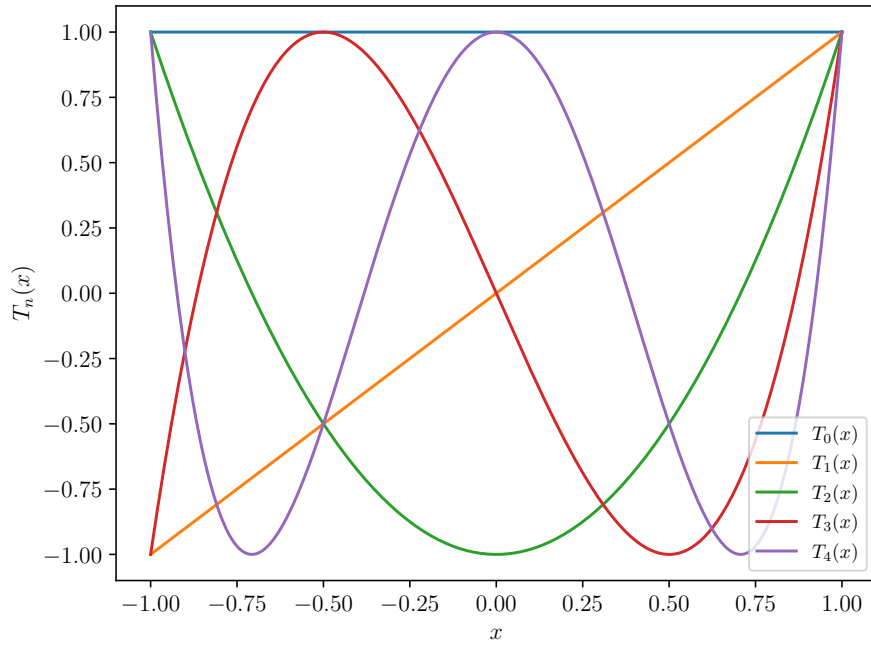


Fig. 2.2 First 5 Chebyshev polynomials

2.4 Machine Learning for PDEs

Physics informed machine learning is a relatively new field attempting to improve upon classical numerical PDE solution schemes. These improvements can be in the form of computational speed-up, simplification of solving methods, transfer learning and more. Some of the new advances in physics informed ML include nonlinear approximants, PINNs, and various methods designed to deal with boundary conditions. The field of parametric operator learning provides new avenues of interest for parametric physics.

2.4.1 Physics Informed Neural Networks

In typical numerical solving schemes, linear approximants find a parametric function satisfying the PDE and boundary conditions. All methods outlined thus far in Sec. 2.3.1 used a linear approximant. That is to say, a linear combination of basis functions. This linearity lends itself very well to the mathematical treatment of numerical solution fields. These systems of linearized equations can be assembled into matrix algebra form, which dominates the field of computational mechanics. In contrast, the major difference between these methods and PINNs (Lagaris et al., 1998, Raissi et al., 2019) is the use of nonlinear approximants (DeVore, 1998). It is important to note that these nonlinear approximant method can still be cast as

weighted residual methods, as shown in Appendix A.5. The particularity of PINNs methods resides in the choice of nonlinear approximant, given as

$$\hat{u}_\theta(x) = (\ell_l \circ \dots \circ \ell_i \circ \dots \circ \ell_0)(x), \quad (2.63)$$

where $\ell_i(x) = \sigma(\mathbf{W}_i x + \mathbf{b}_i)$ and $\mathbf{W}$ a matrix of learnable weights, $\mathbf{b}$ a vector of learnable biases, and σ a chosen activation function. We denote $\theta = \{\mathbf{W}, \mathbf{b}\}$. We have a composition of linear transformations with nonlinear activation functions together into multiple layers. Using collocation points in the PDE domain and on the boundary, we can approximately solve the PDE by minimization as

$$J(\theta) = \sum_i^{N_\Omega} \mathcal{G}(\hat{u}_\theta)(x_i)^2 + \lambda_b \sum_j^{N_{\partial\Omega}} \mathcal{B}(\hat{u}_\theta)(x_j)^2 \quad (2.64)$$

where N_Ω is associated to the set of collocation point inside the domain and $N_{\partial\Omega}$ is associated to the set of points on the boundary and λ_b is a user-specified weighting term. We find

$$\theta^* \in \underset{\theta}{\operatorname{argmin}} J(\theta). \quad (2.65)$$

We obtain an under-specified optimisation problem where there are usually more unknowns than equations. Because of this, there may be any number (possibly infinite) of values of θ that can minimize $J(\theta)$. The optimisation task is typically very high dimensional with the most efficient schemes for these types of problems usually being gradient based optimisers like Adam (Kingma and Ba, 2014). Some works have explored the use of L-BFGS optimisers for fine-tuning results (Krishnapriyan et al., 2021) with some success.

Due to the formulation the optimisation problem, the objective function may be highly multi-modal with non-optimal local minimums. All of these properties makes the mathematical treatment of PINNs very challenging as compared to the rigorous mathematical work that has gone into FEM. However, it has been shown that PINNs may be advantageous in some specific select scenarios over classical methods, even though overall they may pose major difficulties in their use (Markidis, 2021, Grossmann et al., 2023).

2.4.2 Boundary Conditions

The handling of boundary conditions offers an interesting topic in neural network based methods. In FEM we often choose our basis functions to fit the boundary

conditions by definition. In PINNs we cannot treat boundary conditions in the same manner. One solution requires to add a boundary term to the optimisation function as in (2.64). Other methods, developed in Sukumar and Srivastava (2022) impose boundary conditions exactly for certain type of boundary conditions, such as the Dirichlet boundary condition. These can be enforced exactly as

$$\hat{u}_\theta(x) = u_b(x) + d(x)N_\theta(x), \quad (2.66)$$

where $d(x)$ is the boundary nullifier, $u_b(x)$ is the boundary value condition, and $N_\theta(x)$ is a neural network. The boundary nullifier function $d(x) = 0$, iff. $x \in \partial\Omega$. The function $u_b(x)$ satisfies the Dirichlet boundary conditions. By this construction $\hat{u}_\theta(x)$ will satisfy the boundary conditions for any $N_\theta(x)$. Such exact enforcement of boundary conditions yields much more efficient optimisation. Other similar schemes have been developed for other boundary conditions but are not general enough to encompass all types of boundary conditions. We often still need to resort to formulations like the one in (2.64).

2.4.3 Operator Learning

Parametric PDEs arise when a PDE is modulated by a collection of parameters $\mathbf{z}$. In this case, the map $\mathbf{z} \mapsto u(x)$ is called the forward map, and is usually non-linear. The parametric inverse mapping $u(x) \mapsto \mathbf{z}$ is of interest as well. Because the forward map may not be surjective, the inverse mapping is only sensible for physically realizable solution fields. We will look at inverse problems more closely in chapter 3.

A closely related task to learning parametric PDEs is the task of operator learning. In operator learning we learn to approximate the forward operator $z(x) \mapsto u(x)$ for some range of functions $z(x)$. A classic example of operator learning is learning the mapping between diffusion fields to solution fields for a Poisson problem of a given geometry and boundary conditions.

Supervised Operator Learning

ML operators form a class of neural network architectures capable of mappings from one function space to another (Kovachki et al., 2023). The more notable operators are deep operator networks (DeepONets) (Lu et al., 2019), Fourier neural operators (FNO) (Li et al., 2020b), spectral neural operators (SNO) (Fanaskov and Oseledets, 2022), and wavelet neural operator (WNO) (Tripura and Chakraborty, 2022). Each of these may have different areas of specialisation. The most important feature of

neural operators as compared to DeepONets stems from their use of a non-local integral operations usually performed efficiently in frequency space through a Fourier transform (FNO) or a less efficient direct wavelet transform (WNO).

We now focus on FNOs. The architecture for this model is given as

$$F_\theta(\mathbf{z}(\mathbf{x}), \mathbf{x}) = Q_\theta(\cdot) \circ \mathbf{v}_l(\cdot) \circ \dots \circ \mathbf{v}_1(\cdot) \circ P_\theta(\mathbf{z}(\mathbf{x}), \mathbf{x}), \quad (2.67)$$

where $\mathbf{x} \in \mathbb{R}^d$. The function $P_\theta(\mathbf{z}(\mathbf{x}), \mathbf{x}) : [\mathbb{R}^{d_z}, \mathbb{R}^d] \mapsto \mathbb{R}^{d_v}$ is responsible for taking $\mathbf{z}(\mathbf{x}) \in \mathbb{R}^{d_z}$ and project it into a higher dimensional feature space $\mathbf{v}(\mathbf{x}) \in \mathbb{R}^{d_v}$ that is more appropriate for learning of latent features. The function $Q_\theta(\mathbf{v}_l(\mathbf{x}), \mathbf{x}) : [\mathbb{R}^{d_v}, \mathbb{R}^d] \mapsto \mathbb{R}^{d_u}$ is responsible for compressing the high dimensional vector fields of dimension d_v back down to the dimension d_u of the solution field. The FNO layers $\mathbf{v}(\mathbf{x})$ are defined as

$$\mathbf{v}_{i+1}(\mathbf{x}) = \boldsymbol{\sigma}(\mathbf{W}_\theta \mathbf{v}_i(\mathbf{x}) + \mathcal{K}_\theta(\mathbf{v}_i)(\mathbf{x})), \quad (2.68)$$

where the non-local operator is

$$\mathcal{K}_\theta(\mathbf{v}_i)(\mathbf{x}) = \mathcal{F}^{-1}(\mathbf{R}_\theta \mathcal{F}(\mathbf{v}_i)(\mathbf{x})), \quad (2.69)$$

where $\mathcal{F}, \mathcal{F}^{-1}$ are the fast Fourier and fast inverse Fourier transforms respectively and $\mathbf{R}_\theta$ is a tensor of learnable weights. The tensor of weights is constructed in a way that it truncates high mode information up to $k_{\max}$ frequencies which allows for the very interesting property of multi-resolution evaluation. This architecture has been shown to be highly effective at learning patterns in many PDEs such as Burgers equation, Navier-Stokes in vorticity form and Darcy flows (Li et al., 2020b). The operators outlined learn to approximate functions through supervised datasets of solution fields computed using classical numerical solution schemes such as FEM or spectral methods.

Physics Informed Operator Learning

There exists another class of ML operators called physics-informed operators. These are physics informed FNOs (Li et al., 2021) and physics informed DeepONets (Wang et al., 2021a). The idea of physics informed operator learning is to learn mappings between function spaces and solution fields driven by some PDE given a mixture of data and direct PDE enforcement. The physics informed component of these classes of algorithms usually comes from the aforementioned PINN-type loss functions (2.64)

added into the training loss to aid convergence. This type of model will be of use later in this work in chapter 4 and chapter 5.

With operator models using convolutions, such as in FNOs, one has to be careful with the fact that we cannot obtain spatial derivatives of their outputs through automatic differentiation. Rather, we have to resort to some classical numerical differentiation technique such as finite difference or Fourier gradients on the output solution fields.

Relying on repeated evaluations of classical numerical solvers may ultimately limit the potential of ML based methods for PDEs as they will inherit many of their limitations. There is a need to develop new probabilistic methods that can learn forward and inverse mappings between parameters and solutions of PDEs without relying on classical numerical solving methods. The forthcoming chapters will present new frameworks for combining different aspects of probability, ML, and computational mechanics for addressing these challenges.

Chapter 3

Active Learning Surrogates: Accelerating Bayesian Inversion

In many situations in engineering we have a physical system and a numerical model attempting to replicate its behaviour. The behaviour of the physical system can often be measured through sensors, but the properties of interest such as diffusion fields, modulus of elasticity, or geometry of internal elements may not be directly measurable. Through the calibration of various parameters in the model, it is possible to identify the parameters of a numerical model mimicking the behaviour of the physical system. The task of parameter identification for PDEs is called an inverse problem.

In this chapter we develop the notion of inverse problems and how these problems can be treated through Bayesian methods as Bayesian inverse problems (BIPs). We then present state-of-the-art methods for tackling these problems as applied to general PDE scenarios and also discuss methods of surrogate modelling. Following this, we then propose our work on active learning projected surrogates – Stein variational gradient descent, ALPS-SVGD. We conclude the chapter with numerical illustrations of the aforementioned concepts and works.

3.1 Inverse Problems

There exist many types of inverse problems, from recovering functions of integral equations (Vogel, 2002) to applied problems in geophysics (Snieder and Trampert, 1999). The type of inverse problem we are interested in this work, are problems of recovering a “true” latent parameter $\mathbf{z}_T \in \mathcal{Z} \subseteq \mathbb{R}^{d_z}$ of a PDE model $G(\mathbf{z}_T, \mathbf{X})$, through noisy observations (Belov, 2012). Here, $\mathbf{X} \in \Omega^{\otimes n_y}$ are the n_y evaluation

locations on the PDE domain Ω where the solution field is measured. We note here that $G(\mathbf{z}, \mathbf{X}) \neq \mathcal{G}_{\mathbf{z}}(\mathbf{X})$ as $G(\mathbf{z}, \mathbf{X})$ returns the observation of the solution field and $\mathcal{G}_{\mathbf{z}}(\mathbf{X})$ returns the residual. Under a deterministic paradigm we can formulate it as a nonlinear least-square problem (Chavent, 2010) between the numerical model output and the data. This can be written as

$$\mathbf{z}^* = \underset{\mathbf{z}}{\operatorname{argmin}} J_{\mathbf{y}}(\mathbf{z}), \text{ for } \mathbf{z} \in \mathcal{Z}, \quad (3.1)$$

$$J_{\mathbf{y}}(\mathbf{z}) = \frac{1}{2} \|\mathbf{y} - G(\mathbf{z}, \mathbf{X})\|_2^2, \quad (3.2)$$

where $\mathbf{z}$ is the parameter vector of interest, $\mathcal{Z}$ is the set of admissible parameters, $\mathbf{y}$ is the data, and $G(\cdot)$ the numerical model. This objective function will yield the minimal discrepancy value for $\mathbf{z}$ based on data. The hope is that when we obtain $\mathbf{z}^*$, it is approximately equal to $\mathbf{z}_T$.

Among many methods for solving (3.2), one can use first order and second order optimisation and nonlinear least-squares approaches. When optimising such a function, Newton and quasi-Newton methods perform very well. Some of the more successful ones include the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm (and its limited-memory counterpart L-BFGS), the trust region reflective algorithm (trf) and the Levenberg-Marquardt (LM) algorithms, both quasi-Newton type methods (Fletcher, 2013). These two last algorithms are very powerful when we have several measurement locations and an alternative loss function which maintains the dimension of the n measurement locations. Such a function could have the form

$$J_{\mathbf{y}}(z) = [\mathbf{y} - G(\mathbf{z}, \mathbf{X})] \odot [\mathbf{y} - G(\mathbf{z}, \mathbf{X})], \quad (3.3)$$

where “ $\odot$ ” is an element wise operation which is an element-wise least squares problem. These methods can achieve high accuracy results and are usually much faster than gradient descent algorithms for low to medium dimensional problems. However, (3.2) and (3.3) may not yield unique solutions for a given problem.

Wellposedness can be defined as having a unique solution to 3.2 that depends continuously on the data $\mathbf{y}$ (Chavent, 2010). One cannot generally assume uniqueness of $\mathbf{z}^*$ for nonlinear maps $G(\cdot)$ from finitely many discrete noisy observations. To assist the inference, prior knowledge can be incorporated to the problem formulation in (3.2) through a regularization term which biasing the nonlinear least squares

problems towards a specific values deemed *likeliest*. The new objective becomes

$$\mathbf{z}^* = \underset{\mathbf{z}}{\operatorname{argmin}} J_{\mathbf{y},o}(\mathbf{z}), \text{ for } \mathbf{z} \in \mathcal{Z}, \quad (3.4)$$

$$J_{\mathbf{y},o}(\mathbf{z}) = \frac{1}{2\epsilon_y} \|\mathbf{y} - G(\mathbf{z}, \mathbf{X})\|_2^2 + \frac{1}{2\epsilon_o} \|\mathbf{z}\|_2^2, \quad (3.5)$$

where ϵ_o calibrates the strength of the regularization. With these efforts to regularize our problem we obtain a point estimate of $\mathbf{z}$ based on data and *a priori* knowledge. The method may provide ranges of possible values of $\mathbf{z}$ and coherently quantify their relative probability based on data and prior information. This is relevant context for Bayesian inference.

Indeed, in many application scenarios the physical quantity $\mathbf{z}$ can only take values in some admissible set $\mathcal{Z} \subseteq \mathbb{R}^{d_z}$. Often however, through a change of variable, it is possible to transform an unconstrained variable, $\tilde{\mathbf{z}} \in \mathbb{R}^{d_z}$, into a constrained variable $\mathbf{z}$ through $t_{\mathbf{z}}(\tilde{\mathbf{z}}) : \mathbb{R} \mapsto \mathcal{Z}$. This does not affect the formulation of the inverse problem as we can compose the (already) nonlinear model $G(\cdot)$ with this transform as $G(\cdot) \circ t_{\mathbf{z}}(\cdot)$. For example, if $\mathcal{Z} = \mathbb{R}^+$ then we can take $t_{\mathbf{z}}(\cdot) = \exp(\cdot)$. The variable of interest once optimised is identified as $\mathbf{z}^* = t_{\mathbf{z}}(\tilde{\mathbf{z}}^*)$ and in the optimisation scheme we initialise the optimisation at $\tilde{\mathbf{z}}_0 = t_{\mathbf{z}}^{-1}(\mathbf{z}_0)$. In the forgoing sections, when such a map $t_{\mathbf{z}}(\cdot)$ is in use, it will be carried implicitly in the expressions.

3.2 Bayesian Inverse Problems

Bayesian inference revolves around Baye's theorem. It relates prior information, a data informed discrepancy function, and a normalization term to a distribution over parameters given data (Gelman et al., 1995). The relationship between these distributions is given by

$$\text{posterior} = \frac{\text{data-fit} \times \text{prior}}{\text{evidence}}. \quad (3.6)$$

This can be written concisely in terms of probability densities as

$$p(\mathbf{z}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{y})}, \quad (3.7)$$

where $\mathbf{y}$ is our data. The normalization constant is given as

$$p(\mathbf{y}) = \int_{\mathcal{Z}} p(\mathbf{y}|\mathbf{z})p(\mathbf{z}) \, d\mathbf{z}. \quad (3.8)$$

This Bayesian paradigm is applicable to inverse problems for PDE problems (Stuart, 2010a). The advantage of this framework over the deterministic version discussed in Sec. 3.1 is that it provides a probability distribution over model parameters. To apply the Bayesian method to a PDE BIP, we begin by posing our observation model

$$\mathbf{y}_i = G(\mathbf{z}_i, \mathbf{X}_i) + \mathbf{e}, \quad \mathbf{e} \sim \mathcal{N}(\mathbf{0}, \epsilon_y^2 \mathbf{I}). \quad (3.9)$$

This observation model assumes independent identically distributed (i.i.d) noise over the multiple observations $\mathbf{y}_i$. When denoting the collection of observation vectors, $\mathbf{y}_i$ for $i = 1 : N_d$, we drop the subscript and write $\mathbf{y}$. From the observation model, the likelihood of $\mathbf{z}$ for a given model $G(\cdot)$ becomes,

$$p(\mathbf{y}|\mathbf{z}) = \prod_{i=0}^{N_d} p(\mathbf{y}_i|\mathbf{z}) = \prod_{i=0}^{N_d} \exp \left(-\frac{1}{2\epsilon_y^2} \|\mathbf{y}_i - G(\mathbf{z}, \mathbf{X}_i)\|_2^2 \right). \quad (3.10)$$

Because of the nonlinearity of the operator $G(\cdot)$, we cannot write out analytically the normalization constant $p(\mathbf{y})$. It is also computationally intractable due to the cost of high dimensional integration. So, we write out a proportionality

$$p(\mathbf{z}|\mathbf{y}) \propto p(\mathbf{y}|\mathbf{z})p(\mathbf{z}). \quad (3.11)$$

We choose our prior $p(\mathbf{z})$ to be a MVN in this case. In nearly all computational applications we will prefer to work with the log of densities to avoid numerical under and over-flow (Rogers and Girolami, 2016). Choosing $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{z}_o, \epsilon_o^2 \mathbf{I})$ and taking the log of the likelihood function we obtain

$$\log p(\mathbf{y}|\mathbf{z}) \propto -\sum_{i=0}^{N_d} \frac{1}{2\epsilon_y^2} \|\mathbf{y}_i - G(\mathbf{z}, \mathbf{X}_i)\|_2^2 - \frac{1}{2\epsilon_o^2} \|\mathbf{z} - \mathbf{z}_o\|_2^2, \quad (3.12)$$

which has a nearly identical form to (3.4). Maximising the first term is called a maximum likelihood estimation, and maximising (3.12) is called a maximum a posteriori estimate (MAP). Of course, different forms of likelihood and priors can be chosen.

This derivation does not provide a normalization constant. We will now look at methods for obtaining $p(\mathbf{z}|\mathbf{y})$ despite a missing normalizing constant.

3.2.1 Sampling

When we are interested in obtaining an expression for $p(\mathbf{z}|\mathbf{y})$, we need to compute the marginal likelihood,

$$p(\mathbf{y}) = \int_{\mathbf{z}} p(\mathbf{y}|\mathbf{z})p(\mathbf{z}) \, d\mathbf{z}, \quad (3.13)$$

which often is analytically intractable as well as computationally intractable due to the curse of dimensionality. However, with a set of samples from $p(\mathbf{z}|\mathbf{y})$, we can use them to compute most of their statistics such as means, variances, covariances, and marginal distributions. Options for sampling from such unnormalized distributions offer different properties.

Markov Chain Monte Carlo

Samples can be approximately drawn from BIP distributions by constructing markov chains with stationary distribution as the posterior measure (Brooks et al., 2011). The important feature of the Markov chains is that the probability of occurrence of the next sample only depends on the current state,

$$p(\mathbf{z}_{t+1}|\mathbf{z}_t, \mathbf{z}_{t-1}, \mathbf{z}_{t-2}, \dots) = p(\mathbf{z}_{t+1}|\mathbf{z}_t). \quad (3.14)$$

One of the simplest Markov Chain Monte Carlo (MCMC) methods is the Metropolis-Hastings (MH) algorithm. The markov chain moves to a proposed sample $\mathbf{z}_{t+i}$ drawn from a proposal distribution $q(\mathbf{z}_{t+i}|\mathbf{z}_t)$ with probability α where

$$\alpha = \min \left(1, \frac{p(\mathbf{z}_{t+1}|\mathbf{y})q(\mathbf{z}_t|\mathbf{z}_{t+1})}{p(\mathbf{z}_t|\mathbf{y})q(\mathbf{z}_{t+1}|\mathbf{z}_t)} \right), \quad (3.15)$$

. The normalization constant $p(\mathbf{y})$ cancels from the top and bottom, hence we can work with distributions with unknown normalizations.

Other MCMC methods exist and may be better suited than MH for different sampling scenarios. Extensions of the MH algorithm can be created through the use of parallel chains (Llorente et al., 2019). Other sampling schemes only requiring density ratios include the affine invariant ensemble sampler (Goodman and Weare, 2010, Foreman-Mackey et al., 2013) called EMCEE. The affine invariance of this sampler allows for the efficient sampling of distributions with large aspect ratios without gradient information. Another often cited sampler is the Gibbs sampler, useful for sampling from joint distributions when we can easily sample from conditional

distributions of the joint space. With access to gradient information, there are many efficient gradient based sampling schemes. One such algorithm is Hamiltonian Monte-Carlo (HMC) (Betancourt, 2017), a powerful sampling scheme often used as the gold-standard of sampling methods. There exist also adaptive step size methods modified from the HMC framework, such as the very popular No U-turn Hamiltonian sampler (NUTS), proven to be an efficient, easy to use, and robust sampling scheme (Hoffman et al., 2014). Other classes of samplers rely on the discretisation of a stochastic differential equation whose invariant distribution is the target measure. These include the unadjusted stochastic Langevin dynamics (ULA) (Roberts and Rosenthal, 1998), and the metropolis adjusted Langevin algorithm (MALA) (Xifara et al., 2014). Other methods also include the preconditioned Crank-Nicolson (pCN) method (Cotter et al., 2013) and Stein variational gradient descent (SVGD) (Liu and Wang, 2016).

SVGD

SVGD methods are a new class of algorithms that rely on the score function of a target density to deterministically move a cluster of particles from a reference measure to the posterior measure. The score function is the gradient of the log density. As with all VI setups, first we pose our inference problem as an optimisation objective

$$q^* = \operatorname{argmin}_{q \in \mathcal{Q}} D_{\text{KL}}(q||p). \quad (3.16)$$

In the SVGD method we iteratively update the position of our collection n_q randomly initialised samples with

$$T(\mathbf{z}) = \mathbf{z} + \epsilon \phi(\mathbf{z}). \quad (3.17)$$

By taking the directional derivative of the KL-divergence we obtain an optimal descent direction

$$\phi^*(\mathbf{z}) = \mathbb{E}_{\mathbf{z}_q \sim q} [\nabla_{\mathbf{z}} \log p(\mathbf{z}_q) k(\mathbf{z}_q, \mathbf{z}) + \nabla k(\mathbf{z}_q, \mathbf{z})], \quad (3.18)$$

for some choice of kernel $k(\mathbf{z}, \mathbf{z}^*)$. The first term drives the particles towards regions of high density and the second term acts as a repulsive force which pushes particles away from each other (Liu, 2017). Much analysis has so far been done on the convergence properties of SVGD (Korba et al., 2020, Duncan et al., 2019): after enough time and small enough a step-size particles converge to the posterior measure. Being a simple gradient update scheme, many gradient based optimisation algorithms can be

Algorithm 1: SVGD**Input:** $\mathbf{y}$, n_q , $\nabla \log p(\mathbf{z}|\mathbf{y})$ **for** $i < \text{iterations}$ or $\|\hat{\phi}(\mathbf{z})\|_2 < \text{tol}$ **do**

$$\hat{\phi}(\mathbf{z}) = \frac{1}{n_q} \sum_{j=1}^{n_q} [k(\mathbf{z}_j, \mathbf{z}) \nabla_{\mathbf{z}} \log p(\mathbf{z}_j|\mathbf{y}) + \nabla_{\mathbf{z}} k(\mathbf{z}_j, \mathbf{z})] \quad (3.19)$$

$$\mathbf{z}_{i+1}^\ell = \mathbf{z}_i^\ell + \epsilon \hat{\phi}^*(\mathbf{z}_i^\ell) \quad (3.20)$$

end

implemented such as Nesterov (Ruder, 2016), AMSGrad (Reddi et al., 2019), and RMSProp (Zou et al., 2019). In practice, the popular deep learning gradient descent algorithm Adam is not well adapted for this purpose due to its non-convergence properties (Reddi et al., 2019, Kingma and Ba, 2014).

In practice, we sample a collection of n_q samples from the prior, and these samples become our particles that are updated using (3.18). For every n_q particle we compute the update direction (3.19) which requires the evaluation of $\nabla_{\mathbf{z}} \log p(\mathbf{z}|\mathbf{y})$ (also known as the score function). Using this update direction each particle moves in their respective direction with (3.20). As described in Algorithm 1, we repeat these steps until convergence or until the desired number of steps is reached.

In Fig. 3.1 we see a representation of a collection of particles with each their respective kernel bumps used in the SVGD update rules. As in (Liu and Wang, 2016), we use an RBF kernel

$$k(\mathbf{z}, \mathbf{z}') = \exp\left(-\frac{1}{h} \|\mathbf{z} - \mathbf{z}'\|_2^2\right), \quad (3.21)$$

where, as in (3.21), $h = \text{med}^2 / \log(n)$, where med is the median inter-particle distance at time i . The dark blue curve is the sum of the particle kernel function and is responsible for the divergence term in the SVGD particle update rule. This method is referred to as vanilla-SVGD as it is the most basic version of the method.

Beyond Vanilla-SVGD

There exist many other variations and extensions of the vanilla SVGD algorithm described above. We outline below some notable variations.

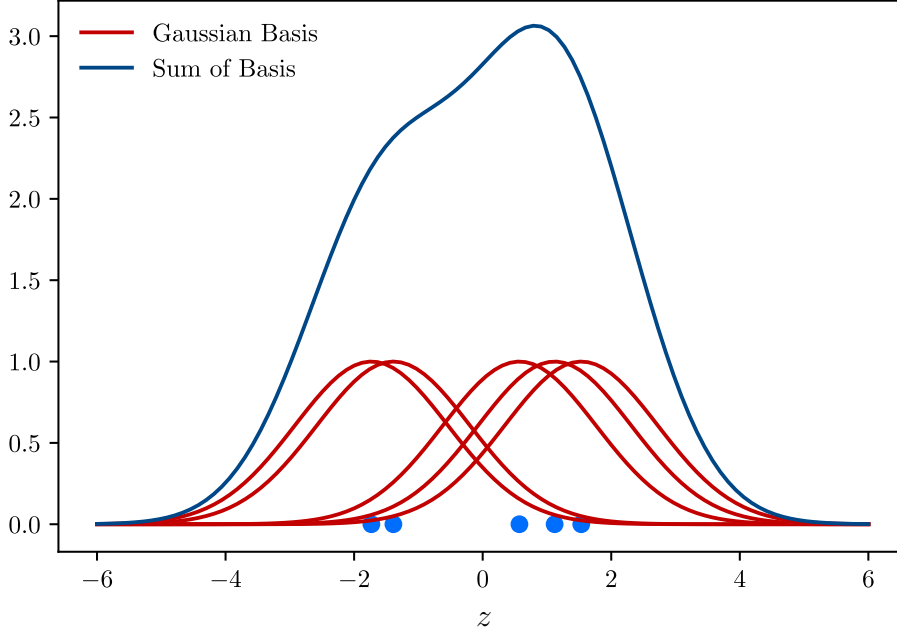


Fig. 3.1 SVGD Gaussian kernel basis

Stein variational Newton methods is proposed by Detommaso et al. (2018). The method extends the gradient update scheme of SVGD to second order information. This requires the solution of systems of linear equations at every iteration to obtain the descent direction. The update step can be expressed as (Wang et al., 2019)

$$\mathbf{z}_{i+1}^\ell = \mathbf{z}_i^\ell + \epsilon \tilde{\mathbf{H}}_\ell^{-1} \phi^*(\mathbf{z}_i^\ell), \quad (3.22)$$

where

$$\tilde{\mathbf{H}}_\ell = \mathbb{E}_{\mathbf{z} \sim q} [\mathbf{H}(\mathbf{z}) k(\mathbf{z}, \mathbf{z}^\ell)^2 + (\nabla_{\mathbf{z}^\ell} k(\mathbf{z}, \mathbf{z}^\ell))^{\otimes 2}], \quad (3.23)$$

where $\mathbf{H}(\mathbf{z})$ is the Hessian of the log-posterior and $\mathbf{x}^{\otimes 2} = \mathbf{x}\mathbf{x}^\top$ is the outer product with itself. A more expressive kernel than the isotropic RBF kernel is defined as

$$k(\mathbf{z}, \mathbf{z}') = \exp \left(-\frac{1}{2d_z} \|\mathbf{z} - \mathbf{z}'\|_{\mathbf{M}}^2 \right), \quad (3.24)$$

where $\mathbf{M} := \frac{1}{n_q} \sum_{i=1}^{n_q} \mathbf{M}(\mathbf{z})$ where $\mathbf{M}(\mathbf{z}) \approx -\nabla^2 \log p(\mathbf{z}|\mathbf{y})$. The Hessian of the posterior density is approximated with a Gauss-Newton matrix as the outer-product of the score function to ensure positive-semi-definiteness. We note that $\|\mathbf{x}\|_{\mathbf{M}}^2 = \mathbf{x}^\top \mathbf{M} \mathbf{x}$.

This method has been shown to be advantageous in approximating distributions with rich curvature information Detommaso et al. (2018).

SVGD with matrix-valued kernels was proposed by Wang et al. (2019) which extends vanilla-svgd with more expressive kernels resulting in a preconditioned SVGD with natural gradients. The update algorithm is similar to vanilla-SVGD, but the kernels make use of three types of preconditioning matrices: constant preconditioning, point-wise preconditioning, and mixture preconditioning. The constant preconditioning matrix kernel is written as

$$K_{\mathbf{Q}}(\mathbf{z}, \mathbf{z}') := \mathbf{Q}^{-1/2} K_o \left(\mathbf{Q}^{1/2} \mathbf{z}, \mathbf{Q}^{1/2} \mathbf{z}' \right) \mathbf{Q}^{-1/2}, \quad (3.25)$$

and with K_o as a RBF kernel, becomes

$$K_{\mathbf{Q}}(\mathbf{z}, \mathbf{z}') = \mathbf{Q}^{-1} \exp \left(-\frac{1}{2h} \|\mathbf{z} - \mathbf{z}'\|_{\mathbf{Q}} \right). \quad (3.26)$$

This provides an update direction given by $\mathbf{Q}^{-1} \phi^*(\mathbf{z})$, where, if we choose $\mathbf{Q}$ to be the Fisher information matrix, is a natural gradient version of SVGD.

Projected SVGD has also been suggested (Chen and Ghattas, 2020, Chen et al., 2019, Zhuo et al., 2018). To alleviate the challenges posed by the curse of dimensionality, they propose to project the high dimensional posterior space into lower-dimensional latent subspace where latent particles can evolve and be projected back to the original space to obtain approximate samples from the high dimensional posterior measure. This subspace is created with a linear projection operator $\mathbf{P}_r : \mathbb{R}^d \mapsto \mathbb{R}^d$ with a subspace given by

$$\mathbf{P}_r \mathbf{z} := \mathbf{\Psi}_r \mathbf{p}, \quad (3.27)$$

where $\mathbf{p}$ is the latent projected variable of $\mathbf{z}$ and $\mathbf{\Psi}_r = [\psi_1, \dots, \psi_r]$ and

$$\mathbf{H} \psi_i = \lambda_i \mathbf{\Gamma} \psi_i, \quad (3.28)$$

which is a generalized eigenvalue problem of the gradient information matrix $\mathbf{H}$ with respect to the prior covariance $\mathbf{\Gamma}$ (Saibaba et al., 2016). Here, $\mathbf{H}$ is given by

$$\mathbf{H} = \int_{\mathcal{Z}} (\nabla_{\mathbf{z}} \log p(\mathbf{y}|\mathbf{z}))^{\otimes 2} p(\mathbf{z}|\mathbf{y}) d\mathbf{z}. \quad (3.29)$$

This method can in some cases minimise the mode collapse that SVGD suffers in higher dimensional settings (Ba et al., 2021). In the update rule for the particle

position, we now use

$$\nabla_{\mathbf{p}} \log p(\mathbf{p}|\mathbf{y}) = \mathbf{\Psi}_r^\top \nabla_{\mathbf{z}} \log p_r(\mathbf{P}_r \mathbf{z}). \quad (3.30)$$

This term updates particle positions through a low dimensional latent representation. Randomness in the update rule can be incorporated with a stochastic version of SVGD.

Stochastic SVGD methods introduce new sources of randomness in the update rules (Li et al., 2020a, Levisyev et al., 2022). One such example is through a stochastic approximation on the expectation in (3.18). This increases computational efficiency and may improve convergence.

Annealed SVGD is a variation of SVGD that contracts and loosens the posterior density to aid in the exploration of multi-modal measures (D’Angelo and Fortuin, 2021).

Each of these schemes might be better suited for certain tasks. However, they all have in common the property of having clusters of moving particles to model the posterior distribution in a non-parametric manner.

This lays the basis for our proposed work on the construction of posterior surrogates of density surfaces for inverse problems; but before we introduce our contribution we need to discuss the computing of score function for PDE BIPs and the general task of surrogate modelling for posterior surfaces.

