Journal of Computational Physics 491 (2023) 112369

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

Fully probabilistic deep models for forward and inverse problems in parametric PDEs

Arnaud Vadeboncoeur [a,*], Ömer Deniz Akyildiz [b], Ieva Kazlauskaite [a], Mark Girolami [a,c], Fehmi Cirak [a]

[a] Department of Engineering, University of Cambridge, 7a JJ Thomson Avenue, Cambridge, CB3 0FA, UK
[b] Department of Mathematics, Imperial College, Exhibition Rd, London, SW7 2AZ, UK
[c] The Alan Turing Institute, 96 Euston Rd, London, NW1 2DB, UK

ARTICLE INFO

Article history:
Received 14 December 2022
Received in revised form 5 July 2023
Accepted 10 July 2023
Available online 16 July 2023

Keywords:
Physics-informed generative models
Latent variable model
Variational inference
Neural networks
Finite elements
Weighted residuals

ABSTRACT

We introduce a physics-driven deep latent variable model (PDDLVM) to learn simultaneously parameter-to-solution (forward) and solution-to-parameter (inverse) maps of parametric partial differential equations (PDEs). Our formulation leverages conventional PDE discretization techniques, deep neural networks, probabilistic modelling, and variational inference 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 parameterized by deep neural networks. The PDE residual is assumed to be an observed random vector of value zero, hence we model it as a random vector with a zero mean and a user-prescribed covariance. The model is trained by maximizing the probability, that is the evidence or marginal likelihood, of observing a residual of zero by maximizing the evidence lower bound (ELBO). Consequently, the proposed methodology does not require any independent PDE solves and is physics-informed at training time, allowing the real-time solution of PDE forward and inverse problems after training. The proposed framework can be easily extended to seamlessly integrate observed data to solve inverse problems and to build generative models. We demonstrate the efficiency and robustness of our method on finite element discretized 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 network (PINN) discretization. We achieve up to three orders of magnitude speed-up after training compared to traditional finite element method (FEM), while outputting coherent uncertainty estimates.

© 2023 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

Partial differential equations (PDEs) are central pillars of many fields of science and engineering. They are typically nonlinear and depend on parameters which modulate their behavior, allowing them to describe a wide range of physical and operational conditions. It is thus of great interest to solve PDEs for a range of parameters to obtain numerical simulations of various scenarios. This amounts to solving the PDE for a fixed parameter and is termed *forward problem*, for which an exten-

* Corresponding author.
E-mail address: av537@cam.ac.uk (A. Vadeboncoeur).

https://doi.org/10.1016/j.jcp.2023.112369
0021-9991/© 2023 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

sive number of solution techniques are available, including finite element method (FEM), finite volumes, finite differences, and spectral methods [1,2]. These methods provide a way to move from a parameter to the solution, thus a `parameter-to-solution` map. 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* [3]. Recent growing interest in inverse problems is driven by the increased availability of sensor data in science and engineering and advances in machine learning (ML) in exploring high-dimensional spaces. Inverse problems are typically more difficult than forward problems due to under-specification, meaning that small errors in observations may lead to large errors in the model parameter, and the parameters explaining the observations may not be unique. This is particularly true in the presence of observation noise and missing physics [4], and high-dimensional parameter spaces [5,6].

More specifically, 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 allows the practitioners to explore a variety of designs and experimental setups efficiently. For example, given a specific modeling 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 obtain a product with desired properties (*e.g.* a structure that does not buckle under a given load). An efficient forward solver allows exploring numerous designs by varying the parameters; 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.

1.1. Contributions

Our goal in this paper is to solve simultaneously the forward and inverse problems 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 modeling parametric PDEs in the forward direction through surrogates because we want to understand the distribution of model confidence between solve instances and within solution domains. UQ in the approximating posterior distributions in inverse problems is also of central importance due to the ill-posedness of many problems [4]. Most importantly, the Bayesian view in modeling PDEs gives us a principled framework to combine machine learning-based methods with classical numerical schemes in a coherent and reasoned manner and has allowed us to construct a model whereby we can train surrogates to jointly learn a variety of probabilistic maps. To achieve this, we develop a probabilistic formulation called physics-driven deep latent variable models (PDDLVM) and uniquely blend techniques from numerical solutions of PDEs, deep learning, and probabilistic ML. Notably, the training of the 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 summarize, we propose a method that (i) simultaneously learns deep probabilistic `parameter-to-solution` (forward) and `solution-to-parameter`/`observation-to-parameter` (inverse) maps, (ii) is free from PDE forward numerical solve operations, (iii) uses the Bayesian view [4] of regularizing the inverse problem to address their inherent ill-posedness while calibrating model uncertainty in the forward direction, and (iv) allows for the incorporation of various neural networks (NNs) tailored to parametric PDEs (such as physics-informed neural networks (PINN)s [7], neural operators [8–11]) 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.

1.2. Related work

The challenges associated with forward and inverse parametric PDE problems led to a proliferation of ML-based techniques that aim to aid the forward simulation of physical processes given some parameters and the estimation of parameters that cannot be directly observed (see, *e.g.*, [12–14]). Although the early approaches were primarily based on training ML methods, such as random forests and deep NNs, on observed or simulated physical data [15] and discovering governing physics laws from the data [16], 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 parameterizations of physical processes with observed/simulated data [17], or through differentiable physics [7,18,19]. The exceptional success of these methods brought tremendous attention to the emerging field of *physics-informed machine learning* [14,20]. We summarize some of these advances and their relevance to our work below.

Physics-informed neural networks (PINN) and related methods. PINN-based methods [7,14,21–24] convert the PDE formulation into a *loss function* with either hard or soft constraints [25,26]. Most of these methods are based on point-wise evaluation of losses and may struggle to handle complex geometries; see, *e.g.*, [27], for treatment of complex meshes. As loss-based models, PINNs are not inherently probabilistic, unlike the method we develop in the current paper.

Learning Forward and Inverse PDE maps. An alternative approach is to learn mappings between parameters/initial/boundary conditions to solutions, typically requiring parameter-to-solution (*i.e.* input-output) pairs in a supervised setting [9,10,28–32] which may be unavailable or costly. Further extensions consider the unsupervised case [33,34], and couple the model with FEM [35].

Probabilistic approaches. Probabilistic formulations of PDE-informed models vary from supervised approaches [36] and mixed semi-supervised formulations [37], to uncertainty quantification procedures based on convolutional neural networks (CNN) [38] to Gaussian processes combined with FEM [39–42]. The most relevant to our work are [43,44] where the authors introduce the weighted residual method (which is a generalization of many classical numerical methods for PDEs [45–50]), and treat the residual term as a pseudo-observation [44]. This approach, however, differs from our framework in that we learn the direct mappings between the parameters and the solutions, while [44] introduces an intermediate low-dimensional embedding that is not readily interpretable. Other related approaches consider flow-based probabilistic surrogates [51]. There are also techniques that use normalizing flows for solving stochastic differential equations in the inverse and forward direction [52]. Another line of work integrates physical and deterministic models within variational autoencoders (VAEs). Some of these models focus on the construction of efficient regularized inverse maps [53,54], while others construct additional latent spaces and relate them to physical observations [55,56]. Related to this line of work, [57] learn disentangled latent representations of physical processes. Among these, the most relevant to our work is the physics-integrated VAE [58], which introduces physics knowledge into a VAE framework. However, our approach is markedly different, as we approximate both forward and inverse maps for parametric PDEs and propose a different variational formulation that constructs a latent space directly characterizing both the solution and the parameters of the physical models.

2. Parametric PDEs and their discretization

We are interested in PDEs defined on a domain Ω with boundary $\partial\Omega$, generalized as

$$\mathcal{G}_z(u,x) = f(x), \qquad \text{for} \qquad x \in \Omega \subset \mathbb{R}^d, \tag{1a}$$

$$\mathcal{B}_z(u,x) = 0, \qquad \text{for} \qquad x \in \partial\Omega, \tag{1b}$$

where $d \in \{1,2,3\}$, $\mathcal{G}_z$ and $\mathcal{B}_z$ are (possibly) nonlinear operators that describe the PDE and its boundary conditions, respectively, $u(x)$ 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 scalar like the wave speed. In the following, we first describe a general way to derive different numerical solution schemes and then introduce FEM and PINNs.

2.1. Weighted residual method

The method of weighted residuals provides a convenient framework for the derivation of many well-known PDE discretization techniques [45,48–50]. Given a PDE of the form (1), we first define the domain residual

$$\mathcal{R}(u,z,f,x) := \mathcal{G}_z(u,x) - f(x). \tag{2}$$

For the sake of brevity, in the following the boundary term (1b) is omitted, *i.e.* it is assumed to be satisfied. We then, multiplying by some arbitrary weight function $w(x)$ and integrating over the domain, obtain a weighted residual functional,

$$\int_\Omega w(x)\left(\mathcal{G}_z(u,x) - f(x)\right)\mathrm{d}x = \int_\Omega w(x)\mathcal{R}(u,z,f,x)\mathrm{d}x. \tag{3}$$

Next we choose a set of test functions $w_{1:n}(x)$ against which we will be integrating the residual. We further pose that $\hat{u}(x)$ is the *approximant* of the solution field $u(x)$. The form of the relationship between the function $\hat{u}(x)$ that is defined everywhere on Ω and the vector $\mathbf{u}$ describing $\hat{u}(x)$ depends on the approximation technique chosen. Similarly, we pose that the PDE parameter field $z(x)$ and forcing field $f(x)$ are represented by the approximants $\hat{z}(x)$ and $\hat{f}(x)$. We then assemble the resulting discrete system of equations into a vectorized form

$$\mathbf{A}_\mathbf{z}(\mathbf{u}) = \left[\int_\Omega w_i(x)\mathcal{G}_{\hat{z}}(\hat{u},x)\mathrm{d}x\right]_{i=1:n}, \tag{4a}$$

$$\mathbf{f}^{\mathrm{WR}} = \left[\int_\Omega w_i(x)\hat{f}(x)\mathrm{d}x\right]_{i=1:n}. \tag{4b}$$

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

$$\mathbf{r} := \mathbf{A}_{\mathbf{z}}(\mathbf{u}) - \mathbf{f}^{\text{WR}}, \tag{5}$$

with $\mathbf{r}, \mathbf{u}, \mathbf{f}^{\text{WR}} \in \mathbb{R}^n$. This residual description is agnostic to the particular method of discretization used in practice.

2.2. Finite elements and parameterizations

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

$$\mathbf{u} \underset{\boldsymbol{\pi}_{\text{CH}}}{\mapsto} \mathbf{u}^{\text{FE}} \underset{\boldsymbol{\pi}_{\text{FE}}}{\mapsto} \hat{u}(x), \tag{6}$$

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

$$\mathbf{u}^{\text{FE}} = \boldsymbol{\pi}_{\text{CH}}(\mathbf{u}, x_{1:n}) = \left[\sum_i T_i(x_j) u_i \right]_{j=1:n}, \tag{7}$$

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

$$\hat{u}(x) = \boldsymbol{\pi}_{\text{FE}}(\mathbf{u}^{\text{FE}}, x) = \sum_i \phi_i(x) u_i^{\text{FE}}, \tag{8}$$

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 (3) prior to finite element discretization [59].

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

$$\mathbf{z} \underset{\boldsymbol{\pi}_{\text{CH}}}{\mapsto} \hat{z}(x), \tag{9}$$

which is chosen to be a spectral parameterization as

$$\hat{z}(x) = \boldsymbol{\pi}_{\text{CH}}(\mathbf{z}, x) = \sum_i T_i(x) z_i. \tag{10}$$

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

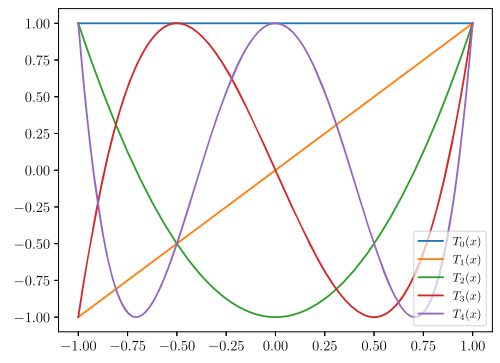
$$\mathbf{f} \underset{\boldsymbol{\pi}_{\text{CH}}}{\mapsto} \mathbf{f}^{\text{FE}} \underset{\boldsymbol{\pi}_{\text{FE}}}{\mapsto} \hat{f}(x), \tag{11}$$

each mapping is then given as

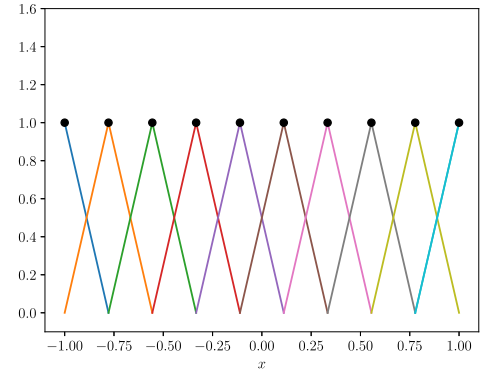
$$\mathbf{f}^{\text{FE}} = \boldsymbol{\pi}_{\text{CH}}(\mathbf{f}, x_{1:n}) = \left[\sum_i T_i(x_j) f_i \right]_{j=1:n}, \tag{12a}$$

$$\hat{f}(x) = \boldsymbol{\pi}_{\text{FE}}(\mathbf{f}^{\text{FE}}, x) = \sum_i \phi_i(x) f_i^{\text{FE}}, \tag{12b}$$

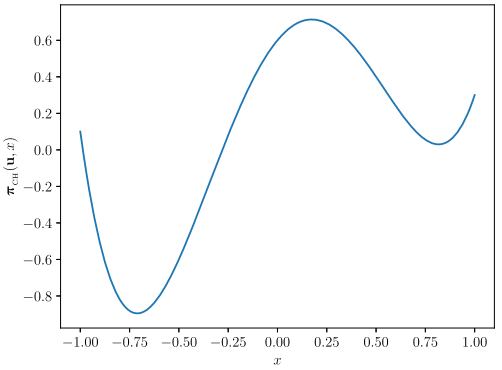
where we use (4b) to map $\mathbf{f}^{\text{FE}}$ to $\mathbf{f}^{\text{WR}}$ through $\hat{f}(x)$ to be used in (5). These various parameterizations allow us to represent solution fields, parameter fields, and forcing fields in a FEM mesh invariant manner. We show in Fig. 1 a representation of 5 spectral 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.



