Deep Probabilistic Models for Forward and Inverse Problems in Parametric PDEs

Arnaud Vadeboncoeur†, Ömer Deniz Akyildiz*, Ieva Kazlauskaite†, Mark Girolami†,⋆, and Fehmi Cirak†

†Department of Engineering, University of Cambridge
⋆The Alan Turing Institute

av537@cam.ac.uk, odakyildiz@turing.ac.uk, {ik394,mag92,fc286}@cam.ac.uk

August 10, 2022

Abstract

We formulate a class of physics-driven deep latent variable models (PDDLVM) to learn parameter-to-solution (forward) and solution-to-parameter (inverse) maps of parametric partial differential equations (PDEs). Our formulation leverages the finite element method (FEM), deep neural networks, and probabilistic modeling to assemble a deep probabilistic framework in which the forward and inverse maps are approximated with coherent uncertainty quantification. Our probabilistic model explicitly incorporates a parametric PDE-based density and a trainable solution-to-parameter network while the introduced amortized variational family postulates a parameter-to-solution network, all of which are jointly trained. Furthermore, the proposed methodology does not require any expensive PDE solves and is physics-informed only at training time, which allows real-time emulation of PDEs and generation of inverse problem solutions after training, bypassing the need for FEM solve operations with comparable accuracy to FEM solutions. The proposed framework further allows for a seamless integration of observed data for solving inverse problems and building generative models. We demonstrate the effectiveness of our method on a nonlinear Poisson problem, elastic shells with complex 3D geometries, and integrating generic physics-informed neural networks (PINN) architectures. We achieve up to three orders of magnitude speed-ups after training compared to traditional FEM solvers, while outputting coherent uncertainty estimates.

1 Introduction

Partial differential equations (PDEs) are central pillars of many fields of science and engineering. These equations are typically nonlinear and depend on parameters which modulate their behaviour, 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 certain parameter and is generally termed the forward problem, for which an extensive number of solution techniques are available, including FEM, finite volumes, finite differences, and spectral methods [1]. 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 stiff, requiring the numerical methods to have high precision, resulting in heavy computational burden. On the other hand, given experimental data of a certain phenomenon, it is often of interest to infer the associated unknown parameters from experimental data to determine the parameter regimes of the observed system, a setting which is termed the inverse problem [2]. The 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 [3], and high-dimensional parameter spaces [4, 5].
Figure 1: Trained PDDLVM of a thin-walled flexible 3D shell bunny. The top row shows (a) the displacement field obtained with a forward FEM solve, (b) mean estimate provided by PDDLVM, (c) absolute error between (a) and (b), (d) standard deviation. In the bottom row, similarly, we have (e) the displacement field obtained from an inverse FEM solve, (f) mean estimate provided by PDDLVM, (g) absolute error, (h) standard deviation. The agreement between (g) and (h) demonstrates that when the difference between our method and an inverse FEM solve is larger, our uncertainty estimates are coherent and demonstrates the regions with larger error accurately. See Sec. 4.2 for more details.

More specifically, many modern engineering applications use expensive FEM solvers which in turn limits the amount of experimentation that is 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 set ups efficiently. For example, given a specific modelling problem (e.g. the thin-wall flexible shell problem) with known governing equations, a designer needs to find the best set of parameters (e.g. material properties, geometry, boundary conditions) to produce an item with desired properties (e.g. an item that does not buckle under a given load). An efficient forward solver allows to explore a large number of 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 require expensive Assemble and Solve operations. In addition, observational data may be available for existing designs (either from lab experiments or from simulations) and needs to be incorporated in the framework.

1.1 Contributions

Our goal in this paper is to solve the forward and inverse problems simultaneously, while at the same time building an interpretable latent space of PDE parameters and providing uncertainty quantification. To achieve this, we develop a probabilistic formulation and uniquely blend techniques from numerical solution of PDEs (specifically, FEM), 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 develop 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 (FEM) forward numerical solve operations, (iii) uses Bayesian view of regularizing the inverse problem to address their inherent ill-posedness, and (iv) enables us to incorporate various neural networks (NNs) tailored for parametric PDEs (such as PINNs [6], neural operators [7, 8, 9, 10]) 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 allows for any architecture to be converted into a probabilistic method. Our trained model allows for real-time solution of forward and inverse
problems, which is relevant to digital twins, e.g., in engineering design and structural health monitoring.

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., [11, 12, 13]). While early approaches were primarily based on training ML methods, such as random forests and deep NNs, on observed or simulated physical data [14, 15] and discovering governing physics laws from data [16], recent methods focus on ML approaches where physics is an integral part of the model. This is achieved through physics (PDE)-informed loss functions that combine the existing PDE-based parameterizations of the physical processes with observed/simulated data [17], or through differentiable physics [6, 18, 19, 20, 21]. The exceptional success of these methods brought tremendous attention to the emerging field of physics-informed machine learning [13, 22]. We summarize some of these advances and their relevance to our work below.

Physics-informed neural networks (PINN) and related methods. PINN-based methods [23, 6, 24, 25, 13, 26] convert the PDE formulation into a loss function with either hard or soft constraints [27, 28]. Most of these methods are based on point-wise evaluation of losses and may struggle to handle complex geometries; see, e.g., [29], 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 [8, 9, 30, 31, 32, 33] which may be unavailable or costly. Further extensions consider the unsupervised case [34], and couple the model with FEM [35].

Probabilistic approaches. Probabilistic formulations of PDE informed models vary from supervised approaches [36], to convolutional neural network (CNN) based uncertainty quantification procedures [37] to Gaussian processes combined with FEM [38, 39, 40, 41]. Most relevant to our work are [42, 43] where authors introduce the weighted residual method (which is a generalization of many numerical methods for PDEs [44, 45, 46, 47, 48, 49]), and treat the residual term as a pseudo-observation [43]. This approach, however, differs from our framework in that we learn the direct mappings between the parameters and the solutions, while [43] introduces an intermediate low-dimensional embedding which is not readily interpretable. Other related approaches consider flow-based probabilistic surrogates [50]. Another line of work integrates physical and deterministic models within variational autoencoders (VAEs), see, e.g., [51, 52, 53, 54, 55, 56]. Most relevant to our work is the physics-integrated VAE [53] which introduces physics knowledge into a VAE framework. However, our approach is markedly different, as we approximate both forward and inverse maps and propose an different variational formulation.