3.2.2 Score Functions

When sampling for posterior distributions it is important to distinguish between the types of information required from candidate algorithms. The main difference in algorithms depends on whether they require the unnormalized log posterior, or the score function of the posterior, or both. Some methods leverage curvature information, which often is approximated with a Gauss-Newton matrix (Girolami and Calderhead, 2011). Computing the score function for a PDE based BIP we use the log posterior

$$\log p(\mathbf{z}|\mathbf{y}) = \log p(\mathbf{y}|\mathbf{z}) + \log p(\mathbf{z}) - \log p(\mathbf{y}), \quad (3.31)$$

and score function is given as

$$\nabla_{\mathbf{z}} \log p(\mathbf{z}|\mathbf{y}) = \nabla_{\mathbf{z}} \log p(\mathbf{y}|\mathbf{z}) + \nabla_{\mathbf{z}} \log p(\mathbf{z}) - \nabla_{\mathbf{z}} \log p(\mathbf{y}). \quad (3.32)$$

Notice that $p(\mathbf{y})$ is not a function of $\mathbf{z}$, hence $\nabla_{\mathbf{z}} \log p(\mathbf{y}) = 0$ which bypasses the computation of the troublesome normalization constant. Furthermore, if the prior $p(\mathbf{z})$ is chosen to be a reasonably simple distribution, then its gradient is easy to compute when the form of the likelihood function follows

$$\log p(\mathbf{y}|\mathbf{z}) = - \sum_{i=0}^{N_d} \frac{1}{2\epsilon_y^2} \|\mathbf{y}_i - G(\mathbf{z}, \mathbf{X}_i)\|_2^2, \quad (3.33)$$

then taking the gradient on both sides and moving it inside the sum of the r.h.s. we obtain

$$\nabla_{\mathbf{z}} \log p(\mathbf{y}|\mathbf{z}) = - \sum_{i=0}^{N_d} \frac{1}{2\epsilon_y^2} \nabla_{\mathbf{z}} \|\mathbf{y}_i - G(\mathbf{z}, \mathbf{X}_i)\|_2^2. \quad (3.34)$$

Using the chain rule we have

$$\nabla_{\mathbf{z}} \|\mathbf{y}_i - G(\mathbf{z}, \mathbf{X}_i)\|_2^2 = -2\|\mathbf{y}_i - G(\mathbf{z}, \mathbf{X}_i)\|_2 \nabla_{\mathbf{z}} G(\mathbf{z}, \mathbf{X}_i), \quad (3.35)$$

and substituting back into our original expression for the score of the posterior (3.32) we obtain

$$\nabla_{\mathbf{z}} \log p(\mathbf{z}|\mathbf{y}) = \sum_{i=0}^{N_d} \frac{1}{\epsilon_y^2} \|\mathbf{y}_i - G(\mathbf{z}, \mathbf{X}_i)\|_2 \nabla_{\mathbf{z}} G(\mathbf{z}, \mathbf{X}_i) + \nabla_{\mathbf{z}} \log p(\mathbf{z}). \quad (3.36)$$

The crux of computing (3.36) is in obtaining the gradients of the observed solution field with respect to the model parameters $\mathbf{z}$, $\nabla_{\mathbf{z}} G(\mathbf{z}, \mathbf{X}_i)$. For a model based on finite elements we have

$$G(\mathbf{z}, \mathbf{X}_i) = \mathbf{P} \mathbf{u}(\mathbf{z}), \quad (3.37)$$

where $\mathbf{P}$ is a projection matrix selecting the observation locations on the FEM mesh, and $\mathbf{u}$ are the displacements given by the FEM model. Looking closer at a linear FEM, we have

$$\mathbf{A}_{\mathbf{z}} \mathbf{u} = \mathbf{f}, \quad (3.38)$$

where $\mathbf{A}_{\mathbf{z}}$ is the stiffness matrix as a function of the model parameters $\mathbf{z}$, and $\mathbf{f}$ is the forcing term. We can rewrite this as

$$\mathbf{u} = \mathbf{A}_{\mathbf{z}}^{-1}\mathbf{f}, \quad (3.39)$$

which leads to the desired quantity of interest

$$\nabla_{\mathbf{z}}G(\mathbf{z}, \mathbf{X}_i) = \nabla_{\mathbf{z}}\mathbf{P}\mathbf{A}_{\mathbf{z}}^{-1}\mathbf{f}. \quad (3.40)$$

With matrix algebra (Petersen et al., 2008) we obtain

$$\nabla_{\mathbf{z}}G(\mathbf{z}, \mathbf{X}_i) = -\mathbf{P}\mathbf{A}_{\mathbf{z}}^{-1}\nabla_{\mathbf{z}}\mathbf{A}_{\mathbf{z}}\mathbf{A}_{\mathbf{z}}^{-1}\mathbf{f}, \quad (3.41)$$

$$\nabla_{\mathbf{z}}G(\mathbf{z}, \mathbf{X}_i) = -\mathbf{P}\mathbf{A}_{\mathbf{z}}^{-1}\nabla_{\mathbf{z}}\mathbf{A}_{\mathbf{z}}\mathbf{u}, \quad (3.42)$$

where $\nabla_{\mathbf{z}}\mathbf{A}_{\mathbf{z}}$ is a third order tensor. In the case where $\mathbf{z}$ directly parametrises the model parameters for each element (n elements mean n entries in $\mathbf{z}$) we can compute $\nabla_{\mathbf{z}}\mathbf{A}_{\mathbf{z}}$ easily if its entries depend linearly on $\mathbf{z}$ (as it does if $\mathbf{z}$ is the diffusion field for some Poisson problem). We then have n highly sparse matrices $\partial_{z_i}\mathbf{A}_{\mathbf{z}}$ that only have entries for the respective element to which z_i belongs. In reality, we may wish to choose some parametrisation of $\mathbf{z}$ related to $\mathbf{A}$ through a function expansion or GP mean. So instead of $\mathbf{A}_{\mathbf{z}}$ we have $\mathbf{A}_{g(\mathbf{z})}$, where $g(\mathbf{z})$ interpolates values of the m coefficients $\mathbf{z}$ over the domain rather than directly parametrisng the elements of the model and being mesh dependent (which is an undesirable quality). We then use the chain rule again to obtain

$$\nabla_{\mathbf{z}}G(\mathbf{z}, \mathbf{X}_i) = -\mathbf{P}\mathbf{A}_{g(\mathbf{z})}^{-1}\nabla_{\mathbf{z}}[\mathbf{A}_{g(\mathbf{z})}]\mathbf{u} \quad (3.43)$$

$$\nabla_{\mathbf{z}}G(\mathbf{z}, \mathbf{X}_i) = -\mathbf{P}\mathbf{A}_{g(\mathbf{z})}^{-1}\nabla_{\mathbf{z}}\mathbf{A}_{g(\mathbf{z})}\nabla_{\mathbf{z}}g(\mathbf{z})\mathbf{u}. \quad (3.44)$$

We now have m matrices with the same sparsity pattern as $\mathbf{A}_{\mathbf{z}}$, whereas before we had n highly sparse matrices. Methods based on adjoint computations reduce the computational cost associated with solving n or m systems of equations (Plessix, 2006, Funke and Farrell, 2013, Mitusch et al., 2019, Farrell et al., 2013). From this simple example on a linear PDE, we can see that computing gradients of solution fields with respect to model parameters may require low-level access of the underlying FEM code, a non-trivial task generally not addressed by common software packages.

3.2.3 Surrogate Modelling of Density Surfaces

When solving Bayesian inverse problems one cannot directly compute the posterior density, and only has access to the unnormalized posterior as in (3.11). One of the most prominent approaches for dealing with this normalizing constant is to ignore it through sampling methods. However, with sampling one needs to evaluate the unnormalized posterior density many times. Depending on the method used, the number of evaluation required might vary between $10^4 - 10^6$ or more evaluations. Aristoff and Bangerth (2021) go as far as creating 2×10^{11} samples from the posterior distribution through 2000 Markov chains of length 10^8 sample each in order to get accurate samples using a simple MH-MCMC scheme (requiring some ~ 30 CPU years of compute time). In order to reduce the computational cost of repeated evaluations of the posterior surface, many have suggested reduced order models of the posterior surface (Bui-Thanh et al., 2008). Advances in ML encourage people to propose GP surrogate to replace expensive computations for MCMC (Kennedy and O’Hagan, 2001). Work has been done to analyse the properties of GP surrogates for unnormalized posterior densities (Teckentrup, 2020, Stuart and Teckentrup, 2018, Lie et al., 2018).

Indeed, in the paper *Calibrate, emulate, sample* (CES), Cleary et al. (2021) propose a scheme to learn surrogates of computer models to compute posterior densities. This was done in a two-step procedure. Firstly, one starts by approximately collecting samples from regions of high density in the posterior measure by using an approximate sampling scheme based on ensemble Kalman methods (Schillings and Stuart, 2017, Garbuno-Inigo et al., 2020). From these samples one then constructs a surrogate of the forward model with a GP and a chosen GP mean function. This forward model surrogate enables fast MCMC sampling, since the evaluation of the log-density is computed efficiently.

Building on these methods and strategies, we can now propose an alternative algorithm that allows for the online construction of GP surrogates of likelihood functions for efficient sampling.

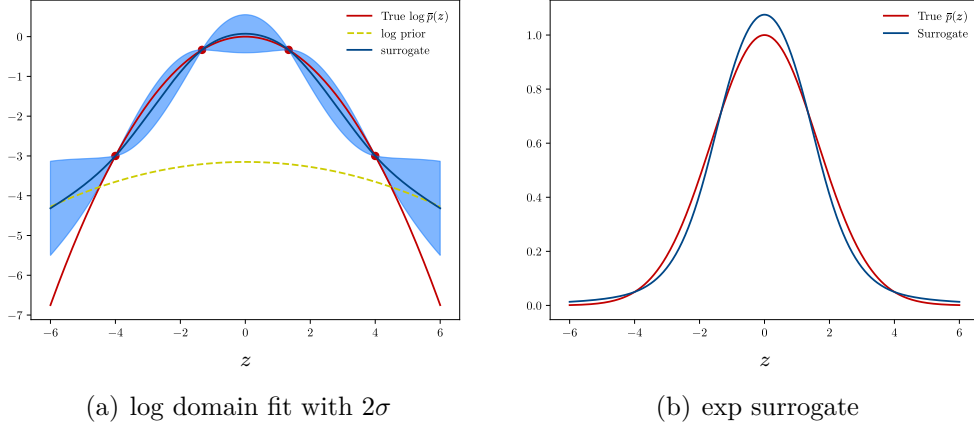


Fig. 3.2 1D log-density surrogate

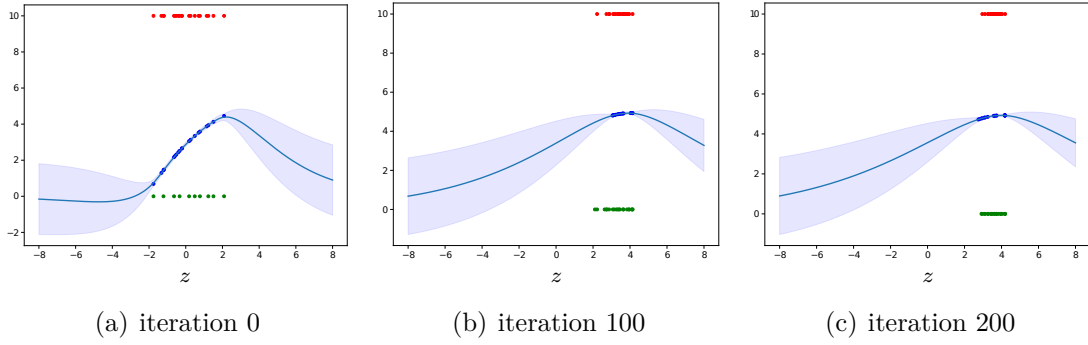


Fig. 3.3 Active Learning Surrogate in log-domain

3.3 Active Learning Projected Surrogates: ALPS-SVGD

For a model with very high dimensional outputs, modelling the log-likelihood function directly reduces the output space from being a high-dimensional vector field to a scalar field, as compared to forward model GPs, similarly to (Cleary et al., 2021).

In Fig. 3.2 we see a GP surrogate of a log-density. The prior mean for the GP is chosen to be the log-prior of the posterior distribution. We propose to take advantage of the cluster of deterministically moving particles in SVGD to construct active learning GP surrogates based on some low-dimensional representation of a PDE Bayesian inverse problem.

Similarly to Yan and Zhou (2021), we construct a local approximation of the posterior surface by aggregating evaluations of the true log-likelihood at points where

Algorithm 2: GP-SVGD

Input: $tol, n_q, n_i, \log p(\mathbf{z}), \log p(\mathbf{y}|\mathbf{z})$

- 1 Sample $\mathbf{Z}_a \sim p(\mathbf{z})^{\otimes n_q}$
- 2 Eval. $\mathbf{Y}_a = \{\log p(\mathbf{y}|\mathbf{z}) | \forall \mathbf{z} \in \mathbf{Z}_a\}$ to construct $\mathcal{D}_a$, (initially we have $\mathbf{S}_a = \mathbf{Z}_a$)
- 3 **while** $\frac{1}{n_q} \sum_i \|\phi^*(\mathbf{z}_i)\|_2 > tol$ **do**
- 4 Fit $\mathcal{GP}(m(\mathbf{z}), k(\mathbf{z}, \mathbf{z}'))$ on $\mathcal{D}_a$ and save hyper-parameters as a warm-state.
- 5 Compute SVGD gradient update:
 $\phi^*(\mathbf{z}_i) = \frac{1}{n_q} \sum_{j=1}^{n_q} [k(\mathbf{z}_j, \mathbf{z}_i) \nabla_{\mathbf{z}_i} (\mathcal{GP}(\mathbf{z}_i) + \log p(\mathbf{z}_i)) + \nabla_{\mathbf{z}_i} k(\mathbf{z}_j, \mathbf{z}_i)]$.
- 6 Compute the particle update via AMSGrad using $\phi^*(\mathbf{z}_i)$.
- 7 Apply (3.46), (3.47), and (3.48).
- 8 Update $\mathcal{D}_a$ where $\mathbf{Y}_a = \{\log p(\mathbf{y}|\mathbf{z}) | \forall \mathbf{z} \in \mathbf{Z}_a\}$.
- 9 **end**

the surrogate approximation is inadequate. This is done through evolving a SVGD system of particles driven by a trained GP informed by an *active dataset*,

$$\mathcal{D}_a = \{\mathbf{Z}_a, \mathbf{Y}_a, \mathbf{S}_a\} \quad \text{s.t.} \quad |\mathbf{Z}_a| \leq n_q, |\mathbf{Y}_a| \leq n_q, |\mathbf{S}_a| \leq n_i, n_i < n_q, \quad (3.45)$$

where $\mathbf{Z}_a$, $\mathbf{Y}_a$ and $\mathbf{S}_a$ are the collection of parameters, associated log-likelihood evaluations, and the collection GP inducing points respectively. The variable $\mathbf{s}$ denotes an inducing point location; n_q and n_i are respectively the maximal size of the dataset and the inducing point set. Sparse GPs enhance the computational efficiency of training the model when there is a large *active dataset*. We perform two actions on the *active dataset*. First, we add log-likelihood evaluations, which require evaluating the PDE model to the active dataset. Second, we remove data-points from the datasets to unburden the GP emulator of points that are far from the regions of high posterior density. This last step is important as GP training is expensive and if we only accumulate a larger and larger dataset, then the GP training might become more expensive than the FEM model itself.

We describe the criteria for adding and removing points for evaluations to the dataset as follows.

1. At iteration i , for every particle $\mathbf{z}$: if

$$\epsilon \|\phi^*(\mathbf{z})\| > \sigma_p(\mathbf{z}) \quad \text{and} \quad \sigma_p(\mathbf{z}) > \sigma_o, \quad (3.46)$$

then append $\{\mathbf{z}, \mathbf{y} = \log p(\mathbf{y}|\mathbf{z})\}$ to the dataset $\mathcal{D}_a$.

2. Ensure

$$\mathbf{Z}^* = \operatorname{argmax}_{n_q \mathbf{z}_j} \sum_k \left| \frac{\partial \sigma_p(\mathbf{z}_k)}{\partial \mathbf{z}_j} \right|, \quad (3.47)$$

$$\mathbf{S}^* = \operatorname{argmax}_{n_i \mathbf{s}_j} \sum_k \left| \frac{\partial \sigma_p(\mathbf{s}_k)}{\partial \mathbf{s}_j} \right|. \quad (3.48)$$

where $\mathbf{z}$ are the particle locations. Here, $\sigma_p(\mathbf{z})$ is the standard deviation of the GP surrogate and σ_o is a chosen uncertainty threshold. Equations (3.46) are responsible for ensuring the active surrogate is selecting particles from the SVGD cluster with the highest surrogate uncertainty that are still moving. This means that particles with surrogate value showing high uncertainty and not moving will not be selected to evaluate the likelihood function and will not be incorporated in the active dataset. Equations (3.47) and (3.48) are sensitivity measures of the uncertainty of the GP surrogate with respect to the dataset points. These equations ensure to find and remove the data points from the dataset that have, in some measure, the least impact on the uncertainty of the surrogate.

In Fig. 3.3 see the GP surrogate in the log-domain. In Fig. 3.4 we see the implementation of our ALPS-SVGD method on a one dimensional Bayesian inverse problem. The forward model is a 16 Element square domain Poisson problem. We can see the iterative construction of our GP surrogate using SVGD samples as locations to evaluate the log-likelihood according to (3.46). The red curve is the target distribution and the blue line is the GP surrogate. The construction of this surrogate is done in the log domain, but was exponentiated and normalized for visualization. The black dots are log-likelihood evaluation locations and the blue dots are the GP-SVGD samples. In Fig. 3.5 we can see the evolution of the magnitude of the gradient update step of the particles for this problem.

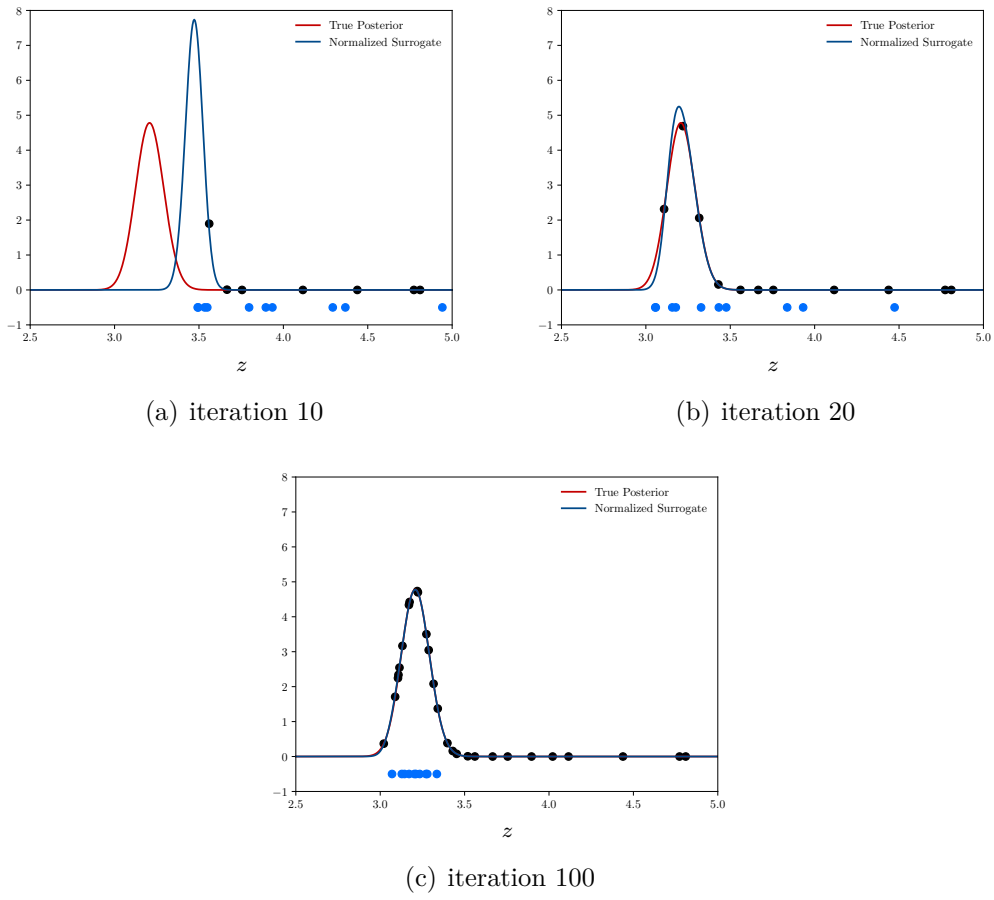


Fig. 3.4 1D sampling with ALPS

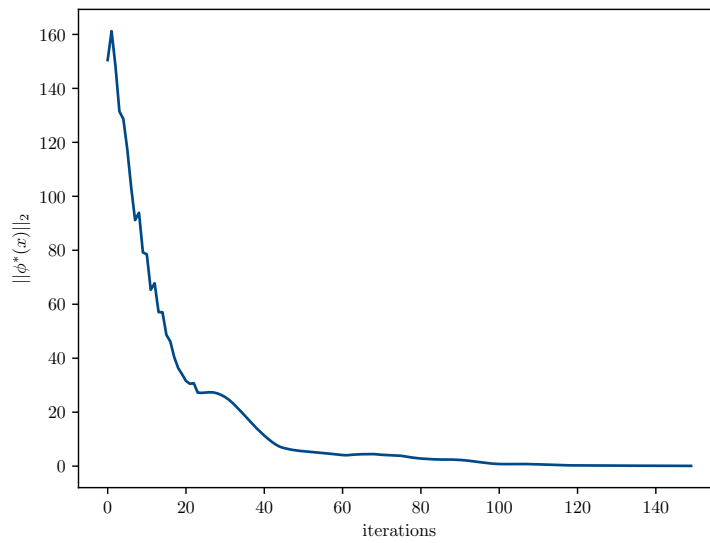


Fig. 3.5 Evolution of gradients

3.4 Numerical Illustration

We illustrate ALPS-SVGD on two examples of Bayesian inversion for two Poisson problems. For both of these examples we use a matrix valued kernel SVGD approach where we can easily compute approximations to Hessians based on the derivative information of the GP surrogate. The Poisson problem can be described as

$$\nabla \cdot (\kappa(x) \nabla u(x)) - f(x) = 0, \quad \text{for } x \in \Omega, \quad (3.49)$$

$$u(x) = 0, \quad \text{for } x \in \partial\Omega, \quad (3.50)$$

where $\kappa(x)$ describes a positive diffusion field by doing inference over the log diffusion. The field is discretised over the FEM elements which would normally induce a high dimensional posterior space for the inverse problem. Indeed, the dimension of the posterior space tends to infinity as the mesh size tends to zero. To reduce the size of the space and its dependence on the FEM discretisation, we project the representation of the diffusion field into a mesh independent space through a GP mean function with anchor point locations that modulate the diffusion field. These anchor points are our $\mathbf{z}$ variables. Hence, in the inverse problem we recover the values of the diffusion field at the anchor points. This, in turn results in a projection type SVGD method.

3.4.1 1D Poisson Problem

For a 1D Poisson problem, from the description in (3.49), we define the domain Ω to be a one dimensional line from 0 to 1. We choose a constant unit forcing function over the domain and the dataset $\mathbf{y}$ is the observation of the solution field at all nodal locations with a noise standard deviation $\sigma_y = 0.01$. The benchmark for this inverse problem is sampled using the affine invariant MCMC algorithm (Foreman-Mackey et al., 2013). In Fig. 3.6, we can see a comparison of the push-forward samples obtained through ALPS-SVGD and the EMCEE affine invariant ensemble sampler. The GP dataset had a max size of 600 data points and 300 inducing points and a log-Gaussian prior mean function with a target uncertainty standard deviation of $\sigma_p = 10^{-3}$. In Fig 3.6 (b) it required a total of 179232 forward model solves for convergence. Our ALPS-SVGD sampling algorithm required 4399 model solves (Fig 3.6 (a)).

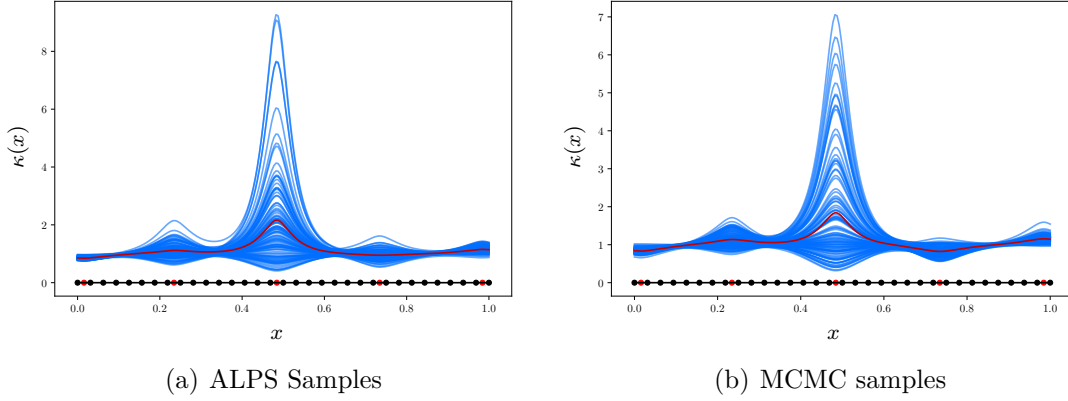


Fig. 3.6 100 samples (blue) of $\kappa(x)$ over a 32 Element Bar (black) with 5 Anchor Points (red) Matérn_{3/2} GP parameterisation, the mean of samples is plotted in red

3.4.2 2D Poisson Problem

For a 2D Poisson problem we show a numerical illustration of ALPS-SVGD. Here, the domain Ω is defined as a unit square $[0, 1] \times [0, 1]$. The domain is discretised into 256 elements and the 2-dimensional diffusion field is parametrised by a GP with 16 anchor point locations. Hence, the posterior measure exists in a 16 dimensional space. We observe the noisy solution field at all nodal locations with a noise standard deviation $\sigma_n = 0.001$. In Fig. 3.7 we see the diffusion field GP anchor point locations, the ground truth diffusion field, and the recovered mean estimate of $\kappa(\mathbf{x})$ obtained through ALPS-SVGD. On this same problem, the NUTS algorithm (Hoffman et al., 2014) required 19963 function and gradient evaluations. Using ALPS-SVGD we needed to evaluate the unnormalized posterior density at 72337 locations but no gradient evaluations. Fig. 3.7 shows the results from sampling the posterior distribution induced by the inverse problem. We notice a difference in the spatial distribution of the diffusion field means, largely explained by the fact that the mean of the distribution does not coincide with the MAP value of the posterior distribution. In Fig. 3.8 we have the evolution of the number of model evaluations queried over the entire inference process as well as the evolution of the magnitude of the particle update steps.

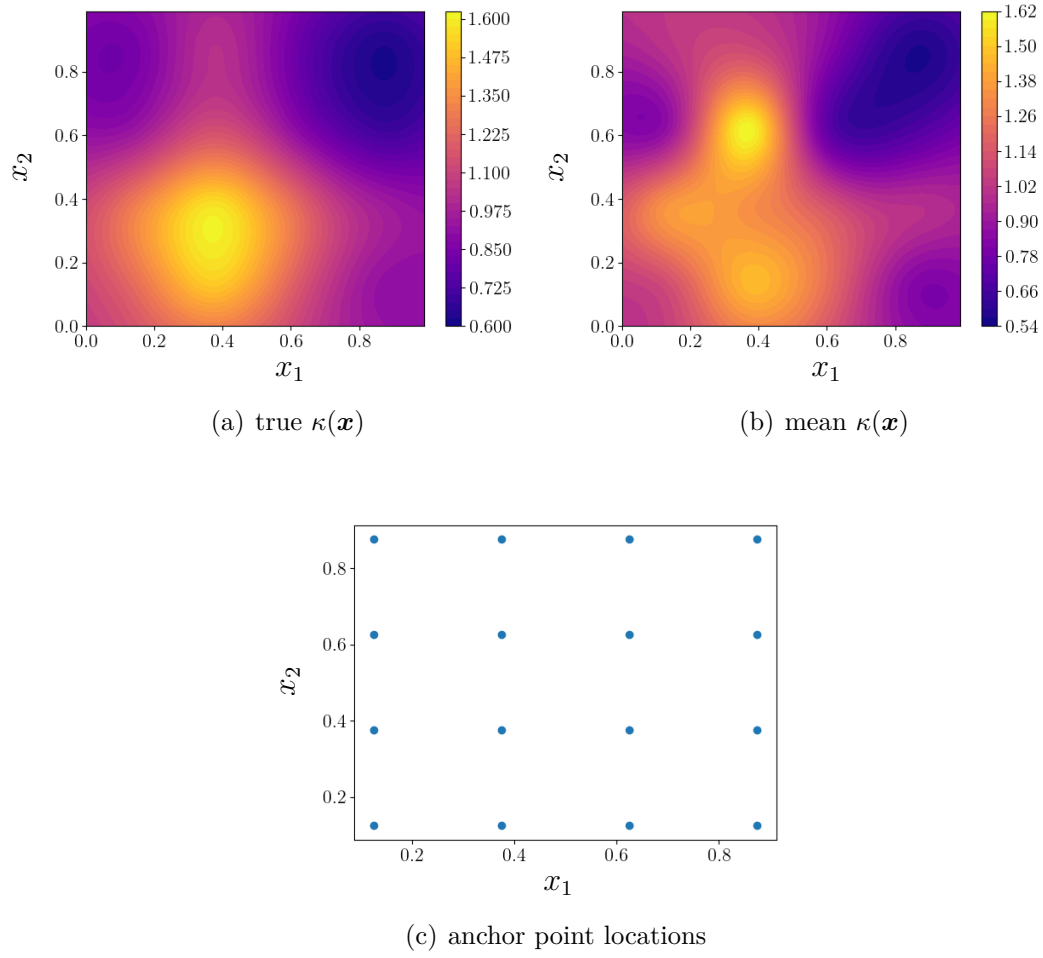


Fig. 3.7 ALPS results 2D diffusion field

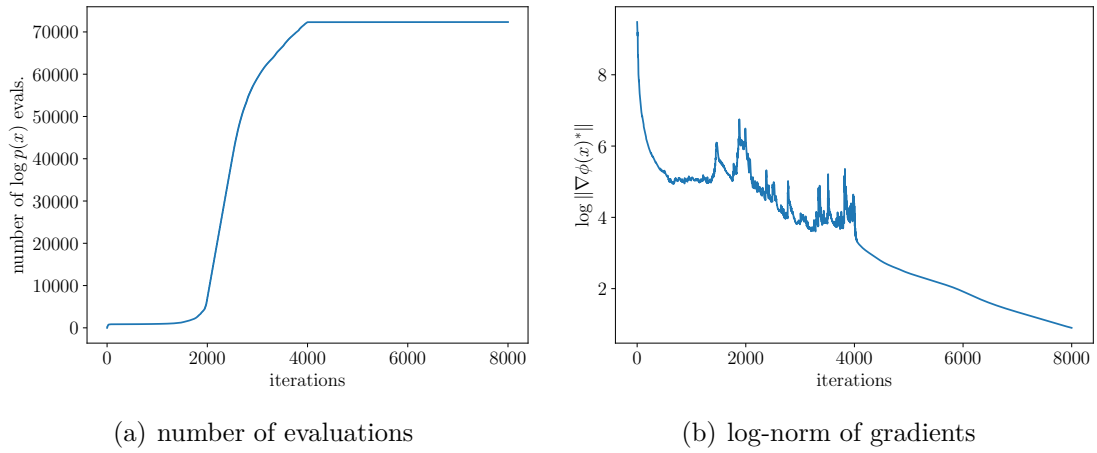


Fig. 3.8 2D Poisson runtime information for ALPS-SVG

3.5 Discussion

In this chapter, we have introduced the framework of Bayesian inverse problems for PDEs. We have discussed methods for approaching such sampling problems through MCMC as well as through SVGD algorithms. We outlined several SVGD schemes that extend vanilla-SVGD for different tasks. We then discussed surrogate modelling of posterior surfaces and the potential benefits of replacing the expensive to evaluate likelihood function with a GP surrogate. From there, we proposed our active learning framework ALPS-SVGD. In testing it we have combined active learning GP surrogates with a variety of SVGD algorithms, from the initial framework proposed in Liu and Wang (2016), to more expressive anisotropic kernels in Detommaso et al. (2018) as well as the matrix valued SVGD of Wang et al. (2019). We found through our method that the active dataset criteria set out in (3.46)-(3.48) behave reasonably well on low dimensional problems. This was seen from the data acquisition curve in Fig. 3.8 (a). The active surrogate gathered an initial amount of data, moved the particles to the approximate region of high posterior density, then refined the approximation with more data until saturation was reached.

To further improve upon the proposed ALPS-SVGD framework several extensions can be proposed. This includes the active learning of basis functions parametrising field information. Basis functions can be constructed from neural networks and these can be learned jointly with the sampling tasks in order to maximise the information captured in the posterior space. Another important extension can derive a neural network parameterised GP surrogate implementation to be trained online. This would alleviate some of the computational burden associated with GPs. If the dimension of the BIP can be reduced to a dozen dimensions and efficient surrogates implemented, then ALPS-SVGD may perform competitively in many sampling tasks.

As the ALPS-SVGD algorithm stands, the GP uncertainty is only incorporated into the learning scheme through the data point selection criteria in the active learning loop. An extension of the method could take advantage of the probabilistic nature of the GP surrogate by sampling random functions from the GP for the evaluation of the log-likelihood gradients (the score). This could potentially result in a more robust particle updating scheme that takes into account the surrogate uncertainty. Because the GP is a posterior over function values, sampling scores from the posterior GP for updating particle positions can be seen as a form of Thomson Sampling (Thompson, 1933, Russo et al., 2018). Basing actions on samples from GP posteriors through Thomson sampling has been shown to be an advantageous

decision scheme, and it has seen success in related problem settings such as Bayesian optimisation (Kandasamy et al., 2018).

Ultimately, from further experimentation on 16 dimensional problems and bigger, we found that the number of data points needed to correctly approximate the posterior surface grew uncontrollably with the number of dimensions of the posterior space. This incurs two problems for higher dimensional problems. Firstly, the number of model evaluations needed to construct the dataset rivals or surpasses the number of evaluations needed for gradient informed MCMC schemes such as NUTS (Hoffman et al., 2014). Secondly, once all these evaluations are collected into an active-dataset, the $\mathcal{O}(n^3)$ training cost of GPs makes them very expensive to use. Because of the nature of the algorithm, it is very difficult to then tune the level of posterior surface approximation done by the GP to use fewer points than what would be used for a high precision surrogate. In short, the active learning aspects of the algorithm make it efficient for low dimensional problems, but it can also make it non-robust for more challenging medium and high dimensional inference tasks.

We have created the proposed ALPS-SVGD algorithm for accelerating the sampling of low-dimensional BIPs. We need to also explore other means of solving PDE inverse problems in a more efficient and stable manner using VI, deep learning, and weighted residuals.

Chapter 4

Learning Parametric PDEs: Physics Driven Deep Latent Variable Models

4.1 Introduction

It is of great engineering interest to solve PDEs for ranges of parameters to obtain numerical simulations of various physical scenarios. The task of solving the PDEs for a particular set of parameter is termed the *forward problem*, for which an extensive number of solution techniques are available, including FEM, finite volumes, finite differences, and spectral methods (Ern and Guermond, 2004, Quarteroni and Valli, 2008). These methods provide a way to move from a parameter to the solution. The forward problem might be challenging to solve when the PDE is high-dimensional or nonlinear, requiring the numerical methods to have high precision, resulting in a heavy computational burden. On the other hand, given the experimental data of a certain phenomenon, it is often of interest to infer the associated unknown parameters from the experimental data to determine the parameter regimes of the observed system, a setting which is termed the *inverse problem* (Tarantola, 2005).

Recent growing interest in inverse problems is driven by the increased availability of sensor data in science and engineering and advances in ML in exploring high-dimensional spaces. Inverse problems are typically more difficult than forward problems due to under-specification and the fact that parameters explaining the observations may not be unique. This is particularly true in the presence of observation noise (Stuart, 2010b), missing physics, and high-dimensional parameter spaces (Vogel, 2002, Arridge et al., 2019).

Many modern engineering design scenarios require computationally expensive FEM solvers, limiting the amount of experimentation possible within a given time frame/computational budget. Being able to emulate both the forward and the inverse maps with emulators can enable practitioners to explore a variety of designs and experimental setups efficiently. For example, given a specific modelling problem (*e.g.* the design of a thin-shell car body) with known governing equations, a designer needs to find the best set of parameters (*e.g.* material properties, geometry, boundary conditions) to design a product with desired properties (*e.g.* a structure that does not buckle under a given load). An efficient forward emulator allows for the exploration of numerous designs by varying the parameter space; this generative process (of proposing design parameters and producing corresponding solutions) would be prohibitively expensive using standard FEM solvers as it requires expensive Assemble and Solve operations. In addition, observational data may be available for existing designs (either from experiments or high-fidelity simulations) and need to be incorporated into the framework. Similarly, constructing an efficient and accurate inverse emulator is of great use in many engineering scenarios. These scenarios include online quality control assessments in manufacturing and general parameter identification tasks in structural integrity assessments.

4.2 Contribution

The objective in this chapter is to simultaneously train emulators for the forward and inverse problem over a range of parameters, while at the same time building an interpretable, physically meaningful low-dimensional latent space of PDE parameters while providing uncertainty quantification (UQ). Such uncertainty quantification is of great value when modelling parametric PDEs in the forward direction through surrogates as we want to understand the distribution of model confidence between solve instances and within solutions.

UQ in the approximating posterior distributions in inverse problems is also of central importance due to the ill-posedness of many problems (Stuart, 2010b) as discussed in chapter 3. Most importantly, the Bayesian view in modelling PDEs gives us a principled framework to combine ML-based methods with classical numerical schemes in a coherent and reasoned manner, and allows the construction of a model whereby we can train surrogates to jointly learn a variety of probabilistic maps.

To achieve this objective, we create a probabilistic formulation and uniquely blend techniques from numerical solutions of PDEs, deep learning, and probabilistic

ML. As opposed to many methods previously outlined, the training of our proposed method is free from the costly forward solve operations; in other words, we do not need to generate PDE solutions to act as training data.

To summarise our plan: we propose a method that (i) simultaneously learns deep probabilistic forward and inverse emulators, (ii) is free from PDE forward numerical solve operations, (iii) uses the Bayesian view (Stuart, 2010b) of regularizing the inverse problem to address the inherent ill-posedness while calibrating model uncertainty in the forward direction, and (iv) allows for the incorporation of various neural network architectures tailored for parametric PDEs (such as PINNs (Raissi et al., 2019), neural operators (Li et al., 2020b, Lu et al., 2021, Anandkumar et al., 2020, Kovachki et al., 2021)) into a coherent probabilistic framework. We demonstrate this last point by showing that we can embed a PINN network into our variational family; thus our framework also allows for any architecture to be converted into a probabilistic method.

4.3 Deep Probabilistic Models for Parametric PDEs

FEM and PINNs allow to solve the forward problem for fixed PDE parameters. In settings that require the solution to PDEs for a variety of parameters, the use of either method can be computationally very expensive. On the other hand, the inverse problem of finding the model parameters $\mathbf{z}$, given some components of the solution $\mathbf{u}$ and the forcing $\mathbf{f}$, may be ill-posed and is generally a harder problem. Solving inverse problems for large collections of observations pertaining to different PDE instances is generally computationally infeasible using classical methods. To overcome these issues, we introduce deep probabilistic models to directly approximate the maps $(\mathbf{z}, \mathbf{f}) \mapsto \mathbf{u}$ and $(\mathbf{u}, \mathbf{f}) \mapsto \mathbf{z}$, while simultaneously providing uncertainty estimates. Once trained, our probabilistic models allow for the online solution of both the forward and inverse problem given a particular forcing $\mathbf{f}$. This is achieved through parametrised distributions which are controlled by a neural network. This network then in turn processes information from a given input to then modulate the distributions. Once trained, our probabilistic models allow for the online solution of both the forward and inverse problems.

We first describe, in Sec. 4.3.2, a method to train a deep probabilistic model to emulate the forward and inverse maps of a PDE without any observed data and without any independent solve operations. The framework is constructed around the maximisation of the probability of observing a zero values residual of the PDE.