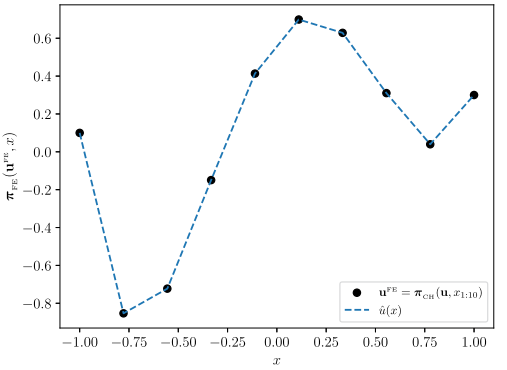
(a) Spectral Chebyshev basis functions $T_i(x)$



(b) Local FEM linear basis functions $\phi_i(x)$



(c) Spectral Chebyshev function $\sum_i T_i(x)u_i$



(d) 10 node FEM linear interpolation

Fig. 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.

2.3. Physics informed neural networks (PINNs)

To construct a physics informed neural network the trial solution $\hat{u}(x)$ is chosen to be the output of a neural network and the spatial locations x are the input. To evaluate the residual (2), we take the relevant gradients of the trial solution on a grid of (collocation) points and construct the discretized differential equation according to (5) and (4). More specifically, we define a mapping $\boldsymbol{\pi}_{\text{NN}}$ between the approximant and the discrete representation of the solution field as

$$\mathbf{u} \underset{\boldsymbol{\pi}_{\text{NN}}}{\longleftarrow} \hat{u}(x_{1:n}). \tag{13}$$

More precisely, we have $\mathbf{u} = \boldsymbol{\pi}_{\text{NN}}(\hat{u}, x_{1:n})$ where

$$\boldsymbol{\pi}_{\text{NN}}(\hat{u}, x_{1:n}) = [\hat{u}(x_1), \ldots, \hat{u}(x_n)],$$

with $x_{1:n}$ is a collection of fixed equidistant collocation points, and

$$\hat{u}(x) = (\ell_l \circ \ldots \circ \ell_i \circ \ldots \circ \ell_0)(x). \tag{14}$$

Here, $\ell_i(x) = \boldsymbol{\sigma}(\mathbf{W}_i x + \mathbf{b}_i)$ and $\mathbf{W}$ is a matrix of learnable weights, $\mathbf{b}$ is a vector of learnable biases and $\boldsymbol{\sigma}$ is a chosen activation function. In this work we only look at PINN implementations with fixed grids. Others works such as [60] look at random grid methods, but this is out of the scope of this current work. The framework could be extended to incorporate repeatedly sampled random grids.

In this work we are interested in PINNs for parametric PDEs. For this we pass PDE parameter $\mathbf{z}$ and forcing term $\mathbf{f}$ as inputs along with the spatial evaluation coordinate. To enforce boundary conditions in this framework a second residual is constructed involving the boundary term (1b) and added in a weighted manner to the PDE residual. In our framework, we keep the boundary residual in vectorized form and append it to the domain residual.

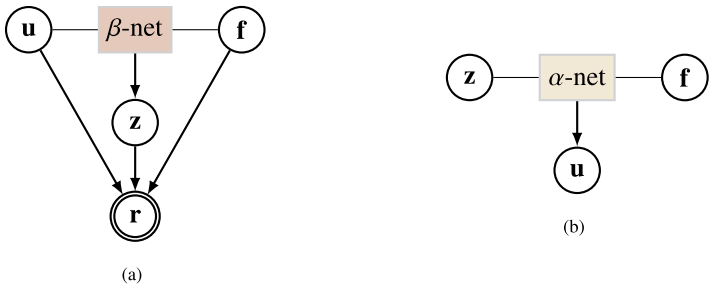


Fig. 2. (a) The probabilistic graphical model with the *inverse* (β) neural network, where the double rings indicate that $\mathbf{r}$ is an observed vector. (b) The probabilistic graphical model with the *forward* (α) neural network.

3. Deep probabilistic models for parametric PDEs

FEM and PINNs allow to solve the well-posed 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 problem of finding $\mathbf{z}$ given some components of $\mathbf{u}$ (the inverse problem) 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 $\mathbf{f}$. We first describe, in Sec. 3.1, 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. Building on this, we introduce in Sec. 3.2 a version of our framework which incorporates observation data.

3.1. Probabilistic model and variational approximation without observed data

Following the weighted residual approach we consider the residual $\mathbf{r} \in \mathbb{R}^n$ in (5) as an auxiliary random variable that is observed and has the value $\mathbf{r} = \mathbf{0}$. We pose the joint distribution over all variables as

$$p_\beta(\mathbf{r}, \mathbf{u}, \mathbf{z}, \mathbf{f}) = p(\mathbf{r}|\mathbf{u}, \mathbf{z}, \mathbf{f}) p_\beta(\mathbf{z}|\mathbf{u}, \mathbf{f}) p(\mathbf{u}) p(\mathbf{f}), \tag{15}$$

where β is the trainable set of parameters in our model, see Fig. 2a. Such a factorization is a modelling choice; the joint distribution could in fact be factorized in different ways so long as the rules of probability are respected. The posed factorization was chosen to obtain a conditional distribution $p(\mathbf{r}|\mathbf{u}, \mathbf{z}, \mathbf{f})$ of the residual $\mathbf{r}$ as well as a conditional distribution $p_\beta(\mathbf{z}|\mathbf{u}, \mathbf{f})$ 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 [61].

We pose the likelihood of the residual as

$$p(\mathbf{r}|\mathbf{u}, \mathbf{z}, \mathbf{f}) = \mathcal{N}(\mathbf{r}; \mathbf{A}_\mathbf{z}(\mathbf{u}) - \mathbf{f}^{\text{WR}}, \boldsymbol{\Sigma}_\mathbf{r}), \tag{16}$$

where $\boldsymbol{\Sigma}_\mathbf{r}$ 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. 2.1, we aim to maximize its marginal likelihood $p(\mathbf{r} = \mathbf{0})$ by integrating out all other variables. We next choose a deep probabilistic model of the form $p_\beta(\mathbf{z}|\mathbf{u}, \mathbf{f}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_\beta(\mathbf{u}, \mathbf{f}), \boldsymbol{\Sigma}_\beta(\mathbf{u}, \mathbf{f}))$, where $\boldsymbol{\mu}_\beta$ and $\boldsymbol{\Sigma}_\beta$ 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 us the flexibility to obtain $\mathbf{z}$ given a solution field $\mathbf{u}$ and forcing $\mathbf{f}$. To this end, we pose the factorization

$$q_\alpha(\mathbf{u}, \mathbf{z}, \mathbf{f}) = q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{f}) p(\mathbf{z}) p(\mathbf{f}), \tag{17}$$

with the approximation $q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{f}) = \mathcal{N}(\mathbf{u}; \boldsymbol{\mu}_\alpha(\mathbf{z}, \mathbf{f}), \boldsymbol{\Sigma}_\alpha(\mathbf{z}, \mathbf{f}))$, where $\boldsymbol{\mu}_\alpha$ and $\boldsymbol{\Sigma}_\alpha$ are represented by neural networks with the set of parameters α consisting of weights and biases, see Fig. 2b. Inspired by the weighted residual approach introduced in Sec. 2.1 (see also [44]), we maximize the marginal likelihood of the residual by fixing $\mathbf{r} = \mathbf{0}$ and deriving an evidence lower bound (ELBO) that lower bounds the log marginal likelihood $\log p(\mathbf{r} = \mathbf{0})$. In order to do this, we first note

$$p(\mathbf{r} = \mathbf{0}) = \int p_\beta(\mathbf{r} = \mathbf{0}, \mathbf{u}, \mathbf{z}, \mathbf{f}) \mathrm{d}\mathbf{u}\mathrm{d}\mathbf{z}\mathrm{d}\mathbf{f}, \tag{18}$$

where the $p(\mathbf{r}=\mathbf{0})$ acts as *evidence*. Our aim is to maximize 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 (18) as

$$p(\mathbf{r}=\mathbf{0}) = \int \frac{p_\beta(\mathbf{r}=\mathbf{0},\mathbf{u},\mathbf{z},\mathbf{f})}{q_\alpha(\mathbf{u},\mathbf{z},\mathbf{f})} q_\alpha(\mathbf{u},\mathbf{z},\mathbf{f})\mathrm{d}\mathbf{u}\mathrm{d}\mathbf{z}\mathrm{d}\mathbf{f}, \tag{19}$$

by dividing and multiplying the integrand in (18) by $q_\alpha(\mathbf{u},\mathbf{z},\mathbf{f})$. Next, we compute the logarithm of this quantity, since maximizing $\log p(\mathbf{r}=\mathbf{0})$ is equivalent to maximizing $p(\mathbf{r}=\mathbf{0})$ and is more numerically stable. Using Jensen's inequality we can obtain an expression of the form

$$\log p(\mathbf{r}=\mathbf{0}) \geq \mathcal{F}(\alpha,\beta), \tag{20}$$

and using (15) and (17), we arrive at the lower bound

$$\mathcal{F}(\alpha,\beta) = \int \log \frac{p(\mathbf{r}=\mathbf{0}|\mathbf{u},\mathbf{z},\mathbf{f})p_\beta(\mathbf{z}|\mathbf{u},\mathbf{f})p(\mathbf{u})}{q_\alpha(\mathbf{u}|\mathbf{z},\mathbf{f})p(\mathbf{z})} q_\alpha(\mathbf{u}|\mathbf{z},\mathbf{f})p(\mathbf{z})p(\mathbf{f})\mathrm{d}\mathbf{u}\mathrm{d}\mathbf{z}\mathrm{d}\mathbf{f}. \tag{21}$$

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 [62]. A full derivation of our PDDLVM framework can be found in Appendix B where we show that our framework minimizes 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 then seek

$$\{\alpha^\star,\beta^\star\} \in \underset{\beta,\alpha}{\operatorname{argmax}}\, \mathcal{F}(\alpha,\beta). \tag{22}$$

We note that the α and β networks (which parameterize $q_\alpha(\mathbf{u}|\mathbf{z},\mathbf{f})$ and $p_\beta(\mathbf{z}|\mathbf{u},\mathbf{f})$, respectively) have intuitive functionalities (see Fig. 2 for the graphical models). In particular, the α network provides a `parameter-to-solution` (forward) map, while the β network provides a `solution-to-parameter` (inverse) map, both with associated uncertainty estimates. Modern variational inference techniques rely on maximizing the lower bound $\mathcal{F}(\alpha,\beta)$, instead of the true evidence $\log p(\mathbf{r}=\mathbf{0})$ to learn the parameters (α,β). To compute (21) 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 reparameterization trick [63]). For this, we rewrite ELBO (21) as a generic integral

$$\mathcal{F}(\alpha,\beta) = \mathbb{E}_{q_\alpha(\mathbf{u}|\mathbf{z},\mathbf{f})p(\mathbf{z})p(\mathbf{f})}[\mathsf{v}_{\alpha,\beta}(\mathbf{u},\mathbf{z},\mathbf{f})], \tag{23}$$

where

$$\mathsf{v}_{\alpha,\beta}(\mathbf{u},\mathbf{z},\mathbf{f}) = \log \frac{p(\mathbf{r}=\mathbf{0}|\mathbf{u},\mathbf{z},\mathbf{f})p_\beta(\mathbf{z}|\mathbf{u},\mathbf{f})p(\mathbf{u})}{q_\alpha(\mathbf{u}|\mathbf{z},\mathbf{f})p(\mathbf{z})}. \tag{24}$$

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

$$\mathsf{w}_\alpha(\mathbf{z},\mathbf{f},\boldsymbol{\varepsilon}) = \boldsymbol{\mu}_\alpha(\mathbf{z},\mathbf{f}) + \boldsymbol{\Sigma}_\alpha^{1/2}(\mathbf{z},\mathbf{f})\boldsymbol{\varepsilon}, \tag{25}$$

and write

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

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

$$\mathcal{F}^N(\alpha,\beta) = \frac{1}{N}\sum_{j=1}^{N} \mathsf{v}_{\alpha,\beta}(\mathsf{w}_\alpha(\mathbf{z}^{(j)},\mathbf{f}^{(j)},\boldsymbol{\varepsilon}^{(j)}),\mathbf{z}^{(j)},\mathbf{f}^{(j)}). \tag{26}$$

The rest of the training is done by computing the gradient of this stochastic loss w.r.t. α,β and running a variant of gradient descent, such as Adam [64]. In many practical applications (and in this paper), $N=1$, *i.e.*, a one-sample estimate of the gradient is utilized [62]. 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.

