PINNtegrate: PINN-based Integral-Learning for Variational and Interface Problems

Frank Ehebrecht[*,1,2] FEHEBRECHT@