BOUNDARY CONDITION ENFORCEMENT WITH PINNS: A COMPARATIVE STUDY AND VERIFICATION ON 3D GEOMETRIES

Conor Rowan
Smead Aerospace Engineering Sciences
3775 Discovery Drive
University of Colorado Boulder
Boulder, CO 80309
conor.rowan@colorado.edu

Kai Hampleman
Smead Aerospace Engineering Sciences
3775 Discovery Drive
University of Colorado Boulder
Boulder, CO 80309
kai.hampleman@colorado.edu

Kurt Maute
Smead Aerospace Engineering Sciences
3775 Discovery Drive
University of Colorado Boulder
Boulder, CO 80309
kurt.maute@colorado.edu

Alireza Doostan
Smead Aerospace Engineering Sciences
3775 Discovery Drive
University of Colorado Boulder
Boulder, CO 80309
alireza.doostan@colorado.edu

ABSTRACT

Since their advent nearly a decade ago, physics-informed neural networks (PINNs) have been studied extensively as a novel technique for solving forward and inverse problems in physics and engineering. The neural network discretization of the solution field is naturally adaptive and avoids meshing the computational domain, which can both improve the accuracy of the numerical solution and streamline implementation. However, there have been limited studies of PINNs on complex three-dimensional geometries, as the lack of mesh and the reliance on the strong form of the partial differential equation (PDE) make boundary condition (BC) enforcement challenging. Techniques to enforce BCs with PINNs have proliferated in the literature, but a comprehensive side-by-side comparison of these techniques and a study of their efficacy on geometrically complex three-dimensional test problems are lacking. In this work, we i) systematically compare BC enforcement techniques for PINNs, ii) propose a general solution framework for arbitrary three-dimensional geometries, and iii) verify the

methodology on three-dimensional, linear and nonlinear test problems with combinations of Dirichlet, Neumann, and Robin boundaries. Our approach is agnostic to the underlying PDE, the geometry of the computational domain, and the nature of the BCs, while requiring minimal hyperparameter tuning. This work represents a step in the direction of establishing PINNs as a mature numerical method, capable of competing head-to-head with incumbents such as the finite element method.

Keywords Physics-informed neural networks · Constrained optimization · Boundary condition enforcement · Numerical methods

1 Introduction

Neural networks were first introduced more than half a century ago as flexible function approximators for data-driven classification and regression tasks [51]. In the past decade, physics-informed neural networks (PINNs) have emerged from the scientific machine learning community as a modification to the standard machine learning workflow. PINNs use physical laws in the form of governing partial differential equations (PDEs) in the process of training a neural network to represent the solution field. Consistent with their data-driven origins, PINNs often supplement the physics-based objective with a second term that enforces agreement with available measurement data. An example of this is the field of inverse problems, where trainable parameters are introduced into the governing equation and estimated by simultaneously minimizing the PDE residual and the mismatch of the solution with measurement data. The efficacy of this method for inverse problems was first demonstrated for estimating the unknown viscosity of the Navier-Stokes equation [49]. It has since been extended to conjugate heat transfer [9], contact mechanics [55], and beam bending [75], to name a few.

While PINNs are often used to blend physics and data, they can also be used as a numerical method analogous to finite element, finite volume, finite difference, or spectral methods. In this setting, no data is used to formulate the objective/loss function, and the neural network simply acts as a discretization of the solution field, with parameters tuned to satisfy the governing PDE. Lagaris et al. performed prescient early studies of neural network discretizations of partial differential equations (PDEs) over two decades ago [32, 31]. In their work, they discretize the PDE solution with a multilayer perceptron (MLP) neural network and determine its weights and biases by minimizing the sum of the squares of the PDE residual at discrete collocation points. More than 20 years later, the term PINNs was coined when, with the help of modern machine learning libraries, these methods were extended to more complex physical models [49]. We note that the "Deep Galerkin method" employs similar strategies and was introduced around the same time [61]. Follow-up works have extended these methods to different problems, such as elasticity [10, 24], contact mechanics [55], heat transfer [41], fluid mechanics [23], homogenization [33], and structural reliability analysis [45]. Owing to concerns about the scalability of the vanilla PINNs approach, domain decomposition methods were introduced and explored in [2, 19, 22]. Various other modifications to the network architecture and loss function have been proposed to improve the accuracy and convergence rates of the learned solution [3, 69, 34, 15, 13].

For many physical systems of engineering interest, the solution of the differential equation corresponds to a minimum of an energy functional. Noting that this energy functional has lower-order spatial derivatives than the corresponding strong form loss—and thus reduces the cost of forward evaluations and backpropagation–the energy defines an attractive objective function for solutions discretized with neural networks. This technique was first introduced as the "Deep Ritz Method" [14] and has been studied in the context of linear elasticity [38], hyperelasticity [1, 46], thermoelasticity [37], and fracture mechanics [43, 16]. We note that such energy functionals do not exist for all physical systems, including but not limited to fluid systems and solids with dissipative constitutive models.

Beyond the strong form and energy loss functions, neural networks can also be trained with the weak form of the governing PDE. Instead of a scalar loss function, the weak form is a system of equations expressing the orthogonality of the PDE residual to a chosen basis of "test" functions. As for the energy formulation, the weak form benefits from lower-order spatial differentiation, yet, because it is derived from the strong form of the governing equation, it exists for all physical systems. Inspired by adversarial training from computer vision, an interesting formulation of the weak form is given in [72], where the test function is a neural network trained to maximize the weighted residual of the PDE. In variational physics-informed neural networks (VPINNs), the spacetime solution is parameterized as a neural network, the strong form is integrated against a suitable basis of test functions, and the neural network parameters are found by minimizing the norm of the weak form residual [25]. Follow-up works have explored the Petrov-Galerkin weak form loss in the context of a variety of other problems from engineering mechanics [26, 59, 58, 27].

Enforcing boundary conditions (BCs) has been one of the primary obstacles to using PINNs for PDEs defined on complex two- and three-dimensional geometries. With the finite element method (FEM), Dirichlet BCs are built into the discretization and Neumann BCs are enforced weakly, meaning that they appear as forcing terms in the governing equation for equilibrium [21]. Given that a PINN solution is mesh-free, it is not straightforward to prescribe the solution along the Dirichlet region of the boundary. When using the variational energy or weak form as an objective, the Neumann BC appears naturally in the objective function, but this is not the case for the strong form loss. Thus, the PINN methodology leaves open important questions about how to best handle BCs. In the original PINN formulation, both Dirichlet and Neumann boundaries are enforced with penalties added to the strong form loss [49]. The penalty approach has been adopted by several other authors for both static and dynamic problems [33, 45, 23]. Noting that multi-objective optimization problems can lead to stiff gradient flow dynamics, various strategies have been proposed to adaptively weight the different terms in the loss function [7, 11, 74]. Of particular relevance, Wang et al. [70] proposed a simple algorithm that dynamically updates penalty weights so that the gradient contributions of each term remain balanced.

Since BCs can be cast as constraints on the solution field, Lagrange multipliers can be used as another enforcement strategy, though this gives the objective function a saddle point structure [54]. A "Deep Uzawa algorithm" is proposed in [42], whereby a stationary point of the Lagrange function objective is found through a staggered descent-ascent strategy. A method that blends penalty methods and Lagrange multipliers is the "Augmented Lagrangian" method, which

enforces constraints by repeatedly solving modified optimization problems in an iterative scheme [18, 4, 64, 40, 73]. Though the constraints are enforced accurately, a downside of the Augmented Lagrangian is that the neural network must be repeatedly trained at different settings of the penalty parameter and Lagrange multiplier. A modification to the Augmented Lagrangian for weak formulations of the PDE solution is "Nitsche's Method," which was investigated for high-dimensional and nonlinear problems in [36]. Though Nitsche's method is intended to improve the convergence properties of standard penalty formulations, the authors in [5] reported that it did not have this effect. Furthermore, Nitsche's method is limited in scope as it only applies to weak and variational formulations of the PDE system. Another related strategy, which combines aspects of penalty methods and Lagrange multipliers, is "Self-adaptive PINNs" (SA-PINNs), which introduces trainable penalty parameters that increase until the BCs are satisfied [44].

All of the above methods rely on modifying the objective function and/or optimization procedure such that the stationary point which the optimizer targets corresponds to a solution that exactly or approximately satisfies the BCs. An alternative approach is to modify the discretization such that the solution field satisfies the BCs automatically. This is possible in the case of Dirichlet BCs, where the neural network discretization of the solution can be multiplied by a "distance function" which is zero along the boundary and positive inside the domain [67, 65, 60, 5]. This enforces homogeneous Dirichlet BCs. For inhomogeneous Dirichlet boundaries, a second function can be added into the discretization after multiplying the neural network by a distance function, thus ensuring that the BCs are satisfied for any setting of the neural network parameters. Though this approach leads to rapid convergence and accurate PDE solutions, it may be non-trivial to construct the distance function and the additive term for the inhomogeneous boundary. Furthermore, it does not extend to Neumann boundaries and becomes especially cumbersome for domains combining Dirichlet and other boundary types. Going forward, we refer to domains with combinations of Dirichlet, Neumann, and/or Robin boundaries as having "mixed" BCs. To the best of the authors' knowledge, using distance functions in this setting of mixed boundaries has not been discussed in the PINNs literature.

Determining best practices for enforcing BCs on complex geometries is a crucial step toward making PINNs competitive with traditional numerical methods such as FEM. Accordingly, the focus of this work will be on studying and improving the capabilities of PINNs as a numerical method with an emphasis on BC enforcement. Our review suggests that a number of formulations of the physics loss and BC enforcement have been proposed, but the scientific machine learning community has not come to a consensus on best practices. Specifically, our goal is to move in the direction of a general-purpose PINN-based numerical solver, which can handle a wide range of governing equations, geometries, and BCs with minimal hyperparameter tuning. To do this, we must address the three fundamental tasks of i) formulating the physics loss, ii) defining the geometry, and iii) enforcing BCs. To this end, our contributions in this paper are as follows:

1. We critically review the three primary formulations of the physics loss (strong form, weak form, variational energy) and suggest that for a PINN-based solver to be generally applicable and to take full advantage of the neural network discretization, it must be based on the strong form loss;

2. We comprehensively survey techniques available for BC enforcement and perform a side-by-side performance comparison of a relevant subset of these techniques on a canonical test problem;

3. Informed by the results of this study, we identify best practices for BC enforcement in the context of PINNs and show how these strategies can be integrated into a framework to solve PDEs on arbitrary three-dimensional geometries;

4. Relying only on a small set of clearly-defined hyperparameters, we verify our proposed methodology on three linear and nonlinear three-dimensional problems from engineering mechanics with complex geometries and BC configurations.

The rest of this paper is organized as follows. In Section 2, we justify our choice to use the strong form loss as the objective function, arguing that the applicability of the Deep Ritz Method is too limited and that the weak form undermines many of the advantages of neural network discretizations. In Section 3, we discuss the various techniques for BC enforcement and perform a side-by-side comparison of a relevant subset of these techniques on a two-dimensional test problem. In Section 4, we build on the results of the comparison to illustrate a general-purpose solution methodology based on PINNs. The few hyperparameters of the solution methodology will be clearly identified and discussed. In Section 5, we verify the method against manufactured solutions on three-dimensional example problems, as well as compare its performance to a second technique for boundary condition enforcement. Finally, in Section 6, we close with concluding remarks and directions for future work.

2 On the strong form loss

As discussed above, several formulations of the physics loss have been explored in the PINNs community. All of these formulations are instances or variants of the strong form, weak form, or energy objective. We argue that the strong form loss provides the most general and flexible foundation for a PINN-based numerical solver intended to handle a broad class of PDEs. We reiterate that the energy objective only exists for some physical systems (steady-state heat transfer, linear and hyperelasticity, variational fracture mechanics, etc.). Notably, even for dynamics problems with variational principles, the stationary point of the energy functional is often a saddle point, not a minimum. The so-called "principle of least action" from classical mechanics is indeed a principle of stationary action [17]. Worse, the structure of the saddle is not known, meaning that, unlike saddle problems from Lagrange multipliers, it is not clear over which variables to maximize and over which to minimize. As such, the analyst is prevented from finding a stationary point with a min-max formulation of the optimization problem. This explains why the Deep Ritz method, which relies on *minimizing* an energy functional, has not been used on time-dependent systems. Thus, not only is Deep Ritz infeasible for fluid systems and non-conservative solid constitutive models, but it also cannot be used as a general strategy for dynamical systems. For any PINN-based solver that aspires to maximize its scope, an energy formulation must be ruled out.

Though the weak form exists for all PDEs, the quality of the solution relies heavily on the choice of test functions. The Bubnov-Galerkin weak form asserts that the PDE residual is orthogonal to the local tangent of the approximation space. For linear discretizations, this reduces to enforcing orthogonality of the residual against each basis function. In nonlinear discretizations—where parameters define both the basis functions and their coefficients—this condition can admit trivial solutions [53]. In such cases, the residual may be reduced not by satisfying the PDE, but by learning poor basis functions that weaken the approximation. Knowingly or not, this has pushed researchers to rely on the Petrov-Galerkin weak form, which makes use of distinct approximation and test spaces. One problem with this approach is that it requires the offline construction of a basis to test the PDE residual against. In [59, 27], the test functions are taken to be piecewise polynomials defined on a finite element mesh, which arguably voids the "mesh-free" promise of PINNs. Mesh-free spectral bases can be used as test functions, but such orthogonal functions are only tabulated for simple domain geometries [26, 6]. One work has investigated using "geometry-informed" neural network discretizations, but this method relies on offline FEM computations to obtain a spectral basis from eigenfunctions of the Laplace operator, which again calls into question whether the method is truly mesh-free [56].