3.2. Probabilistic model and variational approximation with observed data

We now introduce observations into our model to cover data-centric scenarios, as well as to handle multiple forcing terms. 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. 3.1. We denote the learned β network parameter by setting $\beta = \beta_\star$. For a given observation set $\{\mathbf{y}_i\}_{i=1}^m$, where $\mathbf{y}_i \in \mathbb{R}^{n_y}$, and a corresponding solution set $\{\mathbf{u}_i\}_{i=1}^m$, we introduce the observation likelihood

$$p(\mathbf{y}_i|\mathbf{u}_i) = \mathcal{N}(\mathbf{y}_i; \mathbf{g}(\mathbf{u}_i), \boldsymbol{\Sigma}_\mathbf{y}), \qquad \text{for} \quad i = 1, \dots, m, \tag{27}$$

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

$$p(\mathbf{y}_i, \mathbf{u}_i) = p(\mathbf{y}_i|\mathbf{u}_i)p(\mathbf{u}_i). \tag{28}$$

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

$$q_\phi(\mathbf{u}_i|\mathbf{y}_i) = \mathcal{N}(\mathbf{u}_i; \boldsymbol{\mu}_\phi(\mathbf{y}_i), \boldsymbol{\Sigma}_\phi(\mathbf{y}_i)). \tag{29}$$

The marginal likelihood of the data is given by

$$p(\mathbf{y}_i) = \int p(\mathbf{y}_i, \mathbf{u}_i)\mathrm{d}\mathbf{u}_i. \tag{30}$$

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

$$\log p(\mathbf{y}_i) = \log \int \frac{p(\mathbf{y}_i, \mathbf{u}_i)}{q_\phi(\mathbf{u}_i|\mathbf{y}_i)} q_\phi(\mathbf{u}_i|\mathbf{y}_i)\mathrm{d}\mathbf{u}_i. \tag{31}$$

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

$$\mathcal{F}_i(\phi) = \int \log \frac{p(\mathbf{y}_i|\mathbf{u}_i)p(\mathbf{u}_i)}{q_\phi(\mathbf{u}_i|\mathbf{y}_i)} q_\phi(\mathbf{u}_i|\mathbf{y}_i)\mathrm{d}\mathbf{u}_i. \tag{32}$$

Given the dataset $\{\mathbf{y}_i\}_{i=1}^m$, the full ELBO is $\mathcal{F}(\phi) = \sum_{i=1}^m \mathcal{F}_i(\phi)$. Once trained, the variational distribution $q_\phi(\mathbf{u}_i|\mathbf{y}_i)$ is sufficient to obtain an encoding model for observations. The lower bound can be computed in a similar manner to the ELBO in Sec. 3.1 with

$$\mathcal{F}_i(\phi) = \mathbb{E}_{q_\phi(\mathbf{u}_i|\mathbf{y}_i)}[\mathrm{v}_\phi(\mathbf{y}_i, \mathbf{u}_i)], \tag{33}$$

where

$$\mathrm{v}_\phi(\mathbf{y}_i, \mathbf{u}_i) = \log \frac{p(\mathbf{y}_i|\mathbf{u}_i)p(\mathbf{u}_i)}{q_\phi(\mathbf{u}_i|\mathbf{y}_i)} q_\phi(\mathbf{u}_i|\mathbf{y}_i), \tag{34}$$

which can also be computed through the reparametrization trick detailed in (25) as

$$\mathrm{w}_\phi(\mathbf{y}_i, \boldsymbol{\varepsilon}) = \boldsymbol{\mu}_\phi(\mathbf{y}_i) + \boldsymbol{\Sigma}_\phi^{1/2}(\mathbf{y}_i)\boldsymbol{\varepsilon}, \tag{35}$$

and write

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

This can be approximated with Monte-Carlo sampling as done in (26).

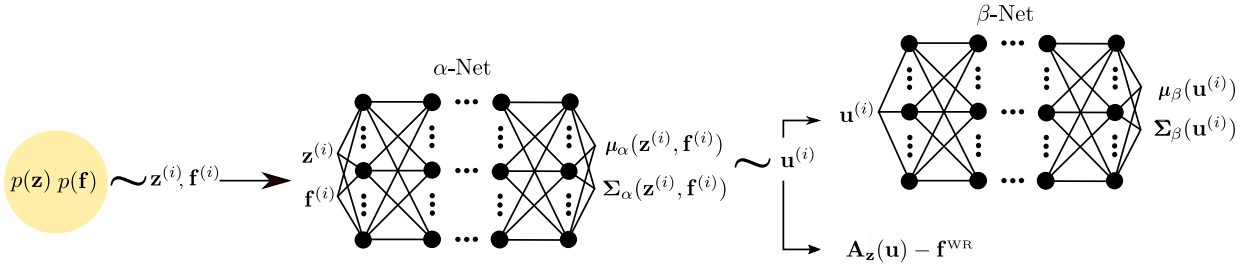
To realize the `observation-to-parameter` map, we can then use the already learned $\beta_\star$-network to marginalize over $\mathbf{u}$ as

$$p(\mathbf{z}|\mathbf{y}, \mathbf{f}) = \int q_{\phi_\star}(\mathbf{u}|\mathbf{y}) p_{\beta_\star}(\mathbf{z}|\mathbf{u}, \mathbf{f})\mathrm{d}\mathbf{u}. \tag{36}$$

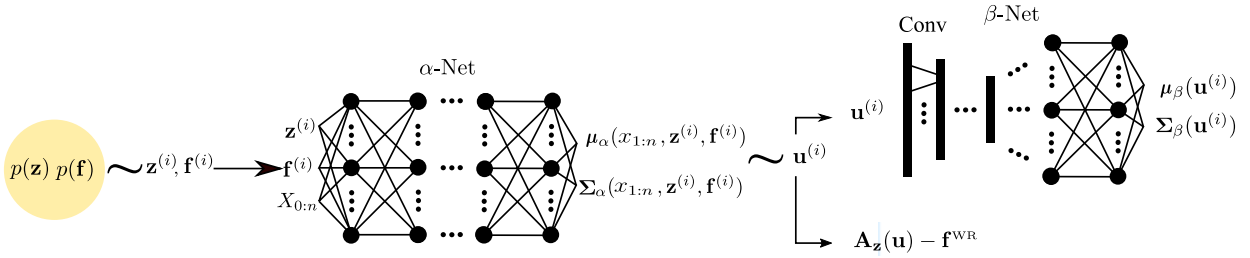
The marginalization of $\mathbf{z}$ over $\mathbf{u}$ can also be easily extended to include the forcing $\mathbf{f}$. We stress that the method we propose only uses data when we are inverting some observable map from a dataset, this extension does not learn the physics of a problem but rather some transformation that is over the underlying physics. The physics are learned using the algorithm described in Sec. 3.1 and this probabilistic physics model can then be used for combined inference as shown in (36).

Algorithm 1 Pseudocode for PDDLVM.

Initialize: α_0, β_0, T (number of iterations), N (number of Monte Carlo samples).
for $t = 1, \dots, T$ **do** ▷ Gradient descent.
  for $i = 1, \dots, N$ **do** ▷ Monte Carlo estimate of the loss (21).
    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 the samples.
  $(\alpha_t, \beta_t) \leftarrow$ OPTIMISER$(\alpha_{t-1}, \beta_{t-1}, \mathcal{F}^N(\alpha, \beta))$ ▷ Update parameters.



(a) Architecture for FEM-PDDLVM



(b) Architecture for PINN-PDDLVM

Fig. 3. The architecture of both the FEM-PDDLVM and the PINN-PDDLVM. The large "∼" 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 reparametrize the samples $\mathbf{u}^{(i)}$ with a global expansion which in the shown examples are Chebyshev expansions of various orders. For the PINN-PDDLVM version we instead evaluate the network at the 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 trial solution evaluated on the grid, it can in essence be interpreted as an image.

3.3. 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). We estimate the ELBOs in (21) using sampling from $\mathbf{z} \sim p(\mathbf{z})$ and $\mathbf{f} \sim p(\mathbf{f})$, $\mathbf{u} \sim q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{f})$ (using the reparameterization trick [63])) and optimize with Adam [64]. When multiple observations exist, *i.e.* if $m \gg 1$, we use only one-sample estimate of ELBO, resulting in a doubly-stochastic gradient. See Algorithm 1 for the full algorithm. We also provide details in Sec. 4 where necessary. In Fig. 3 we show diagrams depicting the flow of samples in the algorithm. With all the quantities computed through the given samples we can then evaluate all the relevant terms in the ELBO.

4. Examples

We demonstrate our methodology on six selected examples.[1] In all experiments, we choose a diagonal covariance for the networks, although other choices can be explored [62]. Furthermore, we make use of a bounded re-parameterization of the log variance of the neural networks [65]. We note that our approach is extrusive in the sense that it needs only the solution and its gradient but is agnostic to the inner workings of the FEM package. The examples in Sec. 4.1, Sec. 4.2, and Sec. 4.3 all use a FEM-PDDLVM residual formulation and the output of the neural network is parameterized with a Chebyshev expansion. The further sections are based on the PINN formulation of the residual. A summary of relevant information for each experiment can be found in Appendix E.

[1] The code is made available at https://github.com/ArnaudVadeboncoeur/PDDLVM.

4.1. 1D linear Poisson

We use the formulation in Sec. 3.1 to train the α and β networks on a linear Poisson equation with Dirichlet boundary conditions

$$\begin{aligned} -\nabla \cdot (\kappa(x)\nabla u(x)) &= f(x), \quad x \in (-1, 1), \\ u(-1) &= a, \quad u(1) = b, \end{aligned} \tag{37}$$

we learn the `parameter-to-solution` map jointly with the `solution-to-parameter` map with uncertainties.

When using (10) to represent a physical quantity which is strictly positive we compose the spectral mapping $\boldsymbol{\pi}_{\text{CH}}$ with a Softplus transformation of the form $\log(1 + \exp(x))$. As the transformation is nonlinear, we make use of the unscented transform [66] 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 Vandermond 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.

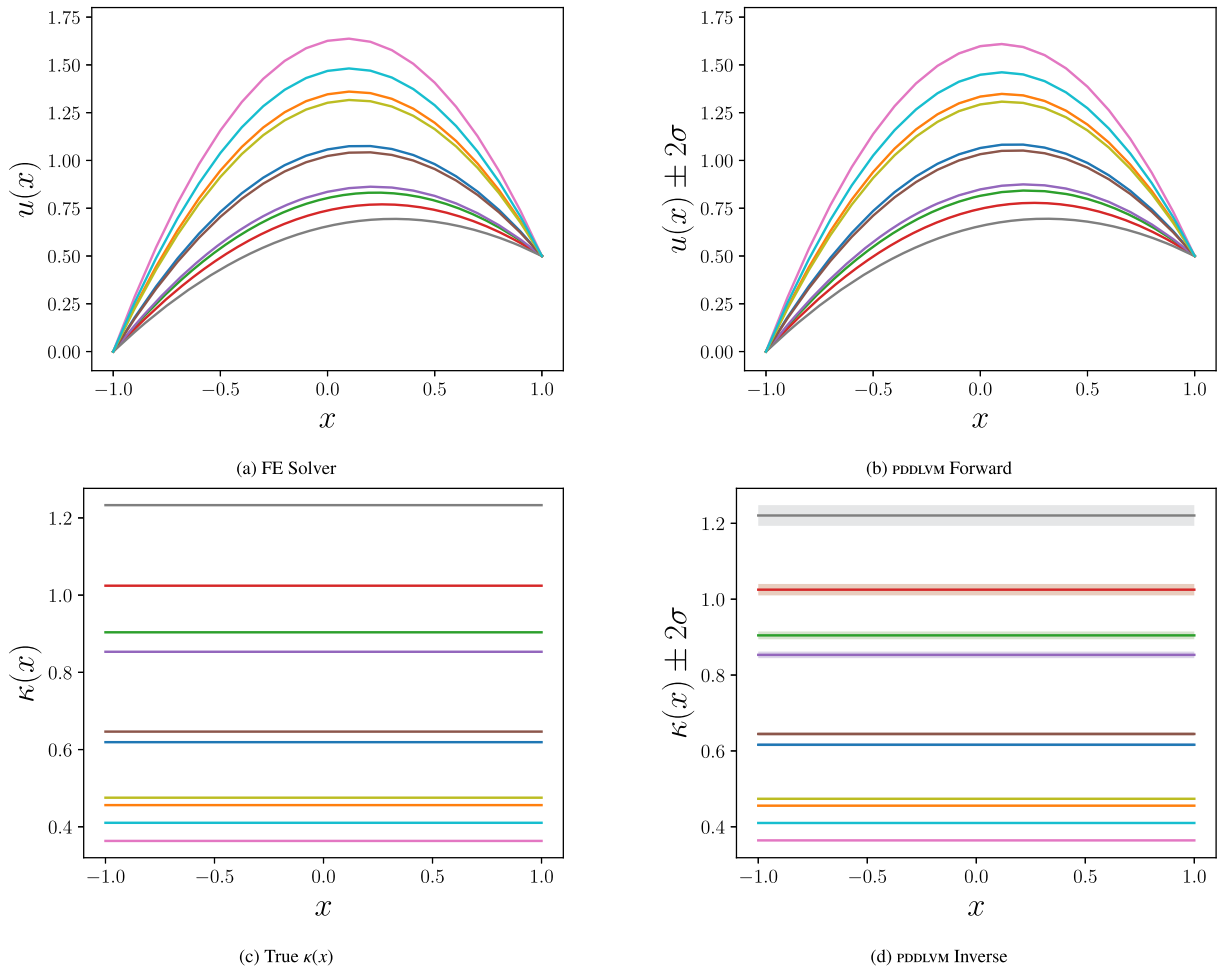
4.1.1. Learning without observations

In this section we demonstrate the use of our method on the linear Poisson problem without observations as in Sec. 3.1. To measure the accuracy of our method after training, we compute the FEM ground truth solution, however, these are *not* used during training. The residual is marginalized over the prior of $\mathbf{z}$ and $\mathbf{f}$ which allow us to learn the PDE over the prior ranges. In this example, our interpretable latent variable $\mathbf{z} = \kappa$ denotes the coefficients of the $\kappa(x)$ expansion. To ensure positivity of the diffusion field, we pass $\kappa(x)$ through a Softplus function of the form $\log(1 + \exp(\kappa(x)))$. In Fig. 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 parameterized as a constant function and the solution field is given by a 4^{th} order Chebyshev expansion. The prior over κ is $\mathcal{N}(\kappa; 0, 1)$. We use a NN architecture of 50 nodes for 3 hidden layers. We use a "swish" activation function and 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. 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.