2 Technical Background

In this work, we are interested in PDEs defined on a domain $\Omega$ with boundary $\partial \Omega$, generalized as

\begin{align}
G_z(u, x) &= f(x), \quad \text{for} \quad x \in \Omega \subset \mathbb{R}^d, \\
B_z(u, x) &= 0, \quad \text{for} \quad x \in \partial \Omega,
\end{align}

where $d \in \{1, 2, 3\}$, $G_z$ and $B_z$ are nonlinear operators describing the PDE and its constraints respectively, $u(\cdot)$ is the solution field, $f(\cdot)$ is the source field, and $z$ is a vector of parameters of the PDE. Below, we describe a general way to derive numerical schemes and then introduce FEM.
2.1 The Weighted Residuals Method

The method of weighted residuals provides a general framework for deriving many well-known PDE discretisation techniques \[44, 47, 48, 49\]. Given the PDE of the form \([1]\), we first write

\[ \mathcal{R}(u, x) := \mathcal{G}_u(u, x) - f(x), \]

where \(\mathcal{R}(u, x)\) is the residual functional. Then, multiplying by some arbitrary weight function \(w(x)\) and integrating over the domain we obtain a weighted residual functional equation as

\[ \int_{\Omega} w(x) (\mathcal{G}_u(u, x) - f(x)) \, dx = \int_{\Omega} w(x) \mathcal{R}(u, x) \, dx. \]  \([3]\)

We then choose a set of arbitrary test functions \(\{w_i(x)\}_{i=0}^n\) and further pose that our solution field \(u(x)\) can be represented as a parameterized function, typically chosen as a weighted combination of linearly independent functions \(\hat{u}(x) = \sum_{j=0}^n u_j \phi_j(x)\). We denote the r.h.s. of \([3]\),

\[ r_i := \int_{\Omega} w_i(x) \mathcal{R}(\hat{u}, x) \, dx. \]  \([4]\)

We can then assemble the resulting system of equations into a vectorized form

\[ \begin{bmatrix} \int_{\Omega} w_0(x) (\mathcal{G}_u(\hat{u}, x) - f(x)) \, dx \\ \vdots \\ \int_{\Omega} w_m(x) (\mathcal{G}_u(\hat{u}, x) - f(x)) \, dx \end{bmatrix} = r. \]  \([5]\)

Setting \(r = 0\) and solving the equations for \(w_{0:j} = u\) solves the PDE. For the sake of clarity, in the foregoing derivation the boundary term \([1b]\) has been omitted.

We note the generality of this approach, as noted by \[43\], as different choices for the weight function \(w(x)\) results in different methods. Setting \(w_i(x) = \delta(x - x_i)\) we have a collocation method \[57\], and for \(w_i(x) = \delta_{ij}\) for \(x \in \Omega_j\) we obtain a subdomain (finite volume) method \[58\]. For \(w_i(x) = \partial \mathcal{R}(\hat{u}, x)/\partial u_j\) we have a least-squares method \[59\], for \(w_i(x) = x^{m-1}\) we have a moment method (used in, e.g., electromagnetics \[60\]). For \(w_i(x) = \phi_i(x)\) we obtain a Galerkin method \[61\]. This could be a spectral method if \(w_i(x) \neq 0\) for \(\forall x \in \Omega\) or FEM if \(w_i(x) \neq 0\) only for \(x \in \Omega_k \subset \Omega\). We also remark that this formulation can be used to describe the PINNs approach by choosing \(w_i(x) = \delta(x - x_i)\) and instead of choosing a trial solution as before we choose a composition of functions of the form \(\hat{u}(x) = (T_1 \circ \ldots \circ T_i \circ \ldots \circ T_0)(x)\) where \(T_i(x) = \sigma(W_i x + b_i)\).

While all of these methods are fully compatible with PDDLM, we choose a subclass of Galerkin methods called FEM which allows us to leverage the tools for discretization (meshing) and preconditioning to solve stiff equations \[62\], such as those arising in the modeling of elastic shells with complex geometries \[63, 64, 65\]. We choose \(w_i(x) = \phi_i(x)\), which are local polynomials joined on the mesh. The coefficients \(u_j\) on the boundaries are chosen to respect the (Dirichlet) boundary conditions. The mesh allows us to both construct a trial space as well as define the dimensionality of our domain. In order to apply the FEM on \([3]\), we use integration by parts to obtain a weak form \[66\]. Once discretized, this weak-form leads to a general system of nonlinear equations whose dimensionality depends on the mesh refinement and polynomial order of the trial function. We obtain \(u, f \in \mathbb{R}^d\), and the equation

\[ A_z(u) = f, \]  \([6]\)

where \(A_z : \mathbb{R}^n \to \mathbb{R}^n\) is a nonlinear map. Finding FEM coefficients \(u\) for fixed \(z\) is the forward problem. Eq. \([6]\) is solved typically using Newton iterations \[67\]. This is computationally intensive, even when \(A_z\) is linear, and requires a matrix inversion. Similarly, the problem of finding \(z\) given some components of \(u\) (the inverse problem) also necessitates nontrivial optimization due to the ill-posed nature of the problem. To resolve these issues, we introduce probabilistic deep models to directly approximate maps \(z \mapsto u\) and \(u \mapsto z\), while also providing uncertainty estimates.
3 Deep Probabilistic Models for Parametric PDEs

In this section, 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 solve operations (i.e. inverting the nonlinear map \( A_d \)). Building on this, we introduce in Sec. 3.2 a version of our framework which incorporates data.

### 3.1 The Model and Variational Family for Parametric PDEs Without Observed Data

Motivated by the weighted residual method, we introduce an auxiliary variable \( r_u \in \mathbb{R}^d \) which we refer to as the “residual” that is treated as a pseudo-observation of a residual between the right and left hand sides of the Eq. (5). We start by posing a likelihood of \( r_u = 0 \) by integrating out all other variables.

We pose the joint distribution as

\[
p(\rho_u | u, z, f) = \mathcal{N}(\rho_u; A_z(u) - f, \Sigma_u),
\]

where \( \Sigma_u \) is the covariance which calibrates the trust in the FEM, similar to a statistical FEM (statFEM) approach [39, 41, 68]. Treating \( r_u \) as observed and setting \( r_u = 0 \) as in Sec. 2.1 we aim to maximize its marginal likelihood \( p(r_u = 0) \) by integrating out all other variables.

We pose the joint distribution as

\[
p_\beta(r_u, u, z, f) = p(r_u | u, z, f)p_\beta(z | u, f)p(u)p(f),
\]

where \( p(u) \) and \( p(f) \) are prior distributions over FEM coefficients and source/force input respectively, and \( \beta \) is a trainable parameter in our model. More specifically, we choose deep probabilistic models of the form \( p_\beta(z | u, f) = \mathcal{N}(z; \mu_\beta(u, f), \Sigma_\beta(u, f)) \), where \( \mu_\beta \) and \( \Sigma_\beta \) are parameterized by neural networks with parameter \( \beta \). This model resembles the variational autoencoder (VAE) structure; however, we use a physics-informed model and formulate the parameter as the latent variable. The direct correspondence between latent variables and physical parameters enables us to have latent variables with clear physical interpretation. We also use a decoder-like distribution to learn the conditional prior of \( z \) given \( u \) which gives us flexibility to simulate \( z \) given a solution field \( u \) and forcing \( f \). To this end, we pose our variational approximation

\[
q_\alpha(u, z, f) = q_\alpha(u | z, f)p(z)p(f),
\]

where \( q_\alpha(u | z, f) = \mathcal{N}(u; \mu_\alpha(z, f), \Sigma_\alpha(z, f)) \) is a Gaussian parameterized by neural networks \( \mu_\alpha, \Sigma_\alpha \). Inspired by the weighted residual method introduced in Sec. 2.1 (see also [43]), we maximize the marginal likelihood of the residual by fixing \( r_u = 0 \) and deriving an evidence lower bound (ELBO) that lower bounds the log marginal likelihood \( \log p(r_u = 0) \). In particular, we have

\[
\log p(r_u = 0) = \log \int \frac{p_\beta(r_u = 0, u, z, f)}{q_\alpha(u, z, f)} q_\alpha(u, z, f) dz df.
\]
We demonstrate our methodology on four challenging examples below. In all experiments, we choose our \( \alpha \) and \( \beta \) networks (that parameterize \( q_\alpha(u|z, f) \) and \( p_\beta(z|u, f) \), respectively) have intuitive functionalities (see Fig. 2 for the graphical models). In particular, \( \alpha \) network provides a parameter-to-solution (forward) map while the \( \beta \) network provides a solution-to-parameter (inverse) map, both with associated uncertainty estimates.

