Amortized Finite Element Analysis for Fast PDE-Constrained Optimization

Tianju Xue [1] **Alex Beatson** [2] **Sigrid Adriaenssens** [1] **Ryan P. Adams** [2]

Abstract

Optimizing the parameters of partial differential equations (PDEs), i.e., PDE-constrained optimization (PDE-CO), allows us to model natural systems from observations or perform rational design of structures with complicated mechanical, thermal, or electromagnetic properties. However, PDE-CO is often computationally prohibitive due to the need to solve the PDE—typically via finite element analysis (FEA)—at each step of the optimization procedure. In this paper we propose amortized finite element analysis (AmorFEA), in which a neural network learns to produce accurate PDE solutions, while preserving many of the advantages of traditional finite element methods. This network is trained to directly minimize the potential energy from which the PDE and finite element method are derived, avoiding the need to generate costly supervised training data by solving PDEs with traditional FEA. As FEA is a variational procedure, AmorFEA is a direct analogue to popular amortized inference approaches in latent variable models, with the finite element basis acting as the variational family. AmorFEA can perform PDE-CO without the need to repeatedly solve the associated PDE, accelerating optimization when compared to a traditional workflow using FEA and the adjoint method.

1. Introduction

Partial differential equations (PDEs) are widely used to describe the properties of physical systems, including heat transfer, electromagnetics, and elasticity. PDE-constrained optimization (PDE-CO) addresses the situation in which an objective function must be minimized or maximized, subject to the constraints of real-world physics as expressed by PDEs. Common examples include optimal design, optimal control, and the identification of parameters to relate a simulation to observed data (Rees et al., 2010; Ulbrich & van Bloemen Waanders, 2018). PDE-CO can be computationally challenging, however, for even moderately sized problems, as it is usually necessary to solve the associated PDE at every iteration of the outer-loop optimization.

As a motivating example, consider a heat conduction problem on a unit disk. The physical system is governed by a Poisson's equation:

$$\begin{aligned} -\Delta u &= \lambda \quad \text{in } \Omega, \\ u &= u_b \quad \text{on } \Gamma, \end{aligned} \tag{1}$$

where $\Delta = \frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2}$ is the Laplace differential operator, $u(x)$ is the temperature field and $\lambda(x)$ is the heat source field (see Fig. 1). Finite element analysis (FEA) is arguably the most powerful method known for computing numerical solutions to this kind of PDE problems. With a given source field $\lambda(x)$, FEA identifies the best approximate solution field $u(x)$ in a piece-wise polynomial function space (Hughes, 2012).

PDE-CO poses a higher level problem, seeking to optimize an objective functional jointly over $u(x)$ and $\lambda(x)$, under the constraint imposed by the governing PDE. In the example problem above, one may reasonably ask: how can we design a source field $\lambda(x)$ so that a desired temperature field $u_d(x)$ is fulfilled while the cost of $\lambda(x)$ is minimized? Such PDE-CO problems are typically high-dimensional (e.g., # of input parameters $= 811$ in the model problem)

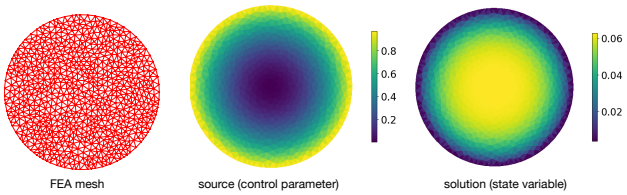

Figure 1. The heat equation on a disk of unit radius. Left: the finite element mesh. Middle: an example source field $\lambda(x) = x_1^2 + x_2^2$. Right: the solution field $u(x)$ associated with the source field solved by FEA. $x = (x_1, x_2) \in \mathbb{R}^2$ denote the spatial coordinates.

[1]Department of Civil and Environmental Engineering, Princeton University, Princeton, NJ, USA [2]Department of Computer Science, Princeton University, Princeton, NJ, USA. Correspondence to: Tianju Xue <txue@princeton.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

and often require iterative procedures that must solve the governing PDE repeatedly using FEA.

We propose a two-stage optimization framework to efficiently tackle sequences of related PDE-constrained optimization problems. At the first stage, we introduce amortized finite element analysis (AmorFEA) to efficiently learn the physics governed by the PDE without requiring supervised data provided by expensive PDE solvers. We borrow the idea of *amortized optimization*, widely used in amortized variational inference (Kingma & Welling, 2013; Gershman & Goodman, 2014; Ravi & Beatson, 2018; Choi et al., 2019). By learning to jump directly to the FEA solution with a neural network, we obtain a surrogate model that is able to predict the solutions directly from the control parameters. At the second stage, we perform gradient-based PDE-CO using the AmorFEA-enabled neural surrogate model. During each optimization iteration, the gradient is efficiently computed via one forward and one backward pass through the neural network, instead of querying an expensive PDE solver as in the traditional adjoint method requires (Cao et al., 2003).

1.1. Related Work

Deep Learning for Solving PDEs The idea of approximating the solution to the PDE by a neural network instead of a piece-wise polynomial function (as in FEA) goes back decades (Lagaris et al., 1998) and has been continuously explored since then (Lopez et al., 2008; Weinan & Yu, 2018; Lu et al., 2019). However, neural networks have only generally shown advantages on high-dimensional PDEs when the finite element mesh is infeasible, as shown by Sirignano & Spiliopoulos (2018). Recently, Zhu et al. (2019) proposed to predict PDE solutions directly from parameter fields by training neural networks with a physically-constrained loss. Since their method essentially integrates with the finite *difference* method (LeVeque, 2007), it only handles structured data in regular domains. Our proposed method, AmorFEA, integrates with the more powerful finite *element* method and benefits from many of the advantages of FEA, such as native support of irregular domains and approximation guarantees (Hughes, 2012).