Using this model, we study the effect of varying the $\epsilon_{\mathbf{r}}$ value of the residual covariance, i.e. $\boldsymbol{\Sigma}_{\mathbf{r}} = \epsilon_{\mathbf{r}}^2 \boldsymbol{I}$, controlling the solution accuracy. In Fig. 5 we plot the log Mean Normalized Squared Error (MNSE) (A.1) for 100 samples of solution fields, diffusion field, and boundary conditions for a variety of κ from the prior. Similarly, in the Appendix Fig. C.13 we plot the mean normalized squared error for 100 independent draws of κ from various ranges of values. The error for the proposed solution field is computed with respect to a true FEM solution. 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 $\epsilon_{\mathbf{r}}$, 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. C.14, changes in the value for $\epsilon_{\mathbf{r}}$ tempers the optimization surface which will affect the training of the algorithm. We can conclude from this that there is no single correct setting of $\epsilon_{\mathbf{r}}$ but in effect is a practitioner's choice which will impact the learning process.

4.1.2. Observable map inversion

We make use of Sec. 3.2 to learn to invert a noisy mapping from the measurements to the solutions in the presence of observations. Here the governing equation is (37) 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. We use a set of 100 observations with a noise covariance of $\boldsymbol{\Sigma}_{\mathbf{y}} = \sigma_y^2 \mathbf{I}$, where $\sigma_y = 0.01$. The $\alpha_\star$ and $\beta_\star$ networks are pretrained on the linear Poisson equation with $\kappa(x)$ expressed as a 3^{rd} order Chebyshev expansion, variable boundary conditions, and variable constant function forcing. The model q_ϕ 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. 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 FE grid is of 5.64×10^{-4} and for $\mathbf{z}$ (the $\kappa(x)$ field) projected on the FE mesh the MNSE is of 8.35×10^{-3}.



(a) FE Solver

(b) PDDLVM Forward

(c) True $\kappa(x)$

(d) PDDLVM Inverse

Fig. 4. Results for the 1D linear Poisson equation. (a) The true solutions given by a linear FEM solver 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).

Table 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%

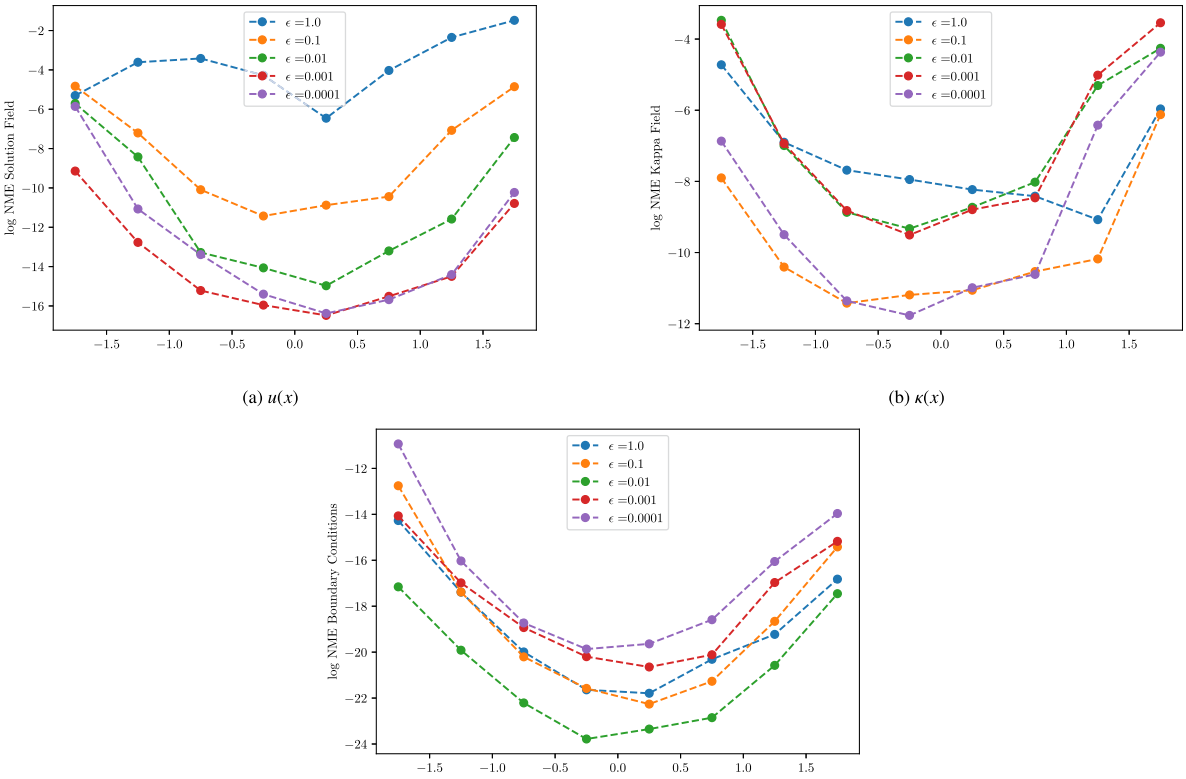
4.2. 1D nonlinear Poisson problem

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

$$-\nabla \cdot (\eta(u,x)\nabla u(x)) = f(x), \tag{38}$$
$$\eta(u,x) = \left(S\left|\frac{\mathrm{d}u}{\mathrm{d}x}\right| + 5\kappa(x)\right)/10,$$

for $x \in (-1,1)$. 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 parameterize $\kappa(x)$ as a 4^{th} order Chebyshev expansion. The solution field is expressed as a 9^{th} 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 0^{th} Chebysehv polynomial. The NN



(a) $u(x)$

(b) $\kappa(x)$

(c) Boundary conditions

Fig. 5. 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 center of this narrow testing distribution, 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 $\epsilon_{\mathbf{r}}$. 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 $p(\kappa)$.

architecture for this problem is a fully connected 4 layers deep network with 100 neurons per hidden layer. The FE model is discretized 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 (Table 1). In Fig. 7, the predictive distributions as well as the solutions are plotted.

4.3. Thin-walled flexible shell

In this example, we consider the deformation of the Stanford bunny with an inhomogeneous Young's modulus subjected to self-weight, see Fig. 8. We analyze this model with an isogeometrically discretized Kirchoff-Love shell finite element scheme [67–69]. 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 $f(x), u(x) : \mathbb{R}^3 \to \mathbb{R}^3$ 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 Kirchoff-Love model we require basis functions with square integrable second derivatives which are obtained from Catmull-Clark subdivision surfaces [70]. 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 reparameterize the output of the network with a vector valued 3D Chebyshev expansion

$$\hat{\boldsymbol{u}}(\boldsymbol{x}) = \boldsymbol{\pi}_{\text{CH}}(\mathbf{u}, \boldsymbol{x}) = \sum_{l,m,n} \mathbf{U}_{ilmn} T_l(x_{(1)}) T_m(x_{(2)}) T_n(x_{(3)}), \tag{39}$$

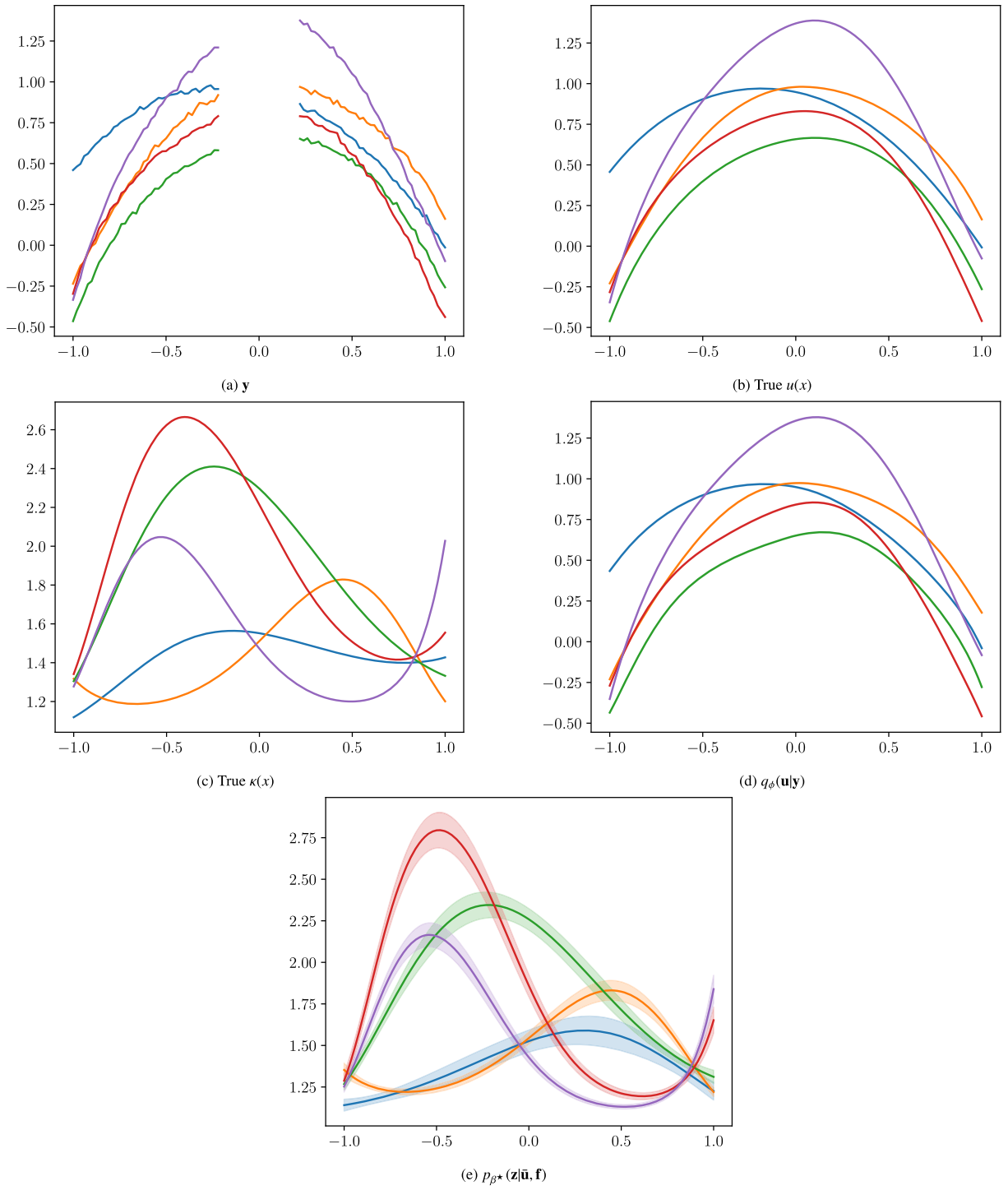
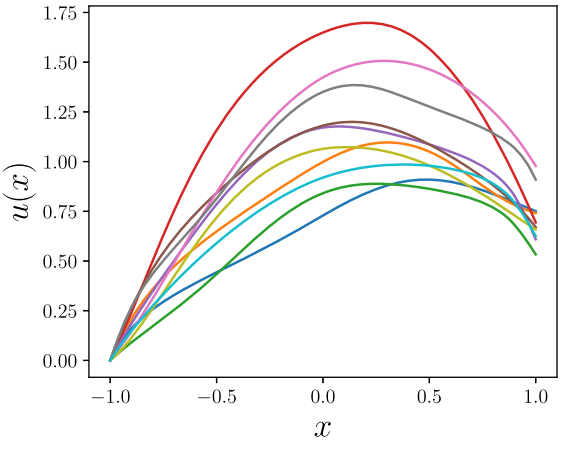
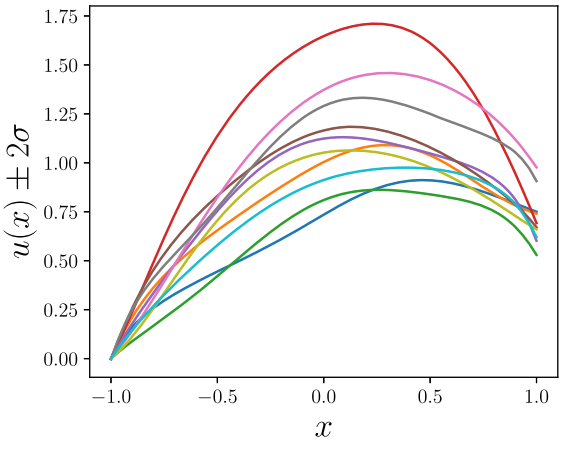


Fig. 6. Example of missing data. (a) Five data samples not observed during training. (b) The true $u(x)$ used to generate the synthetic data in (a). (c) The true parameter coefficient used to generate (b). (d) The predictive distribution of $\mathbf{u}$ given an observation $\mathbf{y}$. (e) The inferred 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 $q_\phi(\mathbf{u}|\mathbf{y})$.