3.2 The Model and Amortized Variational Family with Observed Field Data

We now introduce the \( \beta \) network providing the observation-to-solution map, both with associated uncertainty estimates. Our variational family in this case introduces a probabilistic model for a tuple \((y_i, u_i)\) as

\[
p(y_i, u_i) = p(y_i|u_i)p(u_i).
\]

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

\[
q_\phi(u_i|y_i) = \mathcal{N}(u_i; \mu_\phi(y_i), \Sigma_\phi(y_i)).
\]

Our variational family in this case introduces a \( \phi \)-network which aims at learning the observation-to-solution map and the density. In order to conduct inference, we obtain an ELBO to our marginal likelihood \( \log p(y_i) \) as

\[
\mathcal{F}_i(\phi) = \int \log \frac{p(y_i|u_i)p(u_i)}{q_\phi(u_i|y_i)} q_\phi(u_i|y_i) du_i.
\]

Given a dataset \((y_i)_{n=1}^n\), the full ELBO is \( \mathcal{F}(\phi) = \sum_{i=1}^n \mathcal{F}_i(\phi) \). Once trained, this map is sufficient to obtain an encoding model for observations. To realize the observation-to-parameter map, we can use the already learned \( \beta_* \)-network to marginalize over \( u \) as

\[
p(z|y, f) = \int q_\phi(u|y)p_\beta(z|u, f) du.
\]

The marginalization of \( z \) over \( u \) can also be easily extended to include the forcing \( f \).

3.3 The Algorithm and Implementation Details

**Priors and Monte Carlo ELBO.** In all formulations, we assume a flat prior \( p(u) \propto 1 \) for the solution field. We choose \( p(f) \) and \( p(z) \) as uniform distributions (see Sec. 4). We estimate the ELBOS in eqs. (10) using sampling from \( z \sim p(z) \) and \( f \sim p(f) \), \( u \sim q_\alpha(u|z,f) \) (using the reparameterization trick [71]) and optimize the ELBOS with Adam [70]. When multiple observations exist, i.e. if \( n \gg 1 \), we use only one-sample estimate of ELBO, resulting in a doubly-stochastic gradient. See Algorithm 1 for the full algorithm (see Appendix B for details). We also provide details in Sec. 4 where necessary.

4 Experiments

We demonstrate our methodology on four challenging examples below. In all experiments, we choose a diagonal covariance for the networks, although other choices can be explored [72]. Furthermore, we make use of a bounded re-parameterization of the log variance of the neural networks [72].

Using Jensen’s inequality and using eqs. (8), (9), we arrive at the lower bound

\[
\mathcal{F}(\alpha, \beta) = \int \log \frac{p(r_u = 0|u, z, f)p_\beta(z|u, f)p(u)}{q_\alpha(u|z, f)p(z)} q_\alpha(u|z, f)p(z) du dz df
\]

We note that our \( \alpha \) and \( \beta \) networks (that parameterize \( q_\alpha(u|z, f) \) and \( p_\beta(z|u, f) \), respectively) have intuitive functionalities (see Fig. 2 for the graphical models). In particular, \( \alpha \) network provides a parameter-to-solution (forward) map while the \( \beta \) network provides a solution-to-parameter (inverse) map, both with associated uncertainty estimates.

**The marginalization of**

\[
\int \mathcal{F}(\phi) = \sum_{i=1}^n \mathcal{F}_i(\phi)
\]

**Given a dataset**

\((y_i)_{n=1}^n\), the full ELBO is

\[
\mathcal{F}(\phi) = \sum_{i=1}^n \mathcal{F}_i(\phi)
\]

Once trained, this map is sufficient to obtain an encoding model for observations. To realize the observation-to-parameter map, we can use the already learned \( \beta_* \)-network to marginalize over \( u \) as

\[
p(z|y, f) = \int q_\phi(u|y)p_\beta(z|u, f) du.
\]

The marginalization of \( z \) over \( u \) can also be easily extended to include the forcing \( f \).
4.1 1D Nonlinear Poisson Problem

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

$$-\nabla \cdot (\eta(u, x) \nabla u(x)) = f(x),$$  \hspace{1cm} (16)

where

$$\eta(u, x) = \left( S \left( \frac{du(x)}{dx} \right) + 5\kappa(x) \right)/10,$$

for $x \in (-1, 1)$, where $u(-1) = \omega_l = 0, u(1) = \omega_r = 0.5,$ and $S(x) = 1/(1 + e^{-x})$ is the sigmoid function. For the current example we parameterize $\kappa(x)$ as a fourth order Chebyshev expansion with a softmax+1 transform. The output of the parameter-to-solution map is reparameterized with a 9th order Chebyshev expansion that is then projected onto the FE mesh.

Using (10) we learn the parameter-to-solution map and the solution-to-parameter map with uncertainties. To measure the accuracy of our method after training, we compute true FE solves, however, these are not used during training. The residual is marginalized over the prior of $z$ and $f$ which allow us to learn the PDE over the prior ranges. Here $z = \{\kappa, \omega\}$, and the marginalized priors are: $p(\kappa_i) = N(0, 1)$, $p(\omega_l) = U(0.5, 1)$, $p(\omega_r) = \delta(\omega - 0)$, and $p(f_i) = U(1, 2)$. The NN 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 $\Sigma_u = c_u^2 I$ with $c_u = 10^{-2}$. The model is trained for 1e6 iterations with an initial learning rate of $10^{-3}$ is halved every $2 \times 10^5$ iterations with Adam. In Fig. 3, the predictive distributions as well as the solutions are plotted.

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

|                  | $\alpha$-Net | $\beta$-Net |
|------------------|--------------|-------------|
| MNSE             | $6.82 \times 10^{-4}$ | $2.96 \times 10^{-3}$ |
| % truth in $2\sigma$ | 29.6%        | 87.0%       |

|                  | Forward | Inverse |
|------------------|---------|---------|
| MNSE NN          | 0.00277 | 0.00745 |
| avg. time NN     | 0.00346s | 0.0252s |
| avg. time FE     | 10.4s   |         |

4.2 Thin-Walled Flexible Shell

We use a differential geometric formulation combined with subdivision surface basis functions \[64, 73, 74\] to solve for the displacements of the 3D Stanford bunny \[65\] under self-weight – well suited to FEM as the mesh is the basis for discretization while encoding the geometry of the problem. To build solutions fields with a neural network in a mesh invariant manner, we reparameterize the output of the network with a vector field 3D Chebyshev expansion

$$u^i(x, y, z) = \sum_{l,m,n=0}^N U_{lmn}^i T_l(x) T_m(y) T_n(z),$$  \hspace{1cm} (17)

Figure 4: Thin-walled flexible shell bunny with PDDLVM.
where \( u = \text{vect}(U) \) and \( x, y, z \in \Omega \), where \( i \in \{1, 2, 3\} \) are the three spatial dimensions of the vector field of displacements. With the FEM formulation we can take advantage of well-known preconditioning techniques \cite{75} for iterative solvers, employing a simple preconditioner to greatly accelerate and stabilize convergence. This preconditioning is formulated as a reparametrization of the residual with the stiffness matrix of a representative sample \( z \) of \( p(x) \). We then have \( A_x r = A_x u - f \) where \( A_x \) is assembled with \( z \), the mean of \( p(z) \). For this example \( z \) are the parameters of \( \kappa(y) \), the Young’s modulus of the shell structure which is expressed as a second order Chebyshev expansion, which varies across the height of the shell to mimic material distribution control one might have in 3D printing contexts. Both for \( \alpha \) and \( \beta \) networks are fully connected, have 3 hidden layers of 2'500 neurons with “swish” activation functions \cite{76}. The PDDLVM was trained for 50'000 iterations for a total of 16.6 hours. Once trained we draw new samples of \( \kappa \) from its prior, we then use both the FEM model and the emulator to obtain the solution fields. Fig. 1 shows the trained PDDLVM’s predictions. The top row pertains to the forward operation, the bottom row focuses on the inverse mapping. We plot the FEM solutions, the predictive mean, the standard deviation, as well as the absolute error. Table 4(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 each operation for 100 independent sample from the prior \( p(z) \).