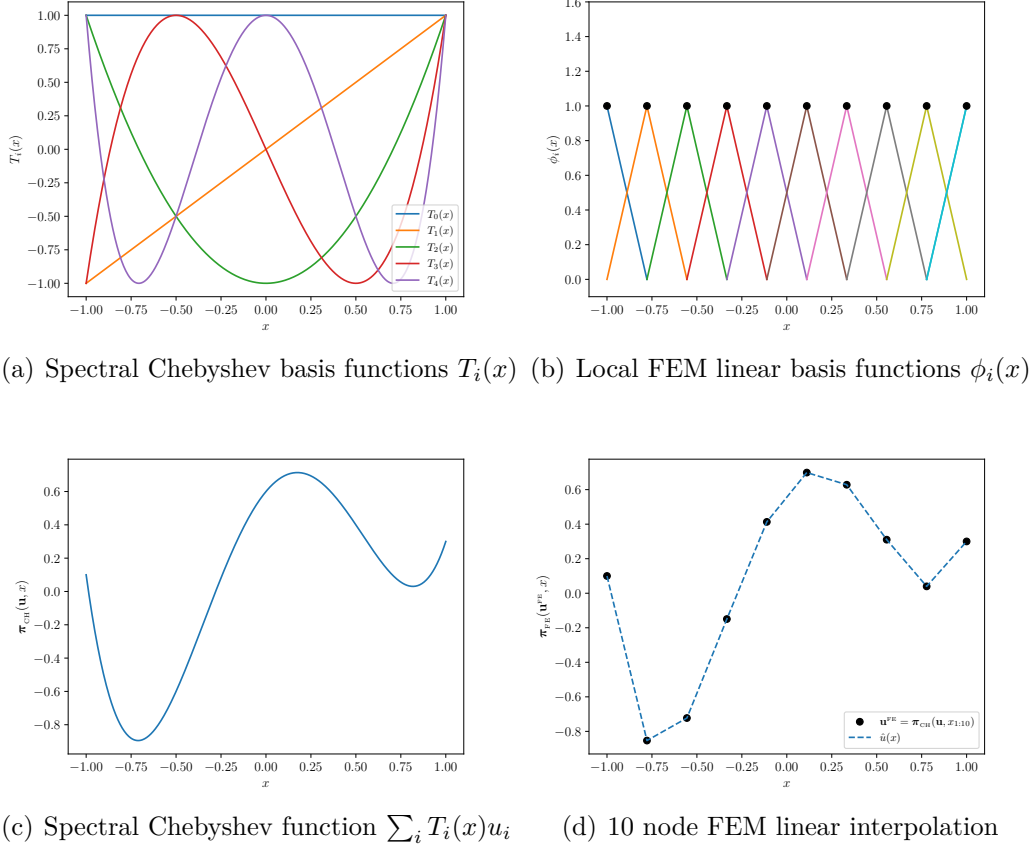


Fig. 4.1 In (a) we show 5 Chebyshev basis function, in (b) 10 linear FEM basis functions, in (c) a function composed of 5 Chebyshev basis functions and in (d) its interpolation using 10 linear FEM basis functions. We can see that we require many more terms in a linear interpolation to approximate smooth functions.

Building on this, in certain cases we will want to go beyond inverting a problem from the solution field $(\mathbf{u}, \mathbf{f}) \mapsto \mathbf{z}$ and invert some nonlinear observation map $g(\cdot)$ acting on $\mathbf{u}$. To this end we introduce in Sec. 4.3.3 a version of our framework which incorporates observational data and learns a variational approximation to inverse the mapping from solution field to observation.

4.3.1 Finite Elements and Parametrisations

It is key to leverage the advantages of various discrete representations of solution fields when constructing probabilistic emulators for the solution of physics problems. Finite elements expansions provide flexible function representations which are versatile, but tend to be very high dimensional. By making use of a low dimensional spectral Chebyshev representation of solution fields we can reduce the number of degrees of

freedom necessary to describe smooth solutions. Spectral expansions can easily be projected onto arbitrary finite element meshes which is a key characteristic when constructing physics emulators as we typically wish to be mesh independent. We describe the various representations of $u(x)$ as through mappings $\boldsymbol{\pi}$ as

$$\mathbf{u} \xrightarrow{\boldsymbol{\pi}_{\text{CH}}} \mathbf{u}^{\text{FE}} \xrightarrow{\boldsymbol{\pi}_{\text{FE}}} \hat{u}(x), \quad (4.1)$$

where $\mathbf{u}$ denotes the spectral representation of solution fields and $\mathbf{u}^{\text{FE}}$ are the finite element coefficients. The mapping $\boldsymbol{\pi}_{\text{CH}}$ is a Chebyshev mapping of the spectral coefficients to a set of spatial locations $x_{1:n}$, given by

$$\mathbf{u}^{\text{FE}} = \boldsymbol{\pi}_{\text{CH}}(\mathbf{u}, x_{1:n}) = \left[\sum_i T_i(x_j) u_i \right]_{j=1:n}, \quad (4.2)$$

where $T_i(x)$ is the i th Chebyshev polynomial. We use a finite element expansion as the approximant solution field

$$\hat{u}(x) = \boldsymbol{\pi}_{\text{FE}}(\mathbf{u}^{\text{FE}}, x) = \sum_i \phi_i(x) u_i^{\text{FE}}, \quad (4.3)$$

where $\phi(x)$ the FEM basis functions. In a Bubnov-Galerkin approach we pose $w_i(x) = \phi_i(x)$. The coefficients u_i on the boundaries $\partial\Omega$ are chosen to respect the (Dirichlet) boundary conditions. The mesh allows us to both define the basis functions and describe the geometry of the domain Ω . We use integration by parts to obtain the *weak form* of (2.42) prior to finite element discretisation (Brenner et al., 2008).

It is similarly advantageous to represent our parameter field $z(x)$ through a spectral approximation

$$\mathbf{z} \xrightarrow{\boldsymbol{\pi}_{\text{CH}}} \hat{z}(x), \quad (4.4)$$

which is chosen to be a spectral parametrisation as

$$\hat{z}(x) = \boldsymbol{\pi}_{\text{CH}}(\mathbf{z}, x) = \sum_i T_i(x) z_i. \quad (4.5)$$

We parametrise forcing fields in a similar way to the solution fields through

$$\mathbf{f} \xrightarrow{\boldsymbol{\pi}_{\text{CH}}} \mathbf{f}^{\text{FE}} \xrightarrow{\boldsymbol{\pi}_{\text{FE}}} \hat{f}(x) \xrightarrow{\text{WR}} \mathbf{f}^{\text{WR}}, \quad (4.6)$$

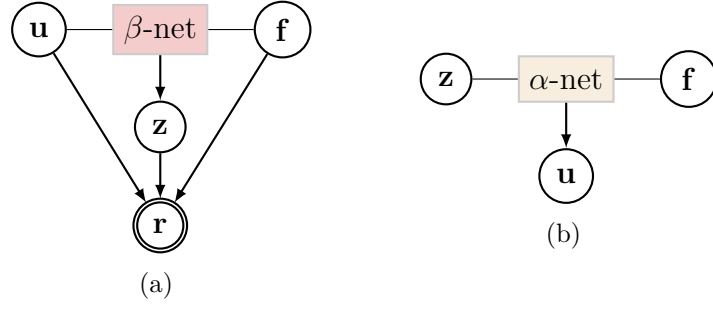


Fig. 4.2 We show the relationship between the solution $\mathbf{u}$, the model parameters $\mathbf{z}$, the forcing $\mathbf{f}$, and the residual $\mathbf{r}$. In (a) we show the probabilistic graphical model with the *inverse* (β) neural network, where the double rings indicate that $\mathbf{r}$ is an observed vector. In (b) we have the probabilistic graphical model with the *forward* (α) neural network.

each mapping is then given as

$$\mathbf{f}^{\text{FE}} = \boldsymbol{\pi}_{\text{CH}}(\mathbf{f}, x_{1:n}) = \left[\sum_i T_i(x_j) f_i \right]_{j=1:n}, \quad (4.7a)$$

$$\hat{f}(x) = \boldsymbol{\pi}_{\text{FE}}(\mathbf{f}^{\text{FE}}, x) = \sum_i \phi_i(x) f_i^{\text{FE}}, \quad (4.7b)$$

where we use (2.44) to map $\mathbf{f}^{\text{FE}}$ to $\mathbf{f}^{\text{WR}}$ through $\hat{f}(x)$ to be used in (2.45). These various parametrisations allow us to represent solution fields, parameter fields, and forcing fields in a FEM mesh invariant manner. We show in Fig 4.1 a representation of 5 spectral Chebyshev basis functions and 9 FEM elements and their local basis functions as well as the interpolation of a Chebyshev function by FEM basis functions.

4.3.2 Probabilistic Model without Observed Data

Following the weighted residual approach in Sec. 2.3.1 we consider the residual $\mathbf{r} \in \mathbb{R}^d$ in (2.45) as an auxiliary random variable that is observed and has the value $\mathbf{r} = \mathbf{0}$. As in Gelman et al. (1995), the first step in constructing a probabilistic model is to pose some joint distributions over the variables of interest. Now, we pose the joint distribution over all variables as

$$p_{\beta}(\mathbf{r}, \mathbf{u}, \mathbf{z}, \mathbf{f}) = p(\mathbf{r}|\mathbf{u}, \mathbf{z}, \mathbf{f}) p_{\beta}(\mathbf{z}|\mathbf{u}, \mathbf{f}) p(\mathbf{u}) p(\mathbf{f}), \quad (4.8)$$

where β is the trainable set of parameters in our model, see Figure 4.2.

Such a factorisation is a modelling choice; the joint distribution could in fact be factorised in different ways so long as the rules of probability are respected. The posed factorisation was chosen to obtain a conditional residual distribution as well as a conditional distribution of $\mathbf{z}$ given the solution $\mathbf{u}$ and the forcing $\mathbf{f}$. The assumption of this particular model is that $\mathbf{u}$ and $\mathbf{f}$ are independent when none of the variables in the model are observed. However, it is easy to show that $\mathbf{u}$ and $\mathbf{f}$ become conditionally dependent when $\mathbf{r} = \mathbf{0}$ is assumed to be observed.

We pose the probability of the residual as

$$p(\mathbf{r}|\mathbf{u}, \mathbf{z}, \mathbf{f}) = \mathcal{N}(\mathbf{r}; \mathbf{A}_{\mathbf{z}}(\mathbf{u}) - \mathbf{f}^{\text{WR}}, \Sigma_{\mathbf{r}}), \quad (4.9)$$

where $\Sigma_{\mathbf{r}}$ is the covariance matrix indicating the desired accuracy of the discrete solution $\mathbf{u}$ expressed as the deviation of the residual from zero. Treating $\mathbf{r}$ as an observed vector and setting $\mathbf{r} = \mathbf{0}$ as in Sec. 2.3.1, we aim to maximise its marginal likelihood $p(\mathbf{r} = \mathbf{0})$ by integrating out all other variables. We next choose deep a probabilistic model of the form $p_{\beta}(\mathbf{z}|\mathbf{u}, \mathbf{f}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_{\beta}(\mathbf{u}, \mathbf{f}), \Sigma_{\beta}(\mathbf{u}, \mathbf{f}))$, where $\boldsymbol{\mu}_{\beta}$ and Σ_{β} are represented by neural networks with the set of parameters β consisting of weights and biases. Our model resembles the variational autoencoder (VAE) structure; however, we use a physics-informed model and formulate the parameter as the latent vector $\mathbf{z}$. The direct correspondence between latent variables and physical parameters enables us to have latent variables with a clear physical interpretation.

We also use a decoder-like distribution to learn the conditional density $p_{\beta}(\mathbf{z}|\mathbf{u}, \mathbf{f})$ which gives the flexibility to obtain $\mathbf{z}$ given a solution field $\mathbf{u}$ and forcing $\mathbf{f}$. To this end, we pose the factorisation

$$q_{\alpha}(\mathbf{u}, \mathbf{z}, \mathbf{f}) = q_{\alpha}(\mathbf{u}|\mathbf{z}, \mathbf{f})p(\mathbf{z})p(\mathbf{f}), \quad (4.10)$$

with the approximation $q_{\alpha}(\mathbf{u}|\mathbf{z}, \mathbf{f}) = \mathcal{N}(\mathbf{u}; \boldsymbol{\mu}_{\alpha}(\mathbf{z}, \mathbf{f}), \Sigma_{\alpha}(\mathbf{z}, \mathbf{f}))$, where $\boldsymbol{\mu}_{\alpha}$ and Σ_{α} are represented by neural networks with the set of parameters α consisting of weights and biases, see Fig. 4.2.

Inspired by the weighted residual approach introduced in Sec. 2.3.1 (see also (Rixner and Koutsourelakis, 2021)), we maximise the marginal likelihood by fixing $\mathbf{r} = \mathbf{0}$ and deriving an ELBO that lower bounds the log marginal likelihood $\log p(\mathbf{r} = \mathbf{0})$. This will in effect tune the parameters α and β in such way as to generate content when sampled that captures the physics specified by the residual.

In order to do this, we note through the rule of marginalization (2.1)

$$p(\mathbf{r} = \mathbf{0}) = \int p_\beta(\mathbf{r} = \mathbf{0}, \mathbf{u}, \mathbf{z}, \mathbf{f}) d\mathbf{u} d\mathbf{z} d\mathbf{f}, \quad (4.11)$$

where the $p(\mathbf{r} = \mathbf{0})$ is the *evidence*. The aim is to maximise the evidence, i.e. the probability of observing $\mathbf{r} = \mathbf{0}$, by optimising the neural network parameters α and β . To this end, we introduce an instrumental variational approximation and rewrite (4.11) as

$$p(\mathbf{r} = \mathbf{0}) = \int \frac{p_\beta(\mathbf{r} = \mathbf{0}, \mathbf{u}, \mathbf{z}, \mathbf{f})}{q_\alpha(\mathbf{u}, \mathbf{z}, \mathbf{f})} q_\alpha(\mathbf{u}, \mathbf{z}, \mathbf{f}) d\mathbf{u} d\mathbf{z} d\mathbf{f}, \quad (4.12)$$

by dividing and multiplying the integrand in (4.11) by $q_\alpha(\mathbf{u}, \mathbf{z}, \mathbf{f})$.

Next, we compute the logarithm of this quantity, since maximising $\log p(\mathbf{r} = \mathbf{0})$ is equivalent to maximising $p(\mathbf{r} = \mathbf{0})$ and is more numerically stable. Using Jensen's inequality (2.34) we can obtain an expression of the form

$$\log p(\mathbf{r} = \mathbf{0}) \geq \mathcal{F}(\alpha, \beta), \quad (4.13)$$

and using (4.8) and (4.10), we arrive at the lower bound

$$\mathcal{F}(\alpha, \beta) = \int \log \frac{p(\mathbf{r} = \mathbf{0} | \mathbf{u}, \mathbf{z}, \mathbf{f}) p_\beta(\mathbf{z} | \mathbf{u}, \mathbf{f}) p(\mathbf{u})}{q_\alpha(\mathbf{u} | \mathbf{z}, \mathbf{f}) p(\mathbf{z})} q_\alpha(\mathbf{u} | \mathbf{z}, \mathbf{f}) p(\mathbf{z}) p(\mathbf{f}) d\mathbf{u} d\mathbf{z} d\mathbf{f}. \quad (4.14)$$

The ELBO obtained in this manner can also be derived using the Kullback–Leibler divergence between the posterior $p(\mathbf{u}, \mathbf{z}, \mathbf{f} | \mathbf{r})$ and the variational approximation $q_\alpha(\mathbf{u}, \mathbf{z}, \mathbf{f})$ to it. This link is especially evident in VAE literature (Kingma and Welling, 2019). A full derivation of our PDDLVM framework can be found in Appendix A.2 where we show that our framework can be derived from the KL divergence between the intractable posterior of our latent variable given the pseudo-observed residual $\mathbf{r} = \mathbf{0}$ and a tractable variational approximation in the form $D_{KL}(q_\alpha(\mathbf{u}, \mathbf{z}, \mathbf{f}) || p_\beta(\mathbf{u}, \mathbf{z}, \mathbf{f} | \mathbf{r}))$. Using the derived ELBO, we seek

$$\{\alpha^*, \beta^*\} \in \operatorname{argmax}_{\beta, \alpha} \mathcal{F}(\alpha, \beta). \quad (4.15)$$

We note that the α and β networks (which parametrise $q_\alpha(\mathbf{u} | \mathbf{z}, \mathbf{f})$ and $p_\beta(\mathbf{z} | \mathbf{u}, \mathbf{f})$, respectively) have intuitive functionalities (see Fig. 4.2 for the graphical models). In particular, the α network provides a forward emulator, while the β network provides an inverse emulator, both of which producing uncertainty estimates associate with the predictions. Modern VI techniques rely on maximising the lower bound $\mathcal{F}(\alpha, \beta)$,

instead of the true evidence $\log p(\mathbf{r} = \mathbf{0})$ to learn the parameters (α, β) . We denote the distribution associated with the forward problem the α -Net and the inverse map is called the β -Net. To compute (4.14) we express the integral as an expectation and use Monte-Carlo approximation of this expectation. The N -sample Monte Carlo ELBO estimate of the gradient of $\mathcal{F}(\alpha, \beta)$ can be obtained by sampling from $p(\mathbf{z})$, $p(\mathbf{f})$ and $q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{f})$ (using the reparametrisation trick, see Sec. 2.2.2). For this, we rewrite ELBO (4.14) as an expectation

$$\mathcal{F}(\alpha, \beta) = \mathbb{E}_{q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{f})p(\mathbf{z})p(\mathbf{f})}[\mathbf{v}_{\alpha, \beta}(\mathbf{u}, \mathbf{z}, \mathbf{f})], \quad (4.16)$$

where

$$\mathbf{v}_{\alpha, \beta}(\mathbf{u}, \mathbf{z}, \mathbf{f}) = \log \frac{p(\mathbf{r} = \mathbf{0}|\mathbf{u}, \mathbf{z}, \mathbf{f})p_\beta(\mathbf{z}|\mathbf{u}, \mathbf{f})p(\mathbf{u})}{q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{f})p(\mathbf{z})}. \quad (4.17)$$

Given that $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, we can sample $\mathbf{u} \sim q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{f})$ by sampling $\boldsymbol{\varepsilon}$ and computing

$$\mathbf{w}_\alpha(\mathbf{z}, \mathbf{f}, \boldsymbol{\varepsilon}) = \boldsymbol{\mu}_\alpha(\mathbf{z}, \mathbf{f}) + \boldsymbol{\Sigma}_\alpha^{1/2}(\mathbf{z}, \mathbf{f})\boldsymbol{\varepsilon}, \quad (4.18)$$

and write

$$\mathcal{F}(\alpha, \beta) = \mathbb{E}_{q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{f})p(\mathbf{z})p(\mathbf{f})}[\mathbf{v}_{\alpha, \beta}(\mathbf{u}, \mathbf{z}, \mathbf{f})] = \mathbb{E}_{q(\boldsymbol{\varepsilon})p(\mathbf{z})p(\mathbf{f})}[\mathbf{v}_{\alpha, \beta}(\mathbf{w}_\alpha(\mathbf{z}, \mathbf{f}, \boldsymbol{\varepsilon}), \mathbf{z}, \mathbf{f})].$$

To construct the Monte Carlo ELBO, we now sample $\boldsymbol{\varepsilon}^{(j)} \sim q(\boldsymbol{\varepsilon})$, $\mathbf{z}^{(j)} \sim p(\mathbf{z})$, $\mathbf{f}^{(j)} \sim p(\mathbf{f})$ for $j = 1, \dots, N$ and obtain

$$\mathcal{F}^N(\alpha, \beta) = \frac{1}{N} \sum_{j=1}^N \mathbf{v}_{\alpha, \beta}(\mathbf{w}_\alpha(\mathbf{z}^{(j)}, \mathbf{f}^{(j)}, \boldsymbol{\varepsilon}^{(j)}), \mathbf{z}^{(j)}, \mathbf{f}^{(j)}). \quad (4.19)$$

The rest of the training is done by computing the gradient of this stochastic loss with respect to α, β , and by running a variant of gradient descent, such as Adam (Kingma and Ba, 2015). In many practical applications and in this work, $N = 1$, *i.e.*, a single-sample estimate of the gradient is utilized (Kingma and Welling, 2019). We note that one of the strengths of this method is that both linear and nonlinear PDEs can be treated the same way as we only need to evaluate the residual term and its gradient.

Modelling the residual of a PDE as an observable for which we want to maximise the probability of observing a desirable quantity ($\mathbf{r} = \mathbf{0}$) has parallels with the probabilistic control method of (Kappen et al., 2012). Here the authors setup a KL divergence based framework relating actions to an arbitrary state dependent control

Algorithm 3: Pseudocode for PDDLVM

Initialise: α_0, β_0, T (number of iterations), N (number of MC samples).
for $t = 1, \dots, T$
 for $i = 1, \dots, N$
 Sample $\mathbf{z}^{(i)} \sim p(\mathbf{z})$
 Sample $\mathbf{f}^{(i)} \sim p(\mathbf{f})$
 Sample $\mathbf{u}^{(i)} \sim q_{\alpha_{t-1}}(\mathbf{u}|\mathbf{z}^{(i)}, \mathbf{f}^{(i)})$
 Compute $\mathcal{F}^N(\alpha, \beta)$ using samples $\triangleright$ Monte Carlo estimate of (4.14)
 $(\alpha_t, \beta_t) \leftarrow \text{OPTIMISER}(\alpha_{t-1}, \beta_{t-1}, \mathcal{F}^N(\alpha, \beta)) \triangleright$ Update parameters

cost. Bringing this to our method, we can interpret our predictions over $\mathbf{u}$ and $\mathbf{z}$ as uncontrolled dynamics that are tied together probabilistically through a graphical structure connecting them to the cost, i.e. the residual.

4.3.3 Probabilistic Model with Observed Data

When observing experimental data it might not be possible to directly observe the solution $\mathbf{u}$ directly, but rather a known nonlinear transformation of the solution field. For this we pose an observational model such as

$$\mathbf{y} = \mathbf{g}(\mathbf{u}) + \mathbf{e}, \quad \mathbf{e} \sim \mathcal{N}(\mathbf{0}, \sigma_n^2 \mathbf{I}). \quad (4.20)$$

We introduce a new model to deal with such observations for this data-centric scenario. Our intent is to approximate some $\mathbf{g}^{-1}(\cdot)$ map while respecting the underlying physics specified by the residual. If $\mathbf{g}(\cdot)$ has a known form this might be a trivial task in the bijective case. If however $\mathbf{g}(\cdot)$ is a non-bijective mapping, such as a sparse observation operator, this is a non-trivial task and must be regularized somehow by physics.

In this section, we assume that the β -network yielding the distribution $p_\beta(\mathbf{z}|\mathbf{u}, \mathbf{f})$ is pretrained as described in Sec. 4.3.2. We denote the learned β network parameter by setting $\beta = \beta_*$. For a given observation set $\{\mathbf{y}_i\}_{i=1}^n$ and the corresponding solution set $\{\mathbf{u}_i\}_{i=1}^n$ we introduce the observation likelihood

$$p(\mathbf{y}_i|\mathbf{u}_i) = \mathcal{N}(\mathbf{y}_i; \mathbf{g}(\mathbf{u}_i), \Sigma_{\mathbf{y}}), \quad \text{for } i = 1, \dots, n, \quad (4.21)$$

where $\mathbf{g} : \mathbb{R}^n \rightarrow \mathbb{R}^{n_y}$ is a *known* observation operator and $\Sigma_{\mathbf{y}}$ is the noise covariance. We pose our probabilistic model for a tuple $(\mathbf{y}_i, \mathbf{u}_i)$ as

$$p(\mathbf{y}_i, \mathbf{u}_i) = p(\mathbf{y}_i|\mathbf{u}_i)p(\mathbf{u}_i). \quad (4.22)$$

Note in this case that our probability model is fully specified, with no trainable parameters. We next describe our variational approximation as

$$q_\gamma(\mathbf{u}_i|\mathbf{y}_i) = \mathcal{N}(\mathbf{u}_i; \boldsymbol{\mu}_\gamma(\mathbf{y}_i), \boldsymbol{\Sigma}_\gamma(\mathbf{y}_i)). \quad (4.23)$$

We write out the marginal likelihood as

$$p(\mathbf{y}_i) = \int p(\mathbf{y}_i, \mathbf{u}_i) d\mathbf{u}_i. \quad (4.24)$$

Incorporating the variational distribution $q_\gamma(\mathbf{u}_i|\mathbf{y}_i)$ for the **observation-to-solution** map we can write for the log marginal likelihood

$$\log p(\mathbf{y}_i) = \log \int \frac{p(\mathbf{y}_i, \mathbf{u}_i)}{q_\gamma(\mathbf{u}_i|\mathbf{y}_i)} q_\gamma(\mathbf{u}_i|\mathbf{y}_i) d\mathbf{u}_i. \quad (4.25)$$

Using Jensen's inequality this expression can be bounded by the ELBO

$$\mathcal{F}_i(\gamma) = \int \log \frac{p(\mathbf{y}_i|\mathbf{u}_i)p(\mathbf{u}_i)}{q_\gamma(\mathbf{u}_i|\mathbf{y}_i)} q_\gamma(\mathbf{u}_i|\mathbf{y}_i) d\mathbf{u}_i. \quad (4.26)$$

Given the dataset $\{\mathbf{y}_i\}_{i=1}^n$, the full ELBO is $\mathcal{F}(\gamma) = \sum_{i=1}^n \mathcal{F}_i(\gamma)$. Once trained, this map is sufficient to obtain an encoding model for observations. The lower bound can be computed similarly to the ELBO in Sec. 4.3.2 with

$$\mathcal{F}_i(\gamma) = \mathbb{E}_{q_\gamma(\mathbf{u}_i|\mathbf{y}_i)}[\mathbf{v}_\gamma(\mathbf{y}_i, \mathbf{u}_i)], \quad (4.27)$$

where

$$\mathbf{v}_\gamma(\mathbf{y}_i, \mathbf{u}_i) = \log \frac{p(\mathbf{y}_i|\mathbf{u}_i)p(\mathbf{u}_i)}{q_\gamma(\mathbf{u}_i|\mathbf{y}_i)} q_\gamma(\mathbf{u}_i|\mathbf{y}_i), \quad (4.28)$$

which can also be computed through the reparametrisation trick detailed in (4.29) as

$$\mathbf{w}_\gamma(\mathbf{y}_i, \boldsymbol{\varepsilon}) = \boldsymbol{\mu}_\gamma(\mathbf{y}_i) + \boldsymbol{\Sigma}_\gamma^{1/2}(\mathbf{y}_i)\boldsymbol{\varepsilon}, \quad (4.29)$$

and write

$$\mathcal{F}_i(\gamma) = \mathbb{E}_{q_\gamma(\mathbf{u}_i|\mathbf{y}_i)}[\mathbf{v}_\gamma(\mathbf{y}_i, \mathbf{u}_i)] = \mathbb{E}_{q(\boldsymbol{\varepsilon})}[\mathbf{v}_\gamma(\mathbf{y}_i, \mathbf{w}_\gamma(\mathbf{y}_i, \boldsymbol{\varepsilon}))].$$

This can be approximated with Monte-Carlo sampling as done in (4.19).

To realize the inverse emulator, we can use the already learned $\beta_\star$ -network to marginalize over $\mathbf{u}$ as

$$p(\mathbf{z}|\mathbf{y}, \mathbf{f}) = \int p_{\beta_\star}(\mathbf{z}|\mathbf{u}, \mathbf{f}) q_{\gamma_\star}(\mathbf{u}|\mathbf{y}) d\mathbf{u}. \quad (4.30)$$

The marginalization of $\mathbf{z}$ over $\mathbf{u}$ can be easily extended to include the forcing $\mathbf{f}$. We stress that the method we propose only uses data when we are inverting some observable map from a dataset, this extension does not learn the physics of a problem but rather some transformation that is over the underlying physics learned with the previously trained α -Net and β -Net. The physics is learned using the algorithm described in Sec. 4.3.2 and this probabilistic physics model can then be used for combined inference as shown in (4.30).

4.3.4 Algorithm and Implementation Details

Priors and Monte Carlo ELBO

In all formulations, we assume a flat prior $p(\mathbf{u}) \propto 1$ for the solution field. We choose $p(\mathbf{f})$ and $p(\mathbf{z})$ as uniform or normal distributions (see Sec. 4.4). We estimate the ELBOs in (4.14) using sampling from $\mathbf{z} \sim p(\mathbf{z})$, $\mathbf{f} \sim p(\mathbf{f})$, and $\mathbf{u} \sim q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{f})$ (using the reparametrisation trick (Kingma and Welling, 2013)) and optimise the ELBOs with Adam (Kingma and Ba, 2015). We use only single-sample estimate of the ELBO, resulting in a doubly-stochastic gradient. See Algorithm 3 for the full algorithm. We also provide details in Sec. 4.4 where necessary.

The frameworks derived in Sec. 4.3.2 and Sec. 4.3.3 are agnostic to the method of computing the residual. In this work we explore two variations. The first is based on a FEM formulation of the residual which uses Lagrange polynomial trial and test function, we call this FEM-PDDLVM. The second variation is a PINN based formulation where the trial solution is given directly by the network and we use collocation points, we call this PINN-PDDLVM.

In Fig. 4.3 we show the architecture of both FEM-PDDLVM and PINN-PDDLVM. The large “ $\sim$ ” symbol denotes the drawing of random samples from the distribution (or from the distribution described by the given parameters) on the left of it. The two variants of PDDLVM differ in the fact that for the FEM version we reparametrise the samples $\mathbf{u}^{(i)}$ with a global expansion. For the PINN-PDDLVM version we evaluate the network at the $x_{1:n}$ collocation gridpoints and sample directly in the solution space of the domain, this yields the simplification $u_j^{(i)} = \hat{u}^{(i)}(x_j)$. In the PINN version the β -Net is implemented as a convolutional neural network taking in the

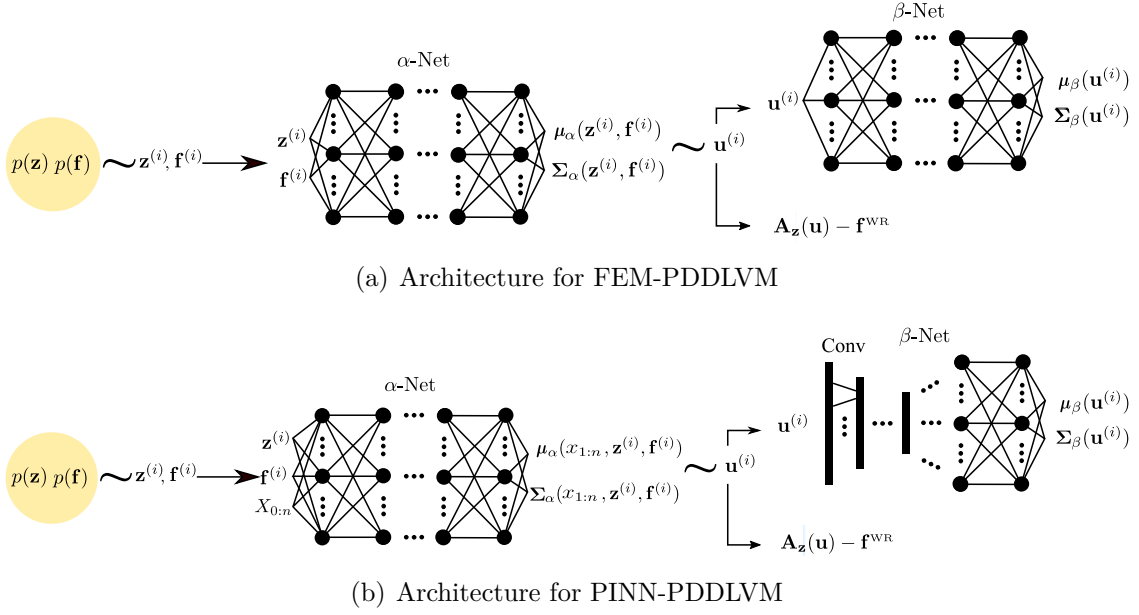


Fig. 4.3 Architectures FEM-PDDLVM and PINN-PDDLVM

trial solution evaluated on the grid, it can in essence be interpreted as an image. With all the quantities computed through the given samples we can then evaluate all the relevant terms in the ELBO.

4.4 Examples

Below, we demonstrate and test our methodology on six selected examples. In all experiments, we choose a diagonal covariance for the networks, although other choices could be explored (Kingma and Welling, 2019). Such neural network controlled variational approximations are known to be occasionally unstable due to exploding gradients. To mitigate this we make use of a bounded re-parameterisation of the log variance of the neural networks (Dehaene and Brossard, 2021).

The examples in Sec. 4.4.1, Sec. 4.4.2, and Sec. 4.4.3 all use a FEM-PDDLVM residual formulation and the output of the neural network is parametrised with a Chebyshev expansion. Sec. 4.4.4 and Sec. 4.4.5 are based on the PINN formulation of the residual. The code for the examples can be found at <https://github.com/ArnaudVadeboncoeur/PDDLVM>.

4.4.1 1D Linear Poisson

We train the α and β networks on a linear Poisson equation with Dirichlet boundary conditions as per the formulation in Sec. 4.3.2 to demonstrate the method on a simple example. We learn the forward emulator jointly with the inverse emulator with uncertainties. The Poisson problem is formulated as

$$\begin{aligned} -\nabla \cdot (\kappa(x) \nabla u(x)) &= f(x), \quad x \in (-1, 1), \\ u(-1) &= a, \quad u(1) = b, \end{aligned} \tag{4.31}$$

where a and b denote the Dirichlet boundary condition values. When using (4.5) to represent a physical quantity which is strictly positive we compose the spectral mapping π_{CH} with a Softplus transformation of the form $\log(1 + \exp(x))$. As the transformation is nonlinear, we make use of the unscented transform (Wan and Van Der Merwe, 2000) for estimation of the first and second moments of the probability density when mapping the probability of $\mathbf{z}$ onto the domain Ω . When we project $\mathbf{z}$ in Chebyshev coefficients onto the finite element mesh as $\hat{z}(x_{1:n})$, we can transform its probability distributions as $p(\hat{z}(x_{1:n})) = \mathcal{N}(\hat{z}(x_{1:n}); \boldsymbol{\mu}_\beta \mathbf{T}^\top, \mathbf{T} \boldsymbol{\Sigma}_\beta \mathbf{T}^\top)$, where $\mathbf{T}$ is the Chebyshev Vandermonde matrix at the mesh locations $x_{1:n}$. The choice of Chebyshev polynomials for this expansion is motivated by their stability (as opposed to monomial or Lagrange polynomials, which suffer from the Runge's phenomenon) and the ease with which they approximate constant, linear, and quadratic functions. Other expansions such as Fourier series might require many terms to approximate these simple functions with accuracy.

Joint Learning Without Observations

In this section, we apply our method on the linear Poisson problem without observations as in Sec. 4.3.2 and measure the accuracy after training by computing the true FEM solves (however, these are not used during training). The residual is marginalized over the prior of $\mathbf{z}$ and $\mathbf{f}$ which allow to learn the PDE over the prior ranges. Our interpretable latent variable $\mathbf{z} = \boldsymbol{\kappa}$ denotes the coefficients of the $\kappa(x)$ expansion.

As per Fig. 4.4, the values for the boundary conditions a and b are kept fixed at 0 and 0.5. We plot the results from the PDDLVM inference for 10 random samples drawn from the prior of κ . The diffusivity field $\kappa(x)$ and forcing $f(x)$ is parametrised as a constant function and the solution field is given by a 3rd order Chebyshev expansion. The prior over each of the elements of κ is $\mathcal{N}(0, 0.5)$. We select a NN

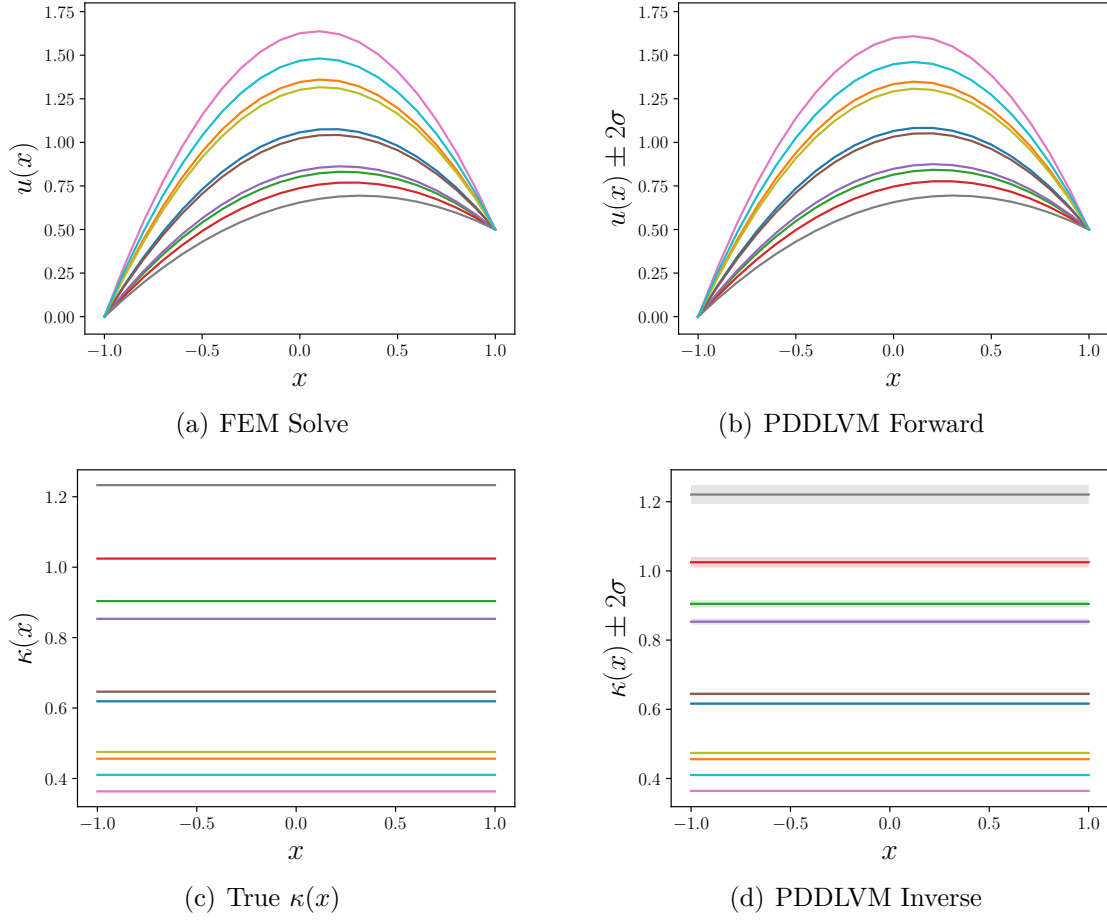


Fig. 4.4 Results for the 1D linear Poisson equation. (a) The true solution given by a linear solve of a FEM discretisation for 10 different constant functions of $\kappa(x)$. (b) The predicted solution with 2σ interval given by the PDDLVM for those same functions $\kappa(x)$. (c) The 10 true $\kappa(x)$ functions used to compute the solution fields in plots (a) and (b). (d) The PDDLVM reconstruction of $\kappa(x)$ from the solution field of plot (a) (identical functions means a perfect reconstruction of the $\kappa(x)$ field).

architecture of 50 nodes for 3 hidden layers with a “swish” activation function. We train each model for 2×10^5 iterations with Adam and a learning rate of 10^{-3} decayed by half 10 times over the training time. The joint training of the α and β networks takes 218 seconds. In Fig. 4.4b the shaded region for 2 standard deviation is very small as the forward problem is well determined and so the confidence region is highly concentrated around the mean function. In contrast to this, the inverse problem is more ill-posed and this is reflected by the larger uncertainty bands around the mean of the diffusion field in Fig. 4.4b.