PDE-constrained Optimization The two main workhorses for PDE-constrained optimization problems are the *all-at-once* approach and the *reduced* approach (Herzog & Kunisch, 2010). The all-at-once approach simultaneously optimizes over both the control parameters and the state variables with a strategy such as sequential quadratic programming (Dennis et al., 1998). Though attractive from an optimization perspective, they are infeasible for large-scale problems with a huge number of state variables to store (van Leeuwen & Herrmann, 2015). The reduced approaches treat the state variables as implicit functions of the control parameters and optimize only over the control parameters. When the control parameters significantly outnumber the objectives (which is often the case), adjoint sensitivity analysis becomes the dominant method under the reduced approaches (Errico, 1997; Cao et al., 2003). In spite of its relative success, the adjoint method requires solutions to the original PDE and the adjoint PDE during each optimization iteration, which is still expensive.

Optimization Using Surrogate Models AmorFEA-based PDE-CO is a surrogate model optimization approach. In optimization problems where the objective function is expensive to evaluate, it is popular to build a surrogate model and perform optimization on the surrogate model instead (Kochenderfer & Wheeler, 2019). There are various approaches for building the surrogate model such as random forests (Criminisi et al., 2011), Gaussian processes (Shahriari et al., 2015) and Student-t processes (Shah et al., 2014). AmorFEA builds a neural network surrogate model for optimization. Although some previous work has used neural network surrogates (e.g., Snoek et al. (2015)), AmorFEA takes a further step to integrate tightly with finite element analysis.

1.2. Contributions

This paper proposes AmorFEA, a framework for training surrogate PDE solvers for finite element analysis, without supervised training data. As AmorFEA is based on an amortized approach to FEA, it inherits the benefits of naturally handling irregular domains and unstructured meshes. As FEA is a variational formalism, we are able to draw useful connections to variational inference tools developed for probabilistic reasoning—AmorFEA inherits FEA's expressive basis, analagous to performing amortized inference with a rich variational family. The speed and differentiability of AmorFEA make it particularly well-suited to PDE-constrained optimization problems and we show that it can outperform the traditional adjoint method in terms of computation time, while still providing solutions of comparable quality. We share and publish our code at `https://github.com/tianjuxue/AmorFEA`.

2. Amortized Finite Element Analysis

Many physical systems governed by PDEs obey variational principles. For a given control field λ (e.g., heat source), the true solution field u (e.g., temperature) is the one that minimizes the total potential energy of the system. Finite element analysis is an approach in which the domain is discretized into a finite set of elements, and the solution u is approximated by a piece-wise polynomial function. This approximation allows us to use a vector $\boldsymbol{u} \in \mathbb{R}^n$ to represent

the FEA solution, since the piece-wise polynomial function space is finite-dimensional and forms an isomorphism with $\mathbb{R}^n$. For a full description of FEA, see Appendix A.

In a FEA problem, given a fixed control vector $\boldsymbol{\lambda} \in \mathbb{R}^m$, we find the solution vector (or the state vector) $\boldsymbol{u} \in \mathbb{R}^n$ by solving an optimization problem:

$$\min_{\boldsymbol{u} \in \mathbb{R}^n} \mathcal{L}(\boldsymbol{u}, \boldsymbol{\lambda}), \tag{2}$$

where $\mathcal{L}(\boldsymbol{u}, \boldsymbol{\lambda})$ denotes the total potential energy.

Our amortized finite element analysis (AmorFEA) approach reframes the per-control-vector optimization process into a shared regression problem. We use a neural network to build a deterministic mapping $g_\psi : \mathbb{R}^m \to \mathbb{R}^n$, whose weights ψ are learned by minimizing the expected potential energy:

$$\min_{\psi} \mathbb{E}_{p(\boldsymbol{\lambda})}[\mathcal{L}(g_\psi(\boldsymbol{\lambda}), \boldsymbol{\lambda})], \tag{3}$$

where $p(\boldsymbol{\lambda})$ is the distribution of typical control parameters associated with specific problems and we hope that $\boldsymbol{u} \approx \widehat{\boldsymbol{u}} = g_\psi(\boldsymbol{\lambda})$. Traditional FEA forces the trial solution to be correct by committing to a class of test functions. In a similar sense, AmorFEA forces the correctness of the model by considering a distribution over control parameters.

Since FEA is an approximate variational procedure (in the general sense of minimizing a functional), it resembles variational inference in latent variable models in that the finite element basis functions are analogous to the variational family of distributions and the FEA solution vector mirrors the variational parameters. It can therefore be helpful to think of AmorFEA as a direct analogue to popular amortized variational inference approaches (Kingma & Welling, 2013; Rezende et al., 2014) in which the solution to the variational problem is produced via function approximation rather than per-example optimization.

2.1. Amortization Suboptimality

For amortized variational inference, Cremer et al. (2018) introduced the notions of *approximation*, *amortization*, and *inference gap*. The approximation gap arises in variational inference due to the inadequacy of the variational family to approximate the true posterior distribution. The amortization gap reflects the difference between the approximate posterior produced by function approximation and the optimal one *within the variational family*. The inference gap is the sum of the approximation and amortization gaps, reflecting the total error in the amortized variational inference scheme.

Similar to gaps for amortized inference, we propose the approximation, amortization and "inference" (total error) gaps for AmorFEA:

$$\Delta_{\text{ap}} = \mathbb{E}_{p(\boldsymbol{\lambda})}\Big[\mathcal{L}(\boldsymbol{u}^\star_{\boldsymbol{\lambda}}, \boldsymbol{\lambda}) - \mathcal{L}(u^e, \lambda)\Big] \tag{4}$$

$$\Delta_{\text{am}} = \mathbb{E}_{p(\boldsymbol{\lambda})}\Big[\mathcal{L}(g_{\psi^\star}(\boldsymbol{\lambda}), \boldsymbol{\lambda}) - \mathcal{L}(\boldsymbol{u}^\star_{\boldsymbol{\lambda}}, \boldsymbol{\lambda})\Big] \tag{5}$$

$$\Delta_{\text{inf}} = \Delta_{\text{ap}} + \Delta_{\text{am}}, \tag{6}$$

where $\psi^\star$ is the optimal solution to Eq. 3, $\boldsymbol{u}^\star_{\boldsymbol{\lambda}}$ is the optimal solution to Eq. 2 for a given $\boldsymbol{\lambda}$, and u^e is the exact solution to the governing PDE which is generally impossible to obtain. Note that we are abusing notation somewhat here in allowing $\mathcal{L}(\cdot, \cdot)$ to take inputs of either functions (e.g., normal font u) or FEA vectors (e.g., bold font $\boldsymbol{u}$).