An even more fundamental problem with the Petrov-Galerkin weak form is that it may not take full advantage of the expressivity of the neural network discretization. To see this, consider the following elliptic boundary value problem:

$$\frac{\partial}{\partial x}\left(\left(1+\frac{1}{2}\sin(20\pi x)\right)\frac{\partial u}{\partial x}\right)+100\sin(\pi x)=0, \quad x\in[0,1], \quad u(0)=u(1)=0. \tag{1}$$

This is the governing equation for the displacement field $u(x)$ of an axially loaded bar of unit length made of a material with multiscale properties. An exact solution can be obtained by directly integrating Eq. (1) and choosing integration constants to enforce the homogeneous Dirichlet BCs. Now, we obtain a PINN-based solution with both strong and weak form objectives. In both cases, the displacement is discretized with

$$\hat{u}(x;\boldsymbol{\theta})=\sin(\pi x)\mathcal{N}(x;\boldsymbol{\theta}), \tag{2}$$

where $\mathcal{N}(x;\boldsymbol{\theta})$ is an MLP neural network with parameters $\boldsymbol{\theta}$. The discretization of Eq. (2) satisfies the Dirichlet boundaries automatically. The strong form loss is given by

$$\mathcal{L}^{\text{STRONG}}(\boldsymbol{\theta})=\int_0^1\left[\frac{\partial}{\partial x}\left(\left(1+\frac{1}{2}\sin(20\pi x)\right)\frac{\partial \hat{u}}{\partial x}\right)+100\sin(\pi x)\right]^2 dx.$$

Defining a set of test functions $\{v_i(x)\}_{i=1}^N$, the weak form loss is the squared magnitude of the residual of the Petrov-Galerkin orthogonality condition, given by

$$\mathcal{L}^{\text{WEAK}}(\boldsymbol{\theta})=\frac{1}{2}\sum_{i=1}^{N}\left(\int_0^1\left[\frac{\partial}{\partial x}\left(\left(1+\frac{1}{2}\sin(20\pi x)\right)\frac{\partial \hat{u}}{\partial x}\right)+100\sin(\pi x)\right]v_i(x)dx\right)^2.$$

The test functions $v_i(x)$ are defined as piecewise linear finite element hat functions corresponding to a solution with homogeneous Dirichlet boundaries:

$$v_i(x) = \max\Big(0, 1 - \Big|x - \frac{i}{N+1}\Big|(N+1)\Big).$$

We discretize the solution with a two-hidden-layer neural network with each layer containing 50 hidden units and use hyperbolic tangent activation functions. Both the strong and weak form losses are unconstrained and are minimized with ADAM optimization. The learning rate is 1×10^{-3}, the optimization is run for 5×10^5 epochs, and there are $N = 18$ hat functions in the test set. See Figure 1 for the results from the strong and weak form solutions to the multiscale bar problem. The minimizer of the strong form loss reproduces the exact solution, whereas the zero of the weak form system leads to a solution with a spuriously large stiffness. Despite the expressivity of the neural network discretization, which is evidenced in the accuracy of the strong form solution, the weak form solution inherits the well-known failure of a coarse finite element solution to capture even the low-frequency component of the multiscale displacement [33]. The expressivity of the neural network is hindered by a poor choice of test functions. Thus, on top of the challenges of obtaining a suitable set of test functions for arbitrary two- and three-dimensional geometries, the weak form solution is only as good as the choice of test functions. We suggest that if the analyst has in hand a set of test functions that is i) adapted to the geometry and BCs of the problem and ii) sufficiently expressive to avoid issues such as those of Figure 1, the neural network discretization offers few advantages, and a Bubnov-Galerkin method should be used with this set of test functions. Given the limitations of the energy objective and the reliance of the weak form on constructing a suitable test set, we believe the strong form formulation to be best-suited for a flexible PINN-based solution methodology. As such, we work with the strong form loss going forward.

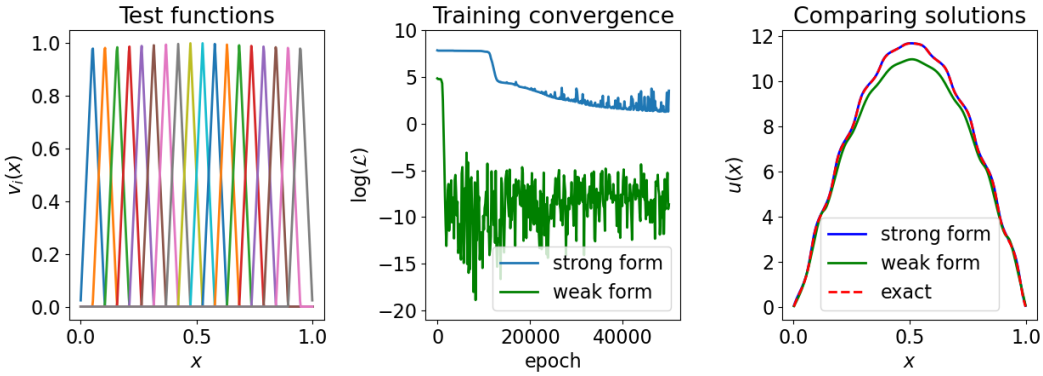

Figure 1: The $N = 18$ finite element hat functions are shown for reference (left), as well as the training dynamics of the strong and weak form losses (center) with the corresponding solutions (right). The Petrov-Galerkin weak form solutions exhibit significant error despite the expressivity of the neural network discretization.

3 Comparison of methods for BC enforcement

To motivate our discussion of BC enforcement, we introduce the following abstract scalar boundary value problem:

$$\begin{aligned} \mathcal{G}\Big(u(\mathbf{x})\Big) + f(\mathbf{x}) = 0, \quad \mathbf{x} \in \omega \\ u(\mathbf{x}) = g(\mathbf{x}), \quad \mathbf{x} \in \partial\omega_D, \\ \nabla u \cdot \hat{\mathbf{n}} = t(\mathbf{x}), \quad \mathbf{x} \in \partial\omega_N, \end{aligned} \tag{3}$$

where $u(\mathbf{x})$ is the solution field, $\mathcal{G}(\cdot)$ is a differential operator, ω is the computational domain, $\partial\omega_D$ is the Dirichlet region of the boundary, $\partial\omega_N$ is the Neumann region, and $\hat{\mathbf{n}}$ is the normal vector. Although Robin conditions will appear in our numerical examples, we restrict attention here to Dirichlet and Neumann boundaries for clarity. We note that, despite the challenges in enforcing BCs with PINNs, a comprehensive survey and comparison of techniques available for Dirichlet and Neumann BC enforcement is lacking in the PINNs literature. An initial step in this direction is [5], where standard penalty methods for Dirichlet BCs are compared against distance functions and Nitsche's method using the weak formulation of the physics loss. Because we choose to work with the strong form loss, Nitsche's method—which is only used for Dirichlet boundaries—is ruled out as a candidate for BC enforcement. Our task is therefore to conduct a comprehensive survey of BC enforcement techniques for PINNs and to carry out a side-by-side comparison of those most promising for three-dimensional geometries. The results of this study will be used to establish our suggestions for best practices.

Coordinate transformation We begin our survey with a discussion of coordinate transformation, which is a technique to map integrals and derivatives defined on one domain to their counterparts on another, often simpler, domain. We briefly describe coordinate transformation as a technique for BC enforcement. This is a topic which is best known in the scientific machine learning literature through [35], in which coordinate transformation is used to map arbitrary geometries to tori in order to use the fast Fourier transform to compute convolution integrals. Let us assume that a quantity of interest $q(\mathbf{x})$ is defined over a complex "current" geometry given by $\mathbf{x} \in \omega$. Assume also that a simpler "reference" geometry is given by $\mathbf{X} \in \Omega$. The deformation map $\boldsymbol{\phi} : \mathbb{R}^3 \to \mathbb{R}^3$ relating the two geometries is given as $\mathbf{x} = \boldsymbol{\phi}(\mathbf{X})$. See Figure 2 for a schematic of the relation between the two configurations. Using the deformation, the change of variables formula for multivariate integration is

$$\int_\omega q(\mathbf{x}) d\mathbf{x} = \int_\Omega q\Big(\boldsymbol{\phi}(\mathbf{X})\Big) \det\left(\frac{\partial \boldsymbol{\phi}}{\partial \mathbf{X}}\right) d\mathbf{X} = \int_\Omega \hat{q}(\mathbf{X}) J(\mathbf{X}) d\mathbf{X}, \tag{4}$$

where $J(\mathbf{X})$ is the "Jacobian determinant" relating volume elements in the two configurations. We note that in the context of PINNs, the quantity of interest $q(\mathbf{x})$ will depend on spatial gradients with respect to the current coordinates. In this case, the derivatives also need to be transformed to the corresponding derivatives in the reference geometry. With this in mind, Eq. (4) is a recipe for computing integrals of a quantity of interest defined in the current geometry entirely in terms of the reference geometry and a new field $\hat{q}(\mathbf{X})$ defined therein. For example, the strong form loss

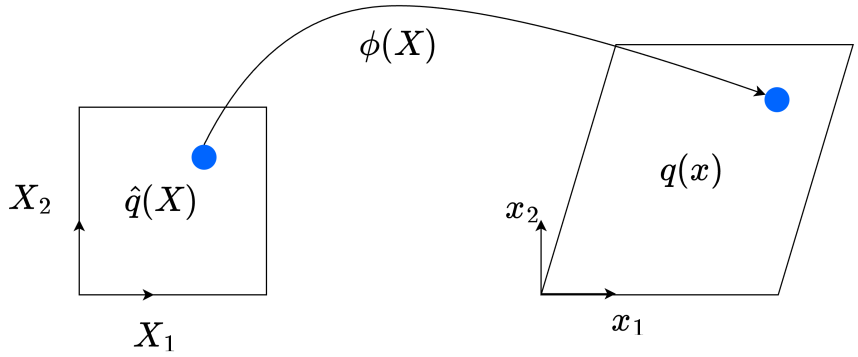

Figure 2: The deformation ϕ relates points $\mathbf{x}$ in the current geometry to corresponding points $\mathbf{X}$ in the reference geometry.

defined on the current geometry ω can be "pulled back" to a simpler geometry Ω to ease the enforcement of BCs. The difficulty with this coordinate transformation approach is threefold: first, we must obtain a deformation map ϕ whose inverse transforms the computational domain ω back to a simple reference geometry (square/cube, circle/sphere, etc.). One way to do this is to use a neural network to represent the deformation and train it by solving another set of partial differential equations [68]. This additional PDE solve introduces significant overhead into the forward solve of interest, in addition to requiring the analyst to specify an appropriate reference geometry. Second, poor choices of reference geometry may lead to Jacobian determinant fields that, through their spatial variation, make the physics of the problem on the reference geometry more complicated. Third, though the reference geometry may be simpler, it is still necessary to devise a method to handle BCs there. We seek a technique for BC enforcement that is general enough to apply to the original problem geometry, and we thus eliminate coordinate transformation as a viable approach.

Distance functions Distance functions have become a popular technique for enforcing Dirichlet BCs with PINNs. Following [67, 65, 60, 5], a distance function method discretizes the PDE solution as

$$\hat{u}(\mathbf{x};\boldsymbol{\theta}) = G(\mathbf{x}) + D(\mathbf{x})\mathcal{N}(\mathbf{x};\boldsymbol{\theta}), \tag{5}$$

where $\mathcal{N}(\mathbf{x};\boldsymbol{\theta})$ is a neural network that need not satisfy the BCs, and the functions $D(\mathbf{x})$ and $G(\mathbf{x})$, respectively, satisfy $D(\mathbf{x}) = 0$ for $\mathbf{x} \in \partial\omega_D$ and $G(\mathbf{x}) = g(\mathbf{x})$ for $\mathbf{x} \in \partial\omega_D$. This ensures that the inhomogeneous Dirichlet BCs are enforced by construction. Note that Eq. (5) does not have any bearing on the Neumann BCs, which, in the context of the strong form loss, need to be handled separately. Furthermore, constructing the two functions $G(\mathbf{x})$ and $D(\mathbf{x})$ may be as significant a bottleneck as obtaining the deformation map for the coordinate transformation approach. Consider the circular geometry shown in Figure 3. The boundary is built up from alternating patches of Dirichlet and Neumann BCs. Analytic distance functions are not readily available for problems of this type, and standard methods used to numerically construct distance functions would need to be modified, given the interspersing of the Neumann patches [65]. As is shown in [60], separate neural networks can be trained to obtain $G(\mathbf{x})$ and $D(\mathbf{x})$, but this introduces additional overhead to the PDE solve and ambiguity as to what conditions these two functions should satisfy along the Neumann boundary.

To remedy this ambiguity, a scalar distance function could be built which computes the distance to the nearest point on the nearest Dirichlet patch, thus ensuring that the function $D(\mathbf{x})$ is non-zero along the Neumann patches. However, having a disconnected set of boundary curves increases the complexity of obtaining the distance function and still leaves Neumann boundaries to be enforced separately. Because our goal is to avoid such pre-processing and to unify the treatment of Dirichlet and Neumann conditions under a single generally-applicable strategy, we do not consider distance functions a viable option for BC enforcement in our framework.

PINN-FEM A final strategy for exactly enforcing Dirichlet boundaries is PINN-FEM, which is a hybrid of a finite element discretization and PINNs relying on domain decomposition [63]. As in the distance function methods, PINN-FEM relies on pre-processing—in this case, meshing—and only handles Dirichlet BCs. Furthermore, we stress again, relying on a mesh undermines one of the main advantages of PINNs, which is its meshless character. Accordingly, this method is also ruled out.

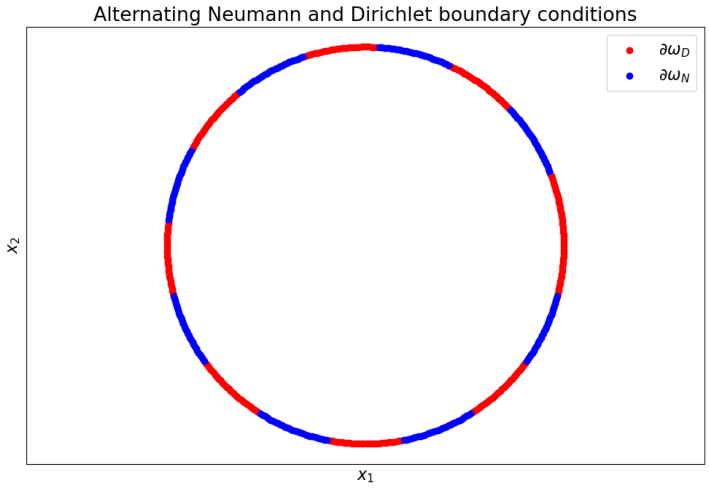

Figure 3: Mixed boundaries present challenges for methods based on distance functions, as the distance function must be zero along $\partial\omega_D$ and non-zero along $\partial\omega_N$.