Using this model we then study the effect of varying the ϵ_r value of the residual covariance, i.e. $\Sigma_r = \epsilon_r^2 \mathbf{I}$, controlling the solution accuracy. In Fig. 4.5 we plot the

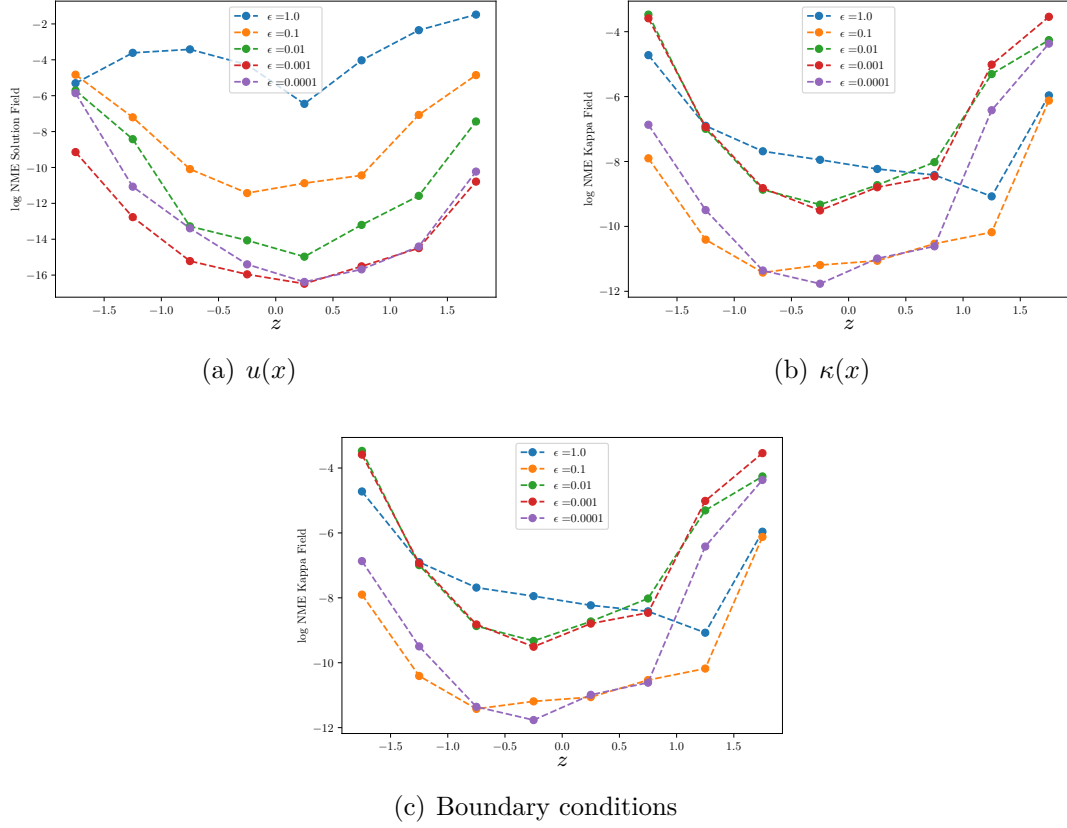


Fig. 4.5 Parametric Study Residual Variance PDDLVM. In (a) we show the error for the solution field, in (b) we show the error in the diffusion field, in (c) we show the error in the boundary condition estimates.

log Mean Normalized Squared Error (MNSE), Appendix (A.1), for 100 samples of solution fields, diffusion fields, and boundary conditions sets of κ drawn from the prior. This is the MNSE of PDDLVM predictions when compared to the ground truth. The algorithm was trained with a prior of $p(\kappa) = \mathcal{N}(0, 0.5)$ and was tested with 100 random draws of new κ from a narrow uniform testing distribution of width 0.5. We show the variation of error as we slide this testing distribution from values of -2 to 2. The points on the graph denote the centre of this narrow testing distribution and so the testing distribution extends in both direction by 0.25. In this manner we compare the model's predictions to the FEM solution as well as the true values of κ and the true values of the boundary conditions. We can see that as we go farther from regions of high prior density of κ , the error increases and vice versa. We plot the log MNSE for (a) the inferred solution field (forward problem), (b) the reconstructed diffusion field from the solution (inverse problem), and (c) the boundary conditions. This is computed for a variety of parameters ϵ_r . The boundary conditions are kept fixed throughout this experiment via a Dirac prior during training. However, we

treat the value of the boundary conditions as variable to be inferred. We can see that error in boundary conditions inferred from inverting the solution field still varies with respect to how close we sample κ to the main probability mass of $p(\kappa)$.

Similarly, in the Appendix Fig. A.1 we plot the mean normalized squared error for 100 independent draws of κ from various ranges of values. For both of these figures, $p(\kappa) = \mathcal{N}(0, 0.5)$. The error for the proposed solution field is computed with respect to a true FEM solve. For the reconstructed quantities κ and the boundary conditions, we compare the generated samples with the reconstructed one. As expected, the MNSE tends to increase as we move away from the regions of highest probability density of our prior distribution of $p(\kappa)$. We see that irrespective of the value of ϵ_r , the error increases for the solution field coefficients, the diffusivity field coefficients, and the boundary conditions as we move away from the main probability mass of the prior. As can also be seen in Fig. A.2, changes in the value for ϵ_r tempers the optimisation surface which will affect the training of the algorithm. We can conclude from this that there is no single correct setting of ϵ_r but in effect is a practitioners choice which will impact the learning process.

Observable Map Inversion

Based on Sec. 4.3.3 the framework learns to invert a noisy mapping from the measurement to the solution in the presence of observations using the γ -Net. Here, the governing equation is (4.31) with varying boundary conditions and the observable forward map is a truncation operation of the middle 20 values on the observable grid of 100 mesh points. The $\alpha_\star$ and $\beta_\star$ networks are pretrained on the linear Poisson equation with $\kappa(x)$ expressed as a 4th order Chebyshev expansion, variable boundary conditions, and variable constant function forcing. For testing, we generate a set of 100 observations with a noise covariance of $\Sigma_y = \sigma_y^2 \mathbf{I}$, where $\sigma_y = 0.01$. The model q_γ reconstructs $\mathbf{u}$ as Chebyshev coefficients from the noisy, partially observed $\mathbf{y}$. We show the results of inference with 5 new data observations not included in the dataset in Fig. 4.6.

To quantify the accuracy of this model we draw 100 independent samples from $p(\mathbf{z})$ and $p(\mathbf{f})$. We then compute the MSNE for those 100 independent draws. The computed MNSE for $\mathbf{u}$ projected on the FEM grid is of 5.64×10^{-4} and for $\mathbf{z}$ (the $\kappa(x)$ field) projected on the FEM mesh the MNSE is of 8.35×10^{-3} .

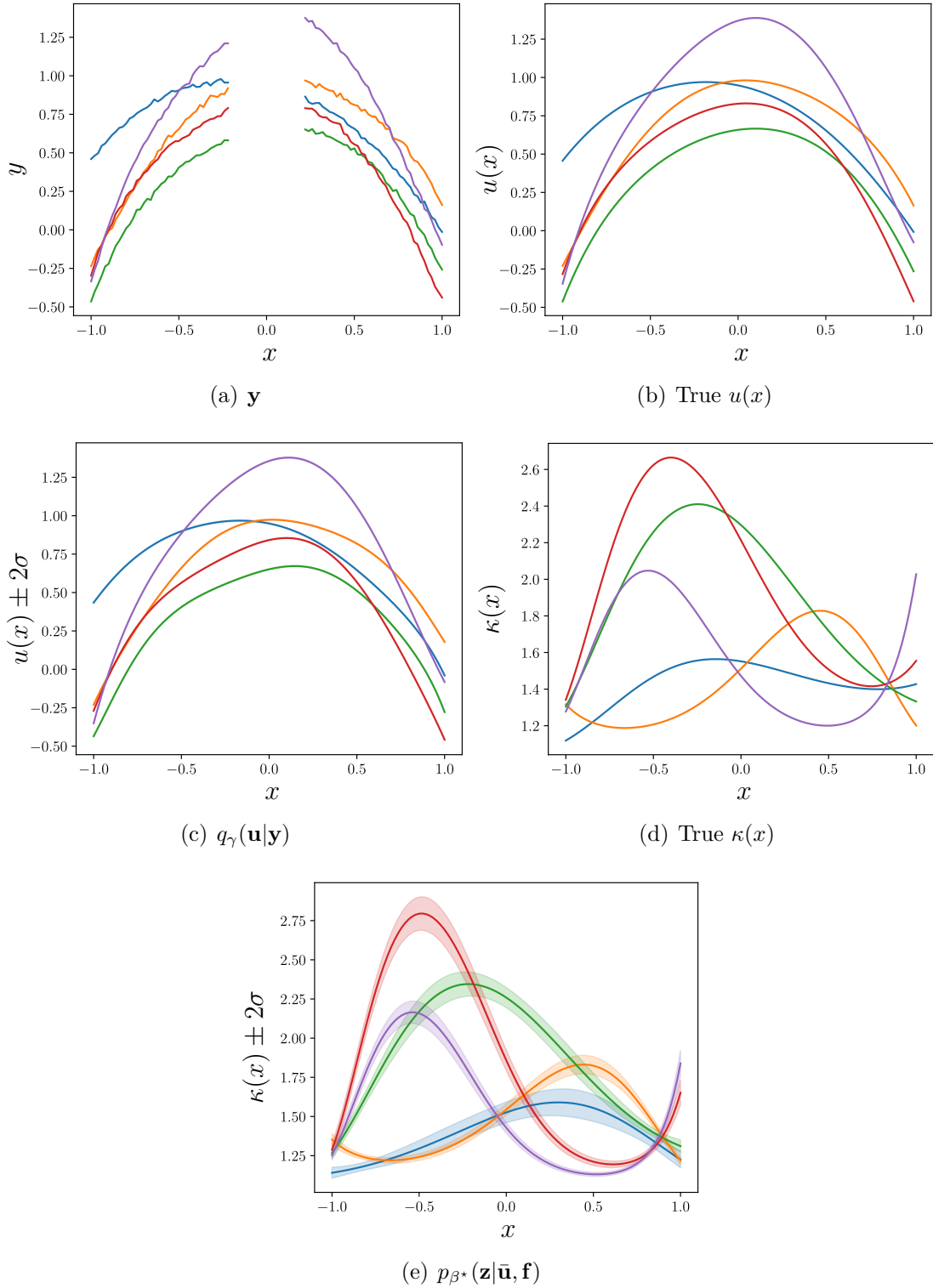


Fig. 4.6 Example of missing data. (a) Five data samples not observed during training. (b) The true $u(x)$ used to generate the synthetic data in (a). (c) The true parameter coefficient used to generate (b). (d) The predictive distribution of $\mathbf{u}$ given an observation $\mathbf{y}$. (e) The inferred distribution over $\kappa(x)$ given the inferred mean of $\mathbf{u}$ in (d) and the forcing. The variable $\bar{\mathbf{u}}$ denotes the mean of the predicted distribution given by $q_\gamma(\mathbf{u}|\mathbf{y})$.

Table 4.1 MNSE for 1D nonlinear Poisson problem and percentage coverage of 2 standard deviations of the output on the ground truth.

	α -Net	β -Net
MNSE	6.82×10^{-4}	2.36×10^{-3}
% truth in 2σ	29.6%	87.0%

4.4.2 1D Nonlinear Poisson Problem

In this example, we apply a PDDLVM for a 1D nonlinear Poisson problem of the form,

$$\begin{aligned}
 -\nabla \cdot (\eta(u, x) \nabla u(x)) &= f(x), \\
 \eta(u, x) &= \left(S \left| \frac{du}{dx} \right| + 5\kappa(x) \right) / 10,
 \end{aligned} \tag{4.32}$$

for $x \in (-1, 1)$. We choose this example to show how PDDLVM seamlessly handles nonlinear PDEs. Here $\mathbf{z} = \{\boldsymbol{\kappa}, a, b\}$ denotes the coefficients of the $\kappa(x)$ expansion and the boundary condition values respectively and the boundary conditions are $u(-1) = a$, $u(1) = b$. The function $S(x) = 1/(1 + e^{-x})$ is the sigmoid function. For the current example we parametrise $\kappa(x)$ as a 4th order Chebyshev expansion. The solution field is expressed as a 9th order Chebyshev expansion through the $\boldsymbol{\pi}_{\text{CH}}$ mapping. The *approximant* $\boldsymbol{\pi}_z(\mathbf{z}, x)$ is then only applied on $\kappa(x)$. The priors are: $p(\boldsymbol{\kappa}_i) = \mathcal{N}(0, 1)$, $p(a) = \delta(a)$, $p(b) = \mathcal{U}(0.5, 1)$, and $f(x)$ has a single expansion term drawn from $p(f) = \mathcal{U}(1, 2)$ as is associated to the 0th Chebyshev polynomial. The NN architecture for this problem is a fully connected 4 layers deep network with 100 neurons per hidden layer. The FE model is discretised with 60 elements. The residual covariance is chosen to be $\boldsymbol{\Sigma}_u = \epsilon_{\mathbf{r}}^2 \mathbf{I}$ with $\epsilon_{\mathbf{r}} = 10^{-2}$. The model is trained for 10^6 iterations with an initial learning rate of 10^{-3} that is halved every 2×10^5 iterations with Adam. In Fig. 4.7, the predictive distributions as well as the solutions are plotted.

4.4.3 Thin-Walled Flexible Shell

The PDDLVM framework can handle complicated geometries. In this example, we consider the deformation of the Stanford bunny with an inhomogeneous Young's modulus subjected to self-weight, see Figure 4.2. We analyse this model with an

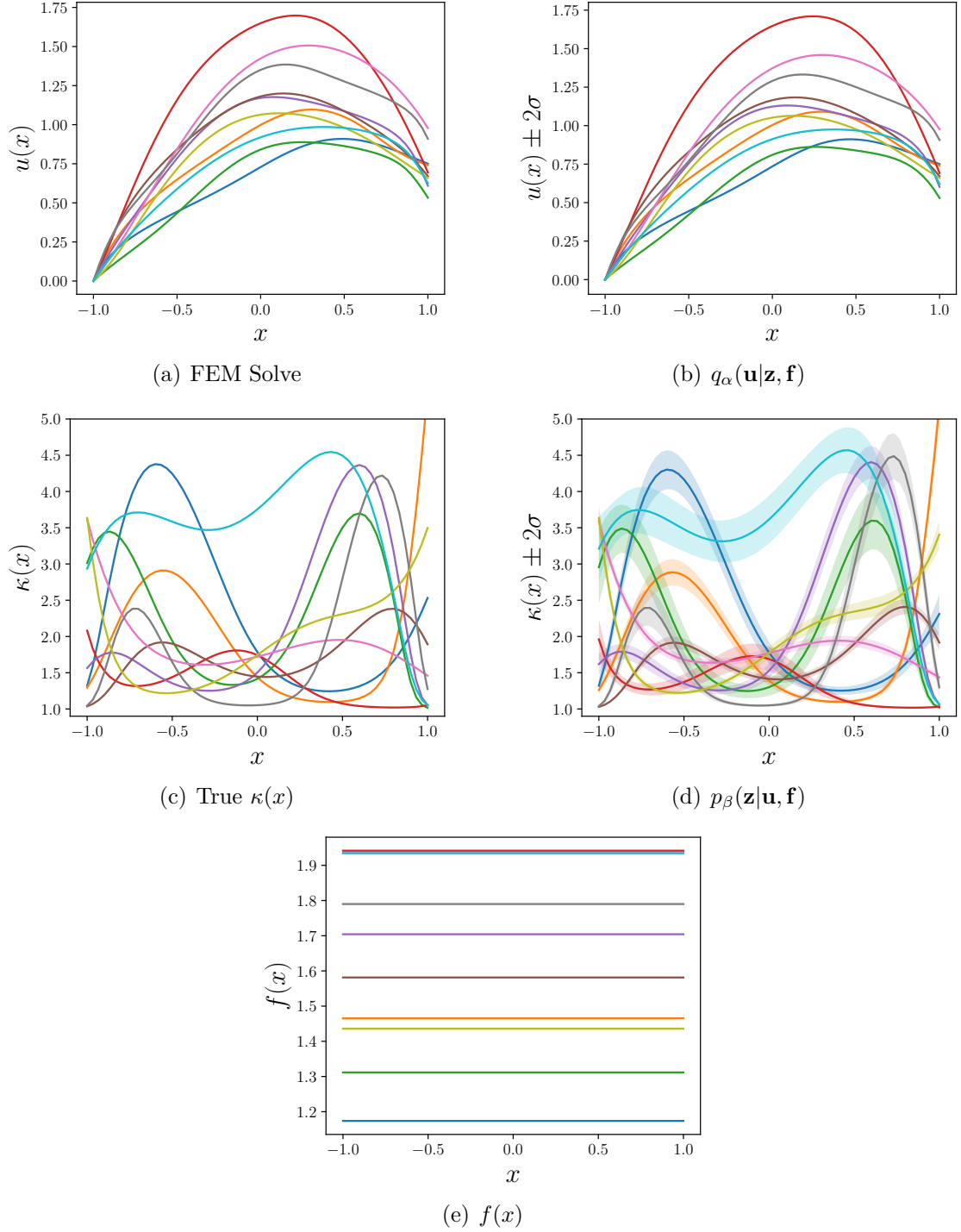


Fig. 4.7 Results for the 1D nonlinear Poisson example. (a) The true solution for 10 different instances of $\{\mathbf{z}, f(x)\}$. (b) The predicted solution with 2σ interval given by the PDDLVM for those same functions. (c) The 10 true $\kappa(x)$ functions used to compute the solution fields of the first 2 subplots. (d) The PDDLVM reconstruction of $\kappa(x)$. (e) The forcing functions $f(x)$.

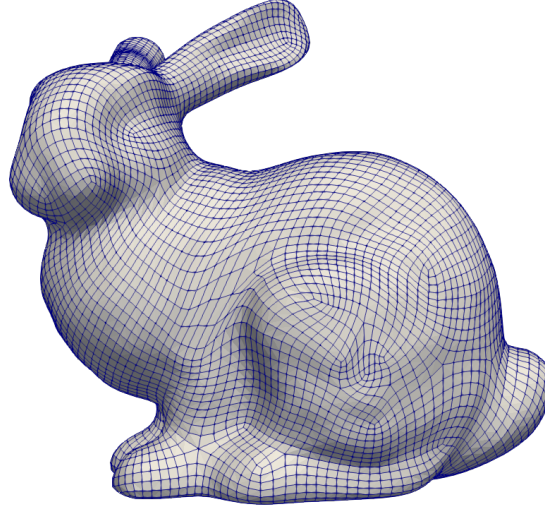


Fig. 4.8 The mesh of the bunny geometry used for the residual term as well as the FEM model, it has 24858 degrees of freedom.

isogeometrically discretised Kirchhoff-Love shell finite element scheme (Cirak et al., 2000, Cirak and Long, 2011, Febrianto et al., 2022). The model is based on the potential energy formulation of a smooth hyper-elastic shell. The weak form of the shell equilibrium equation is obtained by determining the stationary point of the potential energy where $f(x), u(x) : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ and the shell is subject to Dirichlet boundary conditions only. The function $f(x)$ represents the external loading on the shell surface. We choose the forcing to be a uniform vertical pressure representing self-weight. For the Kirchhoff-Love model we require basis functions with square integrable second derivatives which are obtained from Catmull-Clark subdivision surfaces (Zhang et al., 2018). In this example, we choose a Dirichlet boundary condition constraining the displacements at the base of the bunny such that $u(x) = 0$, for $x_2 < b$, where x_2 denotes the vertical coordinate axis and b is the chosen height of the base.

To build solution fields with a neural network in a mesh invariant and low dimensional way, we reparametrise the output of the network with a vector valued 3D Chebyshev expansion

$$\hat{\mathbf{u}}(\mathbf{x}) = \boldsymbol{\pi}_{\text{CH}}(\mathbf{u}, \mathbf{x}) = \sum_{l,m,n} \mathbf{U}_{ilmn} T_l(x_{(1)}) T_m(x_{(2)}) T_n(x_{(3)}), \quad (4.33)$$

where $\hat{\mathbf{u}}(\mathbf{x}) = (\hat{u}_{(1)}(\mathbf{x}), \hat{u}_{(2)}(\mathbf{x}), \hat{u}_{(3)}(\mathbf{x}))^\top$ indexed by $i \in \{1, 2, 3\}$ and the random vector $\mathbf{u} = \text{vect}(\mathbf{U})$ where $\mathbf{U}$ is a four dimensional tensor where the components $\mathbf{U}_{ilmn}$ are Chebyshev coefficients. Furthermore, $\mathbf{x} = (x_{(1)}, x_{(2)}, x_{(3)})^\top \in \Omega \subset \mathbb{R}^3$. The

vector field $\hat{\mathbf{u}}(\mathbf{x})$ denotes the displacement of the bunny surface and is computed on the intersection of the volume and the surface. Describing a 3 component vector field in 3 dimensions this way requires $3N^3$ Chebyshev coefficients, where N denotes the number of basis function per dimension. We choose a sixth order Chebyshev expansion with 7 coefficients per dimension so that we have 1029 Chebyshev expansion coefficients. On the other hand, the quadrilateral Stanford bunny mesh has 8286 nodes and 24858 degrees of freedom. The bunny geometry width, height and length is 154.8, 151.9, and 118.6 respectively. All coordinates plus a 10 unit buffer on all sides are then mapped to the unit cube for the centred Chebyshev reparametrisation. The shell has a thickness of 1, a vertical volumetric pressure field of 10^{-5} , a Poisson ratio of 0.3, and the height of the fixed base is of 7. For this example, the physically interpretable latent variable $\mathbf{z}$ is the set of parameters characterizing $\kappa(x_{(2)})$ which varies across the height of the shell to mimic the control of the material distribution that one might have in 3D printing contexts. It represents the modulus of elasticity expressed as a second order Chebyshev expansion passed through a Softplus function and multiplied by 10^9 .

With the FEM formulation we can take advantage of well-known preconditioning techniques (Grote and Huckle, 1997) for iterative solvers. Employing a simple preconditioner can greatly accelerate and stabilize convergence. This preconditioning is formulated as a reparametrisation of the residual with the stiffness matrix of a representative sample $\mathbf{z}$ of $p(\mathbf{z})$. To do this, we pre-multiply the residual vector $\mathbf{r}$ by $\mathbf{A}_{\bar{\mathbf{z}}}$. The matrix $\mathbf{A}_{\bar{\mathbf{z}}}$ is the stiffness matrix assembled with $\bar{\mathbf{z}}$ which is the mean of $p(\mathbf{z})$. This yields the preconditioned residual formulation as $\mathbf{A}_{\bar{\mathbf{z}}}\mathbf{r}$ which was found to greatly accelerate convergence and improve stability of the optimisation routine. Both α and β networks are fully connected neural networks with 3 hidden layers of 2.5k neurons with “swish” activation functions (Ramachandran et al., 2017). The PDDLVM was trained for 50k iterations for a total of 16.6 hours. Once trained we draw new samples of $\mathbf{z}$ from its prior, we then use both the FEM model and the emulator to obtain the solution fields for comparison. Fig. 4.10 shows the predictions of the trained PDDLVM. The top row pertains to the forward operation, the bottom row focuses on the inverse mapping. We plot the FEM solutions, the predictive mean, the standard deviation, as well as the absolute error. Table 4.2(a) summarises the mean normalized squared errors of the means of both the forward and inverse maps as well as the average time taken per sample for 100 independent draws from the prior $p(\mathbf{z})$.

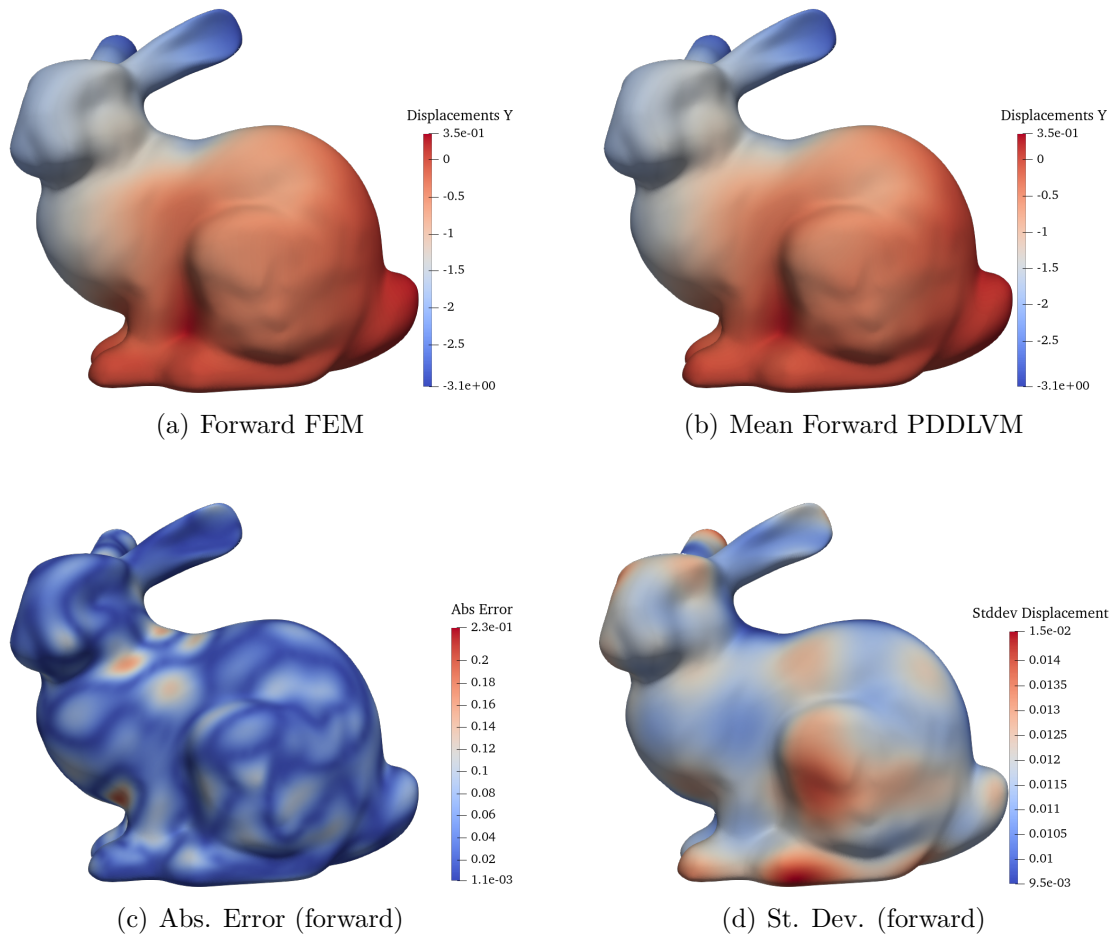


Fig. 4.9 Forward results from a trained FEM-PDDLVM of a thin-walled 3D shell bunny. The top row shows (a) the displacement field obtained with a forward FEM solve, (b) mean estimate provided by PDDLVM, (c) absolute error between (a) and (b), (d) standard deviation.

Table 4.2 Computational runtime of using the FEM-PDDLVM framework on the thin-walled flexible shell bunny as compared to traditional FEM.

	Forward	Inverse
MNSE NN	$2.77 \cdot 10^{-3}$	$7.45 \cdot 10^{-3}$
avg. time NN (s)	$3.46 \cdot 10^{-3}$	$2.52 \cdot 10^{-3}$
avg. time FEM (s)	10.4	—

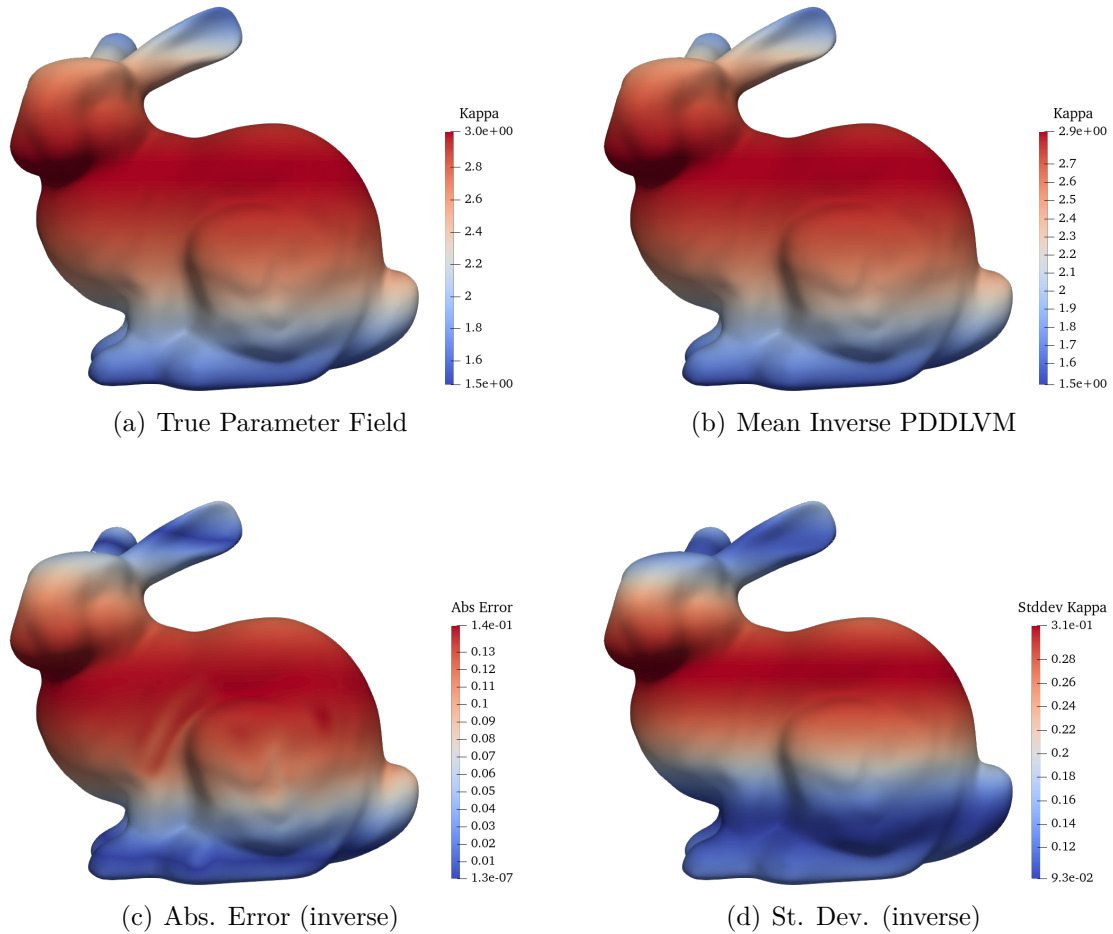


Fig. 4.10 Inverse Results from a trained FEM-PDDLVM of a thin-walled 3D shell bunny. We have (a) true parameter field, (b) mean estimate of the parameters provided by the inverse of PDDLVM, (c) absolute error, (d) standard deviation. The agreement between (c) and (d) demonstrates that when the difference between our method and the true parameter field is larger, our uncertainty estimates are coherent and demonstrates the regions with larger error accurately.

When computing the residual term necessary for FEM-PDDLVM, we need to update the stiffness matrix for new samples of the PDE parameters $\mathbf{z}$. In the governing equation, the domain integral depends linearly on Young’s modulus and we take its value to be constant across each element. Thus, when we compute the residual for a new sample of $\mathbf{z}$ we do not need to recompute the quadrature for the integration of the r.h.s. of the weak form. We can instead stash the element stiffness matrices for a Young’s modulus of 1 and post-multiply the already integrated element matrices with the new values for the Young’s modulus at the centroid of the element. From this we assemble the new matrix, bypassing the expensive quadrature routine. This approach is important when computing the residual tens of thousands of times for large problems. If however, we are interested in marginalising over a quantity that cannot be factorised out of the element stiffness integrals then the quadrature must be computed every iteration.

4.4.4 Nonlinear Heat Equation

To showcase the modularity of the PDDLVM framework, we consider next a time dependent nonlinear heat equation discretised with a PINN (Raissi et al., 2019). We can treat this as a spatio-temporal problem where we work with spatial and temporal dimensions the same way (Raissi et al., 2019). When combined with the PINN framework, our solution field predictive distribution and the inverse distribution are formulated in a general way as

$$\begin{aligned} q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{f}) &= \mathcal{N}\left(\boldsymbol{\mu}_\alpha(x_{1:n}, t_{1:n}, \mathbf{z}, \mathbf{f}), \text{diag}(\sigma_\alpha^2(x_{1:n}, t_{1:n}, \mathbf{z}, \mathbf{f}))\right), \\ p_\beta(\mathbf{z}|\mathbf{u}, \mathbf{f}) &= \mathcal{N}(\boldsymbol{\mu}_\beta(\mathbf{u}), \boldsymbol{\Sigma}_\beta(\mathbf{u})), \end{aligned} \quad (4.34)$$

where $\mathbf{u} \in \mathbb{R}^{nn}$ where $\mathbf{u} \in \mathbb{R}^n$ and are jointly trained as per the framework

Table 4.3 Comparison of PINN-PDDLVM model and a coupled PINN and convolutional inversion network¹.

	PINN & PINN & CONV-Net	PINN-PDDLVM
Mean Residual Forward	2.83×10^{-4}	1.23×10^{-4}
MNSE Inverse	6.35×10^{-3}	2.83×10^{-3}
% truth in $2\sigma_\beta$	n/a	98.06%
total training time	6440s	6808s

in Sec. 4.3.2. The samples $\mathbf{u}$ are given to the various other probability density functions as well as to the residual evaluator in (2.43). Here, the collocation method with $v_i(x) = \delta(x - x_i)\delta(t - t_i)$ reduces the integration to a point evaluation of the PDE on the grid. Automatic differentiation through $\mathbf{u}$ and the neural network, for $\partial_x \hat{u}(x)$, is obtained with the reparametrisation trick. The forward emulator takes in a pair of x, t coordinates along with the PDE parameters. To obtain the distribution across the domain we pass in a grid of x, t coordinates. For the inverse map, we pass a collection of evaluated grid points over the domain as an *image*. A convolutional structure is well adapted for dealing with this kind of spatial data. We construct $\Sigma_{\mathbf{r}} = \text{diag}([\epsilon_{D_0}^2, \dots, \epsilon_{D_d}^2, \epsilon_{B_0}^2, \dots, \epsilon_{B_b}^2, \epsilon_{I_0}^2, \dots, \epsilon_{I_i}^2])$ where each ϵ denotes the confidence for the d domain collocation points, the b boundary points, and the i initial condition points respectively. We choose a residual standard deviation $\epsilon_D = 0.01$ to emphasise the initial and boundary conditions; and we choose $\epsilon_B = 0.001$ and $\epsilon_I = 0.001$. The PDE of interest describes heat dissipation with a temperature dependent conductivity function. The second order nonlinear parametric time dependent PDE is

$$\frac{\partial u(x, t)}{\partial t} = \frac{1}{z_2} \nabla \cdot (\eta(u, z_1) \nabla u(x, t)), \quad \eta(u, z_1) = \left| \frac{u(x, t) z_1^2 + 1}{z_1} \right|, \quad (4.35)$$

for $\Omega = [0, 2\pi]$, with initial conditions and boundary conditions $u(0, t) = 1$, $u(2\pi, t) = 1$, and $u(x, 0) = \sin(x) + 1$. The $\eta(u, z_1)$ function represents a material which conduction increases linearly when the temperature increases, with a slope and intercept governed by the scalar z_1 . The PDE is also parametrised with z_2 which denotes the heat capacity of the material. We train the PDDLVM with marginalization over the prior $p(\mathbf{z}) = \mathcal{U}([1, 5] \times [1, 5])$ where $\mathbf{z} = \{z_1, z_2\}$. We show a sample solution field with standard deviation computed through PDDLVM in Fig. 4.11

In Fig. 4.12 we plot the log PINN residual for a range of PDE parameters z_1 and z_2 and we plot the log mean standard deviation of the α network. The quantities are computed over a 2-dimensional range of values of parameters z_1 and z_2 . The white square denotes the bounds for the observed z_1, z_2 during training as well as the bounds over which the mean residual is computed and is defined by the prior $p(\mathbf{z}) = p(z_1)p(z_2) = \mathcal{U}([1, 5]) \times \mathcal{U}([1, 5])$. In Fig. 4.11 we show the distribution $q_{\alpha}(\mathbf{u}|\mathbf{z})$ for a parameter sample z_1, z_2 for the time dependent nonlinear heat equation as defined in (4.35) and discretised with PINN-PDDLVM. In (a) we show the mean of the predictive field and in (b) we show the 2 standard deviation of the predictive solution

¹For details on the PINN residual formulations, see (4.36)

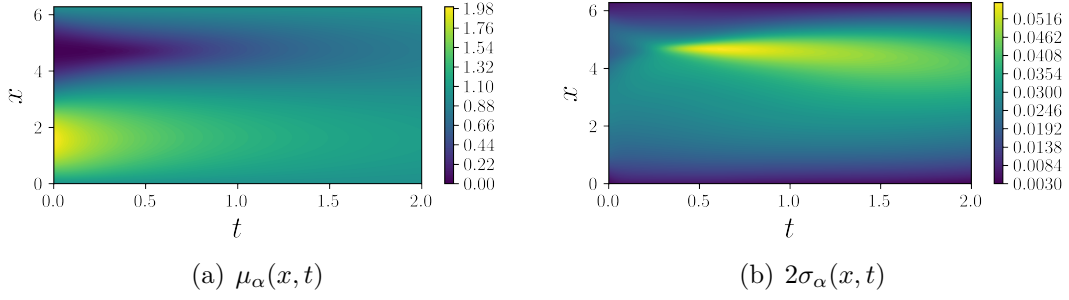


Fig. 4.11 Nonlinear heat results FEM-PDLVM. In (a) we show the mean of the predictive field and in (b) we show the 2 standard deviation of the predictive solution field

field. Notice the uncertainty is lowest at the fixed boundaries at $x = 0$, $x = 2\pi$ and highest in the region of higher non-linearity. We compare results from PINN and PINN-PDDLVM in Table 4.3.