4.3 PINN-PDDLVM

To showcase the modularity of our framework, we implement PDDLVM with a continuous time PINN \[6\]. When combined with the PINN framework our solution field predictive distribution and the inverse distribution are formulated as

\[
q_\alpha(u|z, f) = N(\mu_\alpha(x_{0:m}, t_{0:m}, z, f), \text{diag}(\sigma_\alpha^2(x_{0:m}, t_{0:m}, z, f))), \\
p_\beta(z|u, f) = N(\mu_\beta(u), \Sigma_\beta(u)),
\]

where \( u \in \mathbb{R}^{nm} \). 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. We construct \( \Sigma_u = \text{diag}([\epsilon^2_{D_0}, \ldots, \epsilon^2_{D_\alpha}, \epsilon^2_{A_\beta}, \ldots, \epsilon^2_{D_\beta}, \epsilon^2_{A_\beta}, \ldots, \epsilon^2_{A_\beta}]) \) where each \( \epsilon \) de-

![Figure 5: Comparison of (a) PINN and (b) PDDLVM residuals as well as (c) average PDDLVM standard deviation over the solution domain. The white square denotes the bounds for the observed \( \kappa, \gamma \) during training as well as the bounds over which the mean residual is computed.](image)

Table 2: Comparison of PINN-PDDLVM model and a coupled PINN and convolutional inversion network. For details on the residual formulations, see Appendix C.4
where $\alpha$ is the unsupervised PINN-PDDLVM model. The parametric PDE in question is the inhomogenous wave equation,

$$\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} = \sum_{p} a_p x^p,$$

with an unsupervised PINN-PDDLVM model. We sampled 2500 combinations of $\kappa$, $\gamma$, and $\beta$ for 5 iterations. We then used this trained PINN to output solutions fields and create a supervised training dataset for both the standard PINN and PINN-PDDLVM-\beta networks. We trained the PDDLVM with marginalization over the prior $p(z) = U([1,1],[5,5])$ where $z = \{\kappa, \gamma\}$. In Fig. 5, we plot the log PINN residual for a range of PDE parameters $\kappa$ and $\gamma$ and we plot the log mean standard deviation of the $\alpha$ network.

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 2500 combinations of $\gamma$ and $\kappa$ between $[0,1,5] \times [0,1,0.5]$ and we trained a PINN-network for $5 \times 10^5$ iterations. We then used this trained PINN to output solutions fields and create a supervised training dataset for convolutional inverse network.

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

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

Notes: 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

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

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 PDE is parameterized with $\gamma$ which denotes the heat capacity of the material, and $\eta(u, \kappa)$ which is a temperature dependent conductivity. We train the PDDLVM with marginalization over the prior $p(z) = U([1,1],[5,5])$ where $z = \{\kappa, \gamma\}$. In Fig. 5, we plot the log PINN residual for a range of PDE parameters $\kappa$ and $\gamma$ and we plot the log mean standard deviation of the $\alpha$ network.

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 2500 combinations of $\gamma$ and $\kappa$ between $[0,1,5] \times [0,1,0.5]$ and we trained a PINN-network for $5 \times 10^5$ iterations. We then used this trained PINN to output solutions fields and create a supervised training dataset for convolutional inverse network. Both the PINN-inverse and PINN-PDDLVM-\beta 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 PINN-inverse network or the PINN-PDDLVM-\beta networks respectively. We train the PINN-Inverse network with a maximum likelihood type loss, specified as $\ell(x^*) = \frac{1}{N} \sum_{i=1}^{N} \| f_{NN}(x^*_i) - u_i \|^2$ minibatched one example at a time with Adam and a learning rate of $10^{-3}$.

4.4 Supervised FNO Comparison

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

$$\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} = \sum_{p} a_p x^p,$$

where $a_p$ are the PDE parameters and represent the forcing term parameters. In the current example we allow for 4 terms in the forcing expansion draw from $U(-5,5)$. The solutions for FNO datasets and the validation set are assembled and computed from samples of $a_p$ drawn from this distribution. The FNOs for
the forward and inverse directions have 4 hidden layers with 32 channels and 12 Fourier modes. We train the FNOs for $5 \times 10^4$ iterations for different number of training data. The training data is obtained from a Chebyshev based solver of $21 \times 21$ basis functions. We compare this with an unsupervised PINN-PDDLVM trained for $2.5 \times 10^5$ iterations with a similar architecture to that of Sec. 4.3.

5 Conclusions

PDDLVM is a probabilistic generative model for parametric PDEs that builds an interpretable latent space representation and provides uncertainty estimates on the predictions, which is of particular importance in inverse problems which are inherently ill-posed. The modular nature of the framework allows for an easy transition between meshless (e.g. PINNs) and mesh-based (e.g. FEM) problem statements, 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 preconditions) to solve a stiff set of equations of stresses and deformations in thin shells subject to gravity, providing estimates of the material properties and a framework for novel content generation.

While conventional FEM offers many advantages for solid mechanics problems, for other applications (such as fluid flows), other discretization techniques (such as immersed or particle-based methods [77, 78]) 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 [79].

Acknowledgements

A.V. was supported by the Baxter & Alma Ricard Foundation Scholarship. Ö. D. A. was supported by the Lloyd’s Register Foundation Data Centric Engineering Programme and EPSRC Programme 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.

References

[1] Arieh Iserles. A First Course in the Numerical Analysis of Differential Equations. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2 edition, 2008.
[2] Albert Tarantola. Inverse problem theory and methods for model parameter estimation. SIAM, 2005.
[3] Andrew M Stuart. Inverse problems: A Bayesian perspective. Acta Numerica, 19:451–559, 2010.
[4] Curtis R Vogel. Computational Methods for Inverse Problems. Frontiers in Applied Mathematics. Society for Industrial and Applied Mathematics, 2002.
[5] Simon Arridge, Peter Maass, Ozan Öktem, and Carola-Bibiane Schönlieb. Solving inverse problems using data-driven models. Acta Numerica, 28:1–174, 2019.
[6] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational physics, 378:686–707, 2019.
[7] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. arXiv preprint arXiv:2010.08895, 2020.
[8] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

[9] Anima Anandkumar, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Nikola Kovachki, Zongyi Li, Burigede Liu, and Andrew Stuart. Neural operator: Graph kernel network for partial differential equations. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

[10] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021.