Method	Avoids pre-processing?	Neumann BC?
Coordinate transformation [8]	✗	✗
Distance function [65]	✗	✗
PINN-FEM [63]	✗	✗
Standard penalty [49]	✓	✓
Learning rate annealing [70]	✓	✓
SA-PINNs [44]	✓	✓
Lagrange multipliers [42]	✓	✓
Augmented Lagrangian [64]	✓	✓

Table 1: A list of methods available to enforce BCs for PINN solutions based on the strong form loss.

We have disqualified Nitsche's method because it does not apply to PINN solutions based on the strong form loss. We have also chosen to rule out methods that handle only Dirichlet boundaries and/or rely on significant pre-processing

efforts to prepare the PDE solve. See Table 1 for a list of BC enforcement methods for PINN formulations based on the strong form loss. Thus far, we have discussed the first three methods, and disqualified them as approaches to a general-purpose PINN solver. We reiterate that this disqualification does not call into question the utility of these methods in specific contexts. Rather, our claim is that these methods require pre-processing and/or do not apply to arbitrary combinations of Dirichlet and Neumann boundaries on three-dimensional geometries. The remaining five methods are applicable to both Dirichlet and Neumann boundaries without an offline processing stage: standard penalty methods [49], learning rate annealing [70], SA-PINNs [44], Lagrange multipliers [42], and the Augmented Lagrangian [64]. In order to further explore the pros and cons of each of these methods, we perform a comparison on a two-dimensional test problem with inhomogeneous Dirichlet BCs. For our test problem, we solve the governing equation of forced steady-state heat conduction on the unit circle, given by

$$\begin{aligned} \nabla^2 u(\mathbf{x}) + f(\mathbf{x}) = 0, \quad \mathbf{x} \in \omega, \\ u(\mathbf{x}) = g(\mathbf{x}), \quad \mathbf{x} \in \partial\omega, \end{aligned} \tag{6}$$

where the entire boundary is Dirichlet such that $\partial\omega_D = \partial\omega$. For each method, we first establish the objective functions and the corresponding optimization problem. The temperature field is discretized by an MLP neural network $\mathcal{N}$ as $\hat{u}(\mathbf{x};\boldsymbol{\theta}) = \mathcal{N}(\mathbf{x};\boldsymbol{\theta})$.

Standard penalty A solution to Eq. (6) is obtained using a standard penalty method with

$$\begin{aligned} \mathcal{L}^{\text{PENALTY}}(\boldsymbol{\theta}) = \frac{1}{2}\int_\omega \Big(\nabla^2\hat{u}(\mathbf{x}) + f(\mathbf{x})\Big)^2 d\mathbf{x} + \frac{\beta}{2}\int_{\partial\omega}\Big(\hat{u}(\mathbf{x}) - g(\mathbf{x})\Big)^2 ds, \\ \operatorname*{argmin}_{\boldsymbol{\theta}} \mathcal{L}^{\text{PENALTY}}(\boldsymbol{\theta}), \end{aligned} \tag{7}$$

where β is a penalty hyperparameter [49]. This was the original approach for enforcing BCs with PINNs, and it remains a popular strategy owing to its simplicity.

Learning rate annealing Depending on the choice of β, the two loss terms in Eq. (7) may be unbalanced, leading to stiff gradient flow dynamics. To mitigate this issue, the penalty parameter can be dynamically updated with "learning rate annealing" [70] per

$$\begin{aligned} \mathcal{L}^{\text{LRA}}(\boldsymbol{\theta}) = \frac{1}{2}\int_\omega \Big(\nabla^2\hat{u}(\mathbf{x}) + f(\mathbf{x})\Big)^2 d\mathbf{x} + \frac{\beta(t)}{2}\int_{\partial\omega}\Big(\hat{u}(\mathbf{x}) - g(\mathbf{x})\Big)^2 ds, \\ \hat{\beta}_t = \frac{\max\left(\left|\frac{1}{2}\nabla_{\boldsymbol{\theta}}\int_\omega \Big(\nabla^2\hat{u}(\mathbf{x}) + f(\mathbf{x})\Big)^2 d\mathbf{x}\right|\right)}{\text{mean}\left(\left|\frac{1}{2}\nabla_{\boldsymbol{\theta}}\int_{\partial\omega}\Big(\hat{u}(\mathbf{x}) - g(\mathbf{x})\Big)^2 ds\right|\right)}, \\ \beta_t = (1-\alpha)\beta_t + \alpha\hat{\beta}_t, \\ \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta\nabla_{\boldsymbol{\theta}}\left(\frac{1}{2}\int_\omega \Big(\nabla^2\hat{u}(\mathbf{x}) + f(\mathbf{x})\Big)^2 d\mathbf{x} + \frac{\beta_t}{2}\int_{\partial\omega}\Big(\hat{u}(\mathbf{x}) - g(\mathbf{x})\Big)^2 ds\right), \end{aligned} \tag{8}$$

where $0 < \alpha < 1$ defines a moving average of the penalty parameters over optimization iterations, η is the learning rate, and t indexes the optimization epoch. Though we show a standard gradient descent update in Eq. (8), other optimizers, such as ADAM, can be used as well.

SA-PINNs Another modification to the standard penalty approach is SA-PINNs, which seeks to avoid the dependence of the accuracy of the BC enforcement on the choice of the penalty parameter. The objective function and optimization problem for SA-PINNs are given by

$$\begin{aligned} \mathcal{L}^{\mathrm{SA}}(\boldsymbol{\theta}, \beta) = \frac{1}{2}\int_{\omega} \Big(\nabla^2 \hat{u}(\mathbf{x}) + f(\mathbf{x})\Big)^2 d\mathbf{x} + \frac{1}{2}\int_{\partial\omega} \beta(\mathbf{x}) \Big(\hat{u}(\mathbf{x}) - g(\mathbf{x})\Big)^2 ds, \\ \underset{\beta}{\operatorname{argmax}} \ \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \ \mathcal{L}^{\mathrm{SA}}(\boldsymbol{\theta}, \beta), \end{aligned} \tag{9}$$

where ds is a differential area element on the boundary and $\beta(\mathbf{x})$ is a penalty field defined over the boundary which is driven up according to the pointwise boundary error [44]. In our implementation, as well as that of the original paper, $\beta(\mathbf{x})$ is discretized at integration points.

Lagrange multipliers Departing from penalty methods, another min-max formulation of the PINN solution is that of Lagrange multipliers, with the objective function and optimization problem defined by

$$\begin{aligned} \mathcal{L}^{L}(\boldsymbol{\theta}, \lambda) = \frac{1}{2}\int_{\omega} \Big(\nabla^2 \hat{u}(\mathbf{x}) + f(\mathbf{x})\Big)^2 d\mathbf{x} + \int_{\partial\omega} \lambda(\mathbf{x}) \Big(\hat{u}(\mathbf{x}) - g(\mathbf{x})\Big) ds, \\ \underset{\lambda}{\operatorname{argmax}} \ \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \ \mathcal{L}^{L}(\boldsymbol{\theta}, \lambda), \end{aligned} \tag{10}$$

where $\lambda(\mathbf{x})$ is a Lagrange multiplier field defined over the boundary [42]. As in SA-PINNs, the Lagrange multiplier is discretized at integration points in the numerical implementation. Unlike Eqs. (9), the solution parameters can be chosen to minimize the loss function by making the signed boundary error $\hat{u}(\mathbf{x}) - g(\mathbf{x})$ more negative. This suggests a more complex optimization problem than that of SA-PINNs, for which descent directions of the boundary loss with respect to the penalty parameter correspond to increased BC accuracy.

Augmented Lagrangian A popular hybrid of penalty methods and Lagrange multipliers is the Augmented Lagrangian method. A solution is obtained by solving an iterative sequence of optimization problems given by

$$\begin{aligned} \mathcal{L}^{\mathrm{AL}}(\boldsymbol{\theta}, \lambda_t, \beta_t) = \frac{1}{2}\int_{\omega} \Big(\nabla^2 \hat{u}(\mathbf{x}) + f(\mathbf{x})\Big)^2 d\mathbf{x} + \int_{\partial\omega} \lambda_t(\mathbf{x}) \Big(\hat{u}(\mathbf{x}) - g(\mathbf{x})\Big) ds + \frac{1}{2}\int_{\partial\omega} \beta_t(\mathbf{x}) \Big(\hat{u}(\mathbf{x}) - g(\mathbf{x})\Big)^2 ds, \\ \boldsymbol{\theta}_t = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \ \mathcal{L}^{\mathrm{AL}}(\boldsymbol{\theta}, \lambda_t, \beta_t), \\ \lambda_t(\mathbf{x}) \leftarrow \lambda_t(\mathbf{x}) + \beta_t(\mathbf{x}) \Big(\hat{u}(\mathbf{x}; \boldsymbol{\theta}_t) - g(\mathbf{x})\Big), \\ \beta_t(\mathbf{x}) \leftarrow \gamma \beta_t(\mathbf{x}), \end{aligned} \tag{11}$$

where $\gamma > 1$ defines the monotonically increasing update to the penalty parameter and the iteration over t, which is the index of the "inner loop" optimization problem, continues until user-defined convergence criteria are met [4].

We have now laid out the five candidate formulations of the objective function to handle the inhomogeneous Dirichlet boundary. We first discuss the solution of the optimization problems given in Eqs. (7)-(11) with first-order methods, such as ADAM and gradient descent. We then turn our attention to second-order Newton-type methods. Returning to the test problem of Eq. (6), we take the geometry ω of the test problem to be the unit circle centered at the origin. The source term $f(\mathbf{x})$ and the inhomogeneous Dirichlet boundary $g(\mathbf{x})$—which spans the entirety of the boundary curve—will be determined from an assumed solution $u(\mathbf{x})$ and Eq. (6) using the method of manufactured solutions. The temperature field we assume is

$$u(\mathbf{x}) = 10\Big(x_1 x_2 + \sin(\pi x_1)\sin(\pi x_2)(1 - x_1^2 - x_2^2)\Big).$$

Figure 4: The assumed temperature field used in the method of manufactured solutions, from which the source term and inhomogeneous Dirichlet BCs are computed.

See Figure 4 to visualize this temperature field, which will be used throughout the study. We discretize the temperature field with a standard MLP neural network with two hidden layers, 25 hidden units per layer, and hyperbolic tangent activation functions. We train the neural network corresponding to each objective function for 1×10^4 epochs using ADAM optimization for all gradient updates to the solution parameters $\boldsymbol{\theta}$. The learning rate for ADAM is set at 5×10^{-3} for all methods except the Lagrange multiplier method of Eq. (10), for which it is set at 1×10^{-3}. A 100×100 background integration grid is set up on the square given by $[-1, 1] \times [-1, 1]$, and the points that lie inside the unit circle are taken as integration points. The boundary integration grid consists of 500 equally spaced points around the circumference of the unit circle. We test the penalty method at two different penalty parameters: $\beta = 1$ and $\beta = 100$. For the learning rate annealing method, we set $\alpha = 0.9$ per the suggestions of [70]. In the SA-PINNs method, the initial penalty parameters are all set to 1, and the maximization is done with gradient descent with no momentum and a learning rate of 5×10^{-1}. Using gradient descent over ADAM ensures that the penalty stops increasing if the error is driven down to zero. In the min-max formulation of Lagrange multipliers, the initial values of the Lagrange multipliers

are set to 0, and the maximization is done with stochastic gradient descent without momentum with a learning rate of 1×10^{-2}. In the Augmented Lagrangian, we take the penalty update to be $\gamma = 2$, initialize the penalty parameters at 1, the Lagrange multipliers at 0, and compute an update to the Lagrange multiplier and penalty parameter when the norm of the gradient of the objective falls within 1% of its initial value. We note that these parameter settings are the consequence of experimentation to maximize the performance of each method.

To compare the five formulations of the objective optimized with first-order methods, we define two different error measures, one for the interior error and one for the boundary error:

$$\mathcal{I}(\boldsymbol{\theta}) = \frac{\int_\omega |u(\mathbf{x}) - \hat{u}(\mathbf{x};\boldsymbol{\theta})| d\mathbf{x}}{\int_\omega |u(\mathbf{x})| d\mathbf{x}}, \quad \mathcal{B}(\boldsymbol{\theta}) = \frac{\int_{\partial\omega} |g(\mathbf{x}) - \hat{u}(\mathbf{x};\boldsymbol{\theta})| ds}{\int_{\partial\omega} |g(\mathbf{x})| ds}. \tag{12}$$

See Figure 5 for the results of the comparison. When the penalty is set at a small value ($\beta = 1$), the standard penalty approach does not accurately enforce the BCs, and thus the interior error remains large. The larger penalty value ($\beta = 100$) performs better, but it relies on a choice of hyperparameter which is specific to the problem at hand. For example, if the penalty is made too large, the optimizer will completely ignore the physics loss term. In contrast, by adaptively updating the magnitudes of the two loss terms, the learning rate annealing approach yields low boundary error without relying on a penalty hyperparameter. SA-PINNs performs well and relies only on the choice of the initial penalty parameter and the learning rate for the penalty parameter updates, which, along with the point-wise boundary error, determine the rate at which the optimizer drives them up. The min-max formulation of the Lagrange multiplier problem does not converge within the allotted training period, exhibiting oscillations on both long and short time scales. Similar behavior is observed using ADAM optimization for the Lagrange multiplier, as well as for any choice of learning rates we experimented with. Finally, the Augmented Lagrangian method with the given hyperparameters performs well on our test problem. In our test case, learning rate annealing, SA-PINNs, and the Augmented Lagrangian are the top performers. See Table 2 for a comparison of the average interior and boundary error taken over the last 100 steps of the optimization.

Method	**Interior error $\mathcal{I}$**	**Boundary error $\mathcal{B}$**
Penalty ($\beta = 1$)	2.2×10^{-2}	9.8×10^{-2}
Penalty ($\beta = 100$)	4.8×10^{-3}	5.6×10^{-3}
LR annealing	1.8×10^{-2}	9.0×10^{-3}
SA-PINN	2.9×10^{-3}	8.7×10^{-3}
Lagrange multipliers	3.2×10^{0}	3.8×10^{0}
Augmented Lagrangian	1.2×10^{-3}	1.4×10^{-3}

Table 2: Comparing the techniques for BC enforcement by looking at the interior and boundary errors with the exact solution averaged over the last 100 steps. The three best performers are learning rate annealing, SA-PINNs, and the Augmented Lagrangian method.