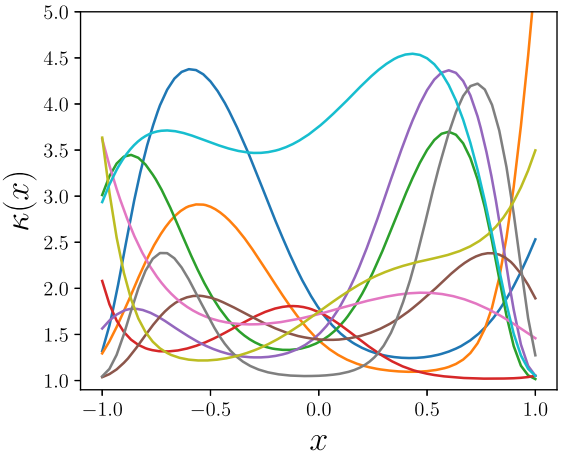
where $\hat{\boldsymbol{u}}(x) = (\hat{u}_{(1)}(\boldsymbol{x}), \hat{u}_{(2)}(\boldsymbol{x}), \hat{u}_{(3)}(\boldsymbol{x}))^\top$ indexed by $i \in \{1, 2, 3\}$ and the random vector $\mathbf{u} = \text{vect}(\mathbf{U})$ where $\mathbf{U}$ is a four dimensional tensor where the components $\mathbf{U}_{ilmn}$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



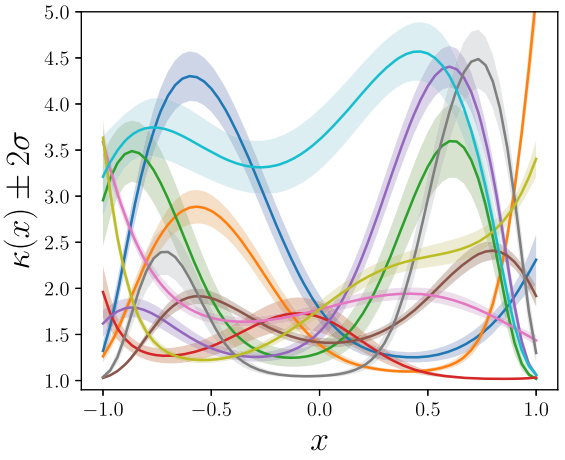
(a) FE Solver



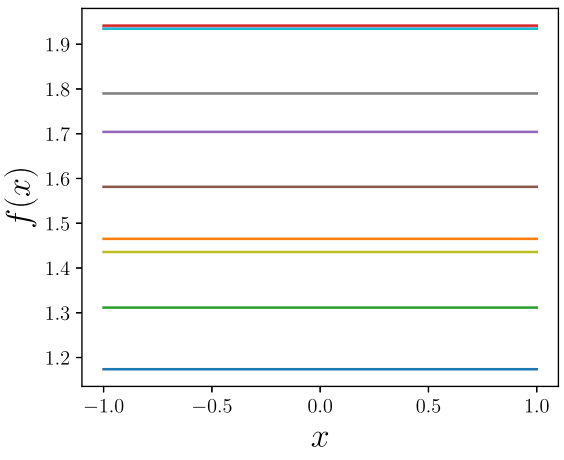
(b) $q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{f})$



(c) True $\kappa(x)$



(d) $p_\beta(\mathbf{z}|\mathbf{u}, \mathbf{f})$

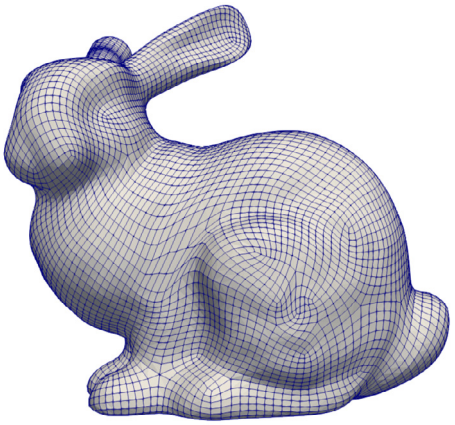


(e) $f(x)$

Fig. 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 $f(x)$.

	Forward	Inverse
MNSE PDDLVM	0.00277	0.00745
avg. time PDDLVM	0.00346s	0.0252s
avg. time FE	10.4s	

(a)



(b)

Fig. 8. (a) Computational runtime of using the FEM-PDDLVM framework on the thin-walled flexible shell bunny as compared to traditional FEM. (b) The mesh of the bunny geometry used for the finite element discretization, it has 24858 degrees of freedom.

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 are 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 centered Chebyshev reparametrization. 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.

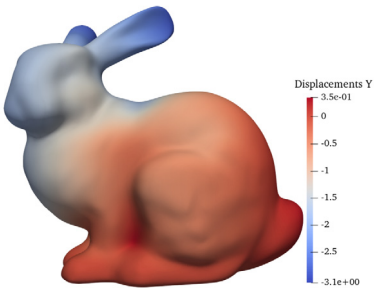
With the FEM formulation we can take advantage of well-known preconditioning techniques [71] for iterative solvers. Employing a simple preconditioner can greatly accelerate and stabilize convergence. This preconditioning is formulated as a reparametrization of the residual with the stiffness matrix of 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 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 optimization routine. Both α and β networks are fully connected neural networks with 3 hidden layers of 2.5k neurons with "swish" activation functions [72]. 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. 9 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 8(a) summarizes 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 $p(\mathbf{z})$.

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 can then assemble the new matrix, bypassing the expensive quadrature routine. This observation is important when computing the residual tens of thousands of times for large problems. However, if we are interested in marginalizing over a quantity that cannot be factorized out of the element stiffness integrals then the quadrature must be computed every iteration.

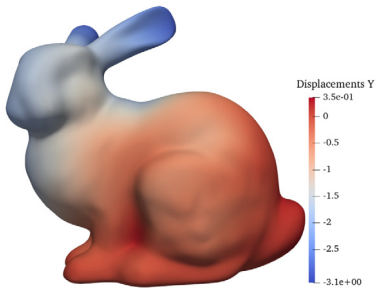
4.4. 1D nonlinear heat equation

To showcase the modularity of the PDDLVM framework, we consider next a time dependent nonlinear heat equation discretized with a PINN [7]. We can treat this as a spatio-temporal problem where we work with spatial and temporal dimensions the same way [7]. When combined with the PINN framework, our solution field predictive distribution and the inverse distribution are formulated in a general way as

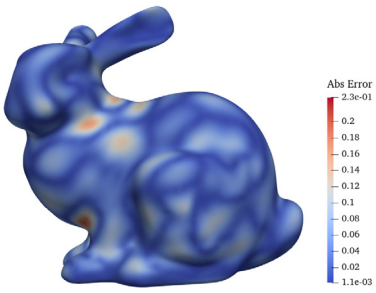
$$q_\alpha(\mathbf{u}|\mathbf{z},\mathbf{f}) = \mathcal{N}\left(\boldsymbol{\mu}_\alpha(x_{1:n},t_{1:n},\mathbf{z},\mathbf{f}), \operatorname{diag}(\sigma^2_\alpha(x_{1:n},t_{1:n},\mathbf{z},\mathbf{f})\right),$$



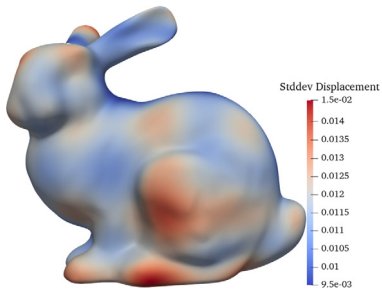
(a) Forward FEM



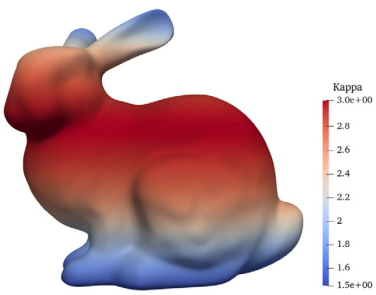
(b) Mean Forward PDDLVM



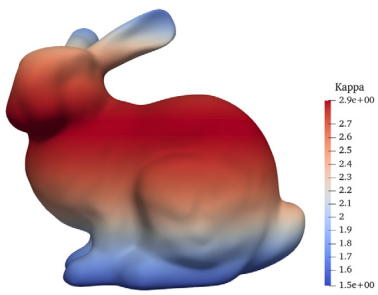
(c) Abs. Error (forward)



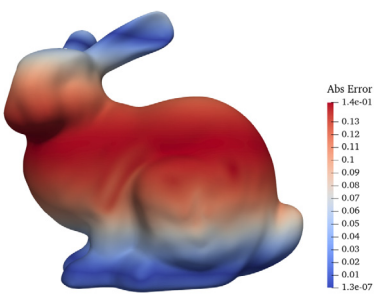
(d) St. Dev. (forward)



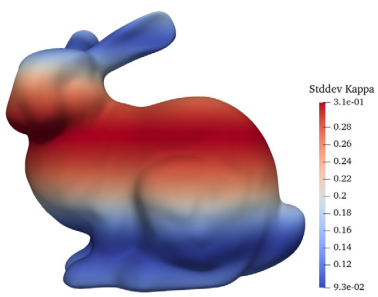
(e) True Parameter Field



(f) Mean Inverse PDDLVM



(g) Abs. Error (inverse)

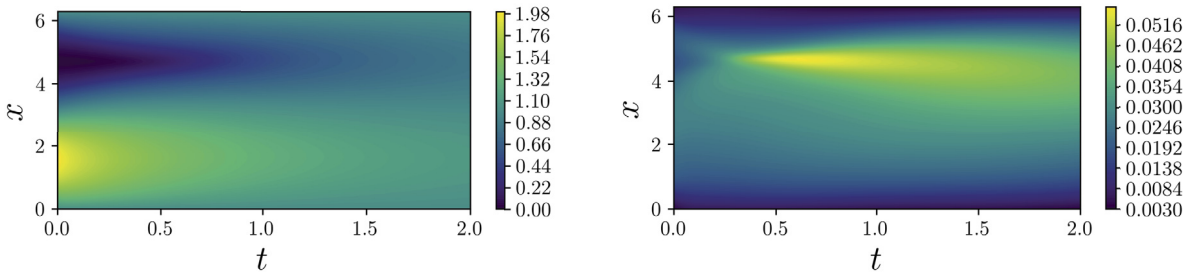


(h) St. Dev. (inverse)

Fig. 9. 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 solution, (b) mean estimate provided by PDDLVM, (c) absolute error between (a) and (b), (d) standard deviation. In the bottom row, similarly, we have (e) true parameter field, (f) mean estimate of the parameters provided by the inverse of PDDLVM, (g) absolute error, (h) standard deviation. The agreement between (g) and (h) 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. See Sec. 4.3 for more details.

$$p_\beta(\mathbf{z}|\mathbf{u},\mathbf{f}) = \mathcal{N}(\boldsymbol{\mu}_\beta(\mathbf{u}), \boldsymbol{\Sigma}_\beta(\mathbf{u})), \tag{40}$$

where $\mathbf{u} \in \mathbb{R}^n$ and are jointly trained as per the framework in Sec. 3.1. The samples $\mathbf{u}$ are then given to the various other probability density functions as well as the residual evaluator in (4). Here the collocation method with $w_i(x,t) = \delta(x - x_i)\delta(t - t_i)$ is used reducing the integration to a point evaluation of the PDE on the grid. We use auto-



(a) $\mu_\alpha(x,t)$ (b) $2\sigma_\alpha(x,t)$

Fig. 10. The distribution $q_\alpha(\mathbf{u}|\mathbf{z},\mathbf{f})$ for a parameter sample γ,κ for the time dependent nonlinear heat equation as defined in (41) and discretized 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 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.

Table 2
Comparison of PINN-PDDLVM model and a coupled PINN and convolutional inversion network. For details on the PINN residual formulations, see (42).

	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	6440 s	6808 s

matic differentiation through variable $\mathbf{u}$ and the neural network to obtain $\partial_x\hat{u}(x)$ with the reparametrization trick. The `parameter-to-solution` map 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 picture of the entire field as a grid of evaluations. A convolutional structure is well adapted for dealing with this kind of spatial data.

The PDE we analyze is the heat equation with a temperature dependent conductivity function. We specify the second order nonlinear parametric time dependent PDE as

$$\frac{\partial u(x,t)}{\partial t} = \frac{1}{\gamma}\nabla\cdot(\eta(u,\kappa)\nabla u(x,t)), \qquad \eta(u,\kappa) = \left|\frac{u(x,t)\kappa^2+1}{\kappa}\right|, \tag{41}$$