2.2. Computational Complexity

Solving a PDE for $\boldsymbol{u} \in \mathbb{R}^n$ by a numerical method such as FEA usually involves solving a large system of equations. A crude estimate of the cost can be described as $\mathcal{O}(dn^r)$, where $d = 1$ if the system is linear and $d > 1$ if it is nonlinear. For the nonlinear system, d can be viewed as the number of iterations required for a nonlinear solver such as Newton-Raphson to converge. The exponent r depends on the sparsity structure of the linear system (or the linearized system for a nonlinear problem) and the algorithm used by the solver. A naïve solver yields $r = 3$, while more efficient solvers usually require case-by-case analysis to take advantage of problem structure.

By amortizing the FEA solving process, we obtain a neural network surrogate function that predicts the state vector $\boldsymbol{u}$ directly from the control vector $\boldsymbol{\lambda}$. Excluding the cost of training time, for a standard multilayer perceptron (MLP) with l equally wide layers of n hidden units, the computational cost is simply the forward pass, which is $\mathcal{O}(ln^2)$. However, AmorFEA requires an up-front training time that we informally think of as $\mathcal{O}(ksln^2)$ where s is the number of training examples and k is the number of training epochs, which tends to be fixed. AmorFEA is able to produce a relatively efficient feed-forward solver, which could be used to save computational resources when an expensive PDE must be solved many times, or when many PDEs must be solved which lie within a given class. In the next subsection, we introduce PDE-CO as such a scenario where AmorFEA can be advantageous.

2.3. PDE-Constrained Optimization

The discretized PDE-constrained optimization is formulated as

$$\begin{aligned} \min_{\boldsymbol{u} \in \mathbb{R}^n, \boldsymbol{\lambda} \in \mathbb{R}^m} \; & \mathcal{J}(\boldsymbol{u}, \boldsymbol{\lambda}) \\ \text{s.t.} \quad & \boldsymbol{c}(\boldsymbol{u}, \boldsymbol{\lambda}) = 0, \end{aligned} \tag{7}$$

where $\mathcal{J}(\cdot, \cdot) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is the objective function and $\boldsymbol{c}(\cdot, \cdot) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is the constraint function im-

posed by the governing PDE. A reduced formulation is often used to embed the PDE constraint as

$$\min_{\boldsymbol{\lambda}\in\mathbb{R}^m} \widehat{\mathcal{J}}(\boldsymbol{\lambda}), \tag{8}$$

where $\widehat{\mathcal{J}}(\boldsymbol{\lambda}) := \mathcal{J}(\boldsymbol{u}(\boldsymbol{\lambda}), \boldsymbol{\lambda})$ and $\boldsymbol{u}(\boldsymbol{\lambda})$ is the implicit function arising from the solution to Eq. 2. Gradient-based optimization algorithms require the evaluation of the derivative of the objective function with respect to the control vector:

$$\frac{d\widehat{\mathcal{J}}}{d\boldsymbol{\lambda}} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{u}} \frac{d\boldsymbol{u}}{d\boldsymbol{\lambda}} + \frac{\partial \mathcal{J}}{\partial \boldsymbol{\lambda}}. \tag{9}$$

A common way to compute this gradient is to use the adjoint method (Cao et al., 2003). Taking the derivative of Eq. 7 with respect to $\boldsymbol{\lambda}$ yields chains of Jacobian matrices:

$$\frac{d\boldsymbol{c}}{d\boldsymbol{\lambda}} = \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{u}} \frac{d\boldsymbol{u}}{d\boldsymbol{\lambda}} + \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{\lambda}} = 0. \tag{10}$$

Hence,

$$\frac{d\boldsymbol{u}}{d\boldsymbol{\lambda}} = -\left(\frac{\partial \boldsymbol{c}}{\partial \boldsymbol{u}}\right)^{-1} \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{\lambda}}. \tag{11}$$

Substitute Eq. 11 to Eq. 9, we obtain

$$\frac{d\widehat{\mathcal{J}}}{d\boldsymbol{\lambda}} = -\overbrace{\frac{\partial \mathcal{J}}{\partial \boldsymbol{u}} \underbrace{\left(\frac{\partial \boldsymbol{c}}{\partial \boldsymbol{u}}\right)^{-1}}_{\text{tangent linear PDE}}}^{\text{adjoint PDE}} \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{\lambda}} + \frac{\partial \mathcal{J}}{\partial \boldsymbol{\lambda}}. \tag{12}$$

Resembling the two different modes of automatic differentiation, we could choose to either solve the adjoint PDE first (reverse-mode) or the tangent linear PDE first (forward-mode). When the size of the control vector is larger than that of the objective (e.g., $m \gg 1$ in our case), it is more efficient to solve the adjoint PDE first, giving the name adjoint method. The adjoint PDE is a linear PDE to solve, but it also relies on the Jacobian matrix $\frac{\partial \boldsymbol{c}}{\partial \boldsymbol{u}}$, which requires the solution vector $\boldsymbol{u}$. As shown, the cost for solving the governing PDE is $\mathcal{O}(dn^r)$, which dominates the total cost of using the adjoint method in one iteration of PDE-CO.