We now discuss using second-order Newton-type methods to optimize the objective function. In particular, in place of the iterative min-max approach to the Lagrange objective of Eq. (10), we explicitly solve the nonlinear system corresponding to stationarity. This avoids the problematic min-max formulation at the cost of requiring second derivative information to perform updates to the solution parameters and Lagrange multiplier. The objective function

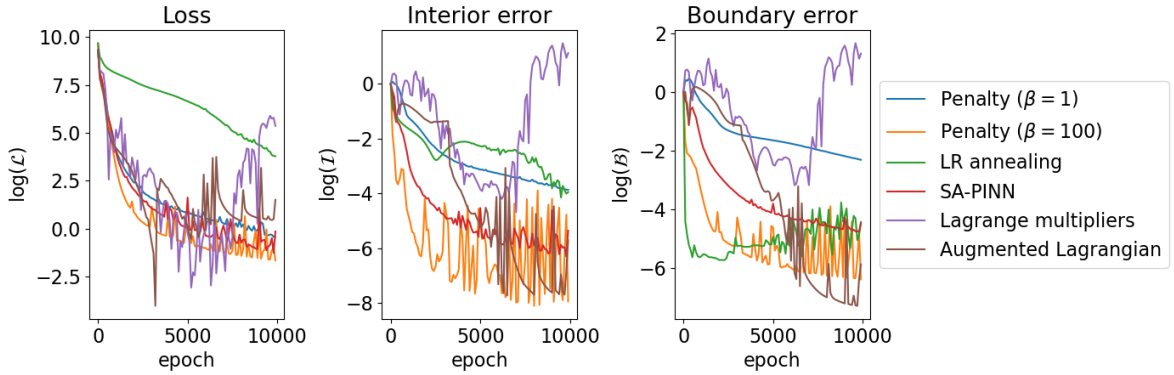

Figure 5: Comparing the five formulations of the loss for enforcing inhomogeneous Dirichlet BCs, optimized with first-order methods. We report the absolute value of the loss with Lagrange multipliers, as it can take on negative values. Across the two error measures, the Augmented Lagrangian and SA-PINNs performs the best, with learning rate annealing also showing good performance.

is the Lagrange objective of Eq. (10), but we denote it with superscript "N" to distinguish this different approach to finding a stationary point. Accordingly, the stationarity condition and the Newton updating are written as

$$\begin{aligned} \mathcal{L}^{\mathrm{N}}(\boldsymbol{\theta}, \lambda) = \frac{1}{2}\int_{\omega}\Big(\nabla^2\hat{u}(\mathbf{x}) + f(\mathbf{x})\Big)^2 d\mathbf{x} + \int_{\partial\omega}\lambda(\mathbf{x})\Big(\hat{u}(\mathbf{x}) - g(\mathbf{x})\Big)ds, \\ \frac{\partial\mathcal{L}^{\mathrm{N}}}{\partial\boldsymbol{\lambda}} = \mathbf{0}, \quad \frac{\partial\mathcal{L}^{\mathrm{N}}}{\partial\boldsymbol{\theta}} = \mathbf{0}, \\ \begin{bmatrix}\boldsymbol{\theta}\\ \boldsymbol{\lambda}\end{bmatrix}^t \leftarrow \begin{bmatrix}\boldsymbol{\theta}\\ \boldsymbol{\lambda}\end{bmatrix}^t - \begin{bmatrix}\partial^2\mathcal{L}^{\mathrm{N}}/\partial\boldsymbol{\theta}\partial\boldsymbol{\theta} & \partial^2\mathcal{L}^{\mathrm{N}}/\partial\boldsymbol{\theta}\partial\boldsymbol{\lambda}\\ \partial^2\mathcal{L}^{\mathrm{N}}/\partial\boldsymbol{\lambda}\partial\boldsymbol{\theta} & \partial^2\mathcal{L}^{\mathrm{N}}/\partial\boldsymbol{\lambda}\partial\boldsymbol{\lambda}\end{bmatrix}^{-1}\begin{bmatrix}\partial\mathcal{L}^{\mathrm{N}}/\partial\boldsymbol{\theta}\\ \partial\mathcal{L}^{\mathrm{N}}/\partial\boldsymbol{\lambda}\end{bmatrix}, \end{aligned} \tag{13}$$

where t indexes the Newton step and with slight abuse of notation we keep the Lagrange multiplier continuous in the objective function but use $\boldsymbol{\lambda}$ (bold) to indicate its discretization at boundary integration points. The Newton updating is continued until the norm of the residual of the system for stationarity falls below a user-specified threshold. As we discuss shortly, an approximation of the Hessian matrix is often used in practice. Second-order methods have received more attention in the context of neural network training in recent years. In [30], Newton optimization is shown to accelerate the training of deep neural networks. In the context of PINNs, second-order optimizers have been systematically compared and shown to provide accelerated convergence [28, 66]. One known issue with Newton methods is that the updates converge to general stationary points in the loss landscape, rather than explicitly looking for minima [12]. Numerical experimentation solving PINN-based optimization problems has suggested that true Newton methods, where the Hessian matrix in Eq. (13) is computed exactly, often exhibit such pathological behavior [52]. As such, we choose to avoid true Newton methods in favor of quasi-Newton methods, where the Hessian is approximated using the BFGS methodology [47]. Returning to the Newton formulation of the test problem given in Eq. (13), we first experiment with the trust constrained SciPy optimizer, which uses the BFGS approximation of the Hessian to build a local quadratic model of the objective but limits the optimization step to a region whose size is adaptively updated

based on the accuracy of the quadratic model. We find that when the number of constraints (the number of boundary integration points, in other words) is very small, the optimizer converges to a solution for which the strong form loss is approximately zero and the BCs are accurately satisfied. However, when the number of boundary integration points takes on realistic values of 100-500, the optimizer gets stuck at solutions with large values of the strong form loss. In Figure 6, we sweep over different sizes of the boundary constraints and show that the optimizer reliably drives the constraint error down to zero but that the strong form loss stagnates once the number of boundary integration points becomes too large. We note that the neural network discretization consists of two hidden layers, each with 30 hidden units, corresponding to 1050 total parameters. This ensures that the problem is not over-constrained, given that the number of constraints in the study is at most 150. Similar results are observed for the Sequential Least Squares Quadratic Programming (SLSQP) optimizer, which iteratively approaches a stationary point of the Lagrange function using a quadratic approximation of the objective with the BFGS Hessian, a linear approximation of the constraints, and line search. This is the one other quasi-Newton method in SciPy that handles constraints.

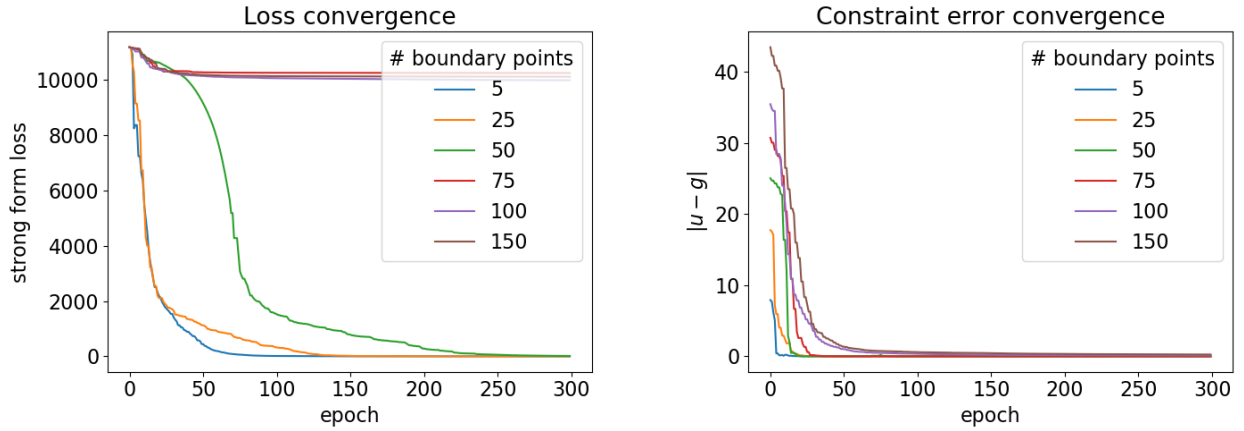

Figure 6: Quasi-Newton constrained optimization with SciPy fails to converge the strong form loss with even a modest number of constraints.

The study of Figure 5 suggests that the three best performing first-order methods are SA-PINNs, the Augmented Lagrangian method, and learning rate annealing. The min-max formulation of Lagrange multipliers does not converge to a solution under any hyperparameter settings we could find, and is thus disqualified as a technique for BC enforcement. While the penalty method with the larger penalty parameter also performs well, we believe SA-PINNs, the Augmented Lagrangian, and learning rate annealing offer certain advantages. All of them involve adaptive balancing of the loss terms to make expensive hyperparameter tuning unnecessary. Thus, on the basis of its sensitivity to hyperparameters, we choose to rule out the standard penalty method. However, in our experience, SA-PINNs can also exhibit sensitivity to hyperparameters, namely to the learning rate for the penalty parameters. If this learning rate is too small, SA-PINNs requires a substantial number of optimization steps for the penalty parameters to stabilize. This increased convergence time is noted in [18]. A second problem with SA-PINNs is the introduction of additional trainable parameters, which balloons the number of optimization variables in two and three spatial dimensions. While the authors in [64] remark

that the method introduces $10,000$ trainable parameters in two dimensions, we remark that at our chosen resolution level, SA-PINNs would introduce up to $50,000$ additional optimization variables. With the neural network architectures we use, treating BCs with SA-PINNs corresponds to a fifty-fold increase in the size of the optimization problem. For these reasons, we narrow the focus of subsequent numerical examples to learning rate annealing and the Augmented Lagrangian method. We choose to emphasize the Augmented Lagrangian given its superior performance in our study. However, each numerical example also provides a comparison to the learning rate annealing method. In the following section, we lay out a strategy for integrating the Augmented Lagrangian method for BC enforcement with arbitrary three-dimensional geometries.

4 Solution framework for PINNs on 3D geometries

Our task is to illustrate a methodology for solving general linear or nonlinear boundary value problems with any combination of Dirichlet, Neumann, and Robin BCs on arbitrarily complex three-dimensional geometries using PINNs. This task is primarily one of synthesis—we have weighed the pros and cons of different formulations of the physics loss, and concluded that the strong form loss is the best-equipped for our goal of a general-purpose PINN-based solver. Similarly, though the BC enforcement strategies we rely on have been proposed in the literature previously, a comprehensive comparison has been lacking. Having committed to the strong form loss, and having narrowed the set of viable constraint enforcement strategies with such a comparison, the remaining ingredient in the numerical solution is the geometry handling. Once again, we do not introduce any novel tools to this end, rather we show how existing strategies can be integrated into a PINN solution workflow. We opt to use level sets to generate three-dimensional geometries, as this offers a streamlined and customizable approach to geometry generation. We note, however, that this choice is made with no loss of generality. Any standard geometry file will come equipped with the information required to build the interior and surface integration grids we extract from the level set. In our construction, all level sets define geometries which are subsets of the unit cube given by $\omega^B = [0,1]^3$, where the superscript "B" refers to the "background" domain. Consider a scalar-valued level set function $\Phi(\mathbf{x})$. The bulk ω and the boundary surface $\partial\omega$ of the three-dimensional geometry are defined by

$$\omega = \{\mathbf{x}|\Phi(\mathbf{x}) < 0\}, \quad \partial\omega = \{\mathbf{x}|\Phi(\mathbf{x}) = 0\}.$$

We define a uniform grid of points on the background domain as $\mathcal{P} = \{\mathbf{x}_i\}_{i=1}^{N_B} \in \Omega^B$ where N_B is the number of integration points. Next, we build an "interior integration grid" $\mathcal{X}$ using the level set with

$$\mathcal{X} = \{\mathbf{x} \in \mathcal{P}|\Phi(\mathbf{x}) < 0\}.$$

This allows us to approximate integrals over the computational domain with

$$\int_\omega f(\mathbf{x})d\mathbf{x} \approx \sum_{i=1}^{N} \Delta V f(\mathcal{X}_i),$$

where N is the number of interior integration points, the set of interior integration points is $\mathcal{X} = \{\mathcal{X}_i\}_{i=1}^N$, and $\Delta V = 1/N$ is the volume element defined by the uniform integration grid. To enforce BCs, we must also compute surface integrals. This is accomplished with the marching cubes algorithm, which provides a triangulation of the zero isocontour of the level set evaluated on the background integration grid [39]. Returned from the marching cubes algorithm are sets of vertex points and the triangular faces formed by these vertex points. We define surface integration points as the centroids of the triangular faces, and compute outward-facing normal vectors by taking the cross product of the vectors defining the edges of the triangular faces. This allows us to approximate surface integrals as

$$\int_{\partial\omega} \mathbf{q}(\mathbf{x}) \cdot \hat{\mathbf{n}} ds \approx \sum_{j=1}^{M} \mathbf{q}(\mathcal{S}_j) \cdot \hat{\mathbf{n}}_j \Delta A_j,$$

where M is the number of surface integration points, $\mathcal{S} = \{\mathcal{S}_j\}_{j=1}^M$ is the surface integration grid obtained from marching cubes, $\hat{\mathbf{n}}_j$ is the outward facing unit normal corresponding to the j-th triangular facet, and ΔA_j is the corresponding area element. The particular implementation we use is Python's `skimage` marching cubes algorithm. We remark that there are more sophisticated algorithms for integrating over volumes and surfaces defined by level sets [57, 20]. In this work, we rely on the grid of background points being sufficiently dense to obtain accurate volume integrals using a uniform weight ΔV. The surface integration grid will be further broken down into the Dirichlet and Neumann components, which are given by $\mathcal{S}^D = \{\mathcal{S}_j^D\}_{j=1}^{M_D}$ and $\mathcal{S}^N = \{\mathcal{S}_k^N\}_{k=1}^{M_N}$ respectively. The superscripts "D" and "N" will be used similarly to index the area elements and normal vectors belonging to the two boundary sets. Figure 7 illustrates a level set function and its zero isocontour, as well as background and interior integration grids, and a boundary split up into Dirichlet and Neumann regions. With the interior integration grid, the boundary integration grid, and the corresponding surface normals, we illustrate a PINN-based solution methodology for BCs on arbitrarily complex three-dimensional geometries using the Augmented Lagrangian method. We discretize the solution and leave the option open to enforce homogeneous Dirichlet BCs on a subset of the domain with a distance function $D(\mathbf{x})$. Thus, the discretization of the solution field is given by

$$\hat{u}(\mathbf{x}; \boldsymbol{\theta}) = D(\mathbf{x}) \mathcal{N}(\mathbf{x}; \boldsymbol{\theta}).$$