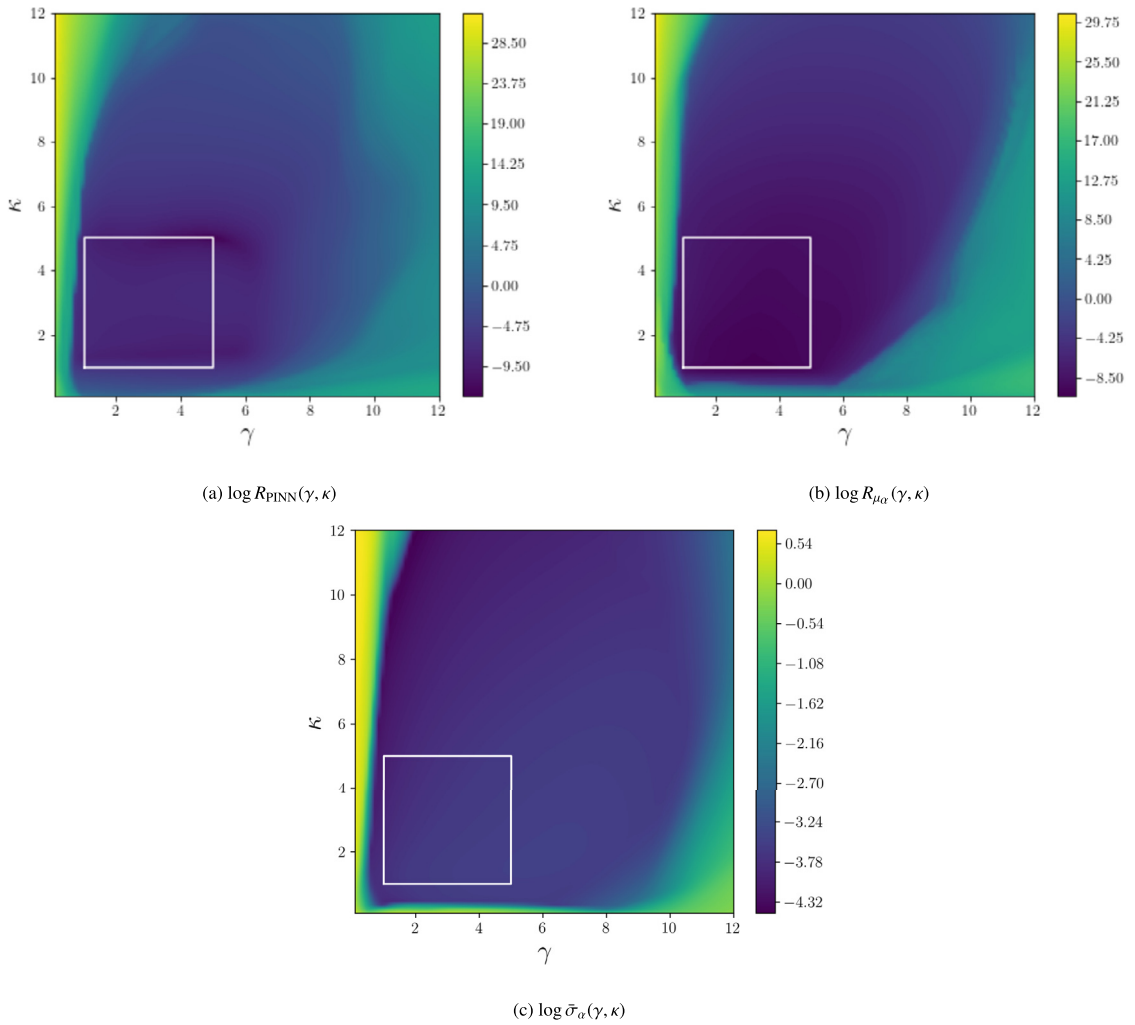
for $x\in(0,2\pi)$, with initial conditions and boundary conditions $u(0,t)=1$, $u(2\pi,t)=1$, and $u(x,0)=\sin(x)+1$. The $\eta(u,\kappa)$ function represents a material whose conductance increases linearly as the temperature increases with a slope and intercept are governed by the scalar κ. We construct $\mathbf{\Sigma_r} = \text{diag}\left([\epsilon^2_{D_0},\ldots,\epsilon^2_{D_d},\epsilon^2_{B_0},\ldots,\epsilon^2_{B_b},\epsilon^2_{I_0},\ldots,\epsilon^2_{I_i}]\right)$ where each ϵ denotes the confidence for the d domain nodes, the b boundary nodes, and the i initial condition nodes respectively. We choose a residual standard deviation $\epsilon_D = 0.01$, while to strongly enforce the initial and boundary conditions, we choose $\epsilon_B = 0.001$ and $\epsilon_I = 0.001$. The PDE is also parameterized with γ 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 $\mathbf{z}=\{\kappa,\gamma\}$.

In Fig. 11 we plot the log PINN residual for a range of PDE parameters κ and γ and we plot the log mean standard deviation of the α network. In Fig. 10 we show a predicted distribution over the solution field of the heat equation for a sample of γ,κ drawn from the prior and computed through the PINN-PDDLVM framework.

The `parameter-to-solution` 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 γ and κ 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, stride $(2,2)$ and [8, 16, 32] filters followed by a final dense layer of 500 neurons ending with either 2 or 4 output neurons for either the 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} (Table 2). When computing the residual for comparison we employ the standard PINN-style [7] residual as

$$R_h(\gamma,\kappa) = \frac{1}{NM}\sum_i^N\sum_j^M\left(-\frac{\partial h(x_i,t_j)}{\partial t} + \frac{1}{\gamma}\nabla\cdot(\eta(h(x_i,t_j),\kappa)\nabla h(x_i,t_j))\right)^2$$



(a) $\log R_{\text{PINN}}(\gamma, \kappa)$

(b) $\log R_{\mu_\alpha}(\gamma, \kappa)$

(c) $\log \bar{\sigma}_\alpha(\gamma, \kappa)$

Fig. 11. A comparison of (a) PINN and (b) PINN-PDDLVM residuals and in (c) the average PINN-PDDLVM standard deviation. The quantities are computed over a 2-dimensional range of values of parameters γ and κ. The white square denotes the bounds for the observed κ, γ during training as well as the bounds over which the mean residual is computed and is defined by the prior $p(\mathbf{z}) = p(\gamma)p(\kappa) = \mathcal{U}([1,5]) \times \mathcal{U}([1,5])$.

$$+ \frac{1}{n}\sum_{k}^{n}(h(x_k,0) - u(x_k,0))^2 + \frac{1}{m}\sum_{l}^{m}\Big[(h(0,t_0) - u(0,t_0))^2 + (h(2\pi,t_0) - u(2\pi,t_0))^2\Big], \tag{42}$$

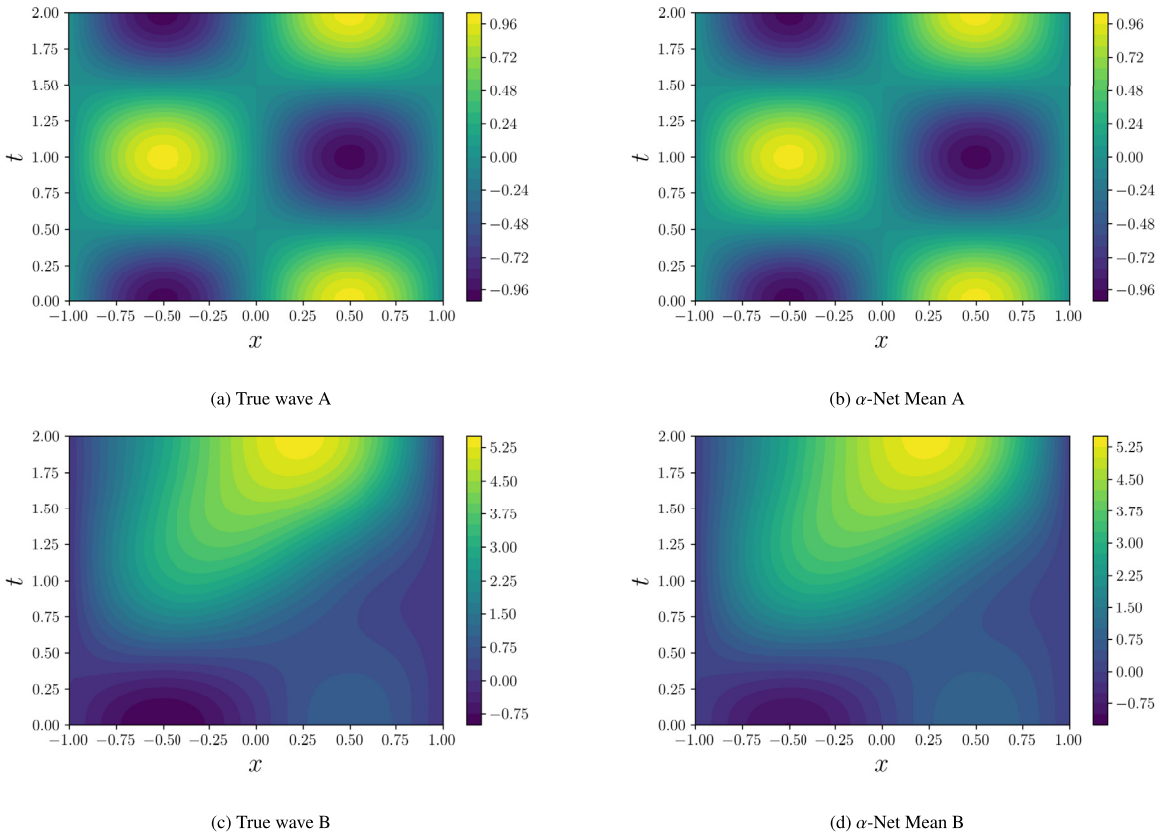
where the second and the third terms are the residuals for the initial and the boundary conditions. It is the log of (42) that is plotted in Fig. 11 but it is not the residual used for *training* the PINN-PDDLVM, for that we use (4).

4.5. Inhomogeneous wave equation

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

$$\begin{aligned} &\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} = f(x), \\ &f(x) = \sum_i f_i x^i. \end{aligned} \tag{43}$$

Here f_i make up the vector $\mathbf{f}$ and are the parameters which make up the polynomial expansion representation of the forcing. In the current example we use 4 terms in the forcing expansion where all components are 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. In Fig. 12 we show 2 example ground truth solutions from the dataset as well as the mean predictions of those



(a) True wave A

(b) α-Net Mean A

(c) True wave B

(d) α-Net Mean B

Fig. 12. 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: $f = -5x^3$.

Table 3
Comparison of MNSE for supervised FNOs with differing number of data points (number following FNO) against an unsupervised PINN-PDDLVM.

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

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. The training data is obtained from a Chebyshev based solver of 21×21 basis functions. In Table 3 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.

5. Conclusions

PDDLVM is a fully probabilistic generative model for parametric PDEs that builds an interpretable latent space representation and provides uncertainty estimates on the predictions. The Bayesian view allows us to construct a collection of probabilistic mappings that are connected together in a sound and principled way. The ability to produce forward and inverse solutions with uncertainty estimates is of significant importance to the practical use of such methods. It is particularly crucial when dealing with inverse problems as they are inherently ill-posed.

The modular nature of the developed framework allows for an easy transition between meshless (*e.g.* PINNs) and mesh-based (*e.g.* FEM) discretizations, making it agnostic to the specific method used to discretize the PDE. For example, we leverage 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 produces a very streamlined framework for learning the behavior of parametric PDEs.

While conventional FEM offers many advantages for solid mechanics problems, for other applications (such as fluid flows), other discretization techniques (such as immersed, particle-based methods [73,74] or spectral methods) might be preferable.

Furthermore, in current work, the variational inference is based on the mean field assumption, which is known to be overconfident in estimating the posterior uncertainty. The work can naturally be 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 [75]. Other directions this work can be expanded is in the use of different architectures for the forward and inverse probabilistic maps, such as Fourier Neural Operators [8], Wavelet Neural Operators [76] or Physics-Informed Graph Neural Galerkin Networks [27]. This work could have significant impact in parametric PDE applications such as design and optimization scenarios and the study of complex systems.

CRediT authorship contribution statement

Arnaud Vadeboncoeur: Conceptualization, Methodology, Software, Writing – original draft. **Ömer Deniz Akyildiz:** Conceptualization, Writing – original draft, Writing – review & editing. **Ieva Kazlauskaite:** Conceptualization, Writing – original draft, Writing – review & editing. **Mark Girolami:** Conceptualization, Writing – review & editing. **Fehmi Cirak:** Conceptualization, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgements

A.V. was supported by the Baxter & Alma Ricard Foundation Scholarship. Ö. D. A. was partly supported by the Lloyd's Register Foundation Data Centric Engineering Program and EPSRC Program Grant EP/R034710/1 (CoSInES). I. K. was funded by a Biometrika Fellowship awarded by the Biometrika Trust. M. G was supported by a Royal Academy of Engineering Research Chair, and EPSRC grants EP/R018413/2, EP/P020720/2, EP/R034710/1, EP/R004889/1, EP/T000414/1. F. C. was supported by Wave 1 of The UKRI Strategic Priorities Fund under the EPSRC Grant EP/T001569/1, particularly the "Digital twins for complex engineering systems" theme within that grant, and The Alan Turing Institute.

Appendix A. 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, the TensorFlow GPU usage was limited between 6 and 13 GBs. The Poisson 1D examples are 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 was run entirely on the GPU as it can best leverage this type of hardware. We also note the definition used for the reported Mean Normalized Squared Error as

$$\mathrm{MNSE}(x^*_{1:N}, x_{1:N}) := \frac{1}{N}\sum_{i}^{N}\frac{\|x^*_i - x_i\|^2}{\|x^*_i\|^2}, \tag{A.1}$$

where x^* denotes the base truth, and x denotes the approximation.

Appendix B. Alternative derivation of PDDLVM

In this section, we show how to derive the PDDLVM from the KL divergence between an intractable posterior and a variational approximation. We start by restating our joint model (15) from Sect. 3.1 which includes the residual observation variable and the trainable inversion network

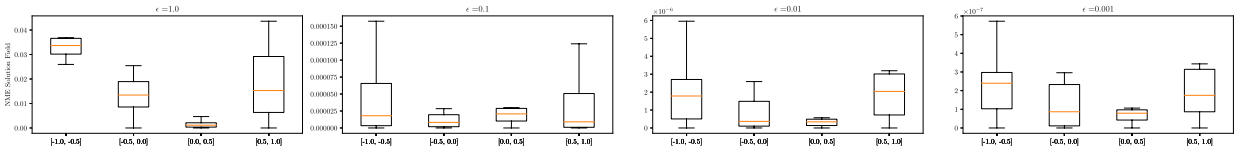
$$p_\beta(\mathbf{r}, \mathbf{u}, \mathbf{z}, \mathbf{f}) = p(\mathbf{r}|\mathbf{u}, \mathbf{z}, \mathbf{f})\, p_\beta(\mathbf{z}|\mathbf{u}, \mathbf{f})\, p(\mathbf{u})\, p(\mathbf{f}). \tag{B.1}$$

We also restate our trainable variational approximation distribution (17),

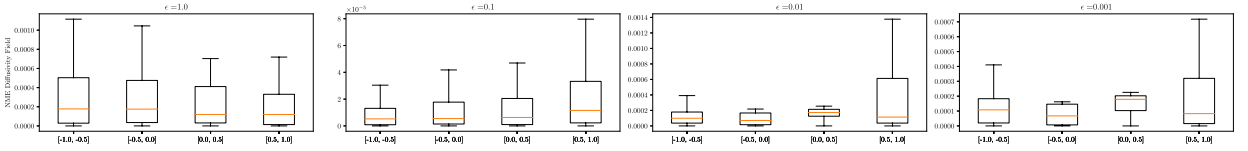
$$q_\alpha(\mathbf{u}, \mathbf{z}, \mathbf{f}) = q_\alpha(\mathbf{u}|\mathbf{z}, \mathbf{f})\, p(\mathbf{z})\, p(\mathbf{f}). \tag{B.2}$$

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

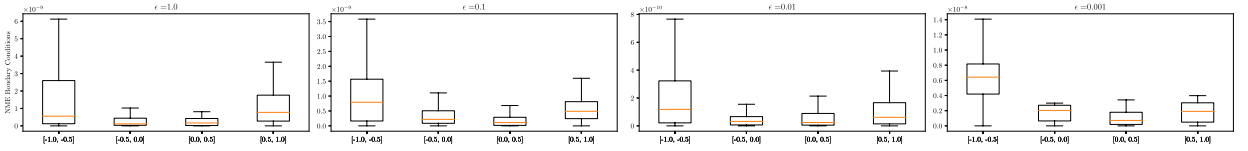
$$p_\beta(\mathbf{u}, \mathbf{z}, \mathbf{f}|\mathbf{r}) = \frac{p_\beta(\mathbf{r}, \mathbf{u}, \mathbf{z}, \mathbf{f})}{p(\mathbf{r})}. \tag{B.3}$$



(a) MNSE of $u(x)$.