We propose accelerating PDE-CO with AmorFEA. AmorFEA yields a differentiable map from the control vector $\boldsymbol{\lambda}$ to the state vector $\boldsymbol{u}$, which does not require solving the governing PDE via traditional FEA, and which can be used to approximate the costly term $\frac{d\boldsymbol{u}}{d\boldsymbol{\lambda}}$ in Eq. 9. With the differentiable neural surrogate model, we can formulate the PDE-constrained optimization problem as

$$\min_{\boldsymbol{\lambda}\in\mathbb{R}^m} \widetilde{\mathcal{J}}(\boldsymbol{\lambda}), \tag{13}$$

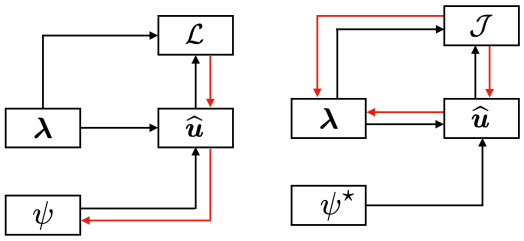

Figure 2. Computation graph for AmorFEA based PDE-CO. Red arrows denote automatic differentiation. Left: AmorFEA training for the surrogate model. Right: PDE-CO with the learned model.

where $\widetilde{\mathcal{J}}(\boldsymbol{\lambda}) := \mathcal{J}(g_{\psi^\star}(\boldsymbol{\lambda}), \boldsymbol{\lambda})$ and $\psi^\star$ are the learned weights of the neural network. The derivative is given by

$$\frac{d\widetilde{\mathcal{J}}}{d\boldsymbol{\lambda}} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{u}} \frac{dg_{\psi^\star}}{d\boldsymbol{\lambda}} + \frac{\partial \mathcal{J}}{\partial \boldsymbol{\lambda}}. \tag{14}$$

We employ reverse-mode automatic differentiation, since $m \gg 1$. By using a standard MLP, the total cost within each iteration of the PDE-CO is $\mathcal{O}(ln^2)$, and so guaranteed to be relatively efficient.

A summary for the procedures of AmorFEA along with AmorFEA based PDE-CO can be found in Fig. 2 by visualizing the computation graph.

3. Linear Models

An important and general class of PDE-CO problems impose a linear relationship between the control parameter and the state variable. These linear problems provide a useful testbed for examining the AmorFEA approach. We focus on the model problem in Section 1, where the source vector $\boldsymbol{\lambda}$ is linearly related to the solution vector $\boldsymbol{u}$. We show that by employing a linear regression model and performing AmorFEA, we are able to fully recover the underlying physics and achieve an amortization gap of zero; this result is unsurprising due to the assumption of a linear relationship between $\boldsymbol{u}$ and $\boldsymbol{\lambda}$. Though simple to analyze, the linear case gives intuition about the proposed scheme. We also compare AmorFEA with supervised learning, where we run FEA simulations to obtain labeled data and train the linear model in a traditional fashion. FEA simulations are carried out using an open source Python package `FEniCS` (Logg et al., 2012). Neural network training is performed in `PyTorch` (Paszke et al., 2019). We show that both AmorFEA and supervised training have the same global optimality condition.

We use AmorFEA to train a single-layer network that predicts $\boldsymbol{u} \in \mathbb{R}^n$ from $\boldsymbol{\lambda} \in \mathbb{R}^m$, where $m = 811$ and $n = 721$ in this case. We assume the distribution over source terms $\boldsymbol{\lambda}$ is a uniform distribution on the hypercube $[-1, 1]^m$. $10,000$ samples were drawn from this distribution to form the training data. FEA converts the minimization problem in Eq. 2

into a linear system to solve:

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{B}\boldsymbol{\lambda}, \tag{15}$$

where $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{B} \in \mathbb{R}^{n\times m}$ arise from the assembly procedure of the weak form in FEA (see details in Appendix A). Let us denote $\boldsymbol{f} = \boldsymbol{B}\boldsymbol{\lambda}$. The linear model we employ has a weight matrix $\boldsymbol{W} \in \mathbb{R}^{n\times n}$ and no bias vector so that $\widehat{\boldsymbol{u}} = \boldsymbol{W}\boldsymbol{f}$. The potential energy in this case is therefore equivalent to the least-squares loss for the solution to the linear system, preconditioned by the stiffness matrix $\boldsymbol{A}$:

$$\mathcal{L}_a(\boldsymbol{W}; \boldsymbol{f}) = \frac{1}{2}(\boldsymbol{W}\boldsymbol{f})^\top \boldsymbol{A}(\boldsymbol{W}\boldsymbol{f}) - \boldsymbol{f}^\top(\boldsymbol{W}\boldsymbol{f}). \tag{16}$$

As an instantiation of Eq. 3 using the Monte Carlo estimate of the expectation, AmorFEA leads to the following optimization problem:

$$\min_{\boldsymbol{W}\in\mathbb{R}^{n\times n}} \overline{\mathcal{L}}_a(\boldsymbol{W}), \tag{17}$$

where

$$\overline{\mathcal{L}}_a(\boldsymbol{W}) = \frac{1}{K}\sum_{k=1}^{K}\Big(\frac{1}{2}(\boldsymbol{W}\boldsymbol{f}_k)^\top \boldsymbol{A}(\boldsymbol{W}\boldsymbol{f}_k) - \boldsymbol{f}_k^\top(\boldsymbol{W}\boldsymbol{f}_k)\Big).$$

Next, we study supervised learning in the same environment. We expand the empirical data set $\mathcal{D} = \{\boldsymbol{\lambda}^{(i)}\}$ to $\mathcal{D}' = \{(\boldsymbol{\lambda}, \boldsymbol{u})^{(i)}\}$ by running FEA simulations. The supervised loss function for each data point is defined as

$$\mathcal{L}_s(\boldsymbol{W}; \boldsymbol{f}) = \frac{1}{2}\|\widehat{\boldsymbol{u}} - \boldsymbol{u}\|_2^2 = \frac{1}{2}\|\boldsymbol{W}\boldsymbol{f} - \boldsymbol{A}^{-1}\boldsymbol{f}\|_2^2. \tag{18}$$

Supervised training solves the following minimization problem:

$$\min_{\boldsymbol{W}\in\mathbb{R}^{n\times n}} \overline{\mathcal{L}}_s(\boldsymbol{W}), \tag{19}$$

where

$$\overline{\mathcal{L}}_s(\boldsymbol{W}) = \frac{1}{K}\sum_{k=1}^{K}\Big(\frac{1}{2}(\boldsymbol{W}\boldsymbol{f}_k - \boldsymbol{A}^{-1}\boldsymbol{f}_k)^\top(\boldsymbol{W}\boldsymbol{f}_k - \boldsymbol{A}^{-1}\boldsymbol{f}_k)\Big).$$

We show that both $\overline{\mathcal{L}}_a(\boldsymbol{W})$ and $\overline{\mathcal{L}}_s(\boldsymbol{W})$ are convex (see proofs in Appendix B).