The forward emulator network for both the standard PINN and PINN-PDDLVM models have 4 fully connected hidden layers of 100 neurons with “swish” activation functions. For the PINNs results we sampled 2.5k combinations of z_2 and z_1 between $[1., 5.] \times [1., 5.]$ and we trained a PINN-network for 5×10^5 iterations. We then used this trained PINN to output solutions fields and create a supervised training dataset for the convolutional inverse network. Both the deterministic inversion network (Conv-Net) and PINN-PDDLVM- β networks are 2D convolutional neural networks with 3 layers of 2D convolutions with swish activation, kernel size of 4, strid (2,2) and 8-16-32 channel filters followed by a final dense layer of 500 neurons ending with either 2 or 4 output neurons for either the PINN & CONV-Net network or the PINN-PDDLVM- β networks respectively.

We train the Conv-Net network with a squared error loss, minibatched one example at a time with Adam and a learning rate of 10^{-3} . When computing the residual for comparison we employ the standard PINN-style (Raissi et al., 2019) residual as

$$\begin{aligned}
 R_h(z_2, z_1) = & \frac{1}{NM} \sum_i^N \sum_j^M \left(-\frac{\partial h(x_i, t_j)}{\partial t} + \frac{1}{z_2} \nabla \cdot (\eta(h(x_i, t_j), z_1) \nabla h(x_i, t_j)) \right)^2 \quad (4.36) \\
 & + \frac{1}{n} \sum_k^n (h(x_k, 0) - u(x_k, 0))^2 \\
 & + \frac{1}{m} \sum_l^m \left[(h(0, t_0) - u(0, t_0))^2 + (h(2\pi, t_0) - u(2\pi, t_0))^2 \right],
 \end{aligned}$$

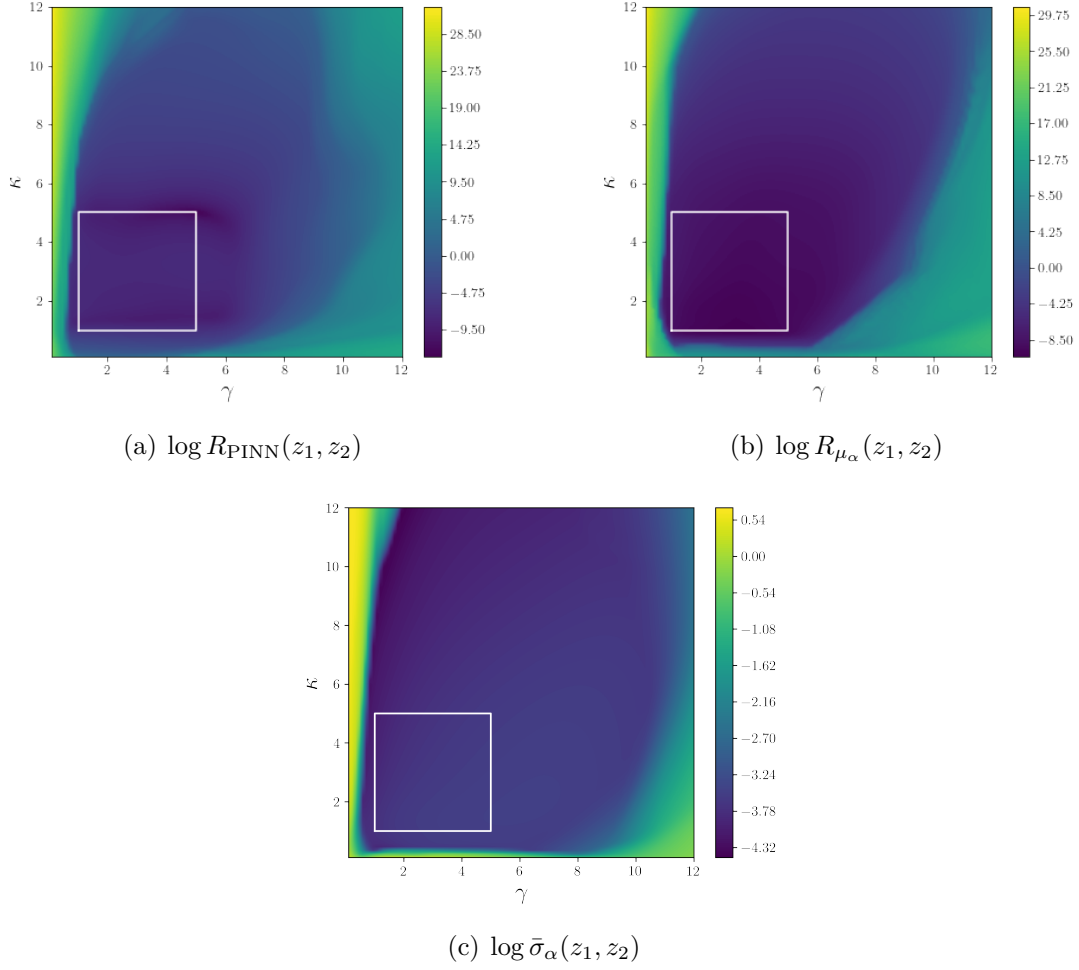


Fig. 4.12 A comparison of (a) PINN and (b) PINN-PDDLVM residuals and in (c) the average PINN-PDDLVM standard deviation.

where the second and the third terms are the residuals for the initial and the boundary conditions. It is the log of (4.36) that is plotted in Fig. 4.12 but it is not the residual used for *training* the PINN-PDDLVM, for that we use (2.43).

4.4.5 Inhomogeneous Wave Equation

Now, we compare the use of a 2D Fourier Neural Operator (FNO) trained on a supervised dataset of PDE solutions with an unsupervised PINN-PDDLVM model. The PDE in question is the inhomogeneous wave equation with parametrised forcing

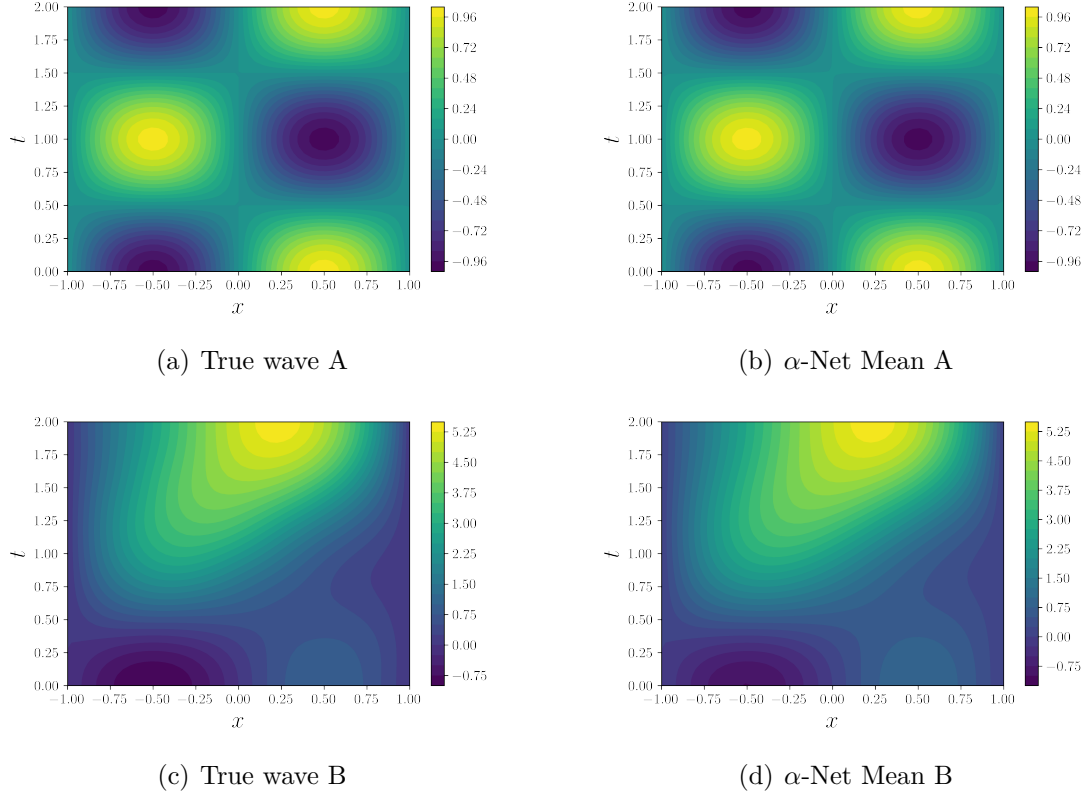


Fig. 4.13 Examples of ground truth and mean predictions from the α -Net of a PINN-PDDLVM for 2 sets of waves with different forcing. Wave A: $f = 0$, Wave B: $f = -5x^3$.

given as

$$\frac{\partial^2 u(x, t)}{\partial t^2} - \frac{\partial^2 u(x, t)}{\partial x^2} = f(x), \quad (4.37)$$

$$f(x) = \sum_i f_i x^i, \quad (4.38)$$

with boundary conditions

$$u(-1, 0) = u(1, 0) = 0, \quad (4.39)$$

$$u(x, 0) = \sin(\pi x). \quad (4.40)$$

Here, f_i are the elements of the vector $\mathbf{f}$ and are the parameters of the polynomial expansion representation of the forcing. In the current example we use 4 terms in the forcing expansion where each component is drawn from $\mathcal{U}(-5, 5)$. The solutions for FNO datasets and the validation set are assembled and computed from samples of $\mathbf{f}$ drawn from this distribution. We treat this $\mathbf{f}$ as the variable $\mathbf{z}$ to be inverted.

Table 4.4 Comparison of MNSE for supervised FNOs with differing number of data points (number following FNO) against an unsupervised PINN-PDDLVM.

MNSE	FNO-50	FNO-100	FNO-500	FNO-1000	PDDLVM
Forward	1.78×10^{-2}	6.20×10^{-4}	1.18×10^{-4}	9.57×10^{-5}	1.29×10^{-4}
Inverse	9.40×10^{-3}	8.67×10^{-4}	1.69×10^{-4}	1.31×10^{-4}	6.51×10^{-4}

In Fig. 4.13, we show 2 example ground truth solutions from the dataset as well as the mean predictions of those two example given by the PINN-PDDLVM. The FNOs for the forward and inverse directions have 4 hidden layers with 32 channels and 12 Fourier modes. We train the FNOs for 5×10^4 iterations for different number of training data points. The training data is obtained from a Chebyshev based solver of 21×21 basis functions. In Table 4.4, we compare this with an unsupervised PINN-PDDLVM trained for 2.5×10^5 iterations with a similar architecture to that of Sec. 4.4.4.

4.5 Discussion

In this chapter we proposed a physics-driven deep latent variable model to learn simultaneously forward and inverse maps of parametric PDEs. Our formulation leveraged conventional PDE discretisation techniques, deep neural networks, probabilistic modelling, and VI to assemble a fully probabilistic coherent framework.

In the posited probabilistic model, both the forward and inverse maps are approximated as Gaussian distributions with a mean and covariance parametrised by deep neural networks. The PDE residual is assumed to be an observed random vector of value zero, hence we modelled it as a random vector with a zero mean and a user-prescribed covariance. We constructed the model in a way that maintained the physical interpretability of the latent space, as this directly characterises physical parameters of our PDE model. From there, the model was trained by maximising the probability, that is the evidence/marginal likelihood, of observing a residual of zero by maximising the ELBO.

The modular nature of the developed framework allows for an easy transition between meshless (*e.g.* PINNs) and mesh-based (*e.g.* FEM) discretisation, making it agnostic to the specific method used to discretise the PDE. For example, we leveraged the FEM formulation (that includes a customized mesh and preconditioners) to

obtain the deformations in thin shells subject to gravity, providing estimates of the material properties and a framework for novel content generation.

As we have shown, the use of our method with PINNs produced a very streamlined framework for learning the behaviour of parametric PDEs. We demonstrated the efficiency and robustness of our method on finite element discretised parametric PDE problems such as linear and nonlinear Poisson problems, elastic shells with complex 3D geometries, and time-dependent nonlinear and inhomogeneous PDEs using a physics-informed neural networks discretisation. We achieved up to three orders of magnitude speed-up after training compared to traditional FEM solvers once the model is trained, while outputting coherent uncertainty estimates.

The computational advantage of making use of the proposed method over a more conventional FEM based solver will largely depend on the problem setting and the downstream uses of the surrogates. A trained PDDLVM can approximately solve a PDE forward or inverse problem in a single neural network pass. That operation is typically 3 orders of magnitude faster than a classical numerical scheme. This is a big advantage when dealing with applications requiring real-time predictions of the behaviour of physical systems, such as in control applications. In offline settings, where we do not need to make real-time predictions, the computational cost of training a PDDLVM may rival the cost of solving a few hundred or thousand of PDE systems with classical numerical methods. For such settings it is only sensible to train a PDDLVM if we require very large numbers of solves of our numerical method. These high evaluation settings come up in optimisation and sampling tasks for inverse problems or forward UQ. Furthermore, the number of parameters and training iterations may grow to be very large for problems with high dimensional parameter spaces. This is due to the growth in complexity of the mapping between the parameter space and the solution field. This is especially true when the parameters that affect the solution of the PDE do not lie on a low-dimensional manifold.

While using conventional FEM offers many advantages for the construction of residuals in the field of solid mechanics and many other applications (such as chemically reactive fluid flows), they do have some shortcomings. One of the biggest drawbacks of using FEM for this purpose is the large difference in computational paradigms associated with the manipulation of FEM meshes, which is typically done on the CPU and easiest coded in serial operations, and the paradigm associated with large dense matrix multiplication associated with deep learning. These operations are then usually carried out on the GPU rather than the CPU. The clash in approaches hinders the training of FEM residual informed deep learning models.

Future work should also look at implementing a spectral Galerkin model, which have dense matrices, and use a modern auto-differentiable library such as JAX for well integrated residual computation for deep learning. In certain physical scenarios other discretisation techniques of FEM could be preferable. Some of these include immersed, particle-based methods (Rüberg and Cirak, 2012, Lind et al., 2020) or spectral methods. Adapting our scheme to use such discretisation would be very compatible with the weighted residual paradigm on which we based our framework.

Another interesting line of future work could be applying PDDLVM models tailored to applications requiring specific discretisation schemes. Given that in the current work, VI uses the mean field assumption, which is known to be overconfident in estimating intractable densities. Our work could be easily extended to utilize the connectivity structure in the FEM formulation to improve the calibration of the uncertainty estimates while leveraging the sparse structure imposed by the FEM (Povala et al., 2022). One could also explore more elaborate covariance approximations such a low-rank decomposition. Other future directions are in the use of different architectures for the forward and inverse probabilistic maps, such as Fourier Neural Operators (Li et al., 2020b), Wavelet Neural Operators (Tripura and Chakraborty, 2022) or Physics-Informed Graph Neural Galerkin Networks (Gao et al., 2022).

Overall, our contribution has significant impact in parametric PDE applications such as design and optimisation scenarios and the study of complicated physical systems. Finally, since the most obvious difficulty of applying the developed method of PDDLVM in practice is its reliance on a unique fixed grid, it makes it difficult to apply to problems on different grids. Chapter 5 will endeavour to address this challenge with an adaptation of our method into a function space view.

Chapter 5

Improved Learning of Parametric PDEs: Random Grids & Gaussian Processes

5.1 Introduction

As discussed in the previous chapters, the main computational problems pertinent to PDE modelling can be categorised into two main classes: The task of obtaining a solution to a PDE for a given set of parameters (*forward problem*) and the task of recovering model parameters from solution fields or from observations (*inverse problem*) (Belov, 2012, Stuart, 2010b). Solving each of these tasks efficiently and accurately remains an open problem in the field of scientific computing. Work has been done in chapter 4 to try and address some of these challenges through the development of PDDLVMs. Now, the central objective is to develop an accurate and uncertainty aware physics informed probabilistic deep learning methods that go beyond *fixed grid* approaches to parametric PDEs. This allows for a synthesis with *noisy* data measured on arbitrary grids for forward and inverse problems, as well as the exploration of the use of more elaborate covariance structures through kernel learning.

5.2 Contribution

Here, we propose a new framework for jointly learning probabilistic mappings of forward and inverse problems in parametric PDEs using ideas from spatial statistics (Ripley, 2005, Cressie and Moores, 2021). This new approach connects neural Gaussian process models of PDE solution fields with physical parameters through

conditioning on random grids. More precisely, we propose (1) a physics driven VI framework based on random grids; (2) kernels for the learning of Gaussian random fields; (3) a new grid invariant architecture to enable learning through random collocation.

Previous work (Chiu et al., 2022) has explored the use of sets of random collocation points drawn for every residual evaluation, but this was studied for single solution field instances, not parametric PDEs, and only for the forward problem. We will demonstrate our methods on three PDEs of increasing complexity. The first is a 1D nonlinear Poisson PDE where we learn the mapping of a spectral representation of a diffusion field to the solution field for a range of forcing conditions. The second PDE is the spatio-temporal Burgers equation learned for a range of diffusion and non-linearity coefficients and of parametrised initial conditions. The third PDE is the incompressible Navier-Stokes lid-driven cavity flow problem where we map density and viscosity coefficients to spatio-temporal 3D solution fields. Furthermore, we demonstrate how to correctly incorporate sparse, *noisy* observations of sample solution fields to improve the predictive capabilities of the model. We also extend our proposed framework to jointly inform the model with data collected from nonlinear transformations of solution fields and PDE expressions.

Finally, we will compare our proposed method to existing methods such as Physics Driven Deep Latent Variable Models (PDDLVM) (Vadeboncoeur et al., 2022), modified DeepONets (Lu et al., 2021), and Physics-Informed Parametric Fourier Feature Networks (Wang et al., 2021b) adapted to solve both forward and inverse problems as described in (Zhao et al., 2022).

5.3 Reframing Φ -ML: Grid Invariance

In this section, we revisit concepts discussed in chapter 2 and chapter 4 for forward and inverse emulators, this time directly considering the grid of collocation points $\mathbf{X} = \{x_i\}_{i=1}^N$ on which the solution fields are observed and computed. This is fundamental to the derivation of the new method.

5.3.1 Physics Informed ML for Forward Problems

We formulate the nonlinear parametric PDE of interest as

$$\mathcal{G}_{\mathbf{z}}^{\mathbf{w}}(u)(x) = 0, \quad x \in \Omega, \quad (5.1)$$

$$\mathcal{B}_{\mathbf{w}}(u)(x) = 0, \quad x \in \partial\Omega, \quad (5.2)$$

where $\Omega \subset \mathbb{R}^d$, $\mathcal{G}_{\mathbf{z}}^{\mathbf{w}}$ is a nonlinear differential operator, $\mathbf{z}$ a set of parameters for which we solve the inverse problem, $\mathbf{w}$ an extra set of model parameters for which we would like to learn the forward and inverse maps, which could parametrise boundary conditions, initial conditions, forcing or PDE dynamics etc.

The inverse problem is defined with respect to $\mathbf{z}$, not $\mathbf{w}$. Similarly, $\mathcal{B}_{\mathbf{w}}$ is the boundary operator. This way, we change the formulation from those in chapter 3 and chapter 4 and absorb all inhomogeneous forcing terms $f(x)$ into $\mathbf{w}$ and do not treat the case where $\mathbf{w}$ is a continuous function $w(x)$; but instead they are treated though coefficients to some spectral expansion. A similar approach is taken with $z(x)$ now only seen as a set of coefficients $\mathbf{z}$.

For the forward problem, we consider the problem of learning a forward parametric emulator $f_{\alpha} : \mathbf{z}, \mathbf{w}, x \mapsto u(x)$ where α denotes the parameters of the emulator. This can be informally written as the approximation problem of finding $\alpha_{\star}$ s.t.

$$\mathcal{G}_{\mathbf{z}}^{\mathbf{w}} \circ f_{\alpha_{\star}}(\mathbf{z}, \mathbf{w})(\mathbf{X}) \approx 0, \quad (5.3)$$

given the boundary conditions, following (5.1). This function operating on $\mathbf{X}$ denotes the batched operation of this function on x . The problem can be formulated as an optimisation statement or could be used to inform a probabilistic model (Kaltenbach et al., 2022, Gao et al., 2022, Zhao et al., 2022). In the context of PDEs, the residual is defined as $r = \mathcal{G}_{\mathbf{z}}^{\mathbf{w}}(u)(x)$. When $r(\mathbf{X}) = \mathbf{0}$ then $u(\mathbf{X})$ is the solution to the PDE (5.1) at locations $\mathbf{X}$. Writing our forward emulator in this fashion expresses the fact that we want to be able to question our emulator not only for new sets of parameters $\mathbf{z}$ and $\mathbf{w}$, but also for new points $\mathbf{X}$ in the domain.

For the forward emulator another important aspect to consider is the boundary conditions. In general, there are many ways to enforce boundary conditions (Raissi et al., 2019). In chapter 4 we enforced boundaries for the PINN-PDDLVM model by using extra boundary residuals added to the overall residual observation. Here, we use the hard enforcement method. We enforce hard boundary conditions for Dirichlet problems (Rao et al., 2021, Sukumar and Srivastava, 2022) with a linear

transformation of the solution field

$$u(x) = b(x) + d(x)f_\alpha(x), \quad (5.4)$$

where $b(x)$ is an arbitrary function that satisfies the boundary conditions and $d(x) = 0$ for $x \in \partial\Omega$, and $d(x) \neq 0$ for $x \in \Omega$. Other formulations exist for mixed enforcement of boundary conditions as well as for more complicated boundary conditions.

5.3.2 Physics Informed ML for Inverse Problems

We now consider the problem of fitting a deterministic parametric inverse emulator denoted as $h_\beta : u(x), \mathbf{w} \mapsto \mathbf{z}$ where we seek $\beta_\star$ s.t.

$$h_{\beta_\star}(f_{\alpha_\star}(\mathbf{z}, \mathbf{w}, \mathbf{X}), \mathbf{w}) \approx \mathbf{z}, \quad (5.5)$$

for any subset $\mathbf{X}$ of the domain Ω . Similarly to the forward case, this can be done using an optimisation or a probabilistic formulation (Vadeboncoeur et al., 2022). We note that the inverse emulator approximation problem proposed here relies on a trained forward emulator and is free from classical numerical solvers. The objective with this model is to obtain an emulator capable of inverting the mapping from solution field to parameters for any set of observation locations $\mathbf{X}$. We want to be mesh invariant in this case. If mesh invariance is achieved, then our method in practice is no longer bound by any specific mesh or input location used in training. In what follows, we therefore develop a comprehensive probabilistic framework to tackle such problems in a principled way.

5.4 Model Derivation

In this section, we derive the training objective for three cases of our model, first in a purely physics informed model (Sec. 5.4.1), second as a physics and data informed model (Sec. 5.4.2), thirdly as a model incorporating observations of nonlinear transformations of solution fields and noisy input parameters as for e.g. measurements of drag coefficients (Sec 5.4.3).

5.4.1 Physics Informed Probabilistic Framework

Function space model: We begin the derivation of the physics informed probabilistic model by introducing all distributions of interest with a random field over

our solution field. Our hierarchical probabilistic model is defined as

$$\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}_{\mathbf{w}}, \boldsymbol{\Sigma}_{\mathbf{w}}). \quad (5.6)$$

$$u \sim \mathcal{GP}(\mu_u(x), k_u(x, x')), \quad (5.7)$$

$$\mathbf{z}|u, \mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}_{\beta}(u), \boldsymbol{\Sigma}_{\beta}(u, \mathbf{w})), \quad (5.8)$$

$$r|u, \mathbf{z}, \mathbf{w} \sim \mathcal{GP}(\mathcal{G}_{\mathbf{z}}^{\mathbf{w}}(u)(x), k_r(x, x'; u, \mathbf{z}, \mathbf{w})), \quad (5.9)$$

In this model, (5.9) defines a probability distribution over the residual and informs the model with physics. Equation (5.8) is the *inverse emulator* to be learned during training for the inverse map. Finally, (5.7) and (5.6) define the priors over u and $\mathbf{w}$. Leveraging the GP view allows to choose from several possible choices of kernels for our different distributions. Some of these include

$$k_{\theta}(x, x') = \epsilon_{\theta} \delta_{x, x'}, \quad (5.10)$$

$$k_{\theta}(x, x') = \lambda_{\theta}(x) \delta_{x, x'}, \quad (5.11)$$

$$k_{\theta}(x, x') = \lambda_{\theta}(x) \delta_{x, x'} + \langle \boldsymbol{\vartheta}_{\theta}(x), \boldsymbol{\vartheta}_{\theta}(x') \rangle. \quad (5.12)$$

Here θ denotes learnable functions or parameters. The kernels in (5.10)–(5.12) correspond respectively to a fixed white noise process, a heteroscedastic white noise process (diagonal kernel), and a degenerate deep low-rank covariance matrix (also known as the left Gram matrix (Rahman et al., 2022, Williams and Rasmussen, 2006)). We elaborate on (5.12) in Sec. 5.5. These kernels are chosen for their favorable scalability properties.

Function space variational family. We next introduce the variational family in function space (to be discretised later). For this, we define

$$\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}_{\mathbf{z}}, \boldsymbol{\Sigma}_{\mathbf{z}}) \quad (5.13)$$

$$u|\mathbf{z}, \mathbf{w} \sim \mathcal{GP}(\mu_{\alpha}(x; \mathbf{z}, \mathbf{w}), k_{\alpha}(x, x'; \mathbf{z}, \mathbf{w})), \quad (5.14)$$

where (5.14) is a flexible variational family parametrised by a neural network to learn the *forward emulator*. Here, (5.13) defines the prior in the variational family while the variational distribution over $\mathbf{w}$ is the same as the prior as in (5.6).

Discretisation through conditioning. To obtain a tractable algorithm, we condition all distributions on a set of N (grid) points $x \in \Omega$ are collected in $\mathbf{X}$. We achieve this by assigning a joint probability measure on $\Omega^{\otimes N}$ we denoted $p(\mathbf{X})$. Sampling from this measure and conditioning on the sample effectively discretises

the GP to a MVN. Given the sample $\mathbf{X}$, we discretise our distributions through conditioning on $\mathbf{X}$ as in (Cressie and Moores, 2021) and convert distributions defined in function space into a multivariate normal distributions MVN (Rudner et al., 2021). More precisely, through conditioning on $\mathbf{X}$, we convert the infinite dimensional model given in (5.9)–(5.6) into a conditional finite-dimensional model where

$$p(\mathbf{r}|\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}) = \mathcal{N}(\mathcal{G}_{\mathbf{z}}^{\mathbf{w}}(\mathbf{u}), \mathbf{K}_r(\mathbf{X}, \mathbf{X}; \mathbf{u}, \mathbf{z}, \mathbf{w})), \quad (5.15)$$

$$p_{\beta}(\mathbf{z}|\mathbf{u}, \mathbf{w}, \mathbf{X}) = \mathcal{N}(\boldsymbol{\mu}_{\beta}(\mathbf{u}, \mathbf{X}, \mathbf{w}), \boldsymbol{\Sigma}_{\beta}(\mathbf{u}, \mathbf{X}, \mathbf{w})), \quad (5.16)$$

$$p(\mathbf{u}|\mathbf{X}) = \mathcal{N}(\boldsymbol{\mu}_u(\mathbf{X}), \mathbf{K}_u(\mathbf{X}, \mathbf{X})), \quad (5.17)$$

where $\mathbf{u} = u(\mathbf{X})$ and $\mathbf{r} \in \mathbb{R}^N$ is the discretised residual. For the conditional residual (5.15) we choose a white noise kernel (5.10) and the mean function is given by the PDE evaluated at locations $\mathbf{X}$. The conditional distribution for $\mathbf{z}$ (5.16) has mean and covariance given as the output of a neural network with learnable β parameters. We call this network the “ β -Net” and it links the probability density of the $\mathbf{z}$ parameter to the solution field $\mathbf{u}$ and the parameters $\mathbf{w}$ given a partition of the domain. The distribution (5.17) is chosen in practice to be uninformative so that the model predictions are purely influenced by the PDE residual. The prior on $p(\mathbf{w})$ is the same as (5.6) since it is independent of the grid. Finally, our joint model can be factorised as

$$p(\mathbf{r}, \mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}) = p(\mathbf{r}|\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X})p_{\beta}(\mathbf{z}|\mathbf{u}, \mathbf{w}, \mathbf{X})p(\mathbf{u}|\mathbf{X})p(\mathbf{w})p(\mathbf{X}). \quad (5.18)$$

From this joint model we are interested in the marginal residual obtained by integrating out all other variables (including the grid) as

$$p(\mathbf{r}) = \int p(\mathbf{r}, \mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}) d\mathbf{u} d\mathbf{z} d\mathbf{w} d\mathbf{X}. \quad (5.19)$$

We know that the PDE is solved if $\mathbf{r} = \mathbf{0}$, hence our aim will be to maximise the marginal probability $p(\mathbf{r} = \mathbf{0})$. For this, we discretise our variational approximation in (5.14)–(5.13) by conditioning on the grid $\mathbf{X}$. In addition, we convert the infinite dimensional variational approximation in (5.14) to a finite dimensional one by conditioning on $\mathbf{X}$ as

$$q_{\alpha}(\mathbf{u}|\mathbf{z}, \mathbf{w}, \mathbf{X}) = \mathcal{N}(\boldsymbol{\mu}_{\alpha}(\mathbf{X}; \mathbf{z}, \mathbf{w}), \mathbf{K}_{\alpha}(\mathbf{X}, \mathbf{X}; \mathbf{z}, \mathbf{w})), \quad (5.20)$$

where (5.20) is a neural network parametrised GP with α learnable weights that relates the probability density of the solution field $\mathbf{u}$ to the two sets of parameters $\mathbf{z}$ and $\mathbf{w}$. This GP is given by the α -Net which is the forward emulator. In our

experiments we alternate between a heteroscedastic white noise kernel (5.11) and a low-rank kernel (5.12). Finally, our joint variational approximation is written as

$$q(\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}) = q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{w}, \mathbf{X})q(\mathbf{z})p(\mathbf{w})p(\mathbf{X}). \quad (5.21)$$

We then set the virtual observable $\mathbf{r} = \mathbf{0}$ (Rixner and Koutsourelakis, 2021, Vadeboncoeur et al., 2022). Using Jensen’s inequality we write out the ELBO

$$\begin{aligned} \log p(\mathbf{r} = \mathbf{0}) \geq \mathcal{F}(\alpha, \beta) = & \int \log \frac{p(\mathbf{r} = \mathbf{0}|\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X})p_\beta(\mathbf{z}|\mathbf{u}, \mathbf{w}, \mathbf{X})p(\mathbf{u}|\mathbf{X})}{q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{w}, \mathbf{X})q(\mathbf{z})} \\ & \times q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{w}, \mathbf{X})q(\mathbf{z})p(\mathbf{w})p(\mathbf{X}) \, d\mathbf{u} \, d\mathbf{z} \, d\mathbf{w} \, d\mathbf{X}. \end{aligned} \quad (5.22)$$

The most important distinction between this objective and all other methods known to us is the marginalization of the spatio-temporal domain through conditioning on random partitions of the domain. This objective can be computed as an expectation of the form

$$\mathcal{F}(\alpha, \beta) = \mathbb{E}_{\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}} \left[\log \frac{p(\mathbf{r} = \mathbf{0}|\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X})p_\beta(\mathbf{z}|\mathbf{u}, \mathbf{w}, \mathbf{X})p(\mathbf{u}|\mathbf{X})}{q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{w}, \mathbf{X})q(\mathbf{z})} \right]. \quad (5.23)$$

The expectation can be approximated with Monte Carlo integration. Other methods treating spatial locations of GPs as random variables to be marginalised include Jadhav et al. (2012), Girard et al. (2002). A potentially more rigorous derivation of the proposed algorithm could possibly be obtained through functional variational inference, such as in Sun et al. (2019), Burt et al. (2020), Wild et al. (2022), Ma and Hernández-Lobato (2021).

5.4.2 Physics and Data Informed Model Derivation

When developing physics emulators to be used in practice there may be noisy observations of real world physics behaviour. Data of this kind can nevertheless be of great use when developing better calibrated and more accurate models (Zhong and Meidani, 2022, Takeishi and Kalousis, 2021). In the current state of the art parametric PDE inference algorithms, there is a lack of methods that map parameters to solution fields adjusted with data in a statistically principled manner. Many methods incorporate data in a Bayesian manner then map parameters to the observation space which may not be the desired output. In this section, we derive a model for direct observation of a noisy solution field with deterministic inputs. The Bayesian

approach for additive Gaussian noise is to pose a model of the form

$$\mathbf{y}_D = G(\mathbf{z}, \mathbf{w}, \mathbf{X}) + \sigma_n \mathbf{e}, \quad \mathbf{e} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \quad (5.24)$$

where $\mathbf{y}_D$ is the data and σ_n is the observational noise, and $G(\cdot)$ is the mapping from parameters to solution field $\mathbf{u}$. For our method we pose this mapping $G(\mathbf{z}, \mathbf{w}, \mathbf{X})$ as a stochastic forward model given the α -Net,

$$\mathbf{y}_D^i = \boldsymbol{\mu}_\alpha(\mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}_D^i) + \mathbf{K}_\alpha(\mathbf{X}_D^i, \mathbf{X}_D^i; \mathbf{z}_D^i, \mathbf{w}_D^i)^{\frac{1}{2}} \mathbf{e}_2 + \sigma_n \mathbf{e}_1, \quad (5.25)$$

where $\mathbf{e}_1, \mathbf{e}_2 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and i indexes the observation set. In our framework we then jointly learn the forward model through the α -Net while adjusting the predictions to match the observed noisy dataset taking into account the relevant uncertainties. Our full joint model can then be written as

$$\begin{aligned} \log p(\mathbf{r} = \mathbf{0}, \mathbf{y}_D | \mathbf{z}_D, \mathbf{w}_D, \mathbf{X}_D) &\geq \mathcal{F}(\alpha, \beta) = \\ &\sum_i^N \log p(\mathbf{y}_D^i | \mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}_D^i) + \mathbb{E}_{\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}} \left[\log \frac{p(\mathbf{r} | \mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}) p_\beta(\mathbf{z} | \mathbf{u}, \mathbf{w}, \mathbf{X}) p(\mathbf{u} | \mathbf{X})}{q_\alpha(\mathbf{u} | \mathbf{z}, \mathbf{w}, \mathbf{X}) q(\mathbf{z})} \right]. \end{aligned} \quad (5.26)$$

The lower bound on the marginal probability of the zero residual and the observed data requires the evaluation of the entire dataset at every iteration. For computational efficiency, we replace the summation over the entire dataset with a mini-batch approximation as

$$\begin{aligned} \mathcal{F}(\alpha, \beta) &\approx \frac{N}{|M|} \sum_{i \in M} \log p(\mathbf{y}_D^i | \mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}_D^i) \\ &+ \mathbb{E}_{\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}} \left[\log \frac{p(\mathbf{r} | \mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}) p_\beta(\mathbf{z} | \mathbf{u}, \mathbf{w}, \mathbf{X}) p(\mathbf{u} | \mathbf{X})}{q_\alpha(\mathbf{u} | \mathbf{z}, \mathbf{w}, \mathbf{X}) q(\mathbf{z})} \right], \end{aligned} \quad (5.27)$$

where M is the mini-batch set randomly partitioned from the dataset. The first term of the ELBO can be efficiently evaluated and the second term can be approximated using a finite sample size through a Monte Carlo approximation of the expectation.