With the Lagrange multiplier field $\lambda(\mathbf{x})$ and penalty field $\beta(\mathbf{x})$ discretized at surface integration points, the discretized Augmented Lagrangian objective at iteration t for the generic scalar BVP of Eq. (3) is

$$\begin{aligned}\hat{\mathcal{L}}^{\text{AL}}(\boldsymbol{\theta}, \boldsymbol{\lambda}_t, \boldsymbol{\beta}_t) = \frac{1}{2} \sum_{i=1}^{N} \Delta V \Big(\mathcal{G}\Big(\hat{u}(\mathcal{X}_i)\Big) + f(\mathcal{X}_i) \Big)^2 + \sum_{j=1}^{M_D} \Delta A_j^D \lambda_{j,t}^D \Big(\hat{u}(\mathcal{S}_j^D) - g(\mathcal{S}_j^D) \Big) \\ + \frac{1}{2} \sum_{j=1}^{M_D} \Delta A_j^D \beta_{j,t}^D \Big(\hat{u}(\mathcal{S}_j^D) - g(\mathcal{S}_j^D) \Big)^2 + \sum_{k=1}^{M_N} \Delta A_k^N \lambda_{k,t}^N \big(\nabla \hat{u}(\mathcal{S}_k^N) \cdot \hat{\mathbf{n}}_k^N - t(\mathcal{S}_k^N) \big) + \\ \frac{1}{2} \sum_{k=1}^{M_N} \Delta A_k^N \beta_{k,t}^N \big(\nabla \hat{u}(\mathcal{S}_k^N) \cdot \hat{\mathbf{n}}_k^N - t(\mathcal{S}_k^N) \big)^2.\end{aligned} \tag{14}$$

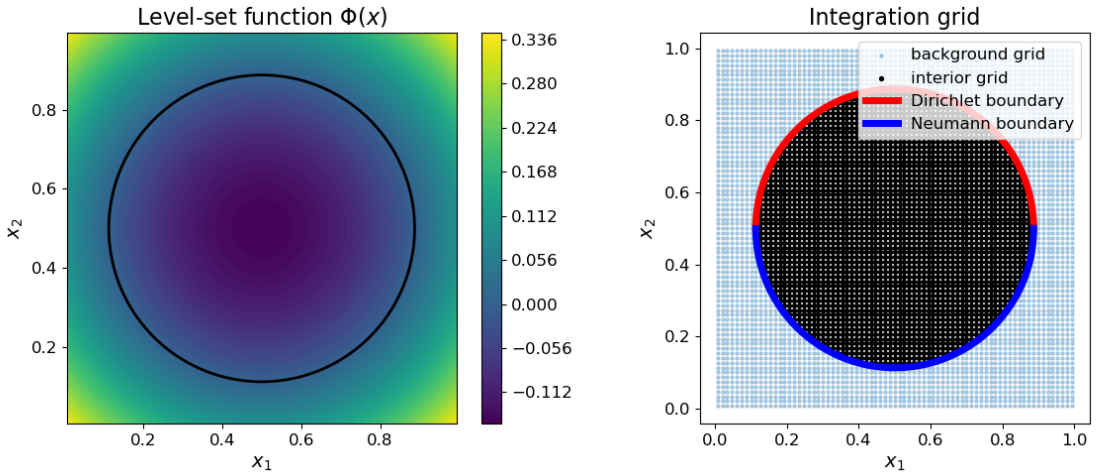

Figure 7: As an example, a two-dimensional level set function is shown with its zero isocontour in black (left). The level set function is used to build an interior integration grid from a background integration, and the boundary is split up into Dirichlet and Neumann regions (right).

Hyperparameter	**Recommended setting**
Learning rate	$(1 \times 10^{-3}) - (5 \times 10^{-3})$
Initial penalty $\boldsymbol{\beta}_0$	$\mathbf{1}$
Initial Lagrange multiplier $\boldsymbol{\lambda}_0$	$\mathbf{0}$
Rate of penalty increase γ	2
Objective convergence $\mathcal{Z}_f$	$(2.5 \times 10^{-3}) - (1 \times 10^{-2})$
Boundary convergence $\mathcal{B}_f$	$(5 \times 10^{-3}) - (1 \times 10^{-2})$
Gradient convergence $\mathcal{R}_f$	$(1 \times 10^{-3}) - (1 \times 10^{-2})$

Table 3: Hyperparameters used for the PINN-based solution framework based on the Augmented Lagrangian method. The learning rate is used in ADAM optimization for the inner-loop problem to find a minimum of the Augmented Lagrangian objective at iteration t. The initial solution parameters $\boldsymbol{\theta}_0$ are obtained with the built-in Xavier initialization in PyTorch.

We remark that the same strategy for computing surface integrals to form penalties on the BCs is used for the learning rate annealing method. We use ADAM optimization to solve the inner loop Augmented Lagrangian problem with no mini-batching of the integration grid. As is standard for the algorithm, the initial penalty parameter $(\boldsymbol{\beta}_0)$ is set at a small value, and the initial Lagrange multiplier $(\boldsymbol{\lambda}_0)$ is set to zero. The rate of increase of the penalty parameter (γ) is set at 2, which is also standard for the method. The most consequential hyperparameters for the Augmented Lagrangian method are the convergence criteria. We define convergence of the inner loop problem to occur when the gradient falls below $\mathcal{R}_f\%$ of its initial value. Convergence of the optimization concludes when this gradient condition is met, when the boundary loss falls below $\mathcal{B}_f$, and when the objective is below $\mathcal{Z}_f\%$ of its initial value. See Table 3 for the list of hyperparameters that specify this algorithm and Algorithm 1 for the breakdown of our implementation of the Augmented Lagrangian method. Note that the notation $\odot$ indicates element-wise multiplication. Numerical experimentation on the forthcoming example problems has informed our choice of recommended settings given in Table 3. In the next section, we showcase the efficacy of this method on three problems from engineering mechanics. We also compare its performance to learning rate annealing. All computations are performed on an AMD Ryzen 9 5950X 16-Core Processor and 64GB unified memory, running Python 3.11.5 and PyTorch 2.1.2.

Algorithm 1 Augmented Lagrangian for PINNs

Require: $\boldsymbol{\theta}_0, \boldsymbol{\beta}_0, \boldsymbol{\lambda}_0$ ▷ initial parameters
Require: $\gamma, \mathcal{B}_f, \mathcal{Z}_f, \mathcal{R}_f$ ▷ penalty update and convergence criteria
$\mathcal{Z}_0 = \mathcal{L}^{\text{AL}}(\boldsymbol{\theta}_0, \boldsymbol{\lambda}_0, \boldsymbol{\beta}_0)$ ▷ initial objective value
$\mathcal{R}_0 = \|\nabla_{\boldsymbol{\theta}} \mathcal{L}^{\text{AL}}(\boldsymbol{\theta}_0, \boldsymbol{\lambda}_0, \boldsymbol{\beta}_0)\|^2$ ▷ initial gradient value
while $\mathcal{B}_t > \mathcal{B}_f$ **and** $\mathcal{Z}_t / \mathcal{Z}_0 > \mathcal{Z}_f$ **do** ▷ outer loop
  while $\|\partial \mathcal{L}^{\text{AL}}(\boldsymbol{\theta}, \boldsymbol{\lambda}_t, \boldsymbol{\beta}_t)/\partial \boldsymbol{\theta}\| > \mathcal{R}_f / \mathcal{R}_0$ **do** ▷ inner loop
    ADAM optimization on $\mathcal{L}^{\text{AL}}(\boldsymbol{\theta}, \boldsymbol{\lambda}_t, \boldsymbol{\beta}_t)$
  end while
  $\boldsymbol{\lambda}^D_{t+1} = \boldsymbol{\lambda}^D_t + \boldsymbol{\beta}^D_t \odot \Big(\hat{u}(\mathcal{S}^D; \boldsymbol{\theta}_t) - g(\mathcal{S}^D)\Big), \quad \boldsymbol{\beta}^D_{t+1} = \gamma \boldsymbol{\beta}^D_t$ ▷ update for Dirichlet boundary
  $\boldsymbol{\lambda}^N_{t+1} = \boldsymbol{\lambda}^N_t + \boldsymbol{\beta}^N_t \odot \Big(\nabla \hat{u}(\mathcal{S}^N; \boldsymbol{\theta}_t) \cdot \hat{\mathbf{n}}^N - t(\mathcal{S}^N)\Big), \quad \boldsymbol{\beta}^N_{t+1} = \gamma \boldsymbol{\beta}^N_t$ ▷ update for Neumann boundary
  $\boldsymbol{\lambda}_{t+1} = [\boldsymbol{\lambda}^D_{t+1}, \boldsymbol{\lambda}^N_{t+1}], \quad \boldsymbol{\beta}_{t+1} = [\boldsymbol{\beta}^D_{t+1}, \boldsymbol{\beta}^N_{t+1}]$
  $\mathcal{Z}_{t+1} = \mathcal{L}^{\text{AL}}(\boldsymbol{\theta}_t, \boldsymbol{\lambda}_t, \boldsymbol{\beta}_t)$ ▷ current objective value
  $\mathcal{B}_{t+1} = \mathcal{B}(\boldsymbol{\theta}_t)$ ▷ current boundary error
  $t = t + 1$
end while

5 Numerical examples

5.1 Fisher-KPP equation on two-way branch

The first test case of the proposed method is a scalar-valued nonlinear static PDE. The Fisher-Kolmogorov-Petrovsky-Psikunov (Fisher-KPP) equation is a nonlinear reaction-diffusion PDE used to model population growth and traveling wave fronts [50]. The dynamic Fisher-KPP equation is given by

$$\frac{\partial u}{\partial t} = \mu \nabla^2 u(\mathbf{x}) + r u(\mathbf{x})(1 - u(\mathbf{x})) + f(\mathbf{x}), \tag{15}$$

where u is the concentration of a chemical substance or population density, $f(\mathbf{x})$ is a source term, μ is the diffusivity, and r is a reaction rate coefficient. Going forward, we set $\mu = 1, r = 1/2$ and work with the steady state of Eq. (15) which is given by

$$\begin{aligned} \nabla^2 u(\mathbf{x}) + r u(\mathbf{x})(1 - u(\mathbf{x})) + f(\mathbf{x}) = 0, \quad \mathbf{x} \in \omega, \\ u(\mathbf{x}) = g(\mathbf{x}), \quad \mathbf{x} \in \partial\omega. \end{aligned} \tag{16}$$

The geometry ω of the problem is that of a two-way branch, inspired by an arterial geometry from biomedical imaging. The following level set can be used to generate this geometry:

$$\Phi(\mathbf{x}) = \Big(\cos 2\pi x_1 - (1 - 2x_3)\Big)^2 + 9(x_2 - 0.5)^2 - 0.5. \tag{17}$$

The geometry is then defined by $\omega = \{\mathbf{x} | \Phi(\mathbf{x}) < 0\}$, as shown in Figure 8. Recall that all geometries are subsets of the unit cube $[0, 1]^3$. We use the marching cubes algorithm to mesh the surface defined by the zero isocontour of the level set given by Eq. (17). We enforce inhomogeneous Dirichlet BCs defined over the surface integration grid and a

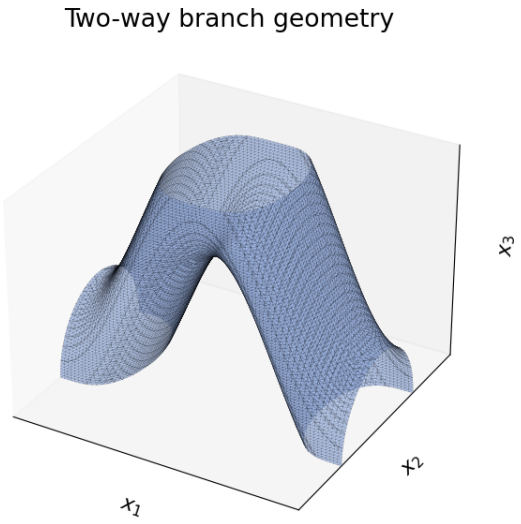

Figure 8: Two-way branch geometry for the steady-state Fisher-KPP problem.

manufactured solution. The assumed solution is

$$u(\mathbf{x}) = 10\sin(3\pi x_3)\sin(2\pi x_1)\sin(\pi x_1)\sin(\pi x_2)\sin(\pi x_3),$$

which enforces that $u(\mathbf{x}) = 0$ along the surface of the unit cube in which the level set is defined. This is relevant, as the zero of the level set extends outside the unit cube; thus, the boundaries defined by the surface of the unit cube will not be picked up by marching cubes. These BCs will be handled in the discretization of the solution field per

$$\hat{u}(\mathbf{x};\boldsymbol{\theta}) = \sin(\pi x_1)\sin(\pi x_2)\sin(\pi x_3)\mathcal{N}(\mathbf{x};\boldsymbol{\theta}).$$

The source term is defined through the manufactured solution and Eq. (16). The solution field is discretized with a two hidden layer network of width 30, and we take the convergence criteria as $\mathcal{B}_f = 1 \times 10^{-2}$, $\mathcal{Z}_f = 5 \times 10^{-3}$, and $\mathcal{R}_f = 1 \times 10^{-2}$. The learning rate for ADAM optimization is 5×10^{-3}. Unless otherwise noted, all further neural network discretizations will use hyperbolic tangent activation functions. The inhomogeneous Dirichlet BCs are defined by evaluating the assumed solution on the surface integration grid from marching cubes. We take a $75 \times 75 \times 75$ integration grid inside the unit cube to build the representation of the geometry. This corresponds to 82537 interior integration points and 33160 surface integration points.