Proposition 1. *The empirical loss function $\overline{\mathcal{L}}_a(\boldsymbol{W})$ of AmorFEA in the linear model is convex.*

Proposition 2. *The empirical loss function $\overline{\mathcal{L}}_s(\boldsymbol{W})$ of supervised learning in the linear model is convex.*

We also see that the first order condition (FOC) for AmorFEA gives

$$\frac{\partial\overline{\mathcal{L}}_a}{\partial\boldsymbol{W}} = (\boldsymbol{A}\boldsymbol{W} - \boldsymbol{I})\Big(\frac{1}{K}\sum_{k=1}^{K}\boldsymbol{f}_k\boldsymbol{f}_k^\top\Big) = \boldsymbol{0}. \tag{20}$$

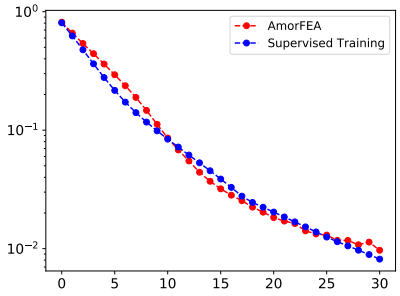

Figure 3. Normalized error versus training epochs. We define the normalized error $\epsilon = \frac{\|\boldsymbol{W}-\boldsymbol{A}^{-1}\|_\infty}{\|\boldsymbol{A}^{-1}\|_\infty}$.

Similarly, the FOC for supervised training gives

$$\frac{\partial\overline{\mathcal{L}}_s}{\partial\boldsymbol{W}} = (\boldsymbol{W} - \boldsymbol{A}^{-1})\Big(\frac{1}{K}\sum_{k=1}^{K}\boldsymbol{f}_k\boldsymbol{f}_k^\top\Big) = \boldsymbol{0}. \tag{21}$$

By Proposition 1 and Proposition 2 along with Eq. 20 and Eq. 21, we conclude that both AmorFEA and supervised learning achieve the global minimum with same condition $\boldsymbol{W} = \boldsymbol{A}^{-1}$. The only difference between these problems is the "physical" preconditioning from $\boldsymbol{A}$. As a supporting experiment, we show in Fig. 3 that both AmorFEA and supervised training have similar capabilities to recover the underlying model.

Since the fully trained linear model is able to produce $\widehat{\boldsymbol{u}} = \boldsymbol{u}$, we have a zero amortization gap as computed by Eq. 5. We do not study PDE-CO in the linear case.

4. Nonlinear Models

We examine two more realistic nonlinear settings and perform PDE-CO in this section. Here the solution vector $\boldsymbol{u}$ and the control vector $\boldsymbol{\lambda}$ have a nonlinear relationship, which naturally motivates the use of a neural network as the predictive mapping.

4.1. Source Field Finding

We consider a two-dimensional optimal source control problem simplified from superconductivity theory (De Melo et al., 1993). Intuitively, we have a known, target field $u_d(x)$ that we want our physical field $u(x)$ to achieve. We accomplish this by imposing an appropriate source field $\lambda(x)$. To save cost, we prefer the magnitude of $\lambda(x)$ to be small. The solution field $u(x)$ and the source field $\lambda(x)$ are related by the governing PDE (Eq. 23).

Mathematically, the problem is to minimize the functional

$$\mathcal{J}(u, \lambda) = \frac{1}{2}\int_\Omega (u - u_d)^2 dx + \frac{\alpha}{2}\int_\Omega \lambda^2 dx \tag{22}$$

subject to a nonlinear Poisson's equation with Dirichlet

Method	α	Optimized objective $[10^{-3}]$	Wall time [ms]
Adjoint Method	10^{-6}	0.338 ± 0.002	728.2 ± 5.5
	10^{-3}	2.599 ± 0.075	511.6 ± 223.2
	1	2.587 ± 0.012	15370.0 ± 1804.3
AmorFEA	10^{-6}	0.340 ± 0.003	39.7 ± 5.0
	10^{-3}	2.613 ± 0.104	71.5 ± 63.1
	1	2.593 ± 0.008	838.5 ± 208.4

Table 1. PDE-CO results for the adjoint method and AmorFEA. For different regularity coefficient α, AmorFEA achieves similar optimal objectives compared with the adjoint method, but with less wall time.

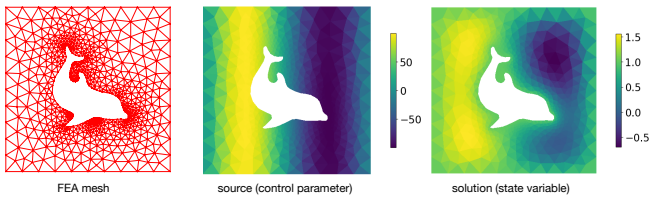

Figure 4. Setup of the problem. Left: the finite element mesh for a irregular domain with a dolphin-shaped hole. Middle: an example source field $\lambda(x) = 100\sin(2\pi x_1)$. Right: the solution field $u(x)$ associated with the source field.

boundary conditions:

$$\begin{aligned} -\Delta u + 10(u + u^3) &= \lambda \quad \text{in } \Omega, \\ u &= u_b \quad \text{on } \Gamma, \end{aligned} \tag{23}$$

where we set a constant boundary condition $u_b = 1$. We use Fig. 4 to further demonstrate the setup of this problem. Note that we employ an irregular domain with a dolphin-shaped hole in the middle. The irregular domain cannot be discretized using a structured mesh (a lattice). Many FEA benchmark cases are performed in this domain with `dolfin` (Logg & Wells, 2010), the finite element computing component of `FEniCS`.