[11] Jiequn Han, Arnulf Jentzen, and Weinan Ee. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115, 07 2017.

[12] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018.

[13] George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

[14] Nils Thuerey, Philipp Holl, Maximilian Mueller, Patrick Schnell, Felix Trost, and Kiwon Um. *Physics-based Deep Learning*. WWW, 2021.

[15] Karthik Raveendran, Chris Wojtan, Nils Thuerey, and Greg Turk. Blending liquids. *ACM Transactions on Graphics*, 33(4), jul 2014.

[16] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.

[17] Byungsoo Kim, Vinicius C Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler. Deep fluids: A generative network for parameterized fluid simulations. In *Computer Graphics Forum*, volume 38, pages 59–70. Wiley Online Library, 2019.

[18] Sungyong Seo and Yan Liu. Differentiable physics-informed graph networks. *arXiv preprint arXiv:1902.02950*, 2019.

[19] Christopher Rackauckas, Alan Edelman, Keno Fischer, Mike Innes, Elliot Saba, Viral B Shah, and Will Tebbutt. Generalized physics-informed learning through language-wide differentiable programming. 2021.

[20] Lynton Ardizzone, Jakob Kruse, Sebastian Wirkert, Daniel Rahner, Eric W Pellegrini, Ralf S Klessen, Lena Maier-Hein, Carsten Rother, and Ullrich Köthe. Analyzing inverse problems with invertible neural networks. *arXiv preprint arXiv:1808.04730*, 2018.

[21] Qingqing Zhao, David B Lindell, and Gordon Wetzstein. Learning to solve pde-constrained inverse problems with graph networks. *arXiv preprint arXiv:2206.00711*, 2022.

[22] Omar Ghattas and Karen Willcox. Learning physics-based models from data: perspectives from inverse problems and model reduction. *Acta Numerica*, 30:445–554, 2021.

[23] Maziar Raissi and George Em Karniadakis. Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, 357:125–141, 2018.

[24] Guofei Pang, Lu Lu, and George Em Karniadakis. fPINNs: Fractional physics-informed neural networks. *SIAM Journal on Scientific Computing*, 41(4):A2603–A2626, 2019.

[25] Zhiping Mao, Ameya D Jagtap, and George Em Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.
[26] Shengze Cai, Zhiping Mao, Zhicheng Wang, Minglang Yin, and George Em Karniadakis. Physics-informed neural networks (PINNs) for fluid mechanics: A review. *Acta Mechanica Sinica*, pages 1–12, 2022.

[27] Lu Lu, Raphael Pestourie, Wenjie Yao, Zhicheng Wang, Francesc Verdugo, and Steven G Johnson. Physics-informed neural networks with hard constraints for inverse design. *SIAM Journal on Scientific Computing*, 43(6):B1105–B1132, 2021.

[28] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.

[29] Han Gao, Matthew J Zahr, and Jian-Xun Wang. Physics-informed graph neural Galerkin networks: A unified framework for solving pde-governed forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 390:114502, 2022.

[30] Kaushik Bhattacharya, Bamdad Hosseini, Nikola B. Kovachki, and Andrew M Stuart. Model Reduction And Neural Networks For Parametric PDEs. *The SMAI journal of computational mathematics*, 7:121–157, 2021.

[31] Rui Wang, Karthik Kashinath, Mustafa Mustafa, Adrian Albert, and Rose Yu. Towards physics-informed deep learning for turbulent flow prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1457–1466, 2020.

[32] Ravi G Patel, Nathaniel A Trask, Mitchell A Wood, and Eric C Cyr. A physics-informed operator regression framework for extracting data-driven continuum models. *Computer Methods in Applied Mechanics and Engineering*, 373:113500, 2021.

[33] Fabian Kröpfl, Roland Maier, and Daniel Peterseim. Operator compression with deep neural networks. *Advances in Continuous and Discrete Models*, 2022(1):1–23, 2022.

[34] Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. *Science advances*, 7(40):eabi8605, 2021.

[35] Minglang Yin, Enrui Zhang, Yue Yu, and George Em Karniadakis. Interfacing finite elements with deep neural operators for fast multiscale modeling of mechanics problems. *Computer Methods in Applied Mechanics and Engineering*, page 115027, 2022.

[36] Yibo Yang and Paris Perdikaris. Conditional deep surrogate models for stochastic, high-dimensional, and multi-fidelity systems. *Computational Mechanics*, 64(2):417–434, 2019.

[37] Nick Winovich, Karthik Ramani, and Guang Lin. ConvPDE-UQ: Convolutional neural networks with quantified uncertainty for heterogeneous elliptic partial differential equations on varied domains. *Journal of Computational Physics*, 394:263–279, 2019.

[38] Jon Cockayne, Chris J Oates, Timothy John Sullivan, and Mark Girolami. Bayesian probabilistic numerical methods. *SIAM review*, 61(4):756–789, 2019.

[39] Mark Girolami, Eky Febrianto, Ge Yin, and Fehmi Cirak. The statistical finite element method (statfem) for coherent synthesis of observation data and model predictions. *Computer Methods in Applied Mechanics and Engineering*, 375:113533, 2021.

[40] Connor Duffin, Edward Cripps, Thomas Stemler, and Mark Girolami. Statistical finite elements for misspecified models. *Proceedings of the National Academy of Sciences*, 118(2), 2021.

[41] Ömer Deniz Akyildiz, Connor Duffin, Sotirios Sabanis, and Mark Girolami. Statistical Finite Elements via Langevin Dynamics. *arXiv preprint arXiv:2110.11131*, 2021.

[42] Sebastian Kaltenbach and Phaedon-Stelios Koutsourelakis. Incorporating physical constraints in a deep probabilistic machine learning framework for coarse-graining dynamical systems. *Journal of Computational Physics*, 419:109673, 2020.
[43] Maximilian Rixner and Phaedon-Stelios Koutsourelakis. A probabilistic generative model for semi-supervised training of coarse-grained surrogates and enforcing physical constraints through virtual observables. *Journal of Computational Physics*, 434:110218, 2021.

[44] Gilbert Strang and LB Freund. Introduction to applied mathematics. *Journal of Applied Mechanics*, 53(2):480, 1986.

[45] Snehashish Chakraverty, Nisha Mahato, Perumandla Karunakar, and Tharasi Dilleswar Rao. *Advanced numerical and semi-analytical methods for differential equations*. John Wiley & Sons, 2019.

[46] Mohammad Hatami. *Weighted residual methods: principles, modifications and applications*. Academic Press, 2017.

[47] Bruce A Finlayson. The method of weighted residuals and variational principles. *SIAM*, 2013.

[48] Lars-Erik Lindgren. From weighted residual methods to finite element methods, 2009.

[49] Curtis F Gerald. *Applied numerical analysis*. Pearson Education India, 2004.

[50] Yinhao Zhu, Nicholas Zabaras, Phaedon-Stelios Koutsourelakis, and Paris Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56–81, 2019.

[51] Daniel O’Malley, John K Golden, and Velimir V Vesselinov. Learning to regularize with a variational autoencoder for hydrologic inverse analysis. *arXiv preprint arXiv:1906.02401*, 2019.

[52] Daniel J Tait and Theodoros Damoulas. Variational autoencoding of PDE inverse problems. *arXiv preprint arXiv:2006.15641*, 2020.