See Figure 9 for the results. Using our proposed methodology, the interior and boundary errors at the converged solution are $\mathcal{B} = 0.89\%$ and $\mathcal{I} = 0.51\%$, respectively. The run time is 1269.8 seconds for 6385 optimization steps. Next, we run the learning rate annealing method with the same network and learning rate for the same number of steps to compare the performance. Per the recommendation of the authors [70], we set the moving average parameter in Eq. (8) to $\alpha = 0.9$. See Figure 10 for the results of this simulation. The interior and boundary errors after the allotted

training period are $\mathcal{B} = 3.6\%$ and $\mathcal{I} = 2.9\%$, and the run time is 1297.8 seconds. In this example, the Augmented Lagrangian achieves markedly better results.

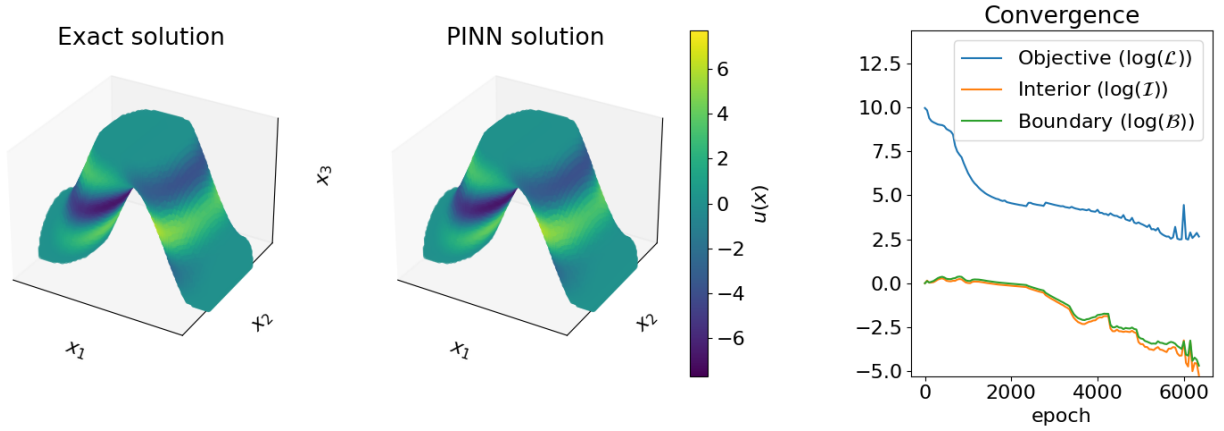

Figure 9: Results of the PINN simulation of the Fisher-KPP equation defined on the two-way branching geometry with inhomogeneous Dirichlet BCs. The Augmented Lagrangian method is used to enforce the boundary conditions.

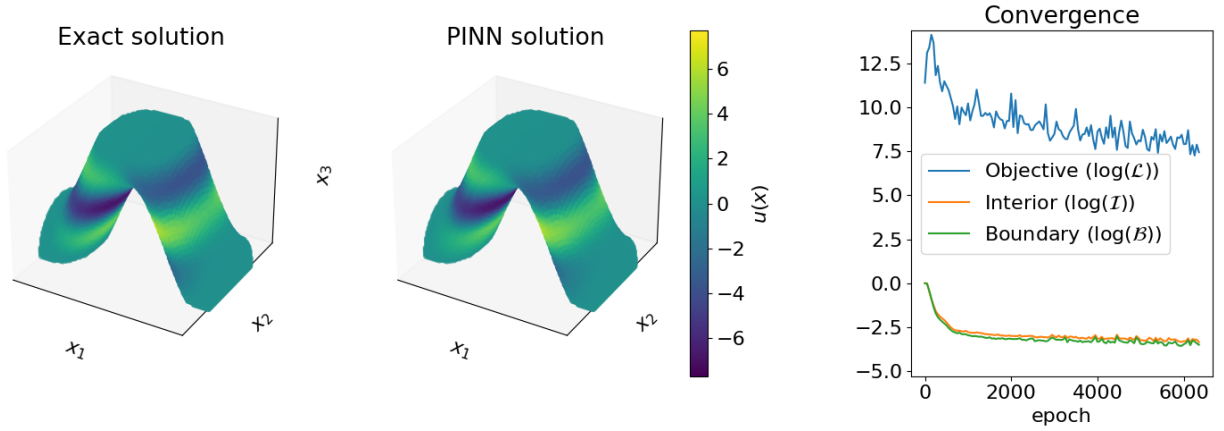

Figure 10: Results of the PINN simulation of the Fisher-KPP equation on the branching geometry using the learning rate annealing method. We see that the interior and boundary error stagnate at large values compared to those obtained by the Augmented Lagrangian.

5.2 Heat transfer with nonlinear Robin boundary and tabletop geometry

The second test case will be another scalar-valued static problem with a nonlinearity in the BC rather than the differential operator. We consider steady-state heat transfer of a functionally graded three-dimensional geometry with a mix of Dirichlet and Robin BCs. The Robin BCs are a combination of a heat flux input and radiative heat loss. The

problem is given by

$$\begin{aligned} \nabla \cdot \Big(\kappa(\mathbf{x}) \nabla u(\mathbf{x})\Big) + f(\mathbf{x}) = 0, \quad & \mathbf{x} \in \omega, \\ u(\mathbf{x}) = g(\mathbf{x}), \quad & \mathbf{x} \in \partial\omega_D, \\ -\nabla u(\mathbf{x}) \cdot \hat{\mathbf{n}} = q(\mathbf{x}) + \sigma u^4, \quad & \mathbf{x} \in \partial\omega_R, \end{aligned} \tag{18}$$

where $\partial\omega_D$ is the Dirichlet region of the boundary, $\partial\omega_R$ is the Robin region, $\kappa(\mathbf{x})$ is the thermal conductivity, $f(\mathbf{x})$ is a volumetric heat source, and $q(\mathbf{x})$ is a prescribed boundary heat flux. Though the Robin boundary is not explicitly discussed in Section 4, it requires only modifying the Neumann section of the boundary to include the solution field in the BC along with the flux. The problem geometry is that of a four-legged "tabletop" defined by the following level set:

$$\Phi(\mathbf{x}) = \Big(\cos 2\pi x - (1 - 5z^2)\Big)^2 + 9(y - 0.5)^2 + \Big(\cos 2\pi y - (1 - 5z^2)\Big)^2 + 9(x - 0.5)^2 - 3.$$

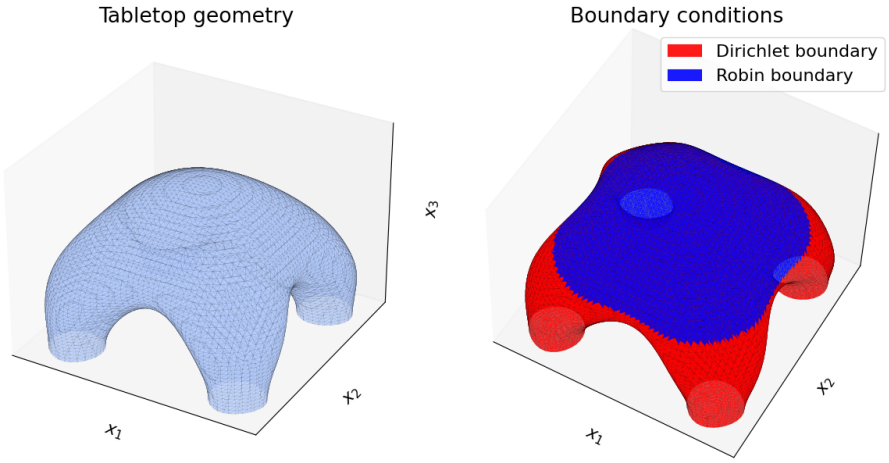

Figure 11: Problem geometry for forced heat transfer with radiation BCs.

The boundary is split into Dirichlet and Robin regions as follows:

$$\partial\omega_D = \{\mathbf{x} | \Phi(\mathbf{x}) = 0, x_3 < 0.5\}, \quad \partial\omega_R = \{\mathbf{x} | \Phi(\mathbf{x}) = 0, x_3 \geq 0.5\}.$$

See Figure 11 to visualize the geometry defined by the zero isocontour of this level set and the two regions of the boundary. Once again, the zero isocontour extends outside of the unit cube on which the background integration grid is defined. In this case, this means that the $x_3 = 0$ boundary surfaces are not detected by marching cubes. We enforce a homogeneous Dirichlet boundary along this surface in the discretization with

$$\hat{u}(\mathbf{x}; \boldsymbol{\theta}) = x_3 \mathcal{N}(\mathbf{x}; \boldsymbol{\theta}).$$

The forcing term and BCs will be computed through a manufactured solution. The assumed temperature field and the graded conductivity are given by

$$u(\mathbf{x}) = 2x_3(1 + \sin(2\pi x_1)\sin(2\pi x_2)), \quad \kappa(\mathbf{x}) = 1 + x_3.$$

The forcing $f(\mathbf{x})$ is computed by substituting the manufactured solution into the governing equation for the bulk of the material. The applied heat flux $q(\mathbf{x})$ is computed at the boundary integration points with the manufactured solution per

$$q(\mathbf{x}) = -\nabla u \cdot \hat{\mathbf{n}} - \sigma u^4, \quad \mathbf{x} \in \partial\omega_R.$$

We discretize the solution with a two-hidden-layer MLP with 30 hidden units per layer and set the emissivity at $\sigma = 0.1$. The learning rate for ADAM optimization is set at 5×10^{-3}. The convergence criteria are set at $\mathcal{B}_f = 7.5 \times 10^{-3}$, $\mathcal{Z}_f = 5 \times 10^{-3}$, and $\mathcal{R}_f = 1 \times 10^{-2}$. The background integration grid is $75 \times 75 \times 75$, corresponding to 117486 interior integration points and 47720 surface integration points. See Figure 12 for results. The run time of this problem is 464.9 seconds for 1702 optimization steps. The two error measures at the converged solution are $\mathcal{B} = 0.66\%$ and $\mathcal{I} = 0.53\%$. The Augmented Lagrangian method converges in a small number of epochs even with a large number of nonlinear constraints on the boundary. We then run the learning rate annealing with the same network and learning rate, using an update weight of $\alpha = 0.9$. We independently balance the gradients of the two loss terms corresponding to the Dirichlet and Robin boundaries per [70]. See Figure 13 for the results of the simulation. After the allotted 1702 steps, the learning rate annealing method obtains a boundary error of $\mathcal{B} = 0.84\%$ and an interior error of $\mathcal{I} = 0.74\%$ with a run time of 587.3 seconds. We note that we loop through parameters to manually set gradients in with learning rate annealing as opposed to using a `.backward()` call in PyTorch, which helps explain the increase in run time. In this example, the Augmented Lagrangian and learning rate annealing methods perform similarly.

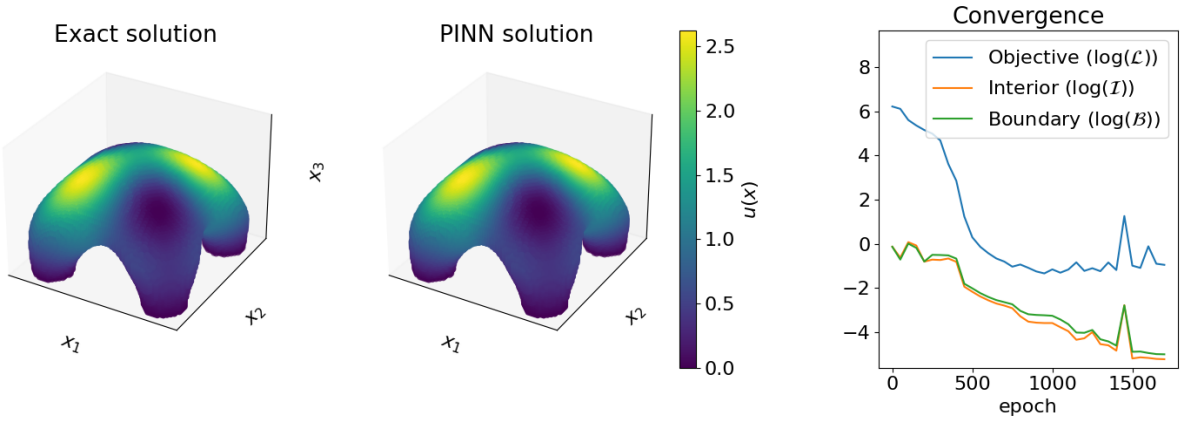

Figure 12: Results of the heat conduction problem using the Augmented Lagrangian method on the tabletop geometry with a mix of inhomogeneous Dirichlet and radiative Robin BCs. The log of the absolute value of the objective function is shown in the convergence plot, given that it becomes negative due to the presence of the Lagrange multiplier term.

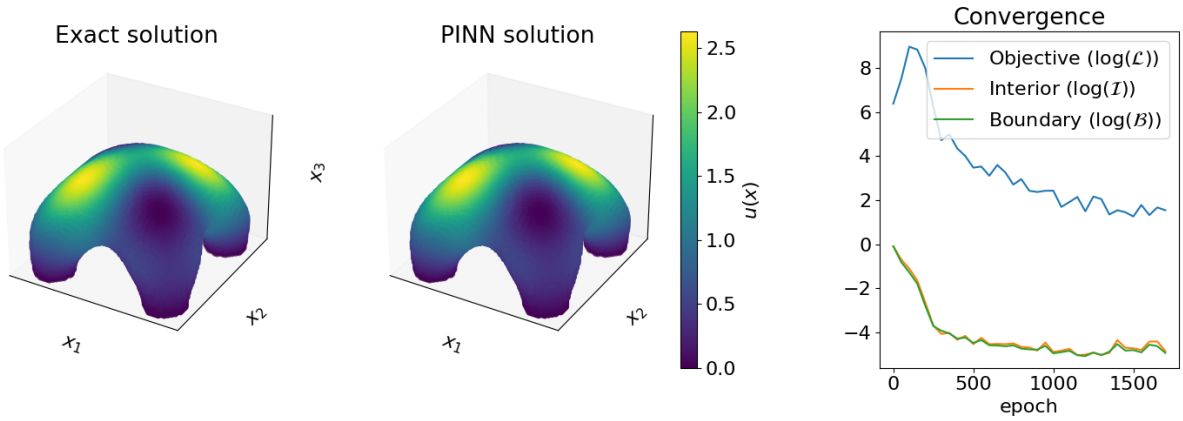

Figure 13: Results of the heat conduction problem using the learning rate annealing method. In this example, there are no notable differences between the performance of the two techniques for BC enforcement.