The potential energy $\mathcal{L}(u, \lambda)$ from which the governing PDE (Eq. 23) can be derived is described in Appendix C. We use AmorFEA to train a neural network that predicts $\boldsymbol{u} \in \mathbb{R}^n$ from $\boldsymbol{\lambda} \in \mathbb{R}^m$, where $m = n = 759$ in this case. The assumed distribution over $\boldsymbol{\lambda}$ is constructed from a zero-mean Gaussian process given by

$$\begin{aligned} f(x) &\sim \mathcal{GP}\big(\mu(\cdot), k(\cdot, \cdot)\big) \\ \mu(x) &= 0 \\ k(x^{(i)}, x^{(j)}) &= \sigma^2 \exp\Big(-\frac{\|x^{(i)} - x^{(j)}\|_2^2}{l^2}\Big), \end{aligned} \tag{24}$$

where we set the output variance $\sigma = 10^2$ and the length-scale $l = 0.1$. To construct the training and testing data from this distribution, $30,000$ source terms were generated. Compared with supervised data generated by expensive FEA simulations, our data are almost free to obtain.

	MLP-0	MLP-1	MLP-2
Δ_{am}	0.1593 ± 0.0032	0.0227 ± 0.0003	0.0156 ± 0.0002
ϵ	0.0468 ± 0.0004	0.0110 ± 0.0002	0.0086 ± 0.0001

Table 2. Test performance. The amortization gap Δ_{am} is computed according Eq. 5. The relative error ϵ is computed as Eq. 25. The number suffix to "MLP" refers to the number of hidden layers.

For the neural network model, we use a MLP with "scaled exponential linear units" (SELUs) for the activation functions (Klambauer et al., 2017). We perform a $90/10$ train-test split for our data. For the 10% test data, we run FEA simulation so that we can report test error by comparing the AmorFEA solutions with the FEA solutions. Since our data represent functions, it is more meaningful to define an error quantification metric for functions than to use simple mean squared error (MSE). We define the relative error ϵ using norms on the $\mathcal{L}^2(\Omega)$ space:

$$\epsilon = \mathbb{E}_{p(\boldsymbol{\lambda})}\Big[\frac{\|u^\star - u^a\|_{\mathcal{L}^2}}{\|u^\star\|_{\mathcal{L}^2}}\Big], \tag{25}$$

where $u^\star$ is the FEA solution and u^a is the AmorFEA solution. With FEA simulations available in the test set, we can also compute the amortization gap introduced in Eq. 5. Since analytical solutions are generally not possible, we cannot explicitly evaluate the approximation gap or the inference gap. However, the approximation gap is a well-studied subject in the literature on finite element analysis (Hughes, 2012).

We run experiments using MLPs with different layers and report the amortization gap and relative error in Table 2. The layers in all MLPs have equal widths. As shown in the table, a single-layer network cannot perfectly solve this nonlinear problem, in contrast to the linear problem in Section 3. We observe that both the amortization gap and the relative error decrease with deeper neural networks. Using bigger neural networks may give us smaller amortization gaps, but there is a trade-off between training time and performance. We show next that a MLP with 2 hidden layers provides adequate accuracy for the PDE-CO problem while being trained cheaply.

We solve the aforementioned PDE-constrained optimization problem using both the adjoint method (see Eq. 12)

and the AmorFEA based method (see Eq. 14). We use `dolfin-adjoint` (Logg & Wells, 2010), an open source package written in the Python interface to `FEniCS` for implementing the adjoint method. We vary the values of the regularity coefficient α in Eq. 22 and run PDE-CO experiments using the conjugate gradient method (Shewchuk, 1994) in `SciPy` (Virtanen et al., 2019). We record the wall-clock time for different cases and show the results in Table 1. Note that since AmorFEA-based PDE-CO solves a surrogate model optimization problem, the objective function is defined differently (see Eq.8 and Eq.13). We have considered this difference and reported the reconstructed true objective function for AmorFEA based PDE-CO in Table 1. As shown, AmorFEA based PDE-CO achieves similar optimized objective values compared with the adjoint method, but with significantly less computation time, excluding time to train the network. However, it took approximately 5 minutes to train the neural network; thus AmorFEA is most interesting when such problems need to be solved repeatedly for, e.g., optimal control of a source.

4.2. Inverse Kinematics of a Soft Robot

In the second nonlinear example, we consider the control of a snake-like soft robot made of elastic material. Such robots represent promising solutions to minimally invasive surgery (Runciman et al., 2019). We control the static position of the robot by expanding or contracting "muscles" on the left and right sides of the robot, quantitatively described by an actuation field $\lambda(x)$ that controls the stretch ratio on the two sides. As shown in Fig. 5, the bottom side of the robot is fixed while the top side is free to deform. While a forward PDE problem solves for the displacement field $u(x)$ with given $\lambda(x)$, we here consider the PDE-constrained optimization problem of inverse kinematics, i.e., determining the appropriate actuation field to achieve an end-effector location. We focus on the middle point x_0 at the top of the robot and specify an arbitrary two-dimensional displacement u_0 that we hope this tip point can achieve by optimizing the actuation field $\lambda(x)$.

Mathematically, the problem is to minimize the functional

$$\mathcal{J}(u, \lambda) = \|u(x_0) - u_0)\|_2^2 \tag{26}$$

subject to an equilibrium equation for hyperelastic material with appropriate boundary conditions:

$$\begin{aligned} \text{Div } P(u) &= 0 \quad \text{in } \Omega, \\ r(u, \lambda) &= 0 \quad \text{on } \Gamma, \end{aligned} \tag{27}$$

where "Div" is the divergence operator, P is the a second-order stress tensor and r is the boundary constraint. For more precise definitions of these terms, refer to Appendix C. Since the boundary constraint in Eq. 27 is unusual, there is no direct implementation in `FEniCS`. We implement both the FEA algorithms and the adjoint method with our custom code for this problem.

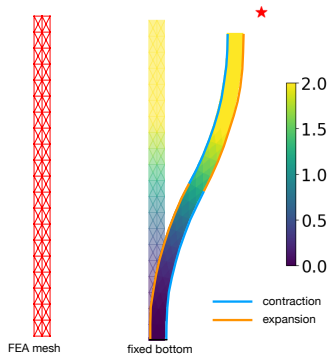

Figure 5. Setup of the problem. Left: the finite element mesh for the soft robot in an undeformed configuration. Right: with the actuation field $\lambda(x)$ set to be half-and-half for contraction and expansion, but upside down for the two sides, the robot can deform to a specific configuration. Colors indicate the displacement magnitude $\|u\|_2$.