[53] Naoya Takeishi and Alexandros Kalousis. Physics-integrated variational autoencoders for robust and interpretable generative modeling. *Advances in Neural Information Processing Systems*, 34:14809–14821, 2021.

[54] Zhaozhi Qian, William Zame, Lucas Fleuren, Paul Elbers, and Mihaela van der Schaar. Integrating expert odes into neural odes: Pharmacology and disease progression. *Advances in Neural Information Processing Systems*, 34:11364–11383, 2021.

[55] Weiheng Zhong and Hadi Meidani. PI-VAE: Physics-informed variational auto-encoder for stochastic differential equations. *arXiv preprint arXiv:2203.11363*, 2022.

[56] Christian Jacobsen and Karthik Duraisamy. Disentangling generative factors of physical fields using variational autoencoders. *Frontiers in Physics*, page 536, 2022.

[57] John C Strikwerda. *Finite difference schemes and partial differential equations*. SIAM, 2004.

[58] Robert Eymard, Thierry Gallouët, and Raphaèle Herbin. Finite volume methods. *Handbook of numerical analysis*, 7:713–1018, 2000.

[59] Bruce A Finlayson and Laurence E Scriven. The method of weighted residuals—a review. *Applied Mechanics Review*, 19(9):735–748, 1966.

[60] Walton C Gibson. *The method of moments in electromagnetics*. Chapman and Hall/CRC, 2021.

[61] Alexandre Ern and Jean-Luc Guermond. *Theory and practice of finite elements*, volume 159. Springer, 2004.

[62] Gustaf Söderlind, Laurent Jay, and Calvo Manuel. Stiffness 1952-2012. Sixty years in search of a definition. 55(2):531–558, 2015.

[63] Qiaoling Zhang and Fehmi Cirak. Manifold-based isogeometric analysis basis functions with prescribed sharp features. *Computer Methods in Applied Mechanics and Engineering*, 359:112659, 2020.
[64] Fehmi Cirak, Michael Ortiz, and Peter Schröder. Subdivision surfaces: a new paradigm for thin-shell finite-element analysis. *International Journal for Numerical Methods in Engineering*, 47(12):2039–2072, 2000.

[65] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’94, New York, NY, USA, 1994.

[66] Susanne C Brenner, L Ridgway Scott, and I. Ridgway Scott. *The mathematical theory of finite element methods*, volume 3. Springer, 2008.

[67] Peter Wriggers. *Nonlinear finite element methods*. Springer Science & Business Media, 2008.

[68] Connor Duffin, Edward Cripps, Thomas Stemler, and Mark Girolami. Low-rank statistical finite elements for scalable model-data synthesis. *Journal of Computational Physics*, page 111261, 2022.

[69] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[70] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[71] Diederik P Kingma and Max Welling. An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*, 2019.

[72] David Dehaene and Rémy Brossard. Re-parameterizing vaes for stability. *arXiv preprint arXiv:2106.13739*, 2021.

[73] Fehmi Cirak and Michael Ortiz. Fully $C^1$-conforming subdivision elements for finite deformation thin-shell analysis. *International Journal for Numerical Methods in Engineering*, 51(7):813–833, 2001.

[74] Qiaoling Zhang, Malcolm Sabin, and Fehmi Cirak. Subdivision surfaces with isogeometric analysis adapted refinement weights. *Computer-Aided Design*, 102:104–114, 2018.

[75] Marcus J Grote and Thomas Huckle. Parallel preconditioning with sparse approximate inverses. *SIAM Journal on Scientific Computing*, 18(3):838–853, 1997.

[76] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[77] Thomas Rüberg and Fehmi Cirak. Subdivision-stabilised immersed b-spline finite elements for moving boundary flows. *Computer Methods in Applied Mechanics and Engineering*, 209–212:266–283, 2012.

[78] Steven J Lind, Benedict D Rogers, and Peter K Stansby. Review of smoothed particle hydrodynamics: towards converged lagrangian flow modelling. *Proceedings of the Royal Society A*, 476(2241):20190801, 2020.

[79] Jan Povala, Ieva Kazlauskaite, Eky Febrianto, Fehmi Cirak, and Mark Girolami. Variational Bayesian approximation of inverse problems using sparse precision matrices. *Computer Methods in Applied Mechanics and Engineering*, 393:114712, 2022.

[80] Eric A Wan and Rudolph 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*, pages 153–158. IEEE, 2000.
Appendix

A The Model and ELBO Derivations

A.1 The Model and ELBO Derivation for Sec. 3.1

As summarized in Sec. 3.1, we have the model with the residual
\[ p(r_u|u, z, f) = N(r_u; A_u(u) - f, \Sigma_u). \]  

(21)

The key idea behind this model is to treat \( r_u \) as observed data (i.e., as virtual observations [43] or pseudo-data [39]) and to set \( r_u = 0 \). Setting the residual to zero is equivalent to solving the PDE and is as explained in Sec. 2.1 used to derive many PDE solvers [44, 45, 46, 47, 48, 49]. As a result, we formulate the inference as the maximization of \( r_u = 0 \) by maximizing its marginal likelihood \( p(r_u = 0) \).

Recall that our model is
\[ p_\beta(r_u, u, z, f) = p(r_u|u, z, f)p_\beta(z|u, f)p(u)p(f), \]  

(22)

where \( p(u) \) and \( p(f) \) are prior distributions over FEM coefficients and source/force input respectively, and \( p_\beta(z|u, f) = N(z; \mu_\beta(u, f), \Sigma_\beta(u, f)) \), where \( \mu_\beta \) and \( \Sigma_\beta \) are parameterized by neural networks with parameters \( \beta \). Our variational approximation is
\[ q_\alpha(u, z, f) = q_\alpha(u|z, f)p(z)p(f), \]  

where \( q_\alpha(u|z, f) = N(u; \mu_\alpha(z, f), \Sigma_\alpha(z, f)) \) is a multivariate Gaussian parameterized by neural networks \( \mu_\alpha, \Sigma_\alpha \). As noted, inspired by the weighted residual method introduced in Sec. 2.1, we maximize the marginal likelihood of the residual by fixing \( r_u = 0 \) and deriving an evidence lower bound (ELBO) that bounds the log marginal likelihood \( \log p(r_u = 0) \). In order to do this, we first note
\[ p(r_u = 0) = \int p_\beta(r_u = 0, u, z, f)dudzdf, \]  

(23)

where the quantity \( p(r_u = 0) \) acts as “evidence”. Our aim is to maximize this quantity. To this end, we introduce an instrumental variational approximation and rewrite (23) as
\[ p(r_u = 0) = \int \frac{p_\beta(r_u = 0, u, z, f)}{q_\alpha(u, z, f)}q_\alpha(u, z, f)dudzdf, \]  

by dividing and multiplying the integrand in (23) by \( q_\alpha(u, z, f) \). Next, we compute the logarithm of this quantity, as maximizing the \( \log p(r_u = 0) \) is equivalent to maximizing \( p(r_u = 0) \) and more stable numerically, to obtain:
\[ \log p(r_u = 0) = \log \int \frac{p_\beta(r_u = 0, u, z, f)}{q_\alpha(u, z, f)}q_\alpha(u, z, f)dudzdf. \]

Using Jensen’s inequality and (8), (9), we arrive at the lower bound
\[ F(\alpha, \beta) = \int \log \frac{p(r_u = 0|u, z, f)p_\beta(z|u, f)p(u)}{q_\alpha(u|z, f)p(z)}q_\alpha(u|z, f)p(z)p(f)dudzdf. \]  