5.4.3 Physics and Indirect Noisy Observations Informed Model

In this section, we now derive a model that deals with indirect observations of the solution field and noisy measurements of the source parameters of the PDE. In this case, we have a general setting where we indirectly observe the solution field through some other known nonlinear mapping. One of the more salient examples

of such a scenario is the noisy measurement of drag coefficients in a fluid flow. The PDE describes the fluid velocity, and from a velocity field, we can compute the drag coefficient. Drag coefficients can also be easier to measure than direct solution fields. For such a model, where the parameters are also noisily measured (or estimated probabilistically through a Bayesian inverse problem), we write out the full observation model as

$$\mathbf{y}_D^i = \mathbf{g}(\tilde{\mathbf{u}}) + \mathbf{e}_{\mathbf{y}_D^i}, \quad \mathbf{e}_{\mathbf{y}_D^i} \sim \mathcal{N}(\mathbf{0}, \epsilon_{\mathbf{y}_D^i}^2 \mathbf{I}), \quad (5.28)$$

$$\mathbf{z}_D^i = \tilde{\mathbf{z}} + \mathbf{e}_{\mathbf{z}^i}, \quad \mathbf{e}_{\mathbf{z}^i} \sim \mathcal{N}(\mathbf{0}, \epsilon_{\mathbf{z}^i}^2 \mathbf{I}), \quad (5.29)$$

$$\mathbf{w}_D^i = \tilde{\mathbf{w}} + \mathbf{e}_{\mathbf{w}^i}, \quad \mathbf{e}_{\mathbf{w}^i} \sim \mathcal{N}(\mathbf{0}, \epsilon_{\mathbf{w}^i}^2 \mathbf{I}), \quad (5.30)$$

$$\mathbf{X}_D^i = \tilde{\mathbf{X}} + \mathbf{e}_{\mathbf{X}^i}, \quad \mathbf{e}_{\mathbf{X}^i} \sim \mathcal{N}(\mathbf{0}, \epsilon_{\mathbf{X}^i}^2 \mathbf{I}). \quad (5.31)$$

where $\tilde{\mathbf{u}}$ is a sample of $q_\alpha(\tilde{\mathbf{u}}|\tilde{\mathbf{z}}, \tilde{\mathbf{w}}, \tilde{\mathbf{X}})$. We assume all noise terms are Gaussian but other models could potentially be used to describe various types of randomness. Our observation model yields a Gaussian likelihood:

$$p(\mathbf{y}_D^i|\tilde{\mathbf{u}}) = \mathcal{N}(\mathbf{y}_D^i|\mathbf{g}(\tilde{\mathbf{u}}), \epsilon_{\mathbf{y}_D^i}^2 \mathbf{I}). \quad (5.32)$$

The $\tilde{\mathbf{u}}, \tilde{\mathbf{z}}, \tilde{\mathbf{w}}, \tilde{\mathbf{X}}$ variables are not directly observed. We can write out the likelihood term of the observation data as

$$p(\mathbf{y}_D|\mathbf{z}_D, \mathbf{w}_D, \mathbf{X}_D) = \prod_i^N p(\mathbf{y}_D^i|\mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}_D^i), \quad (5.33)$$

$$\log p(\mathbf{y}_D|\mathbf{z}_D, \mathbf{w}_D, \mathbf{X}_D) = \sum_i^N \log p(\mathbf{y}_D^i|\mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}_D^i). \quad (5.34)$$

We then take the likelihood function which uses all data-points at once and creates a mini-batch approximation. This drastically increases computational efficiency of the likelihood computation. We write this as

$$\log p(\mathbf{y}_D|\mathbf{z}_D, \mathbf{w}_D, \mathbf{X}_D) \approx \frac{N}{|M|} \sum_{i \in M} \log p(\mathbf{y}_D^i|\mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}_D^i). \quad (5.35)$$

We marginalise out the data variables to obtain the marginal likelihood in terms of the available distributions,

$$\log p(\mathbf{y}_D^i | \mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}_D^i) = \log \int p(\mathbf{y}_D^i, \tilde{\mathbf{u}}, \tilde{\mathbf{z}}, \tilde{\mathbf{w}}, \tilde{\mathbf{X}} | \mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}_D^i) d\tilde{\mathbf{u}} d\tilde{\mathbf{z}} d\tilde{\mathbf{w}} d\tilde{\mathbf{X}}, \quad (5.36)$$

$$\log p(\mathbf{y}_D^i | \mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}_D^i) = \quad (5.37)$$

$$\log \int p(\mathbf{y}_D^i | \tilde{\mathbf{u}}) q_\alpha(\tilde{\mathbf{u}} | \tilde{\mathbf{z}}, \tilde{\mathbf{w}}, \tilde{\mathbf{X}}) p(\tilde{\mathbf{z}} | \mathbf{z}_D^i) p(\tilde{\mathbf{w}} | \mathbf{w}_D^i) p(\tilde{\mathbf{X}} | \mathbf{X}_D^i) d\tilde{\mathbf{u}} d\tilde{\mathbf{z}} d\tilde{\mathbf{w}} d\tilde{\mathbf{X}}. \quad (5.38)$$

We use Jensen's inequality to obtain a lower bound on this marginal likelihood in terms of log distributions for computational convenience and apply the same mini-batching as before,

$$\log p(\mathbf{y}_D^i | \mathbf{z}_D^i, \mathbf{w}_D^i, \mathbf{X}_D^i) \geq \mathbb{E}_{\tilde{\mathbf{u}}, \tilde{\mathbf{z}}, \tilde{\mathbf{w}}, \tilde{\mathbf{X}}} [\log p(\mathbf{y}_D^i | \tilde{\mathbf{u}})], \quad (5.39)$$

$$\log p(\mathbf{y}_D | \mathbf{z}_D, \mathbf{w}_D, \mathbf{X}_D) \geq \sum_i^N \mathbb{E}_{\tilde{\mathbf{u}}, \tilde{\mathbf{z}}, \tilde{\mathbf{w}}, \tilde{\mathbf{X}}} [\log p(\mathbf{y}_D^i | \tilde{\mathbf{u}})], \quad (5.40)$$

$$\approx \frac{N}{|M|} \sum_{i \in M} \mathbb{E}_{\tilde{\mathbf{u}}, \tilde{\mathbf{z}}, \tilde{\mathbf{w}}, \tilde{\mathbf{X}}} [\log p(\mathbf{y}_D^i | \tilde{\mathbf{u}})]. \quad (5.41)$$

From previously derived result in Sec. 5.4.2 we obtain the complete ELBO on the marginal probability of the data and the residual,

$$\log p(\mathbf{r}, \mathbf{y}_D | \mathbf{z}_D, \mathbf{w}_D, \mathbf{X}_D) \geq \mathcal{F}(\alpha, \beta) \quad (5.42)$$

$$\begin{aligned} &= \sum_i^N \mathbb{E}_{\tilde{\mathbf{u}}, \tilde{\mathbf{z}}, \tilde{\mathbf{w}}, \tilde{\mathbf{X}}} [\log p(\mathbf{y}_D^i | \tilde{\mathbf{u}})] \\ &\quad + \mathbb{E}_{\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}} \left[\log \frac{p(\mathbf{r} | \mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}) p_\beta(\mathbf{z} | \mathbf{u}, \mathbf{w}, \mathbf{X}) p(\mathbf{u} | \mathbf{X})}{q_\alpha(\mathbf{u} | \mathbf{z}, \mathbf{w}, \mathbf{X}) q(\mathbf{z})} \right], \end{aligned} \quad (5.43)$$

Using a mini-batch approximation we have

$$\begin{aligned} \mathcal{F}(\alpha, \beta) &\approx \frac{N}{|M|} \sum_{i \in M} \mathbb{E}_{\tilde{\mathbf{u}}, \tilde{\mathbf{z}}, \tilde{\mathbf{w}}, \tilde{\mathbf{X}}} [\log p(\mathbf{y}_D^i | \tilde{\mathbf{u}})] \\ &\quad + \mathbb{E}_{\mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}} \left[\log \frac{p(\mathbf{r} | \mathbf{u}, \mathbf{z}, \mathbf{w}, \mathbf{X}) p_\beta(\mathbf{z} | \mathbf{u}, \mathbf{w}, \mathbf{X}) p(\mathbf{u} | \mathbf{X})}{q_\alpha(\mathbf{u} | \mathbf{z}, \mathbf{w}, \mathbf{X}) q(\mathbf{z})} \right]. \end{aligned} \quad (5.44)$$

With this model, the method is generalized to incorporate data from multiple sources and not just direct observations of the solution field. This also opens the possibility of learning missing dynamics not described in the PDEs in an experimentally feasible way.

5.5 Low-Rank Covariance Matrix

In the previous sections, we have shown how different kernels can be used in inference for the variational approximation. In this section, we discuss in detail how to use a low-rank covariance kernel matrix. The benefit of using such a covariance kernel is that it might be able to express richer covariance structures than a simple diagonal model while still being computationally efficient. Of course a full covariance matrix would be more expressive but such full covariances are far too computationally expensive for most applications.

The low-rank kernel is expressed as

$$k(x, x') = \lambda(x)\delta_{x,x'} + \langle \boldsymbol{\vartheta}(x), \boldsymbol{\vartheta}(x') \rangle, \quad (5.45)$$

where $\boldsymbol{\vartheta}(x) \in \mathbb{R}^l$ and l denotes the column dimension of the block matrices resulting from this kernel. The left Gram matrix $\mathbf{V}\mathbf{V}^\top$ results from this kernel

$$(\mathbf{V}\mathbf{V}^\top)_{ij} = \sum_l \mathbf{V}_{il} \mathbf{V}_{jl}, \quad (5.46)$$

$$= \sum_l \boldsymbol{\vartheta}(x_i)_l \boldsymbol{\vartheta}(x_j)_l, \quad (5.47)$$

$$= \langle \boldsymbol{\vartheta}(x_i), \boldsymbol{\vartheta}(x_j) \rangle. \quad (5.48)$$

Expanding out the definition of the kernel for a collection $\mathbf{X}, \mathbf{X}'$ of points we obtain

$$\mathbf{K}(\mathbf{X}, \mathbf{X}) = \boldsymbol{\Lambda} + \mathbf{V}\mathbf{V}^\top, \quad (5.49)$$

$$\boldsymbol{\Lambda} = \text{diag}(\boldsymbol{\lambda}), \quad (5.50)$$

$$\mathbf{V} \in \mathbb{R}^{n \times l}, \quad (5.51)$$

where $\boldsymbol{\lambda} \in \mathbb{R}^n$ and $\mathbf{V} \in \mathbb{R}^{n \times l}$. This matrix kernel $\mathbf{K}(\mathbf{X}, \mathbf{X})$ forms a dense matrix, but in computing the log density of the Gaussian associated with this covariance, and in sampling, we never need to compute the explicit dense covariance matrix. We show below how we can sample and evaluate the log-density efficiently without ever constructing the dense $n \times n$ matrix.

Sampling the Low-Rank Covariance

Sampling from a typical dense covariance matrix requires a Cholesky decomposition of the Matrix, an $\mathcal{O}(n^3)$ operation. However, with the special structure of the

low-rank kernel in (5.49) we can sample efficiently with

$$\boldsymbol{\epsilon} = \boldsymbol{\lambda}^{1/2} \odot \boldsymbol{\epsilon}_\lambda + \mathbf{V}\boldsymbol{\epsilon}_\mathbf{V}, \quad (5.52)$$

$$\boldsymbol{\epsilon}_\lambda \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_n), \quad \boldsymbol{\epsilon}_\lambda \in \mathbb{R}^n, \quad (5.53)$$

$$\boldsymbol{\epsilon}_\mathbf{V} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_l), \quad \boldsymbol{\epsilon}_\mathbf{V} \in \mathbb{R}^l. \quad (5.54)$$

This allows to easily draw samples from a Gaussian with this low-rank covariance matrix for very large n ; something not possible for dense covariance matrices as storage and computational costs grow $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$. In this case the operation grows $\mathcal{O}(n)$.

Evaluating the Log-Density

Evaluating the log-density requires two main operations which are computationally expensive: solving a linear system of equations and evaluating a log determinant. Usually, both of these two operations are computed with a Cholesky decomposition which costs $\mathcal{O}(n^3)$. We show how we can evaluate the log density more efficiently by using the form of (5.49). Firstly we write the log density of a MVN,

$$\log \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\frac{n}{2}(2\pi) - \frac{1}{2} \log(|\mathbf{K}(\mathbf{X}, \mathbf{X})|) - \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \mathbf{K}(\mathbf{X}, \mathbf{X})^{-1}(\mathbf{x} - \boldsymbol{\mu}). \quad (5.55)$$

We look at how to efficiently solve the linear system by making use of the Woodbury (Petersen et al., 2008) identity,

$$\mathbf{K}(\mathbf{X}, \mathbf{X})^{-1} \mathbf{a} = \boldsymbol{\Lambda}^{-1} \mathbf{a} - \boldsymbol{\Lambda}^{-1} \mathbf{V}(\mathbf{I}_l + \mathbf{V}^\top \boldsymbol{\Lambda}^{-1} \mathbf{V})^{-1} \mathbf{V}^\top \boldsymbol{\Lambda}^{-1} \mathbf{a}, \quad (5.56)$$

where $\mathbf{a} = \mathbf{x} - \boldsymbol{\mu}$, and use

$$(\boldsymbol{\Lambda}^{-1} \cdot \mathbf{a})_i = \lambda_i^{-1} \mathbf{a}_i, \quad (5.57)$$

$$(\boldsymbol{\Lambda}^{-1} \cdot \mathbf{V})_{ij} = \lambda_i^{-1} \mathbf{V}_{ij}. \quad (5.58)$$

The largest matrix is $l \times n$ and requires inverting an $l \times l$ matrix where $l \ll n$. To efficiently compute the determinant of the low-rank covariance matrix, using the matrix determinant lemma (Ding and Zhou, 2007) we have

$$|\mathbf{K}(\mathbf{X}, \mathbf{X})| = \det(\boldsymbol{\Lambda} + \mathbf{V}\mathbf{V}^\top) = \det(\mathbf{I}_l + \mathbf{V}^\top \boldsymbol{\Lambda}^{-1} \mathbf{V}) \prod_i \lambda_i. \quad (5.59)$$

This only requires taking the determinant of a $l \times l$ matrix by computing the Cholesky of the inner matrix as,

$$\mathbf{L} = \text{chol}(\mathbf{I}_l + \mathbf{V}^\top \mathbf{\Lambda}^{-1} \mathbf{V}), \quad (5.60)$$

and computing the log determinant as,

$$\log |\mathbf{\Sigma}| = 2\text{Tr}(\log(\mathbf{L})). \quad (5.61)$$

To form the covariance matrix in (5.49) from the neural network output of the α -Net we have

$$N_\alpha(\mathbf{z}, \mathbf{w}, \mathbf{X}) = \mathbf{O}, \quad \mathbf{O} \in \mathbb{R}^{n \times 2+l}, \quad (5.62)$$

where $N_\alpha(\cdot)$ is the α -Net and

$$\mathbf{O}_{1:n,1} = \boldsymbol{\mu}_\alpha \in \mathbb{R}^n, \quad (5.63)$$

$$\boldsymbol{\sigma}_T(\mathbf{O}_{1:n,2}) = \boldsymbol{\lambda}_\alpha \in \mathbb{R}^n, \quad (5.64)$$

$$\mathbf{O}_{1:n,3:l} = \mathbf{V}_\alpha \in \mathbb{R}^{n \times l}. \quad (5.65)$$

with $\boldsymbol{\sigma}_T(\cdot)$ as an exp-linear function (Dehaene and Brossard, 2021) constraining the value in a set positive interval typically chosen to be $[10^{-5}, 1]$. Later (Sec. 5.7.4) we will construct distributions for a vector field with the distribution simply written as

$$q_\alpha(\mathbf{u}_{(1)}, \dots, \mathbf{u}_{(d)} | \mathbf{z}, \mathbf{w}, \boldsymbol{\omega}) = \mathcal{N} \left(\begin{bmatrix} \boldsymbol{\mu}_{\alpha,1}(\dots) \\ \vdots \\ \boldsymbol{\mu}_{\alpha,d}(\dots) \end{bmatrix}; \text{diag} \begin{bmatrix} \boldsymbol{\lambda}_{\alpha,1}(\dots) \\ \vdots \\ \boldsymbol{\lambda}_{\alpha,d}(\dots) \end{bmatrix} + \begin{bmatrix} \mathbf{V}_{\alpha,1}(\dots) \\ \vdots \\ \mathbf{V}_{\alpha,d}(\dots) \end{bmatrix} \begin{bmatrix} \mathbf{V}_{\alpha,1}(\dots) \\ \vdots \\ \mathbf{V}_{\alpha,d}(\dots) \end{bmatrix}^\top \right). \quad (5.66)$$

We note that the columns of $\mathbf{V}$ must be unique, else the matrix will be singular.

5.6 Grid Invariant Inversion Networks

Central to this new framework are random collocation grids sampled every iteration which poses unique challenges for the inversion network.

Convolutional neural networks (CNNs) require information to be given on a fixed uniform grid and are thus unsuitable for our task. A natural solution for passing information originating on random grids to a CNN would be to use a kernel

interpolation method such as the Nadaraya–Watson kernel estimator (Cai, 2001) to interpolate points at random locations to fixed input locations. However, to capture information from different length scales, a very fine projection grid is required. Such grids are computationally expensive and scale as $\mathcal{O}((nm)^d)$, where m is the number of points on the projection grid. Borrowing ideas from Li et al. (2020b) and kernel feature space methods (Scholkopf et al., 1999), we develop a scalable alternative to fine interpolation grids.

Indeed, we first project the spatially dependent inputs $\mathbf{u}(\mathbf{x}) \in \mathbb{R}^{d_u}$ (a vector field for generality, reduced to a scalar field for the nonlinear Poisson and Burgers equations) to a learned higher dimensional space through a small fully connected neural network $P(\mathbf{u}(\mathbf{x}), \mathbf{x}) = \mathbf{v}(\mathbf{x})$ where $\mathbf{v}(\mathbf{x}) \in \mathbb{R}^{d_v}$ and $\mathbf{x} \in \Omega \subset \mathbb{R}^d$. We then project each dimension of $\mathbf{v}(\mathbf{x})$ at random locations $\mathbf{x}$ onto its own fixed location coarse grid $\mathbf{x}^*$ through the Nadaraya–Watson estimator

$$I_k(\mathbf{v}(\mathbf{x}), \mathbf{x}^*) = \frac{\sum_i \phi_k(\mathbf{x}^*, \mathbf{x}_i) \mathbf{v}(\mathbf{x}_i)}{\sum_i \phi_k(\mathbf{x}^*, \mathbf{x}_i)}, \quad (5.67)$$

where each of the d_v kernels has its own learnable length scale initialized at several different orders of magnitude. A visual representation of this can be seen in Fig. 5.1. Each coarse grid can then represent information at different lengthscales and the complexity grows as $\mathcal{O}(d_v(nm')^d)$ where m' is the new grid density of the interpolation grid chosen to be arbitrarily coarse, and in practice chosen as coarse as a 10 points/dimension lattice. We choose the kernel $\phi_k(\cdot, \cdot)$ to be any number of distance measuring functions with learnable parameters such as RBF, Matérn, etc. We then define the grid invariant convolutional network (GICNet) as

$$G(\mathbf{u}, \mathbf{w}, \mathbf{x}) = \text{Conv}(\cdot, \mathbf{w}) \circ I_{0:k}(\cdot, \mathbf{x}^*) \circ P(\cdot, \mathbf{x}) \circ \mathbf{u}(\mathbf{x}) \quad (5.68)$$

where the projection layer $P(\mathbf{u}(\mathbf{x}), \mathbf{x})$, the interpolation layer $I(\mathbf{v}(\mathbf{x}), \mathbf{x})$ along with the kernel $\phi(\mathbf{x}^*, \mathbf{x})$ lengthscale and the convolutional and fully connected layers are jointly trained.

There are similarities between the proposed model and convolutional neural processes (Gordon et al., 2019) taking in points $\mathbf{X}$ coming in at random locations and amortizing learning across these function observation datasets.

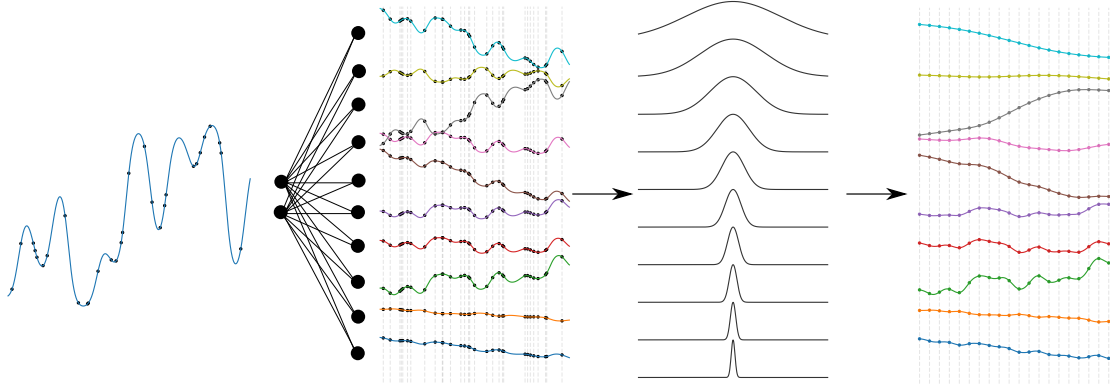


Fig. 5.1 A visual representation of the lifting of a signal sampled at random locations into a higher dimensional feature space where it is then interpolated with various kernels onto a fixed grid. This rich feature space evaluated at a fixed location is then passed to a convolutional neural network with a matching channel dimension.

5.7 Experiments

Our experiments were all run using TensorFlow, and all networks use “swish” activation functions (Ramachandran et al., 2017). The inversion networks were implemented with 1D, 2D, and alternated 1D & 2D convolutional networks for the nonlinear Poisson, Burgers, and Navier-Stokes problems, respectively. A visual representation of the GICNet interpolation layers is shown in Fig. 5.1. The code for these examples can be found on the github repo <https://github.com/ArnaudVadeboncoeur/phys-rgnp>.

5.7.1 The PDEs

The PDEs used in the testing of the method are described below, along with the relevant boundary conditions and their discretisation.

Nonlinear Poisson 1D

The first testing setup is a nonlinear 1D Poisson problem. The variable $\mathbf{z}$ which we want to learn in the inverse problem are coefficients to a Chebyshev expansion describing the diffusion field. The variable w over which we are marginalizing is a scalar representing a constant forcing over the domain. We write out the PDE

$\mathcal{G}_{\mathbf{z}}^w(u)(x)$ as

$$0 = \frac{\partial}{\partial x} \left(k(u, x) \frac{\partial u(x)}{\partial x} \right) - w, \quad (5.69)$$

$$k(u, x) = \log \left(1 + \exp \left(u(x) \sum_{i=0}^{n_z} z_i \phi_i(x) \right) \right) + 0.1. \quad (5.70)$$

for $\Omega = [-1, 1]$, where the Dirichlet boundary conditions are $x(-1) = 0$, $x(1) = 0$. Furthermore, we enforce boundary conditions in an exact manner with,

$$\bar{\mathbf{u}} = b(\mathbf{X}) + d(\mathbf{X})\mathbf{u}, \quad \mathbf{u} \sim q_{\alpha}(\mathbf{u}|\mathbf{z}, \mathbf{w}, \mathbf{X}) \quad (5.71)$$

where $b(x)$ captures the boundary conditions. The prior distributions are $p(z_i) = \mathcal{U}(-1, 1)$ and $p(w) = \mathcal{U}(1, 2)$.

Burgers Equation

The second PDE to test the method is the parametric Burgers equation with 2 spatio-temporal dimensions (one space, one time). The Burgers equation is a convection-diffusion equation (Bateman, 1915); it is an interesting case study looking at nonlinear convection and linear diffusion. It allows for the understanding of shockwave formation when fluid viscosity tends to zero. It also appears in gas dynamics and traffic flow (Musha and Higuchi, 1978). We include in this example the viscous Burger's equation with a parametrised diffusion and convection term. Here, the $\mathbf{z}$ parameters control the scaling of the nonlinear and the diffusive terms and the $\mathbf{w}$ scalar modifies the parametric initial conditions. This allows us to learn forward and inverse mappings for a continuum of initial conditions. The PDE $\mathcal{G}_{\mathbf{z}}^w(u)(x)$ is defined as

$$\frac{\partial u(x, t)}{\partial t} + z_1 u(x, t) \frac{\partial u(x, t)}{\partial x} - z_0 \frac{\partial^2 u(x, t)}{\partial x^2} = 0, \quad (5.72)$$

and the boundary and initial conditions are

$$u(-1, t) = u(1, t) = 0, \quad (5.73)$$

$$u(x, 0) = \sin(2\pi w x) \sin(\pi x), \quad (5.74)$$

for a set domain given as $\Omega[-1, 1]$, $0 \leq t \leq 1$ (Duffin, 2022). The priors are: $p(z_0) = \mathcal{U}(10^{-2}, 10^{-1})$, $p(z_1) = \mathcal{U}(0.5, 1)$, $p(w) = \mathcal{U}(0.5, 2)$. As in the nonlinear Poisson example we enforce boundary conditions in an exact manner with (5.71).

In this example, we make the standard deviation of the residual kernel value (ϵ_θ in (5.10)) a learnable parameter which drastically increases stability of the learning algorithm for this example, while maintaining accuracy. It converges to a value in the range of 10^{-1} .

Navier-Stokes Equations

The last PDE testing the proposed framework is the incompressible Navier-Stokes non-stationary lid-driven cavity flow. This is a setup for studying fundamental aspects of confined fluid-flows (Botella and Peyret, 1998). Learning mappings for forward and inverse problems of Navier-Stokes is of great interest to the engineering sciences. The solution field is a 3-dimensional vector field defined as the velocity and pressure $[\mathbf{u}, p]^\top$, where $\mathbf{u} = [u_1, u_2]^\top$ are the horizontal and vertical velocities.

We do not define a $\mathbf{w}$ variable for these experiments. The equations for $\mathcal{G}_z(\mathbf{u})(\mathbf{x})$ describing the dynamics are given by

$$z_1 \frac{\partial \mathbf{u}}{\partial t} + z_1 \mathbf{u} \cdot \nabla \mathbf{u} + \nabla p - z_2 \nabla^2 \mathbf{u} = 0, \quad (5.75)$$

$$\nabla \cdot \mathbf{u} = 0, \quad (5.76)$$

over a square domain, $\Omega(\mathbf{x}, t)$ between 0 and 1 where $\mathbf{x} = [x_{(1)}, x_{(2)}]^\top$. We define a non-stationary boundary condition on the top edge of the square domain as

$$u_1(x_{(1)}, 1, t) = (1 - (2x_{(1)} - 1)^6)t, \quad (5.77)$$

and all other boundary conditions of u_1, u_2 are zero. The α and β distributions are then defined as

$$u_1, u_2, p | \mathbf{z} \sim \mathcal{GP}(\mu_\alpha(\mathbf{x}; \mathbf{z}), k_\alpha(\mathbf{x}, \mathbf{x}'; \mathbf{z})), \quad (5.78)$$

$$\mathbf{z} | u_1, u_2, p \sim \mathcal{N}(\boldsymbol{\mu}_\beta(u_1, u_2, p), \boldsymbol{\Sigma}_\beta(u_1, u_2, p)). \quad (5.79)$$

The z_1 variable corresponds to the fluid density and the z_2 parameter corresponds to the dynamic viscosity. Here $p(z_1) = \mathcal{U}(0.8, 1)$ and $p(z_2) = \mathcal{U}(0.1, 1)$. For this example, we enforce exact boundary conditions with (5.71) but enforce a soft divergence as an extra residual appended in the residual vector (scaled $10\times$).

5.7.2 Comparisons of Physics Informed Model

To address the physics informed comparisons for the nonlinear Poisson and Burgers equation, we compare our diagonal and low-rank covariance methods (RGNP-D, RGNP-LR) to a fixed grid PDDLVM (Vadeboncoeur et al., 2022), a Fourier Feature Net (FFNet) forward emulator (Wang et al., 2021b) with a GICNet inversion network on a fixed grid, and a physics informed DeepONet (Wang et al., 2021a) with a test-time kernel interpolation (K-I) layer with a convolutional neural net on a fixed grid. Methods other than ours rely on creating a dataset of 1k input pairs of $\mathbf{z}, \mathbf{w}$ variables for the Poisson problem, and 10k samples for the Burgers example. We report the MNSE as in Appendix (A.1) and its standard deviation over the 1000 testing samples (see Table 5.1). We also report the percentage of predictions within 2σ of the ground truth solution and the total training times (see Table 5.2).

We use a batch size of 50 samples for the Poisson problem, and a batch size of 25 for the Burgers equation (this is reduced to a batch size of 5 for the 10k collocation DeepONet & kernel-interpolation case because of excessive memory consumption). We train the Poisson problem for 20k gradient updates and we train the Burgers setup for 80k gradient update steps. All learning is done using the Adam optimiser (Kingma and Ba, 2014) with a decaying learning rate. In Fig. 5.2 we show an example output from the α -Net for the solution field of the nonlinear Poisson 1D problem. In Fig. 5.3 we show example outputs for a solution field for the diagonal model from the α -Net for the Burgers equation.

5.7.3 Comparison of Physics & Data Informed Model

To compare the results from the physics and data informed model to the nonlinear Poisson 1D problem, the setups are similar as in the previous section but now the loss for the proposed model is given by (5.27) and we incorporate noisy observation of solution fields in the inference. We use 1k noisy sample solutions measured at 60 locations with a noise standard deviation $\sigma_n = 0.05$.

In Fig. 5.4 we have 5 samples of observed solution fields where the scatter points are the data passed to the algorithm. The algorithm only sees the scatter points, not the solid ground-truth line. The losses for the other methods against which we compare are modified to include a data-fit term as in (Wang et al., 2021a) to have a

²We used a reduced batched size of 5 collocation sets because of memory constraints. The same number of gradient updates were used hence the similar run-times for the 2.5k and 10k examples.

Table 5.1 Comparisons of physics informed models: MNSEs

METHOD	N	MNSE u	MNSE z
NL Poisson			
rgnp-d	30	$9.21 \cdot 10^{-5} \pm 8.66 \cdot 10^{-4}$	$1.48 \cdot 10^{-2} \pm 2.41 \cdot 10^{-2}$
rgnp-lr $\alpha 2, \beta 1$	30	$2.63 \cdot 10^{-4} \pm 1.88 \cdot 10^{-3}$	$6.72 \cdot 10^{-2} \pm 5.96 \cdot 10^{-2}$
PDDLVM	30	$1.69 \cdot 10^{-4} \pm 1.36 \cdot 10^{-3}$	$8.10 \cdot 10^{-3} \pm 1.36 \cdot 10^{-3}$
FFNET-GICNET	30	$1.90 \cdot 10^{-4} \pm 2.04 \cdot 10^{-3}$	$6.75 \cdot 10^{-1} \pm 1.57 \cdot 10^{-0}$
DONET & K-I.	30	$6.62 \cdot 10^{-4} \pm 3.41 \cdot 10^{-3}$	$5.58 \cdot 10^{-2} \pm 4.62 \cdot 10^{-2}$
DONET & K-I.	100	$6.62 \cdot 10^{-4} \pm 3.95 \cdot 10^{-3}$	$3.66 \cdot 10^{-2} \pm 2.48 \cdot 10^{-2}$
DONET & K-I.	300	$7.69 \cdot 10^{-4} \pm 3.90 \cdot 10^{-3}$	$2.58 \cdot 10^{-2} \pm 2.58 \cdot 10^{-2}$
Burgers			
rgnp-d	225	$1.63 \cdot 10^{-4} \pm 1.58 \cdot 10^{-4}$	$9.05 \cdot 10^{-3} \pm 1.73 \cdot 10^{-2}$
rgnp-lr $\alpha 2, \beta 0$	225	$9.51 \cdot 10^{-5} \pm 9.73 \cdot 10^{-5}$	$8.04 \cdot 10^{-3} \pm 1.66 \cdot 10^{-2}$
PDDLVM	225	$2.70 \cdot 10^{-1} \pm 2.12 \cdot 10^{-1}$	$5.32 \cdot 10^{-2} \pm 6.49 \cdot 10^{-2}$
FFNET-GICNET	225	$7.11 \cdot 10^{-1} \pm 1.70 \cdot 10^{-1}$	$1.19 \cdot 10^{-1} \pm 1.75 \cdot 10^{-1}$
DONETS & K-I.	225	$9.44 \cdot 10^{-1} \pm 4.68 \cdot 10^{-1}$	$8.66 \cdot 10^{-2} \pm 1.45 \cdot 10^{-1}$
DONET & K-I.	900	$4.16 \cdot 10^{-1} \pm 2.07 \cdot 10^{-1}$	$9.79 \cdot 10^{-1} \pm 7.15 \cdot 10^{-1}$
DONET & K-I.	2.5K	$9.65 \cdot 10^{-2} \pm 1.05 \cdot 10^{-1}$	$5.04 \cdot 10^{-1} \pm 2.32 \cdot 10^{-1}$
DONET & K-I. ²	10K	$3.71 \cdot 10^{-3} \pm 8.22 \cdot 10^{-3}$	$2.01 \cdot 10^{-2} \pm 5.35 \cdot 10^{-2}$

Table 5.2 Comparisons of physics informed models: UQ & Time

METHOD	N	$u \in 2\sigma$	$\mathbf{z} \in 2\sigma$	TIME
NL Poisson				
rgnp-d	30	95.6%	99.7%	6.31
rgnp-lr $\alpha 2, \beta 1$	30	94.7%	68.1%	6.19
PDDLVM	30	92.3%	95.2%	4.83
FFNET-GICNET	30	—	—	5.06
DONET & K-I.	30	—	—	3.31
DONET & K-I.	100	—	—	5.36
DONET & K-I.	300	—	—	11.34
Burgers				
rgnp-d	225	99.9%	100.0%	73.56
rgnp-lr $\alpha 2, \beta 0$	225	99.9%	100.0%	73.45
PDDLVM	225	97.7%	63.0%	59.02
FFNET-GICNET	225	—	—	66.45
DONETS & K-I.	225	—	—	36.00
DONET & K-I.	900	—	—	105.61
DONET & K-I.	2.5K	—	—	281.14
DONET & K-I.	10K	—	—	284.90

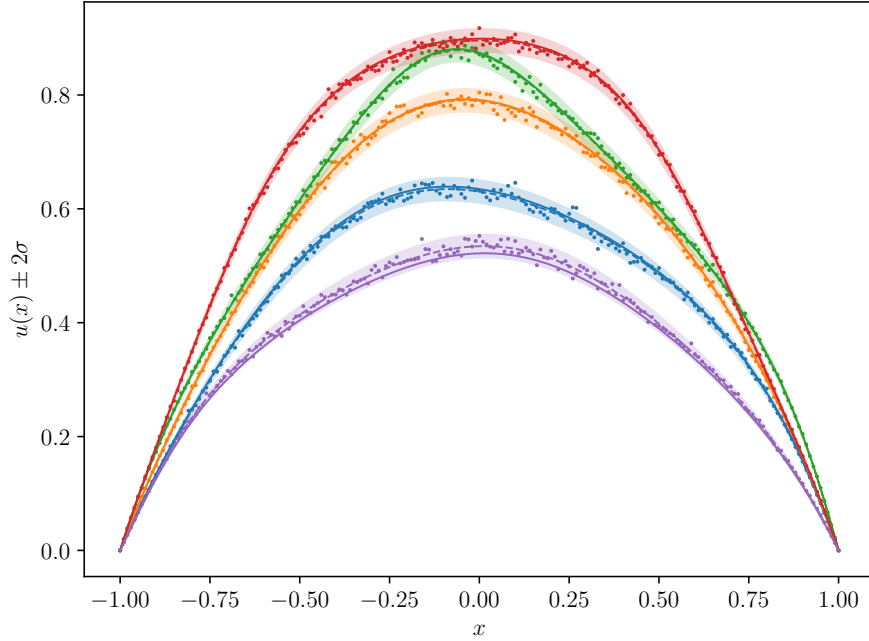


Fig. 5.2 Sample results from the Nonlinear Poisson 1D setup for the diagonal model given by the $q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{w}, \mathbf{X})$ distribution. The solid line is the FEM solution, the dashed line is the mean estimate, and the scatter points are the random samples; the shaded area is the 2σ uncertainty.

trade-off between fitting the differential operator and the observations (see Table 5.3 and Table 5.4).

5.7.4 Navier-Stokes Lid Driven Cavity Flow

We also applied our method to a challenging parametric Navier-Stokes problem. We defined the parametric differential operator and boundary conditions for a time-dependent lid-driven cavity flow example using the incompressible Navier-Stokes equations. Here the parameters corresponded to the dynamic viscosity and the fluid density. In Tables 5.5 and 5.6 we computed these quantities for 100 independent samples of $\mathbf{z}$ drawn from the prior. All models were run for 100k gradient update steps.

We note that obtaining reliable and accurate reference solutions to Navier-Stokes problems is very challenging. Some results from the diagonal model are shown in Fig. 5.5.

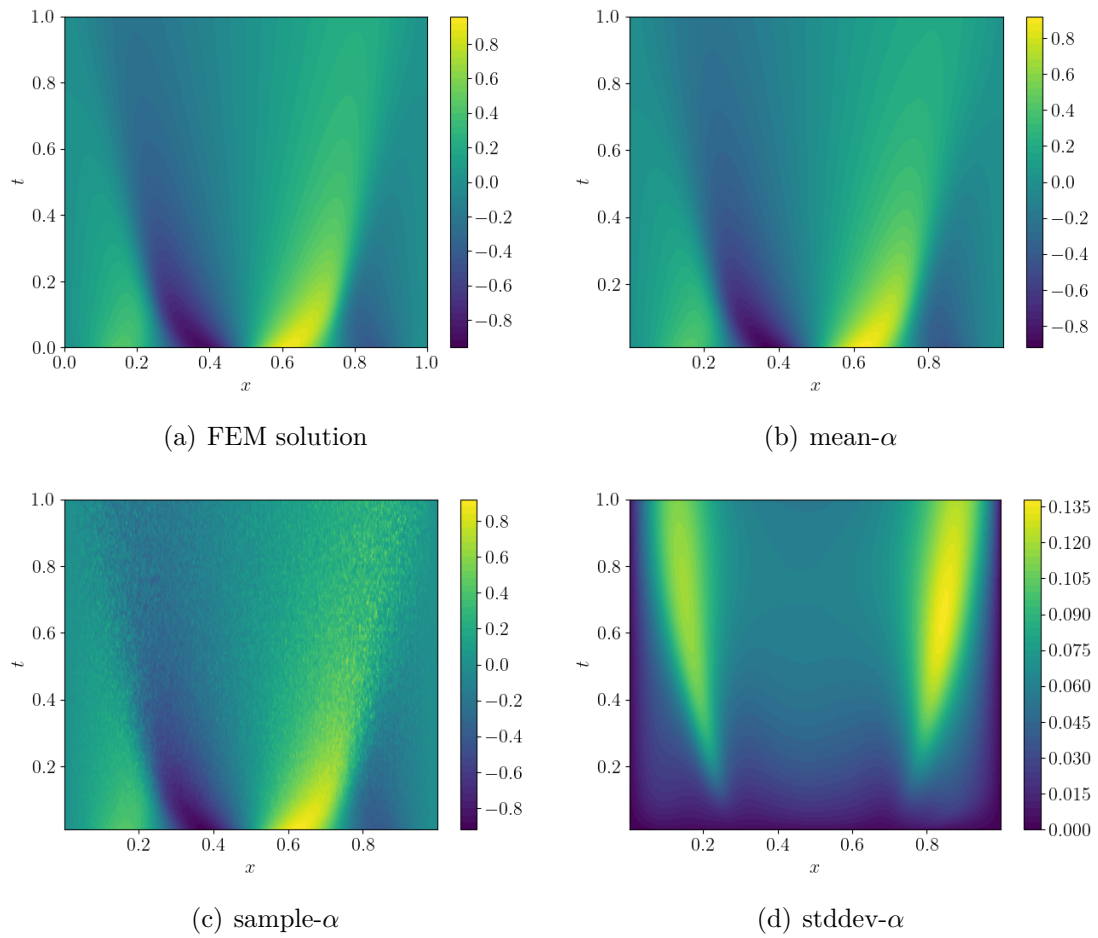


Fig. 5.3 Sample result from the α -Net with a diagonal kernel for the Burgers equation. We show the FEM solution, the mean estimate, the standard deviation field, and a sample from $q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{w}, \mathbf{X})$.