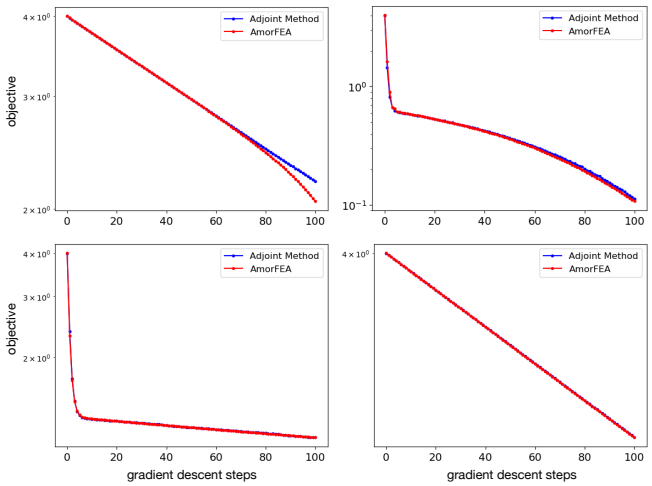

Figure 6. Objective function versus gradient descent steps for both AmorFEA and the adjoint method. The four cases correspond to the same scenarios in Fig. 7.

The potential energy $\mathcal{L}(u, \lambda)$ from which the governing PDE (Eq. 27) can be derived is known as the strain energy, fully described in Appendix C. We use AmorFEA to train a neural network to predict $\boldsymbol{u} \in \mathbb{R}^n$ from $\boldsymbol{\lambda} \in \mathbb{R}^m$, where $m = 40$ and $n = 206$ in this case. The assumed distribution over actuation fields is constructed from a uniform distribution over the hypercube $[-a, a]^m$ with $a > 0$. $30,000$ training data were generated from this distribution. Training is similar to the source-finding case in Section 4.1. However, the potential energy $\mathcal{L}(u, \lambda)$ in this case is sensitive to the displacement u. For a displacement field that causes any overlap of the deformed robot, the energy will be infinite.

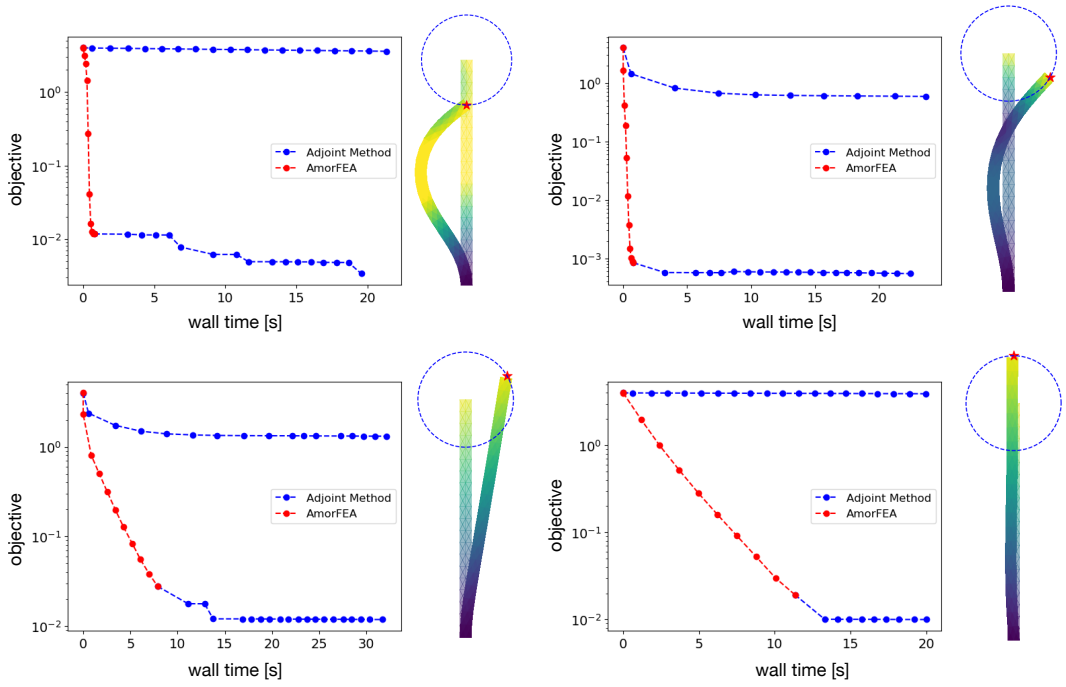

Figure 7. Wall time measurement for both the adjoint method and AmorFEA. There are four target displacements for the tip point displacement $u(x_0)$ to achieve. In these plots, one blue dot indicate one gradient descent step of the adjoint method, while one red dot indicate multiple gradient steps for AmorFEA. Following the row-first order, the numbers of steps per red dot are $(40, 40, 400, 600)$. The gradient descent step size is set to be consistent within each case, but different across the four cases: $(10^{-2}, 10^{-2}, 2 \times 10^{-3}, 2 \times 10^{-3})$.

We take a warm-start strategy to train the network by varying a from 0.1, 0.2, 0.4 until 0.8, and train the network successively. Training took $5 - 10$ minutes.

To demonstrate the ability of the resulting neural network to enable solutions to the inverse kinematics problem via PDE-CO of the actuation field, we pick four equally-spaced target points on a circle centered at the robot tip and perform PDE-CO to find actuations. As shown in Fig. 7, we use gradient descent for both the adjoint method and AmorFEA. AmorFEA consumes significantly less time than the adjoint method. However, since AmorFEA-based PDE-CO uses an approximate surrogate model, we may expect the objective to plateau before reaching the true minimum. One may view this as an asymptotic bias, which can be remedied by "fine tuning" the AmorFEA result with the adjoint method. From this perspective, AmorFEA provides a fast and accurate initial guess for the non-convex PDE-CO problem.