(24)

Modern variational inference techniques rely on maximizing this lower bound, instead of the true evidence \( \log p(r_u = 0) \) to learn parameters \( (\alpha, \beta) \). The \( N \)-sample Monte Carlo ELBO estimate of the gradient of \( F \)
can be obtained by sampling from \( p(z), p(f) \) and \( q_{\alpha}(u|z, f) \) (using the reparameterization trick). For this, we rewrite ELBO (24) as a generic integral
\[
F(\alpha, \beta) = \mathbb{E}_{q_{\alpha}(u|z, f) p(z)} \left[ \nabla_{\alpha, \beta}(u, z, f) \right],
\]
where
\[
\nabla_{\alpha, \beta}(u, z, f) = \log \frac{p(r_u = 0|u, z, f)p_z(z|u, f)p(u)}{q_{\alpha}(u|z, f)p(z)}.
\]

Given that \( \varepsilon \sim \mathcal{N}(0, 1) \), we can sample \( u \sim q_{\alpha}(u|z, f) \) by sampling \( \varepsilon \) and computing
\[
w_{\alpha}(z, f, \varepsilon) = \mu_{\alpha}(z, f) + \Sigma^{1/2}_{\alpha}(z, f)\varepsilon,
\]
and write
\[
F(\alpha, \beta) = \mathbb{E}_{q_{\varepsilon}(\varepsilon)p(z)} \left[ \nabla_{\alpha, \beta}(w_{\alpha}(z, f, \varepsilon), z, f) \right].
\]

To construct the Monte Carlo ELBO, we now sample \( \varepsilon^{(j)}, z^{(j)}, f^{(j)} \sim q(\varepsilon)p(z)p(f) \) for \( j = 1, \ldots, N \) and obtain
\[
F^N(\alpha, \beta) = \frac{1}{N} \sum_{j=1}^{N} \nabla_{\alpha, \beta}(w_{\alpha}(z^{(j)}, f^{(j)}, \varepsilon^{(j)}, z^{(j)}, f^{(j)})�).
\]

The rest of the training is done by computing the derivatives of this stochastic loss w.r.t. \( \alpha, \beta \) and running a variant of gradient descent, such as Adam [70]. In many practical applications (and in this paper), \( N = 1 \), i.e., a one-sample estimate of the gradient is utilized.

### B Algorithmic Details

**NN parameterization.** For a FEM discretization we have
\[
r = A_u^F(u^F) - f,
\]
where \( u^F \) would be the coefficients of the trial solution \( \hat{u}(x) \) in the finite element expansion. These expansions tend to be very high dimensional which can make their use with NNs difficult. Furthermore, the finite element expansion is discretization dependent, so a new mesh would require a new NN architecture. To mitigate both these issues we parameterize the coefficients \( u^F \) with the mapping
\[
u^F_i = P_u(u) = \sum_{k=0}^{N_u} u_k T_k(x_i),
\]
where \( u \in \mathbb{R}^N \) is sampled from the variational distribution \( q_{\alpha}(u|z, f) \). We then make the nonlinear operator \( A_x = (A_u^F \circ P_u)(u) \) responsible for using (26) and project the spectral expansion coefficients \( u \) to the finite element coefficient expansion \( u^F \) and we obtain
\[
r = A_x(u) - f.
\]

We parameterize the \( \kappa(x) \) field in a similar way, adding a composition with a non-linear transformation \( \sigma(\cdot) \) to impose desired characteristics such as positivity,
\[
z^F_i = P_{\kappa}(z) = \sigma \left( \sum_{k=0}^{N} z_k T_k(x_i) \right).
\]

We choose \( T_k(x) \) to be the \( k \)th order Chebyshev polynomial of the first kind and \( x_i \) is the location of \( u^F \) on the FEM mesh. 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.

The function $\sigma(\cdot)$ allows us to ensure certain properties in the reparameterized variables such as positivity or constraining within a range. For $\kappa$, the coefficients of the $\kappa(x)$ expansion, $\sigma(\cdot)$ is the softplus transformation. When the transformation $\sigma(\cdot)$ is linear, we have an exact expression for the probability of $u$ at any $x_i$. When it is nonlinear, we make use of the unscented transform [80] for estimation of the first and second moments of the probability density. In the case where we project $u$ in Chebyshev coefficients onto the finite element mesh as $u^e$ we can transform its probability distributions as $p(u^e) = N(\bar{u}^T, T_\Sigma u^T)$, where $T$ is the Chebyshev Vandermond matrix at the finite element nodes.

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 a commercial FEM package.

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

$$
MNSE(x_{0,N}^*,x_{0,N}) := \frac{1}{N} \sum_{i}^{N} \frac{\| x_{i}^* - x_i \|^2}{\| x_i^* \|^2},
$$

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

\section{Experimental Details}

All experiments where run on an AMD Ryzen 9 5950X 16-Core Processor CPU and an NVIDIA GeForce RTX 3090 GPU. For all experiments TensorFlow GPU usage was limited to certain amounts specified in each experiment’s section. The Poisson 1D examples are run on the CPU rather then the GPU as the FEM code is sequential and runs faster on the GPU. 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.

\subsection{Linear PDE}

We use the formulation in Sec. 3.1 to train the $\alpha$ and $\beta$ networks on a linear Poisson equation. The formulation mirrors that of Sec. 4.1 but here $\eta(x) = \kappa(x)$. The diffusivity field and forcing is parameterized as a constant function and the solution field is given by a 3rd order Chebyshev expansion. The prior over $\kappa$ is $\mathcal{N}(0,0.5)$. We use a NN architecture of 50 nodes for 3 hidden layers. We use a “swish” activation function. We train each model for $2 \times 10^5$ iterations with Adam and a learning rate of $10^{-3}$ decayed by half 10 times over the training time. The joint training of the $\alpha$ and $\beta$ networks takes 218 seconds.

\begin{figure}[h!]
    \centering
    \includegraphics[width=\textwidth]{figure7.png}
    \caption{(a) FE Solve \hspace{1cm} (b) PDDIUM Forward \hspace{1cm} (c) True $\kappa(x)$ \hspace{1cm} (d) PDDIUM Inverse}
    \end{figure}

In Fig. [8] we plot the mean normalized squared error for 100 independent draws of $\kappa$ from various ranges of values for which $p(\kappa) = \mathcal{N}(0,0.5)$. The error for the proposed solution field is computed with respect to a true FEM solve. For the reconstructed quantities $\kappa$ and $\omega$, we compare the generating sample.
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 split the ranges of $\kappa$ in the range of $[-2, 2]$ in 0.5 increments. In Fig. 9 we plot the same quantities as in Fig. 8 but in the log scales. We show the

Figure 8: MNSE for different $\epsilon$ values for the ranges of $\kappa$ given on the x-axis.

MNSEs for a variety of parameters of $\epsilon_u$. In Fig. 10 we plot the training of the ELBO for various settings of

Figure 10: The training of the ELBO for a 1D linear Poisson equation for various choices of $\epsilon_u$. We notice that the smaller the choice of $\epsilon_u$, the slower the convergence, but the higher the accuracy.