(b) MNSE of the reconstructed $\kappa(x)$.



(c) MNSE of a, and b (boundary condition values).

Fig. C.13. The MNSE for different $\epsilon_{\mathbf{r}}$ values for the ranges of $\boldsymbol{\kappa}$ given on the x-axis. This was computed for the 1D linear Poisson example of Sec. 4.1 and is computed from the same values as Fig. 5 but not in the log scale. The prior on which the FEM-PDDLVM was trained was $p(\mathbf{z}) = \mathcal{N}(0, 0.5)$.

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

$$D_{KL}(q_\alpha(\mathbf{u},\mathbf{z},\mathbf{f})||p_\beta(\mathbf{u},\mathbf{z},\mathbf{f}|\mathbf{r})) = \int \log \frac{q_\alpha(\mathbf{u},\mathbf{z},\mathbf{f})}{p_\beta(\mathbf{u},\mathbf{z},\mathbf{f}|\mathbf{r})} q_\alpha(\mathbf{u},\mathbf{z},\mathbf{f}) \mathrm{d}\mathbf{u}\mathrm{d}\mathbf{z}\mathrm{d}\mathbf{f}, \tag{B.4}$$
$$= \int \log \frac{q_\alpha(\mathbf{u},\mathbf{z},\mathbf{f}) p(\mathbf{r})}{p_\beta(\mathbf{r},\mathbf{u},\mathbf{z},\mathbf{f})} q_\alpha(\mathbf{u},\mathbf{z},\mathbf{f}) \mathrm{d}\mathbf{u}\mathrm{d}\mathbf{z}\mathrm{d}\mathbf{f}.$$

From this we factorize the evidence of the residual $p(\mathbf{r})$ out of the integral

$$D_{KL}(q_\alpha(\mathbf{u},\mathbf{z},\mathbf{f})||p_\beta(\mathbf{u},\mathbf{z},\mathbf{f}|\mathbf{r})) = \int \log \frac{q_\alpha(\mathbf{u},\mathbf{z},\mathbf{f})}{p_\beta(\mathbf{r},\mathbf{u},\mathbf{z},\mathbf{f})} q_\alpha(\mathbf{u},\mathbf{z},\mathbf{f}) \mathrm{d}\mathbf{u}\mathrm{d}\mathbf{z}\mathrm{d}\mathbf{f} + \log p(\mathbf{r}), \tag{B.5}$$

while taking into account that $\int q_\alpha(\mathbf{u},\mathbf{z},\mathbf{f})\mathrm{d}\mathbf{u}\mathrm{d}\mathbf{z}\mathrm{d}\mathbf{f} \equiv 1$. We then note that the KL divergence is a non-negative quantity so that

$$D_{KL}(q_\alpha(\mathbf{u},\mathbf{z},\mathbf{f})||p_\beta(\mathbf{u},\mathbf{z},\mathbf{f}|\mathbf{r})) \geq 0. \tag{B.6}$$

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

$$\log p(\mathbf{r} = \mathbf{0}) \geq \int \log \frac{p_\beta(\mathbf{r} = \mathbf{0},\mathbf{u},\mathbf{z},\mathbf{f})}{q_\alpha(\mathbf{u},\mathbf{z},\mathbf{f})} q_\alpha(\mathbf{u},\mathbf{z},\mathbf{f}) \mathrm{d}\mathbf{u}\mathrm{d}\mathbf{z}\mathrm{d}\mathbf{f}. \tag{B.7}$$

Substituting the chosen factorizations (B.1) and (B.2) into the individual joint terms, we obtain the PDDLVM framework derived in Sec. 3.1,

$$\log p(\mathbf{r} = \mathbf{0}) \geq \int \log \frac{p(\mathbf{r} = \mathbf{0}|\mathbf{u},\mathbf{z},\mathbf{f}) p_\beta(\mathbf{z}|\mathbf{u},\mathbf{f}) p(\mathbf{u})}{q_\alpha(\mathbf{u}|\mathbf{z},\mathbf{f}) p(\mathbf{z})} q_\alpha(\mathbf{u}|\mathbf{z},\mathbf{f}) p(\mathbf{z}) p(\mathbf{f}) \mathrm{d}\mathbf{u}\mathrm{d}\mathbf{z}\mathrm{d}\mathbf{f}. \tag{B.8}$$

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

Appendix C. Additional results for 1D nonlinear Poisson

In Fig. C.13 we plot the same quantities as in Fig. 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. C.14 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.

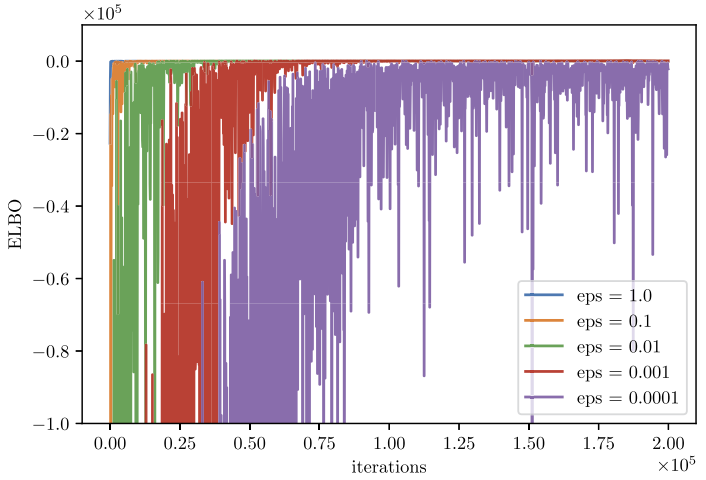


Fig. C.14. The convergence of the ELBO estimate for a 1D linear Poisson equation for various choices of $\epsilon_{\mathbf{r}}$.

Appendix D. PINN-PDDLVM

We stress the generality of using the weighted residual formulation, as noted by [44], as different choices for the weight function $w(x)$ results in different well-known discretization methods. Setting $w_i(x) = \delta(x - x_i)$ we have a collocation method [77], and for $w_i(x) = \boldsymbol{\delta}_{ij}$ for $x \in \Omega_j$ we obtain a subdomain (finite volume) method [78]. For $w_i(x) = \partial\mathcal{R}(\hat{u}, \ldots)/\partial u_j$ we have a least-squares method [79], for $w_i(x) = x^{m-1}$ we have a moment method (used in, *e.g.*, electromagnetism [80]). For $w_i(x) = \phi_i(x)$ we obtain a Galerkin method [1]. This could be a spectral method if $w_i(x) \neq 0$ for every $x \in \Omega$ or FEM if $w_i(x) \neq 0$ only for $x \in \Omega_k \subset \Omega$.

In the case of PINN we clarify its link with the weighted residual method as a method of least-squares collocation [48]. This is done by choosing $w_i(x) = \frac{\partial\mathcal{R}(\hat{u},x)}{\partial W}\delta(x_i - x)$ with N_1, N_2, N_3 collocation points in $\{\Omega, \Gamma, \mathcal{I}\}$, which are the domain, boundary conditions and initial conditions, respectively. W denotes the collection of NN parameters. We abuse the notation and use $x = \{x, t\}$. As in [79] we motivate the least-squares method as

$$r_i = \int_\Omega w_i(x)\mathcal{R}_\Omega(\hat{u}, x)\mathrm{d}x + \int_\Gamma w_i(x)\mathcal{R}_\Gamma(\hat{u}, x)\mathrm{d}x + \int_\mathcal{I} w_i(x)\mathcal{R}_\mathcal{I}(\hat{u}, x)\mathrm{d}x. \tag{D.1}$$

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

$$\begin{aligned} r_i = \int_\Omega \frac{\partial\mathcal{R}_\Omega}{\partial W}\delta(x_i - x)\mathcal{R}_\Omega(\hat{u}, x)\mathrm{d}x + \int_\Gamma \frac{\partial\mathcal{R}_\Gamma}{\partial W}\delta(x_i - x)\mathcal{R}_\Gamma(\hat{u}, x)\mathrm{d}x \\ + \int_\mathcal{I} \frac{\partial\mathcal{R}_\mathcal{I}}{\partial W}\delta(x_i - x)\mathcal{R}_\mathcal{I}(\hat{u}, x)\mathrm{d}x. \end{aligned} \tag{D.2}$$

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

$$\begin{aligned} r_i = \frac{1}{2}\frac{\partial}{\partial W}\Bigg[\int_\Omega \delta(x_i - x)\mathcal{R}_\Omega^2(\hat{u}, x)\mathrm{d}x + \int_\Gamma \delta(x_i - x)\mathcal{R}_\Gamma^2(\hat{u}, x)\mathrm{d}x \\ + \int_\mathcal{I} \delta(x_i - x)\mathcal{R}_\mathcal{I}^2(\hat{u}, x)\mathrm{d}x\Bigg]. \end{aligned} \tag{D.3}$$

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

$$I_i(W) = \mathcal{R}_\Omega^2(\hat{u}, x_i) + \mathcal{R}_\Gamma^2(\hat{u}, x_i) + \mathcal{R}_\mathcal{I}^2(\hat{u}, x_i). \tag{D.4}$$

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

$$I(W) = \frac{1}{N_1}\sum_{x_i \in \Omega} \mathcal{R}_\Omega^2(\hat{u}, x_i) + \frac{1}{N_2}\sum_{x_i \in \Gamma} \mathcal{R}_\Gamma^2(\hat{u}, x_i) + \frac{1}{N_3}\sum_{x_i \in \mathcal{I}} \mathcal{R}_\mathcal{I}^2(\hat{u}, x_i), \tag{D.5}$$

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

Appendix E. Additional details

In this subsection we collect information from the paper for easy referencing (Tables E.4–E.11).

1D linear Poisson – Sec. 4.1.1

Table E.4
Basic information 1D Linear Poisson.

dim **u**	dim **r**	dim **z**	**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. 4.1.2

Table E.5
Basic information 1D Linear Poisson Observable Map Inversion.

dim **u**	dim **r**	dim **z**	dim **f**	dim **y**	**z**	σ_y	$\epsilon_{\mathbf{r}}$	α-Net	β-Net	iters
8	101	4	1	80	$\{\boldsymbol{\kappa}, a, b\}$	10^{-2}	10^{-2}	$4l$, $50n$,	$4l$, $50n$,	1×10^6

Table E.6
Prior information 1D Linear Poisson Observable Map Inversion.

$p(\kappa_i)$	$p(\mathbf{f})$	$p(a)$	$p(b)$
$\mathcal{U}(-1,1)$	$\mathcal{U}(1,5)$	$\mathcal{U}(-0.5,0.5)$	$\mathcal{U}(-0.5,0.5)$

1D Nonlinear Poisson – Sec. 4.2

Table E.7
Basic information 1D Nonlinear Poisson.

dim **u**	dim **r**	dim **z**	dim **f**	**z**	$\epsilon_{\mathbf{r}}$	α-Net	β-Net	iters
10	61	5	1	$\{\boldsymbol{\kappa}, b\}$	10^{-2}	$4l$, $100n$	$4l$, $100n$	1×10^6

Table E.8
Prior information 1D Nonlinear Poisson.

$p(\mathbf{z})$	$p(\mathbf{f})$	$p(b)$
$\mathcal{N}(0,1)$	$\mathcal{U}(1,1)$	$\mathcal{U}(0.5,1)$

Thin-Walled Flexible Shell – Sec. 4.3

Table E.9
Basic information Thin-Walled Flexible Shell.