Table 5.3 Comparisons of physics and data informed models: MNSEs

METHOD	N	MNSE u	MNSE $\mathbf{z}$
NL Poisson			
rgnp-d	30	$1.59 \cdot 10^{-5}$ $\pm 5.26 \cdot 10^{-5}$	$5.45 \cdot 10^{-3}$ $\pm 6.25 \cdot 10^{-3}$
rgnp-lr $\alpha 2, \beta 1$	30	$3.56 \cdot 10^{-5}$ $\pm 2.48 \cdot 10^{-4}$	$3.43 \cdot 10^{-2}$ $\pm 7.40 \cdot 10^{-2}$
PDDLVM	30	$1.67 \cdot 10^{-5}$ $\pm \mathbf{3.60 \cdot 10^{-5}}$	$7.26 \cdot 10^{-3}$ $\pm 7.23 \cdot 10^{-3}$
FFNET-GICNET	30	$2.85 \cdot 10^{-4}$ $\pm 8.76 \cdot 10^{-4}$	$4.36 \cdot 10^{-1}$ $\pm 5.76 \cdot 10^{-1}$
DONETS & K-I.	30	$1.55 \cdot 10^{-4}$ $\pm 2.77 \cdot 10^{-4}$	$9.70 \cdot 10^{-2}$ $\pm 8.99 \cdot 10^{-2}$
DONETS & K-I.	100	$2.17 \cdot 10^{-4}$ $\pm 2.86 \cdot 10^{-4}$	$1.29 \cdot 10^{-1}$ $\pm 1.52 \cdot 10^{-1}$
DONETS & K-I.	300	$3.27 \cdot 10^{-4}$ $\pm 4.44 \cdot 10^{-4}$	$6.97 \cdot 10^{-2}$ $\pm 4.56 \cdot 10^{-1}$

Table 5.4 Comparisons of physics and data informed models: UQ & Time

METHOD	N	$u \in 2\sigma$	$\mathbf{z} \in 2\sigma$	TIME
NL Poisson				
rgnp-d	30	97.2%	99.7%	6.21
rgnp-lr $\alpha 2, \beta 1$	30	98.6%	62.9%	6.47
PDDLVM	30	95.3%	97.6%	4.23
FFNET-GICNET	30	—	—	5.59
DONETS & K-I.	30	—	—	3.94
DONETS & K-I.	100	—	—	5.67
DONETS & K-I.	300	—	—	11.70

Table 5.5 Results for physics informed models on Navier-Stokes: MNSEs

METHOD	N	MNSE u_1	MNSE u_2	MNSE $\mathbf{z}$
rgnp-d	512	$2.37 \cdot 10^{-2} \pm 1.16 \cdot 10^{-2}$	$3.59 \cdot 10^{-2} \pm 1.55 \cdot 10^{-2}$	$6.12 \cdot 10^{-3} \pm 6.00 \cdot 10^{-3}$
rgnp-lr $\alpha 2, \beta 0$	512	$2.39 \cdot 10^{-2} \pm 1.16 \cdot 10^{-2}$	$3.62 \cdot 10^{-2} \pm 1.54 \cdot 10^{-2}$	$4.07 \cdot 10^{-3} \pm 3.98 \cdot 10^{-3}$

Table 5.6 Results for physics informed models on Navier-Stokes: UQ & Times

METHOD	N	$u_1 \in 2\sigma$	$\mathbf{z} \in 2\sigma$	TIME
		MNSE u_2		$u_2 \in 2\sigma$
rgnp-d	512	38.3%	66.5%	297.6
			49.8%	
rgnp-lr $\alpha 2, \beta 0$	512	50.4%	66.5%	306.2
			58.0%	

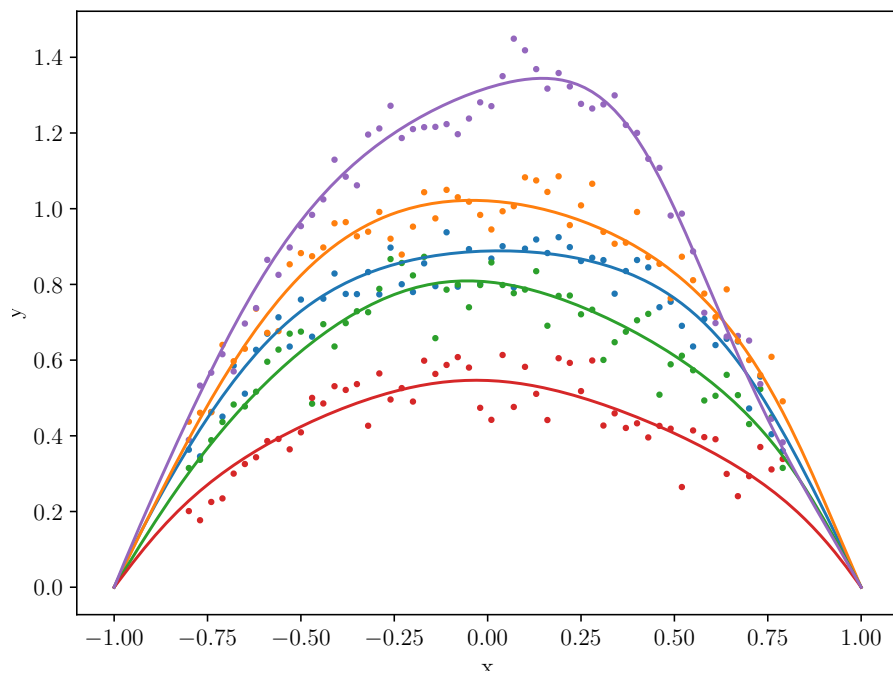


Fig. 5.4 Five example data samples. We show the true solution given by the solver in a solid line, and the noisy scatter data $\mathbf{y}_D$.

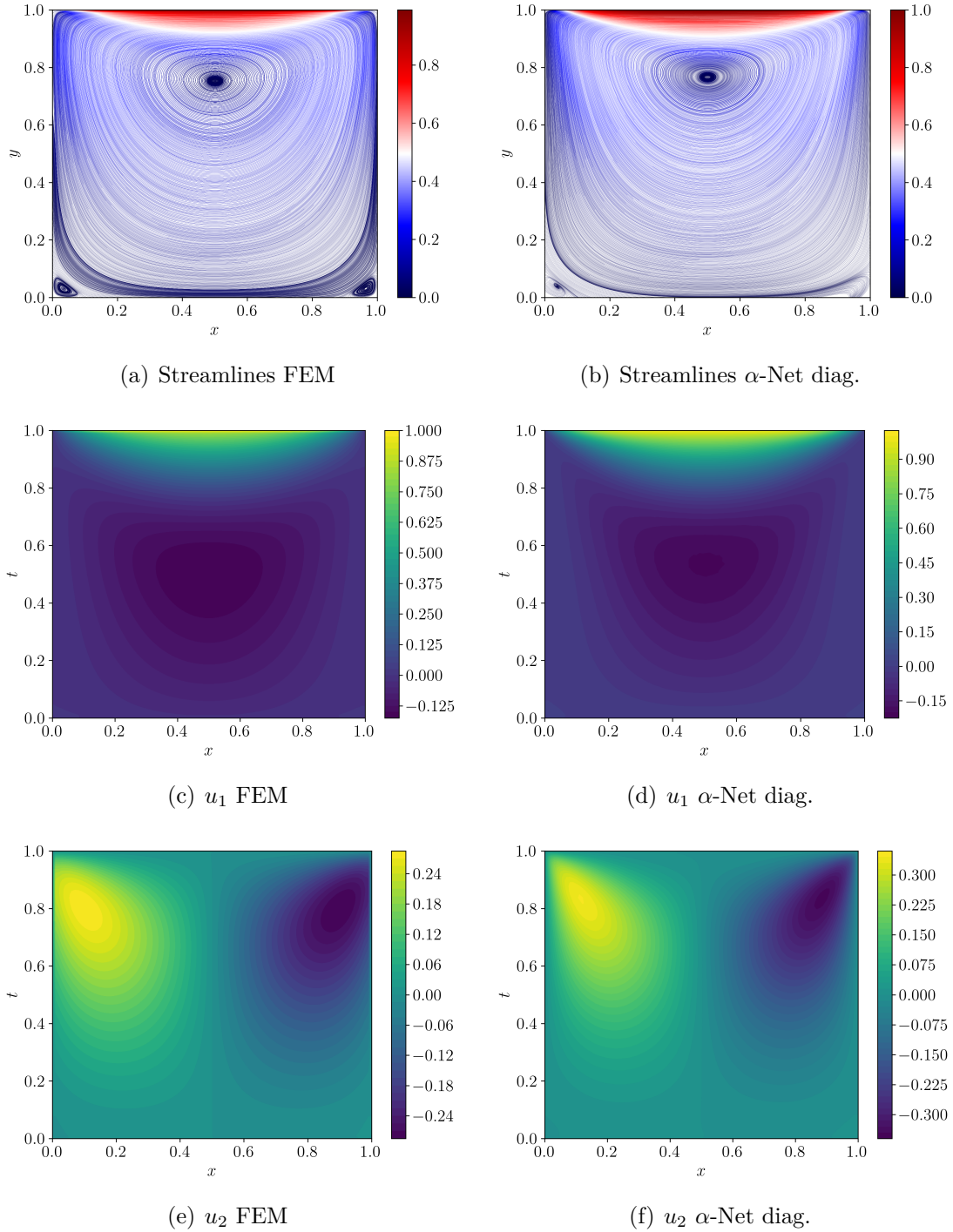


Fig. 5.5 A comparison of the mean solution fields of the α -Net with a diagonal covariance and the FEM solution.

5.8 Discussion

In this chapter, we have proposed a new framework leveraging random grids and GPs for solving forward and inverse parametric physics problems. Our method leverages spatial statistics, VI, GPs, and we proposed a grid-invariant convolutional network to solve forward and inverse problems in a probabilistically coherent manner. Our method is physics informed at training time and incorporates noisy observations of solutions fields from arbitrary grids in a statistically principled way. We tested our method on a nonlinear Poisson problem, the Burgers equation, and the incompressible Navier-Stokes equations. We further compared our method with a series of alternative physics and data-informed methods.

We found our method was highly competitive with other approaches in terms of accuracy, uncertainty quantification, and compute time. It strongly supports the probabilistic treatment of collocation grids for the physics-informed solution of parametric PDEs.

Similarly to the proposed PDDLVM in Ch. 4, the computational advantage of this method over classical solvers will depend largely on the problem setting. However, when comparing the proposed RGNP method to other neural network based surrogates for solving forward and inverse problems, the stochastic treatment of collocation grids should scale more favorably as the spatio-temporal domain of the PDE increases. This is due to the poor scaling properties of uniform grids with respect to dimensionality.

The performance of our approach on the Navier-Stokes examples pointed to some limitations in using uniform distributions over domains for sampling the collocation points. Possibly, the distributions $p(\mathbf{X})$ could be adapted to sample more points close to boundaries to better capture complex boundary effects and be implemented within our framework. Divergence enforcing (Richter-Powell et al., 2022) methods could potentially also be used to accelerate convergence. Chebyshev network architectures could also be investigated (Tang et al., 2023) as an alternative architecture for faster and more regularized residual computation.

Further possible extensions could include the incorporation of CAN-PINN collocation methods (Chiu et al., 2022). The GP formulation of our method implies that other GP methods such as deep kernels (Wilson et al., 2016) and sparse GPs could be leveraged (Snelson and Ghahramani, 2005, Titsias, 2009). We could also make use of variational weak forms in the residual (Kharazmi et al., 2019) to lower

the differentiability order of the PDEs and potentially increase learning stability. Therefore, the generality of our proposed framework implies that it can be readily extended to incorporate many of the newest advances in physics informed ML.

Chapter 6

General Discussion

We now discuss the work presented in this thesis. We review our aim, summarise our contributions, discuss limitations while providing possible avenues of improvement, and finally describe implications and applications.

6.1 Aim

The aim of this thesis was to develop novel methodology to improve the capabilities of simulating and investigating parametric PDEs in engineering. To this end we identified two main areas of interest: The efficient probabilistic estimation of parameters in inverse problems and the construction of scalable parametric PDE probabilistic emulators.

After reviewing some core concepts and tools of probabilistic ML and computational mechanics, we proposed three methods for addressing these challenges by leveraging the tools of weighted residuals, Gaussian processes, deep learning, and variational inference.

We pursued three projects: first we developed active learning surrogates for accelerating BIPs, second we derived a physics driven deep latent variable model, third we improved parametric PDE learning through a random grid neural network parametrised GP. Finally, we tested our framework and compared it with key standard methods.

6.2 Summary of Contributions

Our first main contribution was to develop the surrogate based sampling scheme ALPS-SVGD. We have created a new framework for the non-parametric variational

construction of likelihood function surrogates for accelerating posterior sampling of inverse problems related to parametric PDEs (Chapter 3). We established the criteria by which to guide an active learning GP that assimilates data in an online manner from evaluations of a numerical model. We also set criteria for thinning this active dataset for efficient training of the active surrogate. Our method was shown to be effective for low-dimensional inverse problems where the forward model might be very expensive and full MCMC requiring hundreds of thousands of model evaluations would be infeasible.

In chapter 4 we presented our PDDLVM framework. A novel VI construction that can simultaneously learn forward and inverse probabilistic emulators for parametrised PDEs. We posed a probabilistic model mapping the inverse problems through a factorisation of the joint distribution over variables characterising the PDE. We then posed a variational approximation to this joint model that allowed for the learning of the forward map of the PDE. We then maximised the marginal likelihood of observing a zero residual, which in turn, captured the physics of the problem at hand. Our model can be physics informed through FEM or PINNs depending on ease of implementation and application. We demonstrated the effectiveness of our method on a series of parametric PDEs. We showed that the uncertainties given by the methods were consistent with the desired behaviour of a probabilistic physics emulator, and we compared our methods to non-probabilistic approaches.

Finally, in chapter 5 we fundamentally extended our work on PDDLVM into a function space view with our random grids and neural network parameterised GPs for parametric PDEs. This novel methodology allowed for the probabilistic grid invariant learning of parametric PDE emulators in the forward and inverse direction. In order to implement this framework, we developed a new neural network architecture called GICNet to map solution fields on random grids to the set of parameters associated with the observed solution field. We compared our method to a wide array of alternatives for a collection of PDEs and demonstrated its competitiveness. We also demonstrated that we could handle complicated PDEs as shown in a lid driven cavity flow example, solving for the incompressible Navier-Stokes equations.

In all, our novel work provides new tools to implement machine learning in parametric PDEs for physics driven engineering.

6.3 Limitations and Future Work

We now outline some of the broad limitations of the work while also proposing specific avenues for future work with suggestions to address some of these shortcomings where possible.

Firstly, in as much as ALPS-SVGD has been found to be effective for low-dimensional simple posterior sampling tasks, the overall approach of actively constructing surrogates of posterior surface can lead to unstable algorithms. Some of the instabilities can be alleviated in the case of GP surrogates through the use of carefully constructed mean functions and learnable parameters. Even so, if tasked with sampling from multi-modal distributions or highly complex posterior geometries, the Kriging provided by the GP may not be adequately learned in an online manner. Overall the task of learning an emulator for the actual physics itself and using this physics emulator in the likelihood functions may be a better approach as was used in Cleary et al. (2021). Future work should look at the residual based active learning of physics emulation guided by the exploration of a posterior distribution with some Markov chain targeting this distribution. Such methods would benefit from not needing to solve the PDE itself but learn over the sampling iterations how to emulate the parametric PDE through the residual as with PDDLVM.

Then, in the work related to PDDLVM, a few limitations were addressed in the subsequent chapter on random grids and GPs including the dependence of the method on fixed grids and the use of only diagonal covariance matrices. There were also challenges related to dealing with more complicated PDEs in the case of PINN-PDDLVM due to the high density of collocation points needed causing high computational demands. Further shortcomings are present in the use of FEM-PDDLVM. Many of the operations for FEM used in FEM-PDDLVM were coded in such a way that they are purely sequential. This is too great a limitation for use in a large scale PDE implementation. An area of future work should be to develop methods to parallelise the updating scheme for the stiffness matrix and possibly move these operations onto the GPU for better integration with deep learning libraries.

Also, the work on random grids and GPs has still to overcome some of the issues in PDDLVM. Mapping solution fields to parameters is much easier than mapping noisy sparse observations of solution fields. An interesting area of future research would be to see what kind of problems arise if one inverts sparse noisy observations of $u(x)$ rather than $u(x)$ itself. We might then benefit from using mixture model distributions to capture multi-modality.

Taking the developed framework and applying it to more complicated fluid-structure scenarios would also be of great interest to demonstrate the potential use of the methods in various engineering scenarios.

6.4 Implications and Applications

The methods proposed in this thesis can have far-reaching impact in the engineering sciences. We identify a few potential areas of engineering practice for which our contributions can have a substantial impact.

The algorithm ALPS-SVGD can be used to accelerate Bayesian inversion for expensive to evaluate physics models for parameter spaces of a dozen or so dimensions. When structures are exposed to extreme weather events or natural disasters, they might undergo permanent deformations. Some of these changes may not be readily observable, such as changes happening in steel reinforcement in concrete beams. With accurate numerical models of these objects and observational data, we can reconstruct their internal state through Bayesian inversion to then assess their suitability for continued use. Classical Bayesian inversion may be prohibitively expensive in the case of expensive to evaluate models, and ALPS-SVGD is a tool that may alleviate some of this computational burden making such analyses possible.

The physics driven deep latent variable model outlined in chapter 4, PDDLVM, is developed for the joint training of PDE forward and inverse emulators based on physics that is well simulated by classical numerical solving schemes such as FEM. This can have an important impact in the design process of shell structures such as in structural engineering, car body design, and aerospace applications. The tools developed can aid designers explore the possible combinations of parameters yielding the appropriate physical properties. Once a PDDLVM is trained, the model can then be used to investigate the changes in properties of the physical object and be used for inverse computation. In this way, we can create numerical imitations of design objects where we can efficiently map from physical behaviour to model parameters while capturing uncertainties in the predictive capabilities of the model. Engineered assets can then be deployed alongside their PDDLVM counterpart to inform us of their internal state and predicted behaviour.

The random grid PDE framework presented in chapter 5, is of particular use for parametric fluid applications. The fluid domain is typically a fixed quantity, but the properties of the fluid might be a design variable of interest such as in bio-mechanical applications. RGNP can be used to construct a forward and inverse emulator of this

physical process and can be tailored to the specific instance of a system or patient for physical investigation. In this framework, we can push our inference capabilities further by informing our RGNP model with noisy observations of the physical object. In this way, we are physics and data informed about our systems. This is of particular interest when engineers suspect that the available physical understanding the system is incomplete.

Overall, these contributions have wide-ranging and varied applications all centred on the purpose and bettering of our abilities to treat parametric PDE problems in engineering science.

Appendix A

PDDLVM

A.1 MNSE

We note the definition for the reported Mean Normalized Squared Error as

$$\text{MNSE}(\mathbf{x}_{0:N}^*, \mathbf{x}_{0:N}) := \frac{1}{N} \sum_i^N \frac{\|\mathbf{x}_i^* - \mathbf{x}_i\|^2}{\|\mathbf{x}_i^*\|^2}, \quad (\text{A.1})$$

where $\mathbf{x}^*$ denotes the base truth, and $\mathbf{x}$ denotes the approximation.

A.2 Alternative Derivation of PDDLVM

To derive the PDDLVM from the KL divergence between an intractable posterior and a variational approximation we restate our joint model (4.8) from Sect. 4.3.2 which includes the residual observation variable and the trainable inversion network

$$p_\beta(\mathbf{r}, \mathbf{u}, \mathbf{z}, \mathbf{f}) = p(\mathbf{r}|\mathbf{u}, \mathbf{z}, \mathbf{f})p_\beta(\mathbf{z}|\mathbf{u}, \mathbf{f})p(\mathbf{u})p(\mathbf{f}). \quad (\text{A.2})$$

We also restate our trainable variational approximation distribution (4.10)

$$q_\alpha(\mathbf{u}, \mathbf{z}, \mathbf{f}) = q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{f})p(\mathbf{z})p(\mathbf{f}). \quad (\text{A.3})$$

Using Bayes' rule we write for the intractable posterior of the latent variable given the observed residual

$$p_\beta(\mathbf{u}, \mathbf{z}, \mathbf{f}|\mathbf{r}) = \frac{p_\beta(\mathbf{r}, \mathbf{u}, \mathbf{z}, \mathbf{f})}{p(\mathbf{r})}. \quad (\text{A.4})$$

We then consider the KL divergence between this intractable posterior and our tractable variational approximation

$$\begin{aligned} D_{KL}(q_\alpha(\mathbf{u}, \mathbf{z}, \mathbf{f}) || p_\beta(\mathbf{u}, \mathbf{z}, \mathbf{f} | \mathbf{r})) &= \int \log \frac{q_\alpha(\mathbf{u}, \mathbf{z}, \mathbf{f})}{p_\beta(\mathbf{u}, \mathbf{z}, \mathbf{f} | \mathbf{r})} q_\alpha(\mathbf{u}, \mathbf{z}, \mathbf{f}) \, d\mathbf{u} d\mathbf{z} d\mathbf{f}, \\ &= \int \log \frac{q_\alpha(\mathbf{u}, \mathbf{z}, \mathbf{f}) p(\mathbf{r})}{p_\beta(\mathbf{r}, \mathbf{u}, \mathbf{z}, \mathbf{f})} q_\alpha(\mathbf{u}, \mathbf{z}, \mathbf{f}) \, d\mathbf{u} d\mathbf{z} d\mathbf{f}. \end{aligned} \quad (\text{A.5})$$

From this, we factorise the evidence of the residual $p(\mathbf{r})$ out of the integral and by leveraging that $\int q_\alpha(\mathbf{u}, \mathbf{z}, \mathbf{f}) \, d\mathbf{u} d\mathbf{z} d\mathbf{f} = 1$ we obtain

$$\begin{aligned} D_{KL}(q_\alpha(\mathbf{u}, \mathbf{z}, \mathbf{f}) || p_\beta(\mathbf{u}, \mathbf{z}, \mathbf{f} | \mathbf{r})) &= \\ &\int \log \frac{q_\alpha(\mathbf{u}, \mathbf{z}, \mathbf{f})}{p_\beta(\mathbf{r}, \mathbf{u}, \mathbf{z}, \mathbf{f})} q_\alpha(\mathbf{u}, \mathbf{z}, \mathbf{f}) \, d\mathbf{u} d\mathbf{z} d\mathbf{f} + \log p(\mathbf{r}). \end{aligned} \quad (\text{A.6})$$

We note that the KL divergence is a non-negative quantity so that

$$D_{KL}(q_\alpha(\mathbf{u}, \mathbf{z}, \mathbf{f}) || p_\beta(\mathbf{u}, \mathbf{z}, \mathbf{f} | \mathbf{r})) \geq 0. \quad (\text{A.7})$$

This allows to define the lower bound on the log marginal residual, as we further specify that $\mathbf{r} = \mathbf{0}$. We obtain a lower bound on the marginal probability of solving the parameterised PDE for all parameter values specified by their priors,

$$\log p(\mathbf{r} = \mathbf{0}) \geq \int \log \frac{p_\beta(\mathbf{r} = \mathbf{0}, \mathbf{u}, \mathbf{z}, \mathbf{f})}{q_\alpha(\mathbf{u}, \mathbf{z}, \mathbf{f})} q_\alpha(\mathbf{u}, \mathbf{z}, \mathbf{f}) \, d\mathbf{u} d\mathbf{z} d\mathbf{f}. \quad (\text{A.8})$$

Substituting the chosen factorisations (A.2) and (A.3) into the individual joint terms, we obtain the PDDLVM framework derived in Sec. 4.3.2,

$$\log p(\mathbf{r} = \mathbf{0}) \geq \int \log \frac{p(\mathbf{r} = \mathbf{0} | \mathbf{u}, \mathbf{z}, \mathbf{f}) p_\beta(\mathbf{z} | \mathbf{u}, \mathbf{f}) p(\mathbf{u})}{q_\alpha(\mathbf{u} | \mathbf{z}, \mathbf{f}) p(\mathbf{z})} q_\alpha(\mathbf{u} | \mathbf{z}, \mathbf{f}) p(\mathbf{z}) p(\mathbf{f}) \, d\mathbf{u} d\mathbf{z} d\mathbf{f}. \quad (\text{A.9})$$

In order to relate the proposed probabilistic model to classical non-probabilistic approaches it is instructive to compare them after introducing the probability densities and taking their logarithms the resulting expressions.

A.3 Experimental Details

All experiments were run on an AMD Ryzen 9 5950X 16-Core Processor CPU and an NVIDIA GeForce RTX 3090 GPU. For all experiments TensorFlow GPU usage was limited between 6 and 13 GBs. The Poisson 1D examples were run on the CPU rather than the GPU as the FEM code is sequential and runs faster on the CPU.

The 3D Bunny example was run on a mix of CPU and GPU. The PINN example were run entirely on the GPU as it can best leverage this type of hardware.

A.4 Additional Results for 1D Nonlinear Poisson

In Fig. A.1 we plot the same quantities as in Fig. 4.5 but not in the log scale and in box plot format. We show the MNSEs for a variety of parameters of $\epsilon_{\mathbf{r}}$. In Fig. A.2 we plot the training of the ELBO for various settings of $\epsilon_{\mathbf{r}}$. We notice that the smaller the choice of $\epsilon_{\mathbf{r}}$, the slower the convergence, but the higher the accuracy.

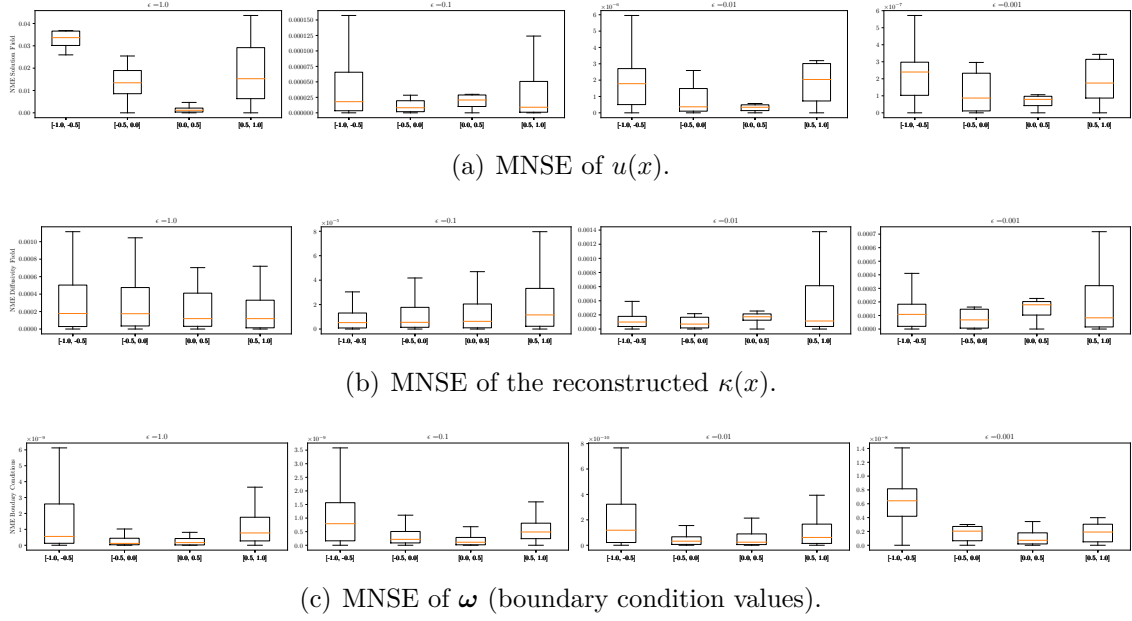


Fig. A.1 The MNSE for different ϵ_r values for the ranges of κ given on the x-axis. This was computed for the 1D linear Poisson example of Sec. 4.4.1 and is computed from the same values as Fig. 4.5 but not in the log scale. The prior on which the FEM-PDDLVM was trained was $p(\mathbf{z}) = \mathcal{N}(0, 0.5)$.

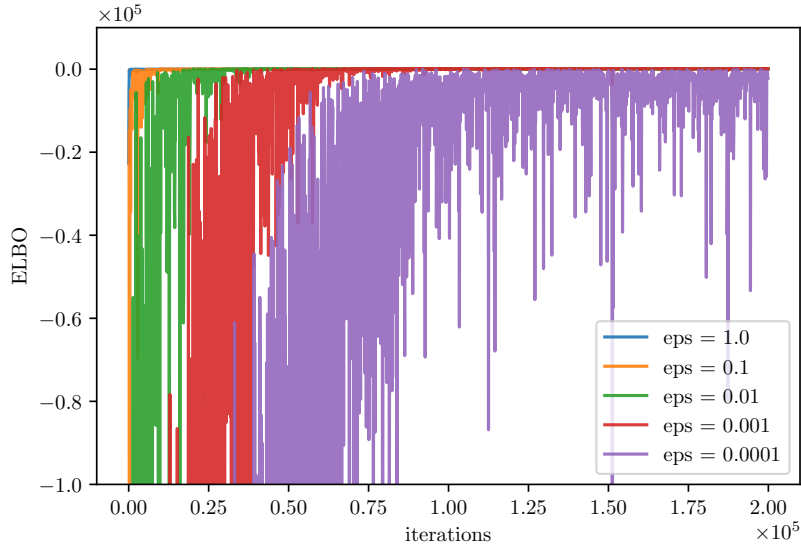


Fig. A.2 The convergence of the ELBO estimate for a 1D linear Poisson equation for various choices of ϵ_r .

A.5 PINN-PDDLVM and Weighted Residuals

In the case of PINN, we clarify its link with the weighted residual method as a method of least-squares collocation (Finlayson, 2013). This is done by choosing $w_i(\mathbf{x}) = \frac{\partial \mathcal{R}(\hat{u}, \mathbf{x})}{\partial W} \delta(\mathbf{x}_i - \mathbf{x})$ with N_1, N_2, N_3 collocation points in $\{\Omega, \Gamma, \mathcal{I}\}$, which respectively are the domain, boundary conditions and initial conditions. W denotes the collection of NN parameters. As a simplification we write $\mathbf{x} = \{\mathbf{x}, t\}$. As in (Finlayson and Scriven, 1966) we motivate the least-squares method as

$$r_i = \int_{\Omega} w_i(\mathbf{x}) \mathcal{R}_{\Omega}(\hat{u}, \mathbf{x}) d\mathbf{x} + \int_{\Gamma} w_i(\mathbf{x}) \mathcal{R}_{\Gamma}(\hat{u}, \mathbf{x}) d\mathbf{x} + \int_{\mathcal{I}} w_i(\mathbf{x}) \mathcal{R}_{\mathcal{I}}(\hat{u}, \mathbf{x}) d\mathbf{x}. \quad (\text{A.10})$$

We then substitute the definition of $w_i(\mathbf{x}) = \frac{\partial \mathcal{R}(\hat{u}, \mathbf{x})}{\partial W} \delta(\mathbf{x}_i - \mathbf{x})$,

$$\begin{aligned} r_i = \int_{\Omega} \frac{\partial \mathcal{R}_{\Omega}}{\partial W} \delta(\mathbf{x}_i - \mathbf{x}) \mathcal{R}_{\Omega}(\hat{u}, \mathbf{x}) d\mathbf{x} + \int_{\Gamma} \frac{\partial \mathcal{R}_{\Gamma}}{\partial W} \delta(\mathbf{x}_i - \mathbf{x}) \mathcal{R}_{\Gamma}(\hat{u}, \mathbf{x}) d\mathbf{x} \\ + \int_{\mathcal{I}} \frac{\partial \mathcal{R}_{\mathcal{I}}}{\partial W} \delta(\mathbf{x}_i - \mathbf{x}) \mathcal{R}_{\mathcal{I}}(\hat{u}, \mathbf{x}) d\mathbf{x}. \end{aligned} \quad (\text{A.11})$$

After factorising out the partial derivative with respect to W and introducing the factor of one-half, we obtain

$$\begin{aligned} r_i = \frac{1}{2} \frac{\partial}{\partial W} \left[\int_{\Omega} \delta(\mathbf{x}_i - \mathbf{x}) \mathcal{R}_{\Omega}^2(\hat{u}, \mathbf{x}) d\mathbf{x} + \int_{\Gamma} \delta(\mathbf{x}_i - \mathbf{x}) \mathcal{R}_{\Gamma}^2(\hat{u}, \mathbf{x}) d\mathbf{x} \right. \\ \left. + \int_{\mathcal{I}} \delta(\mathbf{x}_i - \mathbf{x}) \mathcal{R}_{\mathcal{I}}^2(\hat{u}, \mathbf{x}) d\mathbf{x} \right]. \end{aligned} \quad (\text{A.12})$$

Since r_i is required to be zero for all i , all residuals must be zero. We define

$$I_i(W) = \mathcal{R}_{\Omega}^2(\hat{u}, \mathbf{x}_i) + \mathcal{R}_{\Gamma}^2(\hat{u}, \mathbf{x}_i) + \mathcal{R}_{\mathcal{I}}^2(\hat{u}, \mathbf{x}_i). \quad (\text{A.13})$$

If we average over the points in each residual domain and assemble into a single objective, we obtain

$$I(W) = \frac{1}{N_1} \sum_{\mathbf{x}_i \in \Omega} \mathcal{R}_{\Omega}^2(\hat{u}, \mathbf{x}_i) + \frac{1}{N_2} \sum_{\mathbf{x}_i \in \Gamma} \mathcal{R}_{\Gamma}^2(\hat{u}, \mathbf{x}_i) + \frac{1}{N_3} \sum_{\mathbf{x}_i \in \mathcal{I}} \mathcal{R}_{\mathcal{I}}^2(\hat{u}, \mathbf{x}_i), \quad (\text{A.14})$$

which is the commonly used training objective for PINN-type models.

A.6 Additional Details

In this subsection collect information from the paper for easy referencing.

1D Linear Poisson – Sec. 4.4.1

Table A.1 Basic information 1D Linear Poisson

$\dim \mathbf{u}$	$\dim \mathbf{r}$	$\dim \mathbf{z}$	$\mathbf{z}$	$p(\mathbf{z})$	$\epsilon_{\mathbf{r}}$	α -Net	β -Net	iters
5	21	1	$\{\kappa\}$	$\mathcal{N}(0, 1)$	10^{-2}	$3l, 50n$	$3l, 50n$	2×10^5

1D Linear Poisson Observable Map Inversion – Sec. 4.4.1

Table A.2 Basic information 1D Linear Poisson Observable Map Inversion I

$\dim \mathbf{u}$	$\dim \mathbf{r}$	$\dim \mathbf{z}$	$\dim \mathbf{f}$	$\dim \mathbf{y}$	$\mathbf{z}$	σ_y	$\epsilon_{\mathbf{r}}$
8	101	4	1	80	$\{\kappa, a, b\}$	10^{-2}	

Table A.3 Basic information 1D Linear Poisson Observable Map Inversion II

$p(\kappa_i)$	$p(\mathbf{f})$	$p(a)$	$p(b)$	α -Net	β -Net	iters
$\mathcal{U}(-1, 1)$	$\mathcal{U}(1, 5)$	$\mathcal{U}(-0.5, 0.5)$	$\mathcal{U}(-0.5, 0.5)$	$4l, 50n,$	$4l, 50n,$	1×10^6

1D Nonlinear Poisson – Sec. 4.4.2

Table A.4 Basic information 1D Nonlinear Poisson I

$\dim \mathbf{u}$	$\dim \mathbf{r}$	$\dim \mathbf{z}$	$\dim \mathbf{f}$	$\mathbf{z}$	$\epsilon_{\mathbf{r}}$
10	61	5	1	$\{\kappa, b\}$	10^{-2}

Table A.5 Basic information 1D Nonlinear Poisson II

$p(\mathbf{z})$	$p(\mathbf{f})$	$p(b)$	α -Net	β -Net	iters
$\mathcal{N}(0, 1)$	$\mathcal{U}(1, 1)$	$\mathcal{U}(0.5, 1)$	$4l, 100n$	$4l, 100n$	1×10^6

Thin-Walled Flexible Shell – Sec. 4.4.3

Table A.6 Basic information Thin-Walled Flexible Shell

$\dim \mathbf{u}$	$\dim \mathbf{r}$	$\dim \mathbf{z}$	$\mathbf{z}$	$p(\mathbf{z})$	$\epsilon_{\mathbf{r}}$	α -Net	β -Net	iters
1029	24858	3	$\{\kappa\}$	$\mathcal{U}(-2, 2)$	10^{-2}	$3l, 2500n$	$3l, 2500n$	5×10^4

Nonlinear Heat Equation – Sec. 4.4.4

Table A.7 Basic information Nonlinear Heat Equation I

$\dim \mathbf{u}$	$\dim \mathbf{r}$	$\dim \mathbf{z}$	$\mathbf{z}$	$p(\mathbf{z})$	$\epsilon_{\mathbf{r}}$
2×10^5	2×10^5	2	$\{\gamma, \kappa\}$	$\mathcal{U}(1, 5) \times \mathcal{U}(1, 5)$	10^{-2}

Table A.8 Basic information Nonlinear Heat Equation II

α -Net	β -Net	iters
$4l, 100n$	$3c, 1l - 500n$	5×10^5

Inhomogeneous Wave Equation – Sec. 4.4.5

Table A.9 Basic information Inhomogeneous Wave Equation I

$\dim \mathbf{u}$	$\dim \mathbf{r}$	$\dim \mathbf{z}$	$\mathbf{z}$	$p(\mathbf{z})$	$\epsilon_{\mathbf{r}}$
2.5×10^3	2.5×10^3	4	$\{\mathbf{f}\}$	$\mathcal{U}(-5, 5)$	10^{-2}

Table A.10 Basic information Inhomogeneous Wave Equation II

α -Net	β -Net	iters
$4l, 100n$	$3c, 1l - 500n$	2.5×10^5

Appendix B

RGNP

B.1 Experimental Details

B.1.1 Nonlinear Poisson

The $d(x)$ function used to enforce the Dirichlet boundary conditions for this problem is

$$d(x) = \cos\left(\frac{x\pi}{2}\right). \quad (\text{B.1})$$

Architecture Details:

- number of hidden layers: 7
- number of neurons / hidden layer: 300
- activation: swish
- GICNet channel dimension: 20
- GICNet point/dim lattice: 20

We generated a dataset of 1000 evaluations for parameters drawn from the priors as the ground truth. It took 12.0 minutes to generate 1000 solutions using FEniCS.