18
C.2 Observable Map Inversion

In this example we make use of Sec. 3.2 a set of 100 observations with a noise of $\sigma_y = 0.01$ and a non-invertible observation map which truncates the middle 20 observation points of $u^F$. The $\alpha_\ast$ and $\beta_\ast$ networks are pretrained on the linear Poisson equation with $\kappa(x)$ expressed as a 4 term Chebychev expansion, variable boundary conditions, and variable constant function forcing. The model $q_\phi$ reconstructs $u$ as Chebychev coefficients from the noisy, partially observed $y$. We show the results of inference with 5 new data observations not included in the dataset. To quantify the accuracy of this model we draw 100 independent samples from $p(z)$ and $p(f)$. We then compute the MSNE for those 100 independent draws. The computed MNSE for $u$ projected on the FE grid is of $5.64 \times 10^{-4}$ and for $z$ (the $\kappa$ field) projected on the FE mesh it is of $8.35 \times 10^{-3}$.

C.3 3D Thin-Walled Flexible Shell

We model the thin elastic shell as a surface, i.e. 2-manifold, embedded in 3D space. Consequently, its governing equations are best expressed in a surface-bound curvilinear coordinate system with two coordinates, see e.g. [64].

In the following, Greek indices take the values $\{1, 2\}$, covariant differentiation of a tensor field on the surface with respect to the coordinate $\alpha$ is denoted by $|\alpha$, and the summation convention over repeated indices applies. The equilibrium equations for the membrane forces $n^{\alpha\beta}$ and moments $m^{\alpha\beta}$ of the shell surface with the second fundamental form $b^\gamma_{\alpha\beta}$ and prescribed external forces $f^\alpha$ and $f^\beta$ are given by

\begin{align}
-(n^{\alpha\beta} + b^\gamma_{\alpha\beta} m^{\gamma\beta})|\beta - b^\gamma_{\alpha\beta} m^{\gamma\beta}|\beta &= f^\alpha, \\
m^{\alpha\beta}|_{\alpha\beta} - b^\gamma_{\alpha\beta} b_{\gamma\beta} m^{\alpha\beta} - b_{\alpha\beta} n^{\alpha\beta} &= f^\beta.
\end{align}

(30a) (30b)

The membrane force and moment tensors are defined as

\begin{align}
n^{\alpha\beta} &= \ell a^{\alpha\beta\gamma\tau} \gamma_{\alpha\tau}, \\
m^{\alpha\beta} &= \ell a^{\alpha\beta\gamma\tau} \rho_{\alpha\tau},
\end{align}

(31a) (31b)

where $\ell$ is the thickness of the shell and $a^{\alpha\beta\gamma\tau}$ is a fourth order tensor depending on the two elastic material constants, the Young’s modulus and the Poisson’s ratio. The membrane strain and bending strain tensors $\gamma_{\alpha\tau}$ and $\rho_{\alpha\tau}$ of a shell with the deflection $u$ are given by

\begin{align}
\gamma_{\alpha\tau} &= \frac{1}{2} [a_{\alpha\beta}(u) - a_{\alpha\beta}]^{\text{lin}}, \\
\rho_{\alpha\tau} &= [b_{\alpha\beta}(u) - b_{\alpha\beta}]^{\text{lin}}.
\end{align}

(32) (33)

Here, $a_{\alpha\beta}$ and $b_{\alpha\beta}$ are the standard first and second fundamental forms of the surface. The two strain tensors measure the change in the first and second fundamental forms in the original and deflected surfaces. Furthermore, the tensors $a_{\alpha\beta}(u)$ and $b_{\alpha\beta}(u)$ for the deflected surface depend nonlinearly on the deflection $u$ and are linearised. Finally, in deriving the finite element equations for the shell it is usually more expedient to discretise directly the corresponding energy functional rather than to start with the above differential equations, see [64] for the details.

We use a 3D Chebychev immersion to reduce the number of degrees of freedom used to describe a solution. The vector field $u^{ijk}(x, y, z)$ denotes the displacement on the surface of the geometry and is...
We then substitute for our definition of \( w \) and compute the PINN-style residual as
\[
R_h(\gamma, \kappa) = \frac{1}{NM} \sum_{i} \sum_{j} \left( -\frac{\partial h(x_i, t_j)}{\partial t} + \frac{1}{\gamma} \nabla \cdot (\eta(h(x_i, t_j), \kappa) \nabla h(x_i, t_j)) \right)^2 + \frac{1}{n} \sum_{k} (h(x_k, 0) - u(x_k, 0))^2 + \frac{1}{m} \sum_{l} \left[ (h(0, t_l) - u(0, t_l))^2 + (h(2\pi, t_l) - u(2\pi, t_l))^2 \right].
\]  

This residual can be constructed from the weighted residual method as a method of least-squares-collocation \[47\]. This is done by choosing \( w_i(x) = \frac{\partial R(u, x)}{\partial W} \delta(x_i - x) \) where \( N_1, N_2, N_3 \) are the number of points in \( \{\Omega, \Gamma, I\} \) 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 \[59\] we motivate the least-squares method as
\[
r_i = \int_{\Omega} w_i(x) R_{\Omega}(\hat{u}, x) dx + \int_{\Gamma} w_i(x) R_{\Gamma}(\hat{u}, x) dx + \int_{I} w_i(x) R_{I}(\hat{u}, x) dx.
\]  

We then substitute for our definition of \( w_i(x) = \frac{\partial R(u, x)}{\partial W} \delta(x_i - x), \)
\[
r_i = \int_{\Omega} \frac{\partial R_{\Omega}}{\partial W} \delta(x_i - x) R_{\Omega}(\hat{u}, x) dx + \int_{\Gamma} \frac{\partial R_{\Gamma}}{\partial W} \delta(x_i - x) R_{\Gamma}(\hat{u}, x) dx + \int_{I} \frac{\partial R_{I}}{\partial W} \delta(x_i - x) R_{I}(\hat{u}, x) dx.
\]  

We can then factorize out the partial derivative w.r.t. \( W \) and the factor of one-half,
\[
r_i = \frac{1}{2} \frac{\partial}{\partial W} \left[ \int_{\Omega} \delta(x_i - x) R_{\Omega}^2(\hat{u}, x) dx + \int_{\Gamma} \delta(x_i - x) R_{\Gamma}^2(\hat{u}, x) dx + \int_{I} \delta(x_i - x) R_{I}^2(\hat{u}, x) dx \right].
\]  

Since \( r_i \) is set to zero for all \( i \), then we have a stationary point which must be zero. The residual for an \( x_i \) outside the definition of the residual’s defined domain (\( \Omega, \Gamma, \) or \( I \)) is zero. We have
\[
I_i(W) = R_{\Omega}^2(\hat{u}, x_i) + R_{\Gamma}^2(\hat{u}, x_i) + R_{I}^2(\hat{u}, x_i).
\]
If we average over the points in each residual domain and assemble into a single objective, we obtain

\[
I(W) = \frac{1}{N_1} \sum_{x_i \in \Omega} R^2_{\Omega}(\hat{u}, x_i) + \frac{1}{N_2} \sum_{x_i \in \Gamma} R^2_{\Gamma}(\hat{u}, x_i) + \frac{1}{N_3} \sum_{x_i \in I} R^2_{I}(\hat{u}, x_i),
\]

(39)

which is the commonly used training objective for PINNS-type models. In Fig. 12 we plot \(q_{\alpha}(u|z, f)\) trained on the time dependent heat equation (19) for samples \(z, f\) draw from their respective prior distributions.