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

Nonlinear Heat Equation – Sec. 4.4

Table E.10
Basic information Nonlinear Heat Equation.

dim **u**	dim **r**	dim **z**	**z**	$p(\mathbf{z})$	$\epsilon_{\mathbf{r}}$	α-Net	β-Net	iters
2×10^5	2×10^5	2	$\{\gamma, \kappa\}$	$\mathcal{U}(1,5) \times \mathcal{U}(1,5)$	10^{-2}	$4l$, $100n$	$3c$, $1l - 500n$	5×10^5

Inhomogeneous Wave Equation – Sec. 4.5

Table E.11
Basic information Inhomogeneous Wave Equation.

dim **u**	dim **r**	dim **z**	**z**	$p(\mathbf{z})$	$\epsilon_{\mathbf{r}}$	α-Net	β-Net	iters
2.5×10^3	2.5×10^3	4	$\{\mathbf{f}\}$	$\mathcal{U}(-5,5)$	10^{-2}	$4l$, $100n$	$3c$, $1l - 500n$	2.5×10^5

References

[1] A. Ern, J.-L. Guermond, Theory and Practice of Finite Elements, vol. 159, Springer, 2004.
[2] A. Quarteroni, A. Valli, Numerical Approximation of Partial Differential Equations, Springer, 2008.
[3] A. Tarantola, Inverse Problem Theory and Methods for Model Parameter Estimation, SIAM, 2005.
[4] A.M. Stuart, Inverse problems: a Bayesian perspective, Acta Numer. 19 (2010) 451–559, https://doi.org/10.1017/S0962492910000061.
[5] C.R. Vogel, Computational Methods for Inverse Problems, Frontiers in Applied Mathematics, Society for Industrial and Applied Mathematics, 2002.
[6] S. Arridge, P. Maass, O. Öktem, C.-B. Schönlieb, Solving inverse problems using data-driven models, Acta Numer. 28 (2019) 1–174, https://doi.org/10.1017/S0962492919000059.
[7] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[8] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, arXiv preprint, arXiv:2010.08895, 2020.
[9] L. Lu, P. Jin, G. Pang, Z. Zhang, G.E. Karniadakis, Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, Nat. Mach. Intell. 3 (3) (2021) 218–229.
[10] A. Anandkumar, K. Azizzadenesheli, K. Bhattacharya, N. Kovachki, Z. Li, B. Liu, A. Stuart, Neural operator: graph kernel network for partial differential equations, in: ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations, 2020.
[11] N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, A. Anandkumar, Neural operator: learning maps between function spaces with applications to PDEs, J. Mach. Learn. Res. 24 (89) (2023) 1–97.
[12] J. Han, A. Jentzen, W. Ee, Solving high-dimensional partial differential equations using deep learning, Proc. Natl. Acad. Sci. 115 (07 2017), https://doi.org/10.1073/pnas.1718942115.
[13] J. Sirignano, K. Spiliopoulos, DGM: a deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1339–1364.
[14] G.E. Karniadakis, I.G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, Nat. Rev. Phys. 3 (6) (2021) 422–440.
[15] K. Raveendran, C. Wojtan, N. Thuerey, G. Turk, Blending liquids, ACM Trans. Graph. 33 (4) (2014).
[16] S.L. Brunton, J.L. Proctor, J.N. Kutz, Discovering governing equations from data by sparse identification of nonlinear dynamical systems, Proc. Natl. Acad. Sci. 113 (15) (2016) 3932–3937.
[17] B. Kim, V.C. Azevedo, N. Thuerey, T. Kim, M. Gross, B. Solenthaler, Deep fluids: a generative network for parameterized fluid simulations, Comput. Graph. Forum 38 (2019) 59–70, Wiley Online Library.
[18] L. Ardizzone, J. Kruse, S. Wirkert, D. Rahner, E.W. Pellegrini, R.S. Klessen, L. Maier-Hein, C. Rother, U. Köthe, Analyzing inverse problems with invertible neural networks, arXiv preprint, arXiv:1808.04730, 2018.
[19] Q. Zhao, D.B. Lindell, G. Wetzstein, Learning to solve PDE-constrained inverse problems with graph networks, in: ICML 2022 2nd AI for Science Workshop, 2022.
[20] O. Ghattas, K. Willcox, Learning physics-based models from data: perspectives from inverse problems and model reduction, Acta Numer. 30 (2021) 445–554.
[21] M. Raissi, G.E. Karniadakis, Hidden physics models: machine learning of nonlinear partial differential equations, J. Comput. Phys. 357 (2018) 125–141.
[22] G. Pang, L. Lu, G.E. Karniadakis, fPINNs: fractional physics-informed neural networks, SIAM J. Sci. Comput. 41 (4) (2019) A2603–A2626.
[23] Z. Mao, A.D. Jagtap, G.E. Karniadakis, Physics-informed neural networks for high-speed flows, Comput. Methods Appl. Mech. Eng. 360 (2020) 112789.
[24] S. Cai, Z. Mao, Z. Wang, M. Yin, G.E. Karniadakis, Physics-informed neural networks (PINNs) for fluid mechanics: a review, Acta Mech. Sin. (2022) 1–12.
[25] L. Lu, R. Pestourie, W. Yao, Z. Wang, F. Verdugo, S.G. Johnson, Physics-informed neural networks with hard constraints for inverse design, SIAM J. Sci. Comput. 43 (6) (2021) B1105–B1132.
[26] S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient flow pathologies in physics-informed neural networks, SIAM J. Sci. Comput. 43 (5) (2021) A3055–A3081.
[27] H. Gao, M.J. Zahr, J.-X. Wang, Physics-informed graph neural Galerkin networks: a unified framework for solving PDE-governed forward and inverse problems, Comput. Methods Appl. Mech. Eng. 390 (2022) 114502.
[28] K. Bhattacharya, B. Hosseini, N.B. Kovachki, A.M. Stuart, Model reduction and neural networks for parametric PDEs, SMAI J. Comput. Math. 7 (2021) 121–157.
[29] R. Wang, K. Kashinath, M. Mustafa, A. Albert, R. Yu, Towards physics-informed deep learning for turbulent flow prediction, in: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020, pp. 1457–1466.
[30] R.G. Patel, N.A. Trask, M.A. Wood, E.C. Cyr, A physics-informed operator regression framework for extracting data-driven continuum models, Comput. Methods Appl. Mech. Eng. 373 (2021) 113500.
[31] F. Kröpfl, R. Maier, D. Peterseim, Operator compression with deep neural networks, Adv. Cont. Discr. Mod. 2022 (1) (2022) 1–23.
[32] Y. Zhu, N. Zabaras, Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification, J. Comput. Phys. 366 (2018) 415–447.
[33] S. Wang, H. Wang, P. Perdikaris, Learning the solution operator of parametric partial differential equations with physics-informed DeepONets, Sci. Adv. 7 (40) (2021).
[34] A. Vadeboncoeur, I. Kazlauskaite, Y. Papandreou, F. Cirak, M. Girolami, Ö.D. Akyildiz, Random grid neural processes for parametric partial differential equations, arXiv preprint, arXiv:2301.11040, 2023.
[35] M. Yin, E. Zhang, Y. Yu, G.E. Karniadakis, Interfacing finite elements with deep neural operators for fast multiscale modeling of mechanics problems, Comput. Methods Appl. Mech. Eng. (2022) 115027.
[36] Y. Yang, P. Perdikaris, Conditional deep surrogate models for stochastic, high-dimensional, and multi-fidelity systems, Comput. Mech. 64 (2) (2019) 417–434.
[37] Y. Yang, P. Perdikaris, Adversarial uncertainty quantification in physics-informed neural networks, J. Comput. Phys. 394 (2019) 136–152.
[38] N. Winovich, K. Ramani, G. Lin, ConvPDE-UQ: convolutional neural networks with quantified uncertainty for heterogeneous elliptic partial differential equations on varied domains, J. Comput. Phys. 394 (2019) 263–279.
[39] J. Cockayne, C.J. Oates, T.J. Sullivan, M. Girolami, Bayesian probabilistic numerical methods, SIAM Rev. 61 (4) (2019) 756–789.
[40] M. Girolami, E. Febrianto, G. Yin, F. Cirak, The statistical finite element method (statFEM) for coherent synthesis of observation data and model predictions, Comput. Methods Appl. Mech. Eng. 375 (2021) 113533.
[41] C. Duffin, E. Cripps, T. Stemler, M. Girolami, Statistical finite elements for misspecified models, Proc. Natl. Acad. Sci. 118 (2) (2021).
[42] Ö.D. Akyildiz, C. Duffin, S. Sabanis, M. Girolami, Statistical finite elements via Langevin dynamics, SIAM/ASA J. Uncertain. Quantificat. (2022).
[43] S. Kaltenbach, P.-S. Koutsourelakis, Incorporating physical constraints in a deep probabilistic machine learning framework for coarse-graining dynamical systems, J. Comput. Phys. 419 (2020) 109673.
[44] M. Rixner, P.-S. Koutsourelakis, A probabilistic generative model for semi-supervised training of coarse-grained surrogates and enforcing physical constraints through virtual observables, J. Comput. Phys. 434 (2021) 110218.
[45] G. Strang, Introduction to Applied Mathematics, Wellesley-Cambridge Press, 1986.

[46] S. Chakraverty, N. Mahato, P. Karunakar, T.D. Rao, Advanced Numerical and Semi-analytical Methods for Differential Equations, John Wiley & Sons, 2019.
[47] M. Hatami, Weighted Residual Methods: Principles, Modifications and Applications, Academic Press, 2017.
[48] B.A. Finlayson, The Method of Weighted Residuals and Variational Principles, SIAM, 2013.
[49] L.-E. Lindgren, From Weighted Residual Methods to Finite Element Methods, 2009.
[50] C. Gerald, P. Wheatley, Applied Numerical Analysis, Featured Titles for Numerical Analysis, Pearson/Addison-Wesley, 2004.
[51] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, P. Perdikaris, Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data, J. Comput. Phys. 394 (2019) 56–81.
[52] L. Guo, H. Wu, T. Zhou, Normalizing field flows: solving forward and inverse stochastic differential equations using physics-informed flow models, J. Comput. Phys. 461 (2022) 111202.
[53] D. O'Malley, J.K. Golden, V.V. Vesselinov, Learning to regularize with a variational autoencoder for hydrologic inverse analysis, UMBC Fac. Collect. (2019).
[54] D.J. Tait, T. Damoulas, Variational autoencoding of PDE inverse problems, arXiv preprint, arXiv:2006.15641, 2020.
[55] Z. Qian, W. Zame, L. Fleuren, P. Elbers, M. van der Schaar, Integrating expert ODEs into neural ODEs: pharmacology and disease progression, Adv. Neural Inf. Process. Syst. 34 (2021) 11364–11383.
[56] W. Zhong, H. Meidani, PI-VAE: physics-informed variational auto-encoder for stochastic differential equations, Comput. Methods Appl. Mech. Eng. 403 (2023) 115664.
[57] C. Jacobsen, K. Duraisamy, Disentangling generative factors of physical fields using variational autoencoders, Front. Phys. 10 (2022).
[58] N. Takeishi, A. Kalousis, Physics-integrated variational autoencoders for robust and interpretable generative modeling, Adv. Neural Inf. Process. Syst. 34 (2021) 14809–14821.
[59] S.C. Brenner, L.R. Scott, The Mathematical Theory of Finite Element Methods, vol. 3, Springer, 2008.
[60] P.-H. Chiu, J.C. Wong, C. Ooi, M.H. Dao, Y.-S. Ong, CAN-PINN: a fast physics-informed neural network based on coupled-automatic–numerical differentiation method, Comput. Methods Appl. Mech. Eng. 395 (2022) 114909.
[61] C.M. Bishop, Pattern Recognition and Machine Learning, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
[62] D.P. Kingma, M. Welling, An introduction to variational autoencoders, Found. Trends Mach. Learn. 12 (4) (2019) 307–392.
[63] D.P. Kingma, M. Welling, Auto-encoding variational Bayes, in: International Conference on Learning Representations, ICLR 2014, 2014.
[64] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, in: International Conference on Learning Representations, ICLR, 2015.
[65] D. Dehaene, R. Brossard, Re-parameterizing VAEs for stability, arXiv preprint, arXiv:2106.13739, 2021.
[66] E.A. Wan, R. Van Der Merwe, The unscented Kalman filter for nonlinear estimation, in: Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium, IEEE, 2000, pp. 153–158.
[67] F. Cirak, M. Ortiz, P. Schröder, Subdivision surfaces: a new paradigm for thin-shell finite-element analysis, Int. J. Numer. Methods Eng. 47 (12) (2000) 2039–2072.
[68] F. Cirak, Q. Long, Subdivision shells with exact boundary control and non-manifold geometry, Int. J. Numer. Methods Eng. 88 (9) (2011) 897–923.
[69] E. Febrianto, L. Butler, M. Girolami, F. Cirak, Digital twinning of self-sensing structures using the statistical finite element method, Data-Cent. Eng. 3 (2022) e31.
[70] Q. Zhang, M. Sabin, F. Cirak, Subdivision surfaces with isogeometric analysis adapted refinement weights, Comput. Aided Des. 102 (2018) 104–114.
[71] M.J. Grote, T. Huckle, Parallel preconditioning with sparse approximate inverses, SIAM J. Sci. Comput. 18 (3) (1997) 838–853.
[72] P. Ramachandran, B. Zoph, Q.V. Le, Searching for activation functions, arXiv preprint, arXiv:1710.05941, 2017.
[73] T. Rüberg, F. Cirak, Subdivision-stabilised immersed b-spline finite elements for moving boundary flows, Comput. Methods Appl. Mech. Eng. 209–212 (2012) 266–283.
[74] S.J. Lind, B.D. Rogers, P.K. Stansby, Review of smoothed particle hydrodynamics: towards converged Lagrangian flow modelling, Proc. R. Soc. A 476 (2241) (2020) 20190801.
[75] J. Povala, I. Kazlauskaite, E. Febrianto, F. Cirak, M. Girolami, Variational Bayesian approximation of inverse problems using sparse precision matrices, Comput. Methods Appl. Mech. Eng. 393 (2022) 114712.
[76] T. Tripura, S. Chakraborty, Wavelet neural operator for solving parametric partial differential equations in computational mechanics problems, Comput. Methods Appl. Mech. Eng. 404 (2023).
[77] J.C. Strikwerda, Finite Difference Schemes and Partial Differential Equations, SIAM, 2004.
[78] R. Eymard, T. Gallouët, R. Herbin, Finite volume methods, Handb. Numer. Anal. 7 (2000) 713–1018.
[79] B.A. Finlayson, L.E. Scriven, The method of weighted residuals—a review, Appl. Mech. Rev. 19 (9) (1966) 735–748.
[80] W.C. Gibson, The Method of Moments in Electromagnetics, Chapman and Hall/CRC, 2021.