We additionally demonstrate that AmorFEA follows a similar optimization path to the adjoint method, by showing the objective function versus the number of gradient descent steps in Fig. 6 for each of the four cases. Wall time measurements in [s] of these four cases are $(119, 218, 133, 113)$ for the adjoint method and $(0.215, 0.210, 0.226, 0.249)$ for AmorFEA. For completeness, we also explored gradient-free techniques such as the Nelder–Mead method (Lagarias et al., 1998), but they typically converged to an unsatisfactory local minimum.

5. Limitations and future work

Compared to traditional FEA, AmorFEA consumes additional computational resources up front for training the model. To accommodate the training time, AmorFEA only starts to show its edge when a sequence of related problems need to be solved. Additionally, we note that the current formulation requires the governing PDE to be derived from minimization of a potential energy, but this is not applicable for all PDEs, which limits the scope of this work.

Promising directions for future work include a more systematic study of the amortization gap in nonlinear problems, using higher order finite element basis functions for AmorFEA, and parameterization of the domain for topology optimization in PDE-CO.

6. Conclusions

In this work, we proposed a novel formulation (AmorFEA) that amortizes the solving process of classic finite element analysis. AmorFEA enables fast, differentiable prediction of PDE solutions, which accelerates PDE-constrained optimization. We quantitatively studied the amortization gap for

both linear problems and nonlinear problems. Numerical experiments show that our method outperforms the traditional adjoint method on a per-iteration basis.

Acknowledgements

This work was supported by the Princeton Catalysis Initiative at Princeton University.

References

Cao, Y., Li, S., Petzold, L., and Serban, R. Adjoint sensitivity analysis for differential-algebraic equations: The adjoint DAE system and its numerical solution. *SIAM Journal on Scientific Computing*, 24(3):1076–1089, 2003.

Choi, K., Wu, M., Goodman, N., and Ermon, S. Meta-amortized variational inference and learning. *arXiv preprint arXiv:1902.01950*, 2019.

Cremer, C., Li, X., and Duvenaud, D. Inference suboptimality in variational autoencoders. *arXiv preprint arXiv:1801.03558*, 2018.

Criminisi, A., Shotton, J., and Konukoglu, E. Decision forests for classification, regression, density estimation, manifold learning and semi-supervised learning. *Microsoft Research Cambridge, Tech. Rep. MSRTR-2011-114*, 5(6):12, 2011.

De Melo, C. S., Randeria, M., and Engelbrecht, J. R. Crossover from BCS to Bose superconductivity: Transition temperature and time-dependent Ginzburg-Landau theory. *Physical Review Letters*, 71(19):3202, 1993.

Dennis, J., Heinkenschloss, M., and Vicente, L. N. Trust-region interior-point SQP algorithms for a class of nonlinear programming problems. *SIAM Journal on Control and Optimization*, 36(5):1750–1794, 1998.

Errico, R. M. What is an adjoint model? *Bulletin of the American Meteorological Society*, 78(11):2577–2592, 1997.

Gershman, S. and Goodman, N. Amortized inference in probabilistic reasoning. In *Proceedings of the annual meeting of the cognitive science society*, volume 36, 2014.

Herzog, R. and Kunisch, K. Algorithms for PDE-constrained optimization. *GAMM-Mitteilungen*, 33(2): 163–176, 2010.

Hughes, T. J. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2012.

Kingma, D. P. and Welling, M. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. Self-normalizing neural networks. In *Advances in neural information processing systems*, pp. 971–980, 2017.

Kochenderfer, M. J. and Wheeler, T. A. *Algorithms for optimization*. MIT Press, 2019.

Lagarias, J. C., Reeds, J. A., Wright, M. H., and Wright, P. E. Convergence properties of the Nelder–Mead simplex method in low dimensions. *SIAM Journal on optimization*, 9(1):112–147, 1998.

Lagaris, I. E., Likas, A., and Fotiadis, D. I. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5): 987–1000, 1998.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

LeVeque, R. J. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*, volume 98. SIAM, 2007.

Logg, A. and Wells, G. N. DOLFIN: Automated finite element computing. *ACM Transactions on Mathematical Software (TOMS)*, 37(2):1–28, 2010.

Logg, A., Mardal, K.-A., and Wells, G. *Automated solution of differential equations by the finite element method: The FEniCS book*, volume 84. Springer Science & Business Media, 2012.

Lopez, R., Balsa-Canto, E., and Oñate, E. Neural networks for variational problems in engineering. *International Journal for Numerical Methods in Engineering*, 75(11): 1341–1360, 2008.

Lu, L., Meng, X., Mao, Z., and Karniadakis, G. E. Deepxde: A deep learning library for solving differential equations. *arXiv preprint arXiv:1907.04502*, 2019.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pp. 8024–8035, 2019.

Ravi, S. and Beatson, A. Amortized Bayesian meta-learning. 2018.

Rees, T., Dollar, H. S., and Wathen, A. J. Optimal solvers for PDE-constrained optimization. *SIAM Journal on Scientific Computing*, 32(1):271–298, 2010.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Runciman, M., Darzi, A., and Mylonas, G. P. Soft robotics in minimally invasive surgery. *Soft robotics*, 6(4):423–443, 2019.

Shah, A., Wilson, A., and Ghahramani, Z. Student-t processes as alternatives to Gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, pp. 877–885, 2014.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and De Freitas, N. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

Shewchuk, J. R. An introduction to the conjugate gradient method without the agonizing pain, 1994.

Sirignano, J. and Spiliopoulos, K. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018.

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., and Adams, R. Scalable Bayesian optimization using deep neural networks. In *International Conference on Machine Learning*, pp. 2171–2180, 2015.

Ulbrich, M. and van Bloemen Waanders, B. An introduction to partial differential equations constrained optimization. *Optimization and Engineering*, 19(3):515–520, 2018.

van Leeuwen, T. and Herrmann, F. J. A penalty method for PDE-constrained optimization in inverse problems. *Inverse Problems*, 32(1):015007, 2015.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., et al. SciPy 1.0–fundamental algorithms for scientific computing in Python. *arXiv preprint arXiv:1907.10121*, 2019.

Weinan, E. and Yu, B. The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

Zhu, Y., Zabaras, N., Koutsourelakis, P.-S., and Perdikaris, P. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394: 56–81, 2019.