5.3 Linearly elastic non-prismatic pipe

We now perform a test on a vector-valued problem with inhomogeneous boundaries and a problem geometry with a different topology than the previous two examples. The problem is that of linear elastic deformation of a pipe with a varying cross-section. The governing equation for linear elasticity is

$$\begin{aligned} \nabla \cdot \boldsymbol{\sigma} + \mathbf{f}(\mathbf{x}) &= \mathbf{0}, \quad \mathbf{x} \in \omega, \\ \mathbf{u}(\mathbf{x}) &= \mathbf{g}(\mathbf{x}), \quad \mathbf{x} \in \partial\omega_D, \\ \boldsymbol{\sigma} \cdot \hat{\mathbf{n}} &= \mathbf{t}(\mathbf{x}), \quad \mathbf{x} \in \partial\omega_N, \\ \sigma_{ij} = C_{ijk\ell}\epsilon_{k\ell}, \quad \epsilon_{k\ell} &= \frac{1}{2}\left(\frac{\partial u_k}{\partial x_\ell} + \frac{\partial u_\ell}{\partial x_k}\right), \end{aligned} \tag{19}$$

where $C_{ijk\ell} = \lambda\delta_{ij}\delta_{kl} + \mu(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk})$ is the constitutive tensor corresponding to an isotropic, homogeneous linearly elastic material. The two Lame parameters are related to tabulated material parameters by

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}, \quad \mu = \frac{E}{2(1+\nu)},$$

where E is the Young's modulus and ν is the Poisson ratio. The non-prismatic pipe geometry is defined by the following level set:

$$\Phi(\mathbf{x}) = \Big((x_3 - 0.5)^2 + (x_2 - 0.5)^2 - R_2^2 - a^2 \exp\big(-100(x_1 - 0.5)^2\big)\Big)\Big((x_3 - 0.5)^2 + (x_2 - 0.5)^2 - R_1^2\Big),$$

where R_1 and R_2 define the inner and outer radii of the pipe at the two edges of the boundary, and a defines the increase of the outer radius at the center of the pipe. See Figure 14 to visualize the geometry via cross-sections. We assume a manufactured solution corresponding to a radially symmetric expansion of the pipe whose magnitude is

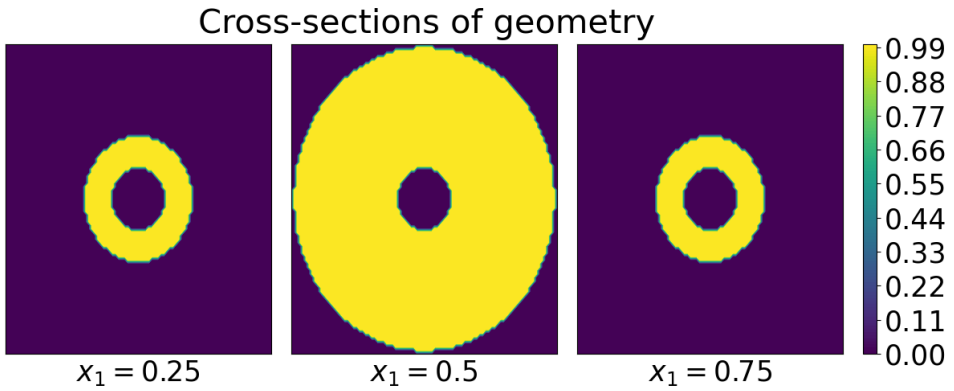

Figure 14: The problem geometry given by the level set is that of a non-prismatic pipe, where the outer radius obtains a maximum at the center. In these plots, the inside of the volume is given by a phase value of 1 and outside by a phase value of 0.

maximum at the center and falls off to zero at the two ends of the pipe. Defining the distance from the axis of the pipe as $r = \sqrt{(x_2 - 0.5)^2 + (x_3 - 0.5)^2}$, the assumed solution is

$$\mathbf{u}(\mathbf{x}) = u_0 \begin{bmatrix} 0 \\ (x_2 - 0.5) r \sin(\pi x_1) \\ (x_3 - 0.5) r \sin(\pi x_1) \end{bmatrix}, \tag{20}$$

where u_0 controls the magnitude of the displacement field. Note that this displacement field corresponds to clamped Dirichlet boundaries on the $x_1 = 0$ and $x_1 = 1$ faces of the domain. This is enforced automatically in the discretized solution with

$$\hat{\mathbf{u}}(\mathbf{x}; \boldsymbol{\theta}) = \sin(\pi x_1) \mathcal{N}(\mathbf{x}; \boldsymbol{\theta}),$$

where $\mathcal{N}(\mathbf{x}; \boldsymbol{\theta})$ is a vector-valued MLP neural network. The rest of the boundary will be inhomogeneous Dirichlet with values computed from evaluating the surface integration grid on the manufactured solution given by Eq. (20). Note that the error measures of Eqs. (12) are modified to accommodate the vector-valued solution field per

$$\mathcal{I}(\boldsymbol{\theta}) = \frac{\int_\omega \|\mathbf{u}(\mathbf{x}) - \hat{\mathbf{u}}(\mathbf{x}; \boldsymbol{\theta})\| d\mathbf{x}}{\int_\omega \|\mathbf{u}(\mathbf{x})\| d\mathbf{x}}, \quad \mathcal{B}(\boldsymbol{\theta}) = \frac{\int_{\partial\omega} \|\mathbf{g}(\mathbf{x}) - \hat{\mathbf{u}}(\mathbf{x}; \boldsymbol{\theta})\| ds}{\int_{\partial\omega} \|\mathbf{g}(\mathbf{x})\| ds}.$$

The Augmented Lagrangian algorithm of Eq. (14) is also modified to handle the vector-valued solution by replacing the squared residual of the strong form loss and the Dirichlet boundary with the squared magnitude of each residual. We take $u_0 = 25$ and use a two-hidden-layer neural network with 50 hidden units per layer and a learning rate of 1×10^{-3}. The material properties are $E = 1$ and $\nu = 0.3$. The convergence criteria are set at $\mathcal{B}_f = 5 \times 10^{-3}$, $\mathcal{Z}_f = 2.5 \times 10^{-3}$, and $\mathcal{R}_f = 1 \times 10^{-2}$, and the background integration grid is $75 \times 75 \times 75$ uniformly spaced points. This corresponds to 83164 interior integration points and 45168 surface integration points. See Figure 15 for a three-dimensional rendering of the geometry and the results of the simulation. The run time of this problem is 1855.8 seconds. The two error measures at the converged solution are $\mathcal{I} = 0.51\%$ and $\mathcal{B} = 0.49\%$. The method converges in 1803 steps. We remark

that in each of the three numerical examples, the inner loop problem in Algorithm 1 converges no more than 11 times, meaning that the maximum penalty parameter used in the Augmented Lagrangian method is $2^{11} = 2048$. Finally, we re-run the simulation using the learning rate annealing method with the same network, learning rate, and number of steps as the Augmented Lagrangian. See Figure 16 for the results. The two error measures from learning rate annealing are $\mathcal{I} = \mathcal{B} = 1.1\%$, which are more than twice that of their counterparts from the Augmented Lagrangian. The run time of the numerical experiment is 1788.1 seconds.

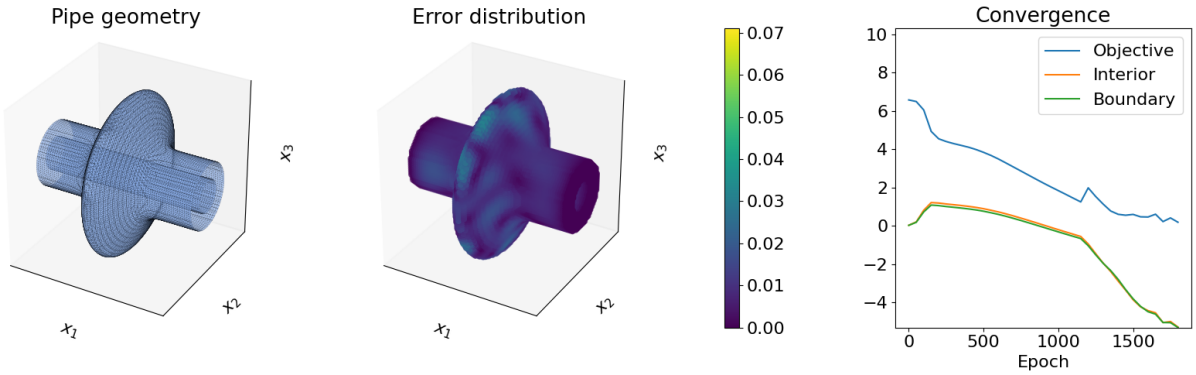

Figure 15: Results of the PINN simulation of the non-prismatic pipe using the Augmented Lagrangian method to enforce inhomogeneous Dirichlet boundaries. Convergence is obtained in 1803 optimization steps.

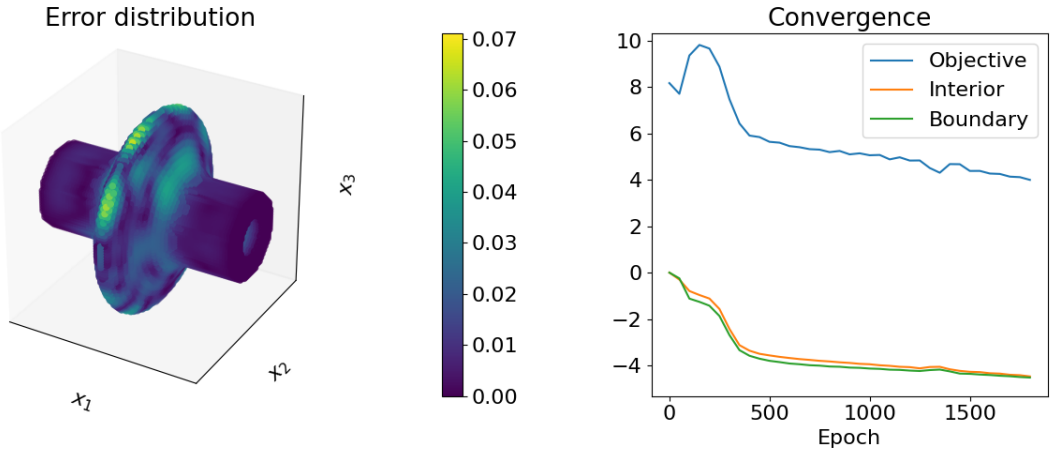

Figure 16: Results of the PINN simulation of the non-prismatic pipe using the learning rate annealing method. We obtain larger errors when running the method for the same number of steps as the Augmented Lagrangian.

6 Conclusion

The goal of this work was to discuss and study aspects of PINNs for PDEs in order to move them closer to a bona fide numerical solution framework, capable of applying to an equally large class of physical models, geometries, and BCs as the finite element method. To this end, we began with a discussion of the loss function, arguing that the strong form loss is the best choice if generality and exploiting the expressivity of the neural network are primary concerns.

Next, we conducted an extensive review of techniques for BC enforcement. We ruled out a number of such methods on first principles, and we studied the remaining methods in the context of a two-dimensional model problem. This led us into a detour discussion of Newton and quasi-Newton optimization methods, and to the conclusion that these methods struggle to handle the large number of nonlinear constraints arising from neural network discretizations. We ultimately settled on the Augmented Lagrangian and learning rate annealing as the two candidate techniques for BC enforcement, but chose to emphasize the Augmented Lagrangian in our exposition of a solution framework for three-dimensional geometries. We then studied the proposed method as well as learning rate annealing on three three-dimensional example problems with complex geometries and BCs. Using the method of manufactured solutions to obtain exact solutions, we showed that both techniques for BC enforcement can obtain accurate approximations with a modest parameter count and run times on the order of minutes. In the two-dimensional study, the Augmented Lagrangian outperformed learning rate annealing. This was also the case in two of the three three-dimensional numerical examples. While these results are not definitive, the Augmented Lagrangian method is the preferred strategy for handling BCs with PINNs in the opinion of the authors.

In synthesizing existent strategies in the literature to enforce Dirichlet, Neumann, and/or Robin BCs on arbitrary three-dimensional domains, our work pushes PINNs one step closer to having the same impressive range of capabilities as incumbents such as FEM. However, many challenges remain. As seen in Figure 6, we encountered convergence issues in the PINN problem when using a quasi-Newton optimizer with constraints. Though second-order optimization methods have recently become popular in the PINNs literature [30, 66, 28], there has been limited discussion of challenges unique to second-order methods in the context of non-convex problems. One example of this is the convergence of Newton methods to saddle points and maxima [12]. The convergence issues we encountered and the tailoring of second-order methods to avoid concave regions of the loss landscape are opportunities for further research. Future work on this solution method will focus on extensions to dynamic PDEs on three-dimensional domains, as well as devising techniques to streamline the training of networks representing high-frequency solution fields. The numerical examples contained in this work show that standard PINN-based solutions are an accurate and efficient technique when the solution field is smooth and minimally oscillatory. However, numerical experimentation indicates that high-frequency solution fields present challenges for PINN solutions over and above the complexity of the geometry and/or BCs. This problem has been noted, and various remedies have been proposed in the PINNs literature [48, 71, 62, 29]. Future work is required to explore the efficacy of these architectural modifications in the context of three-dimensional geometries and the Augmented Lagrangian method for BC enforcement. Devising methods that are accurate, reliable, and efficient on complex geometries and in the presence of high-frequency solutions will represent a decisive step in establishing PINNs as a mature numerical method, capable of competing head-to-head with traditional techniques such as FEM.

Acknowledgments

This work was funded by the National Defense Science and Engineering Graduate Fellowship (NDSEG) through the Department of Defense (DOD) and the Army Research Office (ARO). A. Doostan was supported by the Department of Energy, National Nuclear Security Administration, Predictive Science Academic Alliance Program (PSAAP) Award Number DE-NA0003962.

References

[1] Diab W. Abueidda, Seid Koric, Rashid Abu Al-Rub, Corey M. Parrott, Kai A. James, and Nahil A. Sobh. A deep learning energy method for hyperelasticity and viscoelasticity. *European Journal of Mechanics - A/Solids*, 95:104639, September 2022. arXiv:2201.08690 [cs].