B.1.2 Burgers

For the DeepONet, FNN, and GICNet we train forward net with mean absolute residual as squared mean residual is too unstable. We have 10k training latent parameters. For DeepONet we shorten the number of collocation evaluations for 100×100 grid as we reduce the batch size but keep the same number of iterations.

Generating the 1000 sample dataset using FEniCS takes 33.5 mins. The $d(x, t)$ function used to enforce the Dirichlet boundary conditions for this problem is

$$d(x, t) = \sin(x\pi) t. \quad (\text{B.2})$$

Architecture Details:

- number of hidden layers: 8
- number of neurons / hidden layer: 400
- activation: swish
- GICNet channel dimension: 20
- GICNet point/dim lattice: 10

B.1.3 Navier-Stokes

The $d(\mathbf{x}, t)$ function used to enforce the Dirichlet boundary conditions for this problem is

$$d(x_{(1)}, x_{(2)}, t) = \sin(x_{(1)}\pi) \sin(x_{(2)}\pi)t. \quad (\text{B.3})$$

The full Navier-Stokes equations can be written more explicitly as

$$\begin{aligned} z_0 \frac{\partial u_1}{\partial t} + z_0 \left(u_1 \frac{\partial u_1}{\partial x_{(1)}} + u_2 \frac{\partial u_1}{\partial x_{(2)}} \right) + \frac{\partial p}{\partial x_{(1)}} - z_1 \left(\frac{\partial^2 u_1}{\partial x_{(1)}^2} + \frac{\partial^2 u_1}{\partial x_{(2)}^2} \right) &= 0, \\ z_0 \frac{\partial u_2}{\partial t} + z_0 \left(u_1 \frac{\partial u_2}{\partial x_{(1)}} + u_2 \frac{\partial u_2}{\partial x_{(2)}} \right) + \frac{\partial p}{\partial x_{(2)}} - z_1 \left(\frac{\partial^2 u_2}{\partial x_{(1)}^2} + \frac{\partial^2 u_2}{\partial x_{(2)}^2} \right) &= 0, \\ \frac{\partial u_1}{\partial x_{(1)}} + \frac{\partial u_2}{\partial x_{(2)}} &= 0. \end{aligned} \quad (\text{B.4})$$

The fully defined boundary and initial conditions are

$$\begin{aligned} u_1(x_{(1)}, 1, t) &= (1 - (2x_{(1)} - 1)^6)t, \\ u_1(0, x_{(2)}, t) &= u_1(1, x_{(2)}, t) = u_1(x_{(1)}, 0, t) = 0, \\ u_2(0, x_{(2)}, t) &= u_2(1, x_{(2)}, t) = u_2(x_{(1)}, 0, t) = u_2(x_{(1)}, 1, t) = 0, \\ u_1(x_{(1)}, x_{(2)}, 0) &= u_2(x_{(1)}, x_{(2)}, 0) = 0, \\ p(0, 0, t) &= 0. \end{aligned} \quad (\text{B.5})$$

Architecture Details:

- number of hidden layers: 10
- number of neurons / hidden layer: 200
- activation: swish
- GICNet channel dimension: 30
- GICNet point/dim lattice: 10

B.2 Hardware

The GPU memory usage was limited to 6GB for the nonlinear Poisson problems, 10GB for the Burgers examples, and 20GB for the Navier-Stokes examples.

References

- Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283.
- Akyildiz, Ö. D., Duffin, C., Sabanis, S., and Girolami, M. (2022). Statistical Finite Elements via Langevin Dynamics. *SIAM/ASA Journal of Uncertainty Quantification*.
- Anandkumar, A., Azizzadenesheli, K., Bhattacharya, K., Kovachki, N., Li, Z., Liu, B., and Stuart, A. (2020). Neural operator: Graph kernel network for partial differential equations. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*.
- Ardizzone, L., Kruse, J., Wirkert, S., Rahner, D., Pellegrini, E. W., Klessen, R. S., Maier-Hein, L., Rother, C., and Köthe, U. (2018). Analyzing inverse problems with invertible neural networks. *arXiv preprint arXiv:1808.04730*.
- Aristoff, D. and Bangerth, W. (2021). A benchmark for the bayesian inversion of coefficients in partial differential equations. *arXiv preprint arXiv:2102.07263*.
- Arridge, S., Maass, P., Öktem, O., and Schönlieb, C.-B. (2019). Solving inverse problems using data-driven models. *Acta Numerica*, 28:1–174.
- Asmar, N. H. (2016). *Partial differential equations with Fourier series and boundary value problems*. Courier Dover Publications.
- Astrid, P., Weiland, S., Willcox, K., and Backx, T. (2008). Missing point estimation in models described by proper orthogonal decomposition. *IEEE Transactions on Automatic Control*, 53(10):2237–2251.
- Ba, J., Erdogdu, M. A., Ghassemi, M., Sun, S., Suzuki, T., Wu, D., and Zhang, T. (2021). Understanding the variance collapse of SVGD in high dimensions. In *International Conference on Learning Representations*.
- Bateman, H. (1915). Some recent researches on the motion of fluids. *Monthly Weather Review*, 43(4):163–170.
- Belov, Y. Y. (2012). Inverse problems for partial differential equations. In *Inverse Problems for Partial Differential Equations*. De Gruyter.
- Belytschko, T., Liu, W. K., Moran, B., and Elkhodary, K. (2014). *Nonlinear finite elements for continua and structures*. John wiley & sons.

- Betancourt, M. (2017). A conceptual introduction to hamiltonian monte carlo. *arXiv preprint arXiv:1701.02434*.
- Bhattacharya, K., Hosseini, B., Kovachki, N. B., and Stuart, A. M. (2021). Model Reduction And Neural Networks For Parametric PDEs. *The SMAI Journal of Computational Mathematics*, 7:121–157.
- Bishop, C. M. and Nasrabadi, N. M. (2006). *Pattern recognition and machine learning*, volume 4. Springer.
- Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877.
- Botella, O. and Peyret, R. (1998). Benchmark spectral results on the lid-driven cavity flow. *Computers & Fluids*, 27(4):421–433.
- Box, G. E. and Muller, M. E. (1958). A note on the generation of random normal deviates. *The annals of mathematical statistics*, 29(2):610–611.
- Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018). JAX: composable transformations of Python+NumPy programs.
- Brenner, S. C., Scott, L. R., and Scott, L. R. (2008). *The mathematical theory of finite element methods*, volume 3. Springer.
- Brooks, S., Gelman, A., Jones, G., and Meng, X.-L. (2011). *Handbook of markov chain monte carlo*. CRC press.
- Brunton, S. L., Proctor, J. L., and Kutz, J. N. (2016). Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937.
- Bui-Thanh, T., Willcox, K., and Ghattas, O. (2008). Model reduction for large-scale systems with high-dimensional parametric input space. *SIAM Journal on Scientific Computing*, 30(6):3270–3288.
- Bunch, J. R. and Rose, D. J. (2014). *Sparse matrix computations*. Academic Press.
- Burt, D. R., Ober, S. W., Garriga-Alonso, A., and van der Wilk, M. (2020). Understanding variational inference in function-space. In *Third Symposium on Advances in Approximate Bayesian Inference*.
- Caffisch, R. E. (1998). Monte carlo and quasi-monte carlo methods. *Acta numerica*, 7:1–49.
- Cai, Z. (2001). Weighted nadaraya-watson regression estimation. *Statistics & probability letters*, 51(3):307–318.
- Cartz, L. (1995). Nondestructive testing.
- Chakraverty, S., Mahato, N., Karunakar, P., and Rao, T. D. (2019). *Advanced numerical and semi-analytical methods for differential equations*. John Wiley & Sons.

- Chavent, G. (2010). *Nonlinear least squares for inverse problems: theoretical foundations and step-by-step guide for applications*. Springer Science & Business Media.
- Chen, P. and Ghattas, O. (2020). Projected Stein variational gradient descent. *Advances in Neural Information Processing Systems*, 33:1947–1958.
- Chen, P., Wu, K., Chen, J., O’Leary-Roseberry, T., and Ghattas, O. (2019). Projected Stein variational Newton: A fast and scalable bayesian inference method in high dimensions. *Advances in Neural Information Processing Systems*, 32.
- Chen, Y., Hosseini, B., Owhadi, H., and Stuart, A. M. (2021). Solving and learning nonlinear pdes with Gaussian processes. *Journal of Computational Physics*, 447:110668.
- Chiu, P.-H., Wong, J. C., Ooi, C., Dao, M. H., and Ong, Y.-S. (2022). Can-pinn: A fast physics-informed neural network based on coupled-automatic-numerical differentiation method. *Computer Methods in Applied Mechanics and Engineering*, 395:114909.
- Cirak, F. and Long, Q. (2011). Subdivision shells with exact boundary control and non-manifold geometry. *International Journal for Numerical Methods in Engineering*, 88(9):897–923.
- Cirak, F., Ortiz, M., and Schröder, P. (2000). Subdivision surfaces: a new paradigm for thin-shell finite-element analysis. *International Journal for Numerical Methods in Engineering*, 47(12):2039–2072.
- Cleary, E., Garbuno-Inigo, A., Lan, S., Schneider, T., and Stuart, A. M. (2021). Calibrate, emulate, sample. *Journal of Computational Physics*, 424:109716.
- Cockayne, J., Oates, C. J., Sullivan, T. J., and Girolami, M. (2019). Bayesian probabilistic numerical methods. *SIAM review*, 61(4):756–789.
- Cook, R. D. (1994). *Modeling for Stress Analysis*. New York: Wiley.
- Cotter, S. L., Roberts, G. O., Stuart, A. M., and White, D. (2013). Mcmc methods for functions: modifying old algorithms to make them faster.
- Cottrell, J. A., Hughes, T. J., and Bazilevs, Y. (2009). *Isogeometric analysis: toward integration of CAD and FEA*. John Wiley & Sons.
- Cressie, N. and Moores, M. T. (2021). Spatial statistics. *arXiv preprint arXiv:2105.07216*.
- D’Angelo, F. and Fortuin, V. (2021). Annealed Stein variational gradient descent. *arXiv preprint arXiv:2101.09815*.
- Dehaene, D. and Brossard, R. (2021). Re-parameterizing vaes for stability. *arXiv preprint arXiv:2106.13739*.
- Detommaso, G., Cui, T., Marzouk, Y., Spantini, A., and Scheichl, R. (2018). A Stein variational Newton method. *Advances in Neural Information Processing Systems*, 31.
- DeVore, R. A. (1998). Nonlinear approximation. *Acta numerica*, 7:51–150.

- Diaconis, P. (1988). Bayesian numerical analysis. *Statistical decision theory and related topics IV*, 1:163–175.
- Ding, J. and Zhou, A. (2007). Eigenvalues of rank-one updated matrices with some applications. *Applied Mathematics Letters*, 20(12):1223–1226.
- Duffin, C. (2022). *Statistical finite element methods for nonlinear PDEs*. PhD thesis, The University of Western Australia.
- Duffin, C., Cripps, E., Stemler, T., and Girolami, M. (2021). Statistical finite elements for misspecified models. *Proceedings of the National Academy of Sciences*, 118(2).
- Duncan, A., Nüsken, N., and Szpruch, L. (2019). On the geometry of Stein variational gradient descent. *arXiv preprint arXiv:1912.00894*.
- Dutordoir, V., Saul, A., Ghahramani, Z., and Simpson, F. (2022). Neural diffusion processes. *arXiv preprint arXiv:2206.03992*.
- Ern, A. and Guermond, J.-L. (2004). *Theory and practice of finite elements*, volume 159. Springer.
- Eymard, R., Gallouët, T., and Herbin, R. (2000). Finite volume methods. *Handbook of numerical analysis*, 7:713–1018.
- Fanaskov, V. and Oseledets, I. (2022). Spectral neural operators. *arXiv preprint arXiv:2205.10573*.
- Farrell, P. E., Ham, D. A., Funke, S. W., and Rognes, M. E. (2013). Automated derivation of the adjoint of high-level transient finite element programs. *SIAM Journal on Scientific Computing*, 35(4):C369–C393.
- Febrianto, E., Butler, L., Girolami, M., and Cirak, F. (2022). Digital twinning of self-sensing structures using the statistical finite element method. *Data-Centric Engineering*, 3:e31.
- Fefferman, C. L. (2000). Existence and smoothness of the navier-stokes equation. *The millennium prize problems*, 57:67.
- Finlayson, B. A. (2013). *The method of weighted residuals and variational principles*. SIAM.
- Finlayson, B. A. and Scriven, L. E. (1966). The method of weighted residuals—a review. *Applied Mechanics Review*, 19(9):735–748.
- Fletcher, R. (2013). *Practical methods of optimization*. John Wiley & Sons.
- Foreman-Mackey, D., Hogg, D. W., Lang, D., and Goodman, J. (2013). Emcee: the mcmc hammer. *Publications of the Astronomical Society of the Pacific*, 125(925):306.
- Funke, S. W. and Farrell, P. E. (2013). A framework for automated pde-constrained optimisation. *arXiv preprint arXiv:1302.3894*.

- Gao, H., Zahr, M. J., and Wang, J.-X. (2022). Physics-informed graph neural Galerkin networks: A unified framework for solving pde-governed forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 390:114502.
- Garbuno-Inigo, A., Hoffmann, F., Li, W., and Stuart, A. M. (2020). Interacting langevin diffusions: Gradient structure and ensemble kalman sampler. *SIAM Journal on Applied Dynamical Systems*, 19(1):412–441.
- Garnelo, M., Rosenbaum, D., Maddison, C., Ramalho, T., Saxton, D., Shanahan, M., Teh, Y. W., Rezende, D., and Eslami, S. A. (2018a). Conditional neural processes. In *International Conference on Machine Learning*, pages 1704–1713. PMLR.
- Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S., and Teh, Y. W. (2018b). Neural processes. *arXiv preprint arXiv:1807.01622*.
- Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. (1995). *Bayesian data analysis*. Chapman and Hall/CRC.
- Genton, M. G. (2001). Classes of kernels for machine learning: a statistics perspective. *Journal of machine learning research*, 2(Dec):299–312.
- Gerald, C. F. (2004). *Applied numerical analysis*. Pearson Education India.
- Ghattas, O. and Willcox, K. (2021). Learning physics-based models from data: perspectives from inverse problems and model reduction. *Acta Numerica*, 30:445–554.
- Gibson, W. C. (2021). *The method of moments in electromagnetics*. Chapman and Hall/CRC.
- Gil, A., Segura, J., and Temme, N. M. (2007). *Numerical methods for special functions*. SIAM.
- Girard, A., Rasmussen, C., Candela, J. Q., and Murray-Smith, R. (2002). Gaussian process priors with uncertain inputs application to multiple-step ahead time series forecasting. *Advances in neural information processing systems*, 15.
- Girolami, M. and Calderhead, B. (2011). Riemann manifold langevin and hamiltonian monte carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(2):123–214.
- Girolami, M., Febrianto, E., Yin, G., and Cirak, F. (2021). The statistical finite element method (statfem) for coherent synthesis of observation data and model predictions. *Computer Methods in Applied Mechanics and Engineering*, 375:113533.
- Goodman, J. and Weare, J. (2010). Ensemble samplers with affine invariance. *Communications in applied mathematics and computational science*, 5(1):65–80.
- Gordon, J., Bruinsma, W. P., Foong, A. Y., Requeima, J., Dubois, Y., and Turner, R. E. (2019). Convolutional conditional neural processes. *arXiv preprint arXiv:1910.13556*.
- Gramacy, R. B. (2020). *Surrogates: Gaussian process modeling, design, and optimization for the applied sciences*. Chapman and Hall/CRC.

- Grossmann, T. G., Komorowska, U. J., Latz, J., and Schönlieb, C.-B. (2023). Can physics-informed neural networks beat the finite element method? *arXiv preprint arXiv:2302.04107*.
- Grote, M. J. and Huckle, T. (1997). Parallel preconditioning with sparse approximate inverses. *SIAM Journal on Scientific Computing*, 18(3):838–853.
- Guo, L., Wu, H., and Zhou, T. (2022). Normalizing field flows: Solving forward and inverse stochastic differential equations using physics-informed flow models. *Journal of Computational Physics*, 461:111202.
- Han, J., Jentzen, A., and Ee, W. (2017). Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115.
- Hatami, M. (2017). *Weighted residual methods: principles, modifications and applications*. Academic Press.
- Hensman, J., Fusi, N., and Lawrence, N. D. (2013). Gaussian processes for big data. *arXiv preprint arXiv:1309.6835*.
- Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. (2013). Stochastic variational inference. *Journal of Machine Learning Research*.
- Hoffman, M. D., Gelman, A., et al. (2014). The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623.
- Jacobsen, C. and Duraisamy, K. (2022). Disentangling generative factors of physical fields using variational autoencoders. *Frontiers in Physics*.
- Jadaliha, M., Xu, Y., Choi, J., Johnson, N. S., and Li, W. (2012). Gaussian process regression for sensor networks under localization uncertainty. *IEEE Transactions on Signal Processing*, 61(2):223–237.
- Joyce, J. M. (2011). Kullback-leibler divergence. In *International encyclopedia of statistical science*, pages 720–722. Springer.
- Kaltenbach, S. and Koutsourelakis, P.-S. (2020). Incorporating physical constraints in a deep probabilistic machine learning framework for coarse-graining dynamical systems. *Journal of Computational Physics*, 419:109673.
- Kaltenbach, S., Perdikaris, P., and Koutsourelakis, P.-S. (2022). Semi-supervised invertible deepnets for bayesian inverse problems. *arXiv preprint arXiv:2209.02772*.
- Kandasamy, K., Krishnamurthy, A., Schneider, J., and Póczos, B. (2018). Parallelised bayesian optimisation via thompson sampling. In *International Conference on Artificial Intelligence and Statistics*, pages 133–142. PMLR.
- Kappen, H. J., Gómez, V., and Opper, M. (2012). Optimal control as a graphical model inference problem. *Machine learning*, 87:159–182.
- Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., and Yang, L. (2021). Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440.

- Kass, R. E. (1990). The validity of posterior expansions based on laplace's method. *Bayesian and likelihood methods in statistics and econometrics*, pages 473–487.
- Kennedy, M. C. and O'Hagan, A. (2001). Bayesian calibration of computer models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(3):425–464.
- Kharazmi, E., Zhang, Z., and Karniadakis, G. E. (2019). Variational physics-informed neural networks for solving partial differential equations. *arXiv preprint arXiv:1912.00873*.
- Kim, B., Azevedo, V. C., Thuerey, N., Kim, T., Gross, M., and Solenthaler, B. (2019). Deep fluids: A generative network for parameterized fluid simulations. In *Computer Graphics Forum*, volume 38, pages 59–70. Wiley Online Library.
- Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.
- Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.
- Kingma, D. P. and Welling, M. (2019). An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*.
- Knoblauch, J., Jewson, J., and Damoulas, T. (2019). Generalized variational inference: Three arguments for deriving new posteriors. *arXiv preprint arXiv:1904.02063*.
- Kolmogorov, A. N. and Bharucha-Reid, A. T. (2018). *Foundations of the theory of probability: Second English Edition*. Courier Dover Publications.
- Korba, A., Salim, A., Arbel, M., Luise, G., and Gretton, A. (2020). A non-asymptotic analysis for Stein variational gradient descent. *Advances in Neural Information Processing Systems*, 33:4672–4682.
- Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., and Anandkumar, A. (2021). Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*.
- Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., and Anandkumar, A. (2023). Neural operator: Learning maps between function spaces with applications to pdes. *Journal of Machine Learning Research*, 24(89):1–97.
- Krishnapriyan, A., Gholami, A., Zhe, S., Kirby, R., and Mahoney, M. W. (2021). Characterizing possible failure modes in physics-informed neural networks. *Advances in Neural Information Processing Systems*, 34:26548–26560.
- Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.
- Lagaris, I. E., Likas, A., and Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000.

- LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.
- Lee, B.-J., Hong, S., and Kim, K.-E. (2020). Residual neural processes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4545–4552.
- Leviyev, A., Chen, J., Wang, Y., Ghattas, O., and Zimmerman, A. (2022). A stochastic Stein variational Newton method. *arXiv preprint arXiv:2204.09039*.
- Li, L., Li, Y., Liu, J.-G., Liu, Z., and Lu, J. (2020a). A stochastic version of Stein variational gradient descent for efficient sampling. *Communications in Applied Mathematics and Computational Science*, 15(1):37–63.
- Li, Z., Huang, D. Z., Liu, B., and Anandkumar, A. (2022). Fourier neural operator with learned deformations for pdes on general geometries. *arXiv preprint arXiv:2207.05209*.
- Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. (2020b). Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*.
- Li, Z., Zheng, H., Kovachki, N., Jin, D., Chen, H., Liu, B., Azizzadenesheli, K., and Anandkumar, A. (2021). Physics-informed neural operator for learning partial differential equations. *arXiv preprint arXiv:2111.03794*.
- Lie, H. C., Sullivan, T. J., and Teckentrup, A. L. (2018). Random forward models and log-likelihoods in bayesian inverse problems. *SIAM/ASA Journal on Uncertainty Quantification*, 6(4):1600–1629.
- Lind, S. J., Rogers, B. D., and Stansby, P. K. (2020). Review of smoothed particle hydrodynamics: towards converged lagrangian flow modelling. *Proceedings of the Royal Society A*, 476(2241):20190801.
- Lindgren, L.-E. (2009). From weighted residual methods to finite element methods.
- Liu, Q. (2017). Stein variational gradient descent as gradient flow. *Advances in neural information processing systems*, 30.
- Liu, Q. and Wang, D. (2016). Stein variational gradient descent: A general purpose bayesian inference algorithm. *Advances in neural information processing systems*, 29.
- Llorente, F., Martino, L., and Delgado, D. (2019). Parallel metropolis–hastings coupler. *IEEE Signal Processing Letters*, 26(6):953–957.
- Logg, A., Mardal, K.-A., and Wells, G. (2012). *Automated solution of differential equations by the finite element method: The FEniCS book*, volume 84. Springer Science & Business Media.
- Long, D., Wang, Z., Krishnapriyan, A., Kirby, R., Zhe, S., and Mahoney, M. (2022). AutoIP: A united framework to integrate physics into Gaussian processes. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S., editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 14210–14222. PMLR.

- Louizos, C., Shi, X., Schutte, K., and Welling, M. (2019). The functional neural process. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.
- Lu, L., Jin, P., and Karniadakis, G. E. (2019). Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*.
- Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G. E. (2021). Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229.
- Lu, L., Meng, X., Cai, S., Mao, Z., Goswami, S., Zhang, Z., and Karniadakis, G. E. (2022). A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778.
- Ma, C. and Hernández-Lobato, J. M. (2021). Functional variational inference based on stochastic process generators. *Advances in Neural Information Processing Systems*, 34:21795–21807.
- MacKay, D. J. and Mac Kay, D. J. (2003). *Information theory, inference and learning algorithms*. Cambridge university press.
- Markidis, S. (2021). The old and the new: Can physics-informed deep-learning replace traditional linear solvers? *Frontiers in big Data*, page 92.
- Markou, S., Requeima, J., Bruinsma, W. P., Vaughan, A., and Turner, R. E. (2022). Practical conditional neural processes via tractable dependent predictions. *arXiv preprint arXiv:2203.08775*.
- Martins, J. R. and Ning, A. (2021). *Engineering design optimization*. Cambridge University Press.
- Matérn, B. (2013). *Spatial variation*, volume 36. Springer Science & Business Media.
- Matsumoto, M. and Nishimura, T. (1998). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30.
- Matthews, A. G. d. G., Hensman, J., Turner, R., and Ghahramani, Z. (2016). On sparse variational methods and the kullback-leibler divergence between stochastic processes. In *Artificial Intelligence and Statistics*, pages 231–239. PMLR.
- McElreath, R. (2020). *Statistical rethinking: A Bayesian course with examples in R and Stan*. Chapman and Hall/CRC.
- Mitusch, S. K., Funke, S. W., and Dokken, J. S. (2019). dolfin-adjoint 2018.1: automated adjoints for fenics and firedrake. *Journal of Open Source Software*, 4(38):1292.
- Mohd-Yusof, J. (1997). For simulations of flow in complex geometries. *Annual research briefs*, 317:35.

- Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.
- Musha, T. and Higuchi, H. (1978). Traffic current fluctuation and the burgers equation. *Japanese journal of applied physics*, 17(5):811.
- O’Malley, D., Golden, J. K., and Vesselinov, V. V. (2019). Learning to regularize with a variational autoencoder for hydrologic inverse analysis. *arXiv preprint arXiv:1906.02401*.
- Pang, G., Yang, L., and Karniadakis, G. E. (2019). Neural-net-induced Gaussian process regression for function approximation and pde solution. *Journal of Computational Physics*, 384:270–288.
- Papoulis, A. and Unnikrishna Pillai, S. (2002). *Probability, random variables and stochastic processes*.
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.
- Pelamatti, J., Brevault, L., Balesdent, M., Talbi, E.-G., and Guerin, Y. (2018). How to deal with mixed-variable optimization problems: An overview of algorithms and formulations. In *Advances in Structural and Multidisciplinary Optimization: Proceedings of the 12th World Congress of Structural and Multidisciplinary Optimization (WCSMO12) 12*, pages 64–82. Springer.
- Petersen, K. B., Pedersen, M. S., et al. (2008). The matrix cookbook. *Technical University of Denmark*, 7(15):510.
- Plessix, R.-E. (2006). A review of the adjoint-state method for computing the gradient of a functional with geophysical applications. *Geophysical Journal International*, 167(2):495–503.
- Povala, J., Kazlauskaitė, I., Febrianto, E., Cirak, F., and Girolami, M. (2022). Variational Bayesian approximation of inverse problems using sparse precision matrices. *Computer Methods in Applied Mechanics and Engineering*, 393:114712.
- Qian, Z., Zame, W., Fleuren, L., Elbers, P., and van der Schaar, M. (2021). Integrating expert odes into neural odes: Pharmacology and disease progression. *Advances in Neural Information Processing Systems*, 34:11364–11383.
- Quarteroni, A. and Valli, A. (2008). *Numerical approximation of partial differential equations*. Springer.
- Queipo, N. V., Haftka, R. T., Shyy, W., Goel, T., Vaidyanathan, R., and Tucker, P. K. (2005). Surrogate-based analysis and optimization. *Progress in aerospace sciences*, 41(1):1–28.
- Rackauckas, C., Edelman, A., Fischer, K., Innes, M., Saba, E., Shah, V. B., and Tebbutt, W. (2021). Generalized physics-informed learning through language-wide differentiable programming.

- Rahman, S., Johnson, V. E., and Rao, S. S. (2022). Using the left gram matrix to cluster high dimensional data. *arXiv preprint arXiv:2202.08236*.
- Raissi, M. and Karniadakis, G. E. (2018). Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, 357:125–141.
- Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707.
- Ramachandran, P., Zoph, B., and Le, Q. V. (2017). Searching for activation functions. *arXiv preprint arXiv:1710.05941*.
- Rao, C., Sun, H., and Liu, Y. (2021). Physics-informed deep learning for computational elastodynamics without labeled data. *Journal of Engineering Mechanics*, 147(8):04021043.
- Rasmussen, C. E. and Williams, C. K. (2004). Gaussian processes in machine learning. *Lecture notes in computer science*, 3176:63–71.
- Raveendran, K., Wojtan, C., Thuerey, N., and Turk, G. (2014). Blending liquids. *ACM Transactions on Graphics*, 33(4).
- Reddi, S. J., Kale, S., and Kumar, S. (2019). On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*.
- Reddy, J. N. (2005). *An introduction to the finite element method*, volume 3. McGraw-Hill New York.
- Richter-Powell, J., Lipman, Y., and Chen, R. T. (2022). Neural conservation laws: A divergence-free perspective. In *Advances in Neural Information Processing Systems*.
- Ripley, B. D. (2005). *Spatial statistics*. John Wiley & Sons.
- Rixner, M. and Koutsourelakis, P.-S. (2021). A probabilistic generative model for semi-supervised training of coarse-grained surrogates and enforcing physical constraints through virtual observables. *Journal of Computational Physics*, 434:110218.
- Robert, C. P., Casella, G., and Casella, G. (1999). *Monte Carlo statistical methods*, volume 2. Springer.
- Roberts, G. O. and Rosenthal, J. S. (1998). Optimal scaling of discrete approximations to langevin diffusions. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 60(1):255–268.
- Rogers, S. and Girolami, M. (2016). *A first course in machine learning*. Chapman and Hall/CRC.
- Rossi, S., Heinonen, M., Bonilla, E., Shen, Z., and Filippone, M. (2021). Sparse gaussian processes revisited: Bayesian approaches to inducing-variable approximations. In *International Conference on Artificial Intelligence and Statistics*, pages 1837–1845. PMLR.

- Royden, H. L. and Fitzpatrick, P. (1988). *Real analysis*, volume 32. Macmillan New York.
- Rüberg, T. and Cirak, F. (2012). Subdivision-stabilised immersed b-spline finite elements for moving boundary flows. *Computer Methods in Applied Mechanics and Engineering*, 209–212:266–283.
- Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.
- Rudner, T. G., Chen, Z., Teh, Y. W., and Gal, Y. (2021). Tractable function-space variational inference in bayesian neural networks. In *Advances in Neural Information Processing Systems*.
- Russo, D. J., Van Roy, B., Kazerouni, A., Osband, I., Wen, Z., et al. (2018). A tutorial on thompson sampling. *Foundations and Trends in Machine Learning*, 11(1):1–96.
- Sacks, J., Welch, W. J., Mitchell, T. J., and Wynn, H. P. (1989). Design and analysis of computer experiments. *Statistical science*, 4(4):409–423.
- Saibaba, A. K., Lee, J., and Kitanidis, P. K. (2016). Randomized algorithms for generalized hermitian eigenvalue problems with application to computing karhunen–loève expansion. *Numerical Linear Algebra with Applications*, 23(2):314–339.
- Schilders, W. H., Van der Vorst, H. A., and Rommes, J. (2008). *Model order reduction: theory, research aspects and applications*, volume 13. Springer.
- Schillings, C. and Stuart, A. M. (2017). Analysis of the ensemble kalman filter for inverse problems. *SIAM Journal on Numerical Analysis*, 55(3):1264–1290.
- Scholkopf, B., Mika, S., Burges, C. J., Knirsch, P., Muller, K.-R., Ratsch, G., and Smola, A. J. (1999). Input space versus feature space in kernel-based methods. *IEEE transactions on neural networks*, 10(5):1000–1017.
- Seo, S. and Liu, Y. (2019). Differentiable physics-informed graph networks. *arXiv preprint arXiv:1902.02950*.
- Singh, G., Yoon, J., Son, Y., and Ahn, S. (2019). Sequential neural processes. *Advances in Neural Information Processing Systems*, 32.
- Sirignano, J. and Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364.
- Snelson, E. and Ghahramani, Z. (2005). Sparse Gaussian processes using pseudo-inputs. *Advances in neural information processing systems*, 18.
- Snelson, E. L. (2008). *Flexible and efficient Gaussian process models for machine learning*. University of London, University College London (United Kingdom).
- Snieder, R. and Trampert, J. (1999). Inverse problems in geophysics. In *Wavefield inversion*, pages 119–190. Springer.
- Sobester, A., Forrester, A., and Keane, A. (2008). *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons.

- Stein, M. L. (1999). *Interpolation of spatial data: some theory for kriging*. Springer Science & Business Media.
- Strang, G. and Freund, L. (1986). Introduction to applied mathematics. *Journal of Applied Mechanics*, 53(2):480.
- Strikwerda, J. C. (2004). *Finite difference schemes and partial differential equations*. SIAM.
- Stuart, A. and Teckentrup, A. (2018). Posterior consistency for gaussian process approximations of bayesian posterior distributions. *Mathematics of Computation*, 87(310):721–753.
- Stuart, A. M. (2010a). Inverse problems: a bayesian perspective. *Acta numerica*, 19:451–559.
- Stuart, A. M. (2010b). Inverse problems: A Bayesian perspective. *Acta Numerica*, 19:451–559.
- Sukumar, N. and Srivastava, A. (2022). Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks. *Computer Methods in Applied Mechanics and Engineering*, 389:114333.
- Sun, S., Zhang, G., Shi, J., and Grosse, R. (2019). Functional variational bayesian neural networks. *arXiv preprint arXiv:1903.05779*.
- Tait, D. J. and Damoulas, T. (2020). Variational autoencoding of PDE inverse problems. *arXiv preprint arXiv:2006.15641*.
- Takeishi, N. and Kalousis, A. (2021). Physics-integrated variational autoencoders for robust and interpretable generative modeling. *Advances in Neural Information Processing Systems*, 34:14809–14821.
- Tang, S., Feng, X., Wu, W., and Xu, H. (2023). Physics-informed neural networks combined with polynomial interpolation to solve nonlinear partial differential equations. *Computers & Mathematics with Applications*, 132:48–62.
- Tarantola, A. (2005). *Inverse problem theory and methods for model parameter estimation*. SIAM.
- Teckentrup, A. L. (2020). Convergence of gaussian process regression with estimated hyper-parameters and applications in bayesian inverse problems. *SIAM/ASA Journal on Uncertainty Quantification*, 8(4):1310–1337.
- Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294.
- Thuerey, N., Holl, P., Mueller, M., Schnell, P., Trost, F., and Um, K. (2021). *Physics-based Deep Learning*. WWW.
- Titsias, M. (2009). Variational learning of inducing variables in sparse Gaussian processes. In *Artificial intelligence and statistics*, pages 567–574. PMLR.
- Tripura, T. and Chakraborty, S. (2022). Wavelet neural operator: a neural operator for parametric partial differential equations. *arXiv preprint arXiv:2205.02191*.

- Tronarp, F., Bosch, N., and Hennig, P. (2022). Fenrir: Physics-enhanced regression for initial value problems. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pages 21776–21794. PMLR.
- Vadeboncoeur, A., Akyildiz, Ö. D., Kazlauskaitė, I., Girolami, M., and Cirak, F. (2022). Deep probabilistic models for forward and inverse problems in parametric pdes. *arXiv preprint arXiv:2208.04856*.
- Vadeboncoeur, A., Kazlauskaitė, I., Papandreou, Y., Cirak, F., Girolami, M., and Akyildiz, Ö. D. (2023). Random grid neural processes for parametric partial differential equations. *arXiv preprint arXiv:2301.11040*.
- Vogel, C. R. (2002). *Computational methods for inverse problems*. SIAM.
- Wan, E. A. and Van Der Merwe, R. (2000). The unscented kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium*, pages 153–158. IEEE.
- Wang, D., Tang, Z., Bajaj, C., and Liu, Q. (2019). Stein variational gradient descent with matrix-valued kernels. *Advances in neural information processing systems*, 32.
- Wang, S., Wang, H., and Perdikaris, P. (2021a). Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. *Science advances*, 7(40):eabi8605.
- Wang, S., Wang, H., and Perdikaris, P. (2021b). On the eigenvector bias of fourier feature networks: From regression to solving multi-scale pdes with physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 384:113938.
- Whittle, P. (1954). On stationary processes in the plane. *Biometrika*, pages 434–449.
- Wild, V. D., Hu, R., and Sejdinovic, D. (2022). Generalized variational inference in function spaces: Gaussian measures meet bayesian deep learning. *arXiv preprint arXiv:2205.06342*.
- Williams, C. and Rasmussen, C. (1995). Gaussian processes for regression. *Advances in neural information processing systems*, 8.
- Williams, C. and Seeger, M. (2000). Using the nyström method to speed up kernel machines. *Advances in neural information processing systems*, 13.
- Williams, C. K. and Rasmussen, C. E. (2006). *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA.
- Wilson, A. G., Hu, Z., Salakhutdinov, R., and Xing, E. P. (2016). Deep kernel learning. In *Artificial intelligence and statistics*, pages 370–378. PMLR.
- Winovich, N., Ramani, K., and Lin, G. (2019). ConvPDE-UQ: Convolutional neural networks with quantified uncertainty for heterogeneous elliptic partial differential equations on varied domains. *Journal of Computational Physics*, 394:263–279.
- Wriggers, P. (2008). *Nonlinear finite element methods*. Springer Science & Business Media.

- Xifara, T., Sherlock, C., Livingstone, S., Byrne, S., and Girolami, M. (2014). Langevin diffusions and the metropolis-adjusted langevin algorithm. *Statistics & Probability Letters*, 91:14–19.
- Yan, L. and Zhou, T. (2021). Stein variational gradient descent with local approximations. *Computer Methods in Applied Mechanics and Engineering*, 386:114087.
- Yang, Y. and Perdikaris, P. (2019a). Adversarial uncertainty quantification in physics-informed neural networks. *Journal of Computational Physics*, 394:136–152.
- Yang, Y. and Perdikaris, P. (2019b). Conditional deep surrogate models for stochastic, high-dimensional, and multi-fidelity systems. *Computational Mechanics*, 64(2):417–434.
- Zhang, J., Zhang, S., and Lin, G. (2022). Pagp: A physics-assisted Gaussian process framework with active learning for forward and inverse problems of partial differential equations. *arXiv preprint arXiv:2204.02583*.
- Zhang, Q., Sabin, M., and Cirak, F. (2018). Subdivision surfaces with isogeometric analysis adapted refinement weights. *Computer-Aided Design*, 102:104–114.
- Zhao, Q., Lindell, D. B., and Wetzstein, G. (2022). Learning to solve PDE-constrained inverse problems with graph networks. *arXiv preprint arXiv:2206.00711*.
- Zhong, W. and Meidani, H. (2022). PI-VAE: Physics-informed variational auto-encoder for stochastic differential equations. *arXiv preprint arXiv:2203.11363*.
- Zhu, Y., Zabaras, N., Koutsourelakis, P.-S., and Perdikaris, P. (2019). Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56–81.
- Zhuo, J., Liu, C., Shi, J., Zhu, J., Chen, N., and Zhang, B. (2018). Message passing Stein variational gradient descent. In *International Conference on Machine Learning*, pages 6018–6027. PMLR.
- Zou, F., Shen, L., Jie, Z., Zhang, W., and Liu, W. (2019). A sufficient condition for convergences of adam and rmsprop. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11127–11135.