[2] Ameya D. Jagtap Ameya D. Jagtap and George Em Karniadakis George Em Karniadakis. Extended Physics-Informed Neural Networks (XPINNs): A Generalized Space-Time Domain Decomposition Based Deep Learning Framework for Nonlinear Partial Differential Equations. *Communications in Computational Physics*, 28(5):2002–2041, January 2020.

[3] Sokratis J. Anagnostopoulos, Juan Diego Toscano, Nikolaos Stergiopulos, and George Em Karniadakis. Residual-based attention in physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 421:116805, March 2024.

[4] Shamsulhaq Basir and Inanc Senocak. Physics and equality constrained artificial neural networks: Application to forward and inverse problems with multi-fidelity data fusion. *Journal of Computational Physics*, 463:111301, August 2022.

[5] S. Berrone, C. Canuto, M. Pintore, and N. Sukumar. Enforcing Dirichlet boundary conditions in physics-informed neural networks and variational physics-informed neural networks. *Heliyon*, 9(8):e18820, August 2023.

[6] Stefano Berrone, Claudio Canuto, and Moreno Pintore. Variational Physics Informed Neural Networks: the Role of Quadratures and Test Functions. *Journal of Scientific Computing*, 92(3):100, August 2022.

[7] Rafael Bischof and Michael A. Kraus. Multi-Objective Loss Balancing for Physics-Informed Deep Learning. *Computer Methods in Applied Mechanics and Engineering*, 439:117914, May 2025.

[8] Samuel Burbulla. Physics-informed neural networks for transformed geometries and manifolds, November 2023. arXiv:2311.15940 [cs].

[9] Shengze Cai, Zhicheng Wang, Sifan Wang, Paris Perdikaris, and George Em Karniadakis. Physics-Informed Neural Networks for Heat Transfer Problems. *Journal of Heat Transfer*, 143(060801), April 2021.

[10] Zhiqiang Cai, Jingshuang Chen, Min Liu, and Xinyu Liu. Deep least-squares methods: an unsupervised learning-based numerical method for solving elliptic PDEs. *Journal of Computational Physics*, 420:109707, November 2020.

[11] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks, June 2018. arXiv:1711.02257 [cs].

[12] Yann Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, June 2014. arXiv:1406.2572 [cs].

[13] Yifan Du and Tamer A. Zaki. Evolutional Deep Neural Network. *Physical Review E*, 104(4):045303, October 2021. arXiv:2103.09959 [physics].

[14] Weinan E and Bing Yu. The Deep Ritz method: A deep learning-based numerical algorithm for solving variational problems, September 2017. arXiv:1710.00211 [cs].

[15] 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, February 2022.

[16] Yousef Ghaffari Motlagh, Peter K. Jimack, and René de Borst. Deep learning phase-field model for brittle fractures. *International Journal for Numerical Methods in Engineering*, 124(3):620–638, 2023.

[17] Herbert Goldstein, Charles Poole, John Safko, and Stephen R. Addison. Classical Mechanics, 3rd ed. *American Journal of Physics*, 70(7), July 2002.

[18] Qifeng Hu, Shamsulhaq Basir, and Inanc Senocak. Conditionally adaptive augmented Lagrangian method for physics-informed learning of forward and inverse problems using artificial neural networks, August 2025. arXiv:2508.15695 [cs].

[19] Zheyuan Hu, Ameya D. Jagtap, George Em Karniadakis, and Kenji Kawaguchi. When Do Extended Physics-Informed Neural Networks (XPINNs) Improve Generalization? *SIAM Journal on Scientific Computing*, 44(5):A3158–A3182, October 2022. arXiv:2109.09444 [cs].

[20] S. Hubrich and A. Düster. Numerical integration for nonlinear problems of the finite cell method using an adaptive scheme based on moment fitting. *Computers & Mathematics with Applications*, 77(7):1983–1997, April 2019.

[21] Thomas J. R. Hughes. The Finite Element Method, 2000.

[22] Ameya D. Jagtap, Ehsan Kharazmi, and George Em Karniadakis. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 365:113028, June 2020.

[23] Xiaowei Jin, Shengze Cai, Hui Li, and George Em Karniadakis. NSFnets (Navier-Stokes Flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations. *Journal of Computational Physics*, 426:109951, February 2021.

[24] Vijay Kag and Venkatesh Gopinath. Physics-informed neural network for modeling dynamic linear elasticity, January 2024. arXiv:2312.15175 [cs].

[25] E. Kharazmi, Z. Zhang, and G. E. Karniadakis. Variational Physics-Informed Neural Networks For Solving Partial Differential Equations, November 2019. arXiv:1912.00873 [cs].

[26] Ehsan Kharazmi, Zhongqiang Zhang, and George Em Karniadakis. hp-VPINNs: Variational Physics-Informed Neural Networks With Domain Decomposition. *Computer Methods in Applied Mechanics and Engineering*, 374:113547, February 2021.

[27] Reza Khodayi-Mehr and Michael M. Zavlanos. VarNet: Variational Neural Networks for the Solution of Partial Differential Equations, December 2019. arXiv:1912.07443 [cs].

[28] Elham Kiyani, Khemraj Shukla, Jorge F. Urbán, Jérôme Darbon, and George Em Karniadakis. Optimizing the Optimizer for Physics-Informed Neural Networks and Kolmogorov-Arnold Networks, August 2025. arXiv:2501.16371 [cs].

[29] Seungchan Ko and Sanghyeon Park. VS-PINN: A fast and efficient training of physics-informed neural networks using variable-scaling methods for solving PDEs with stiff behavior. *Journal of Computational Physics*, 529:113860, May 2025.

[30] Mikalai Korbit, Adeyemi D. Adeoye, Alberto Bemporad, and Mario Zanon. Exact Gauss-Newton Optimization for Training Deep Neural Networks, May 2024. arXiv:2405.14402 [cs].

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

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

[33] Wing Tat Leung, Guang Lin, and Zecheng Zhang. NH-PINN: Neural homogenization-based physics-informed neural network for multiscale problems. *Journal of Computational Physics*, 470:111539, December 2022.

[34] Xi'an Li, Jiaxin Deng, Jinran Wu, Shaotong Zhang, Weide Li, and You-Gan Wang. Physical informed neural networks with soft and hard boundary constraints for solving advection-diffusion equations using Fourier expansions, December 2023. arXiv:2306.12749 [math].

[35] Zongyi Li, Daniel Zhengyu Huang, Burigede Liu, and Anima Anandkumar. Fourier Neural Operator with Learned Deformations for PDEs on General Geometries, May 2024. arXiv:2207.05209 [cs].

[36] Yulei Liao and Pingbing Ming. Deep Nitsche Method: Deep Ritz Method with Essential Boundary Conditions. *Communications in Computational Physics*, 29(5):1365–1384, January 2021.

[37] Kuan-Chung Lin, Kuo-Chou Wang, and Cheng-Hung Hu. Investigating deep energy method applications in thermoelasticity. *Engineering Analysis with Boundary Elements*, 159:302–314, February 2024.

[38] Min Liu, Zhiqiang Cai, and Karthik Ramani. Deep Ritz method with adaptive quadrature for linear elasticity. *Computer Methods in Applied Mechanics and Engineering*, 415:116229, October 2023.

[39] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, August 1987.

[40] Lu Lu, Raphaël 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, January 2021. Publisher: Society for Industrial and Applied Mathematics.

[41] Bahae-Eddine Madir, Francky Luddens, Corentin Lothodé, and Ionut Danaila. Physics Informed Neural Networks for heat conduction with phase change, October 2024. arXiv:2410.14216 [math].

[42] Charalambos G. Makridakis, Aaron Pim, and Tristan Pryer. A Deep Uzawa-Lagrange Multiplier Approach for Boundary Conditions in PINNs and Deep Ritz Methods, November 2024. arXiv:2411.08702 [math].

[43] M. Manav, R. Molinaro, S. Mishra, and L. De Lorenzis. Phase-field modeling of fracture with physics-informed deep learning. *Computer Methods in Applied Mechanics and Engineering*, 429:117104, September 2024.

[44] Levi McClenny and Ulisses Braga-Neto. Self-Adaptive Physics-Informed Neural Networks using a Soft Attention Mechanism. *Journal of Computational Physics*, 474:111722, February 2023. arXiv:2009.04544 [cs].

[45] Zeng Meng, Qiaochu Qian, Mengqiang Xu, Bo Yu, Ali Rıza Yıldız, and Seyedali Mirjalili. PINN-FORM: A new physics-informed neural network for reliability analysis with partial differential equation. *Computer Methods in Applied Mechanics and Engineering*, 414:116172, September 2023.

[46] Vien Minh Nguyen-Thanh, Xiaoying Zhuang, and Timon Rabczuk. A deep energy method for finite deformation hyperelasticity. *European Journal of Mechanics - A/Solids*, 80:103874, March 2020.

[47] Jorge Nocedal and Stephen J. Wright. *Quasi-Newton Methods*. Springer, New York, NY, 1999.

[48] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred A. Hamprecht, Yoshua Bengio, and Aaron Courville. On the Spectral Bias of Neural Networks, May 2019. arXiv:1806.08734 [stat].

[49] M. Raissi, P. Perdikaris, and G.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, February 2019.

[50] Franz M. Rohrhofer, Stefan Posch, Clemens Gößnitzer, and Bernhard C. Geiger. Approximating Families of Sharp Solutions to Fisher's Equation with Physics-Informed Neural Networks. *Computer Physics Communications*, 307:109422, February 2025. arXiv:2402.08313 [cs].

[51] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[52] Conor Rowan. Nonlinear discretizations and Newton's method: characterizing stationary points of regression objectives, October 2025. arXiv:2510.11987 [cs].

[53] Conor Rowan, Kurt Maute, and Alireza Doostan. Physics-informed solution reconstruction in elasticity and heat transfer using the explicit constraint force method, May 2025. arXiv:2505.04875 [cs].

[54] Andrzej Ruszczynski. *Nonlinear Optimization*. Princeton University Press, 2006.

[55] T. Sahin, M. von Danwitz, and A. Popp. Solving Forward and Inverse Problems of Contact Mechanics using Physics-Informed Neural Networks. *Advanced Modeling and Simulation in Engineering Sciences*, 11(1):11, May 2024. arXiv:2308.12716 [math].

[56] Francisco Sahli Costabal, Simone Pezzuto, and Paris Perdikaris. PINNs: Physics-informed neural networks on complex geometries. *Engineering Applications of Artificial Intelligence*, 127:107324, January 2024.

[57] R. I. Saye. High-Order Quadrature Methods for Implicitly Defined Surfaces and Volumes in Hyperrectangles. *SIAM Journal on Scientific Computing*, 37(2):A993–A1019, January 2015. Publisher: Society for Industrial and Applied Mathematics.

[58] Yong Shang and Fei Wang. Randomized Neural Networks with Petrov-Galerkin Methods for Solving Linear Elasticity Problems, August 2023. arXiv:2308.03088 [math].

[59] Yong Shang, Fei Wang, and Jingbo Sun. Deep Petrov-Galerkin Method for Solving Partial Differential Equations, January 2022. arXiv:2201.12995 [math].

[60] Hailong Sheng and Chao Yang. PFNN: A penalty-free neural network method for solving a class of second-order boundary-value problems on complex geometries. *Journal of Computational Physics*, 428:110085, March 2021.

[61] Justin Sirignano and Konstantinos Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, December 2018. arXiv:1708.07469 [q-fin].

[62] Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit Neural Representations with Periodic Activation Functions, June 2020. arXiv:2006.09661 [cs].

[63] Nahil Sobh, Rini Jasmine Gladstone, and Hadi Meidani. PINN-FEM: A Hybrid Approach for Enforcing Dirichlet Boundary Conditions in Physics-Informed Neural Networks, January 2025. arXiv:2501.07765 [cs].

[64] Hwijae Son, Sung Woong Cho, and Hyung Ju Hwang. Enhanced physics-informed neural networks with Augmented Lagrangian relaxation method (AL-PINNs). *Neurocomputing*, 548:126424, September 2023.

[65] N. Sukumar and Ankit Srivastava. Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks. *Computer Methods in Applied Mechanics and Engineering*, 389:114333, February 2022.

[66] Jorge F. Urbán, Petros Stefanou, and José A. Pons. Unveiling the optimization process of physics informed neural networks: How accurate and competitive can PINNs be? *Journal of Computational Physics*, 523:113656, February 2025.

[67] Jiaji Wang, Y.L. Mo, Bassam Izzuddin, and Chul-Woo Kim. Exact Dirichlet boundary Physics-informed Neural Network EPINN for solid mechanics. *Computer Methods in Applied Mechanics and Engineering*, 414:116184, September 2023.

[68] Min Wang, Haisheng Li, Haoxuan Zhang, Xiaoqun Wu, and Nan Li. PINN-MG: A physics-informed neural network for mesh generation, March 2025. arXiv:2503.00814 [cs].

[69] Sifan Wang, Shyam Sankaran, and Paris Perdikaris. Respecting causality for training physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 421:116813, March 2024.

[70] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient pathologies in physics-informed neural networks, January 2020. arXiv:2001.04536 [cs].

[71] Sifan Wang, Hanwen Wang, and Paris Perdikaris. On the eigenvector bias of Fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 384:113938, October 2021. arXiv:2012.10047 [cs].

[72] Yaohua Zang, Gang Bao, Xiaojing Ye, and Haomin Zhou. Weak Adversarial Networks for High-dimensional Partial Differential Equations. *Journal of Computational Physics*, 411:109409, June 2020. arXiv:1907.08272 [math].

[73] Zhaoyang Zhang, Qingwang Wang, Yinxing Zhang, Tao Shen, and Weiyi Zhang. Physics-informed neural networks with hybrid Kolmogorov-Arnold network and augmented Lagrangian function for solving partial differential equations. *Scientific Reports*, 15(1):10523, March 2025. Publisher: Nature Publishing Group.

[74] Chenhong Zhou, Jie Chen, Zaifeng Yang, and Ching Eng Png. Dual-Balancing for Physics-Informed Neural Networks, May 2025. arXiv:2505.11117 [cs].

[75] Wei Zhou and Y. F. Xu. Data-Guided Physics-Informed Neural Networks for Solving Inverse Problems in Partial Differential Equations, July 2024. arXiv:2407.10836 [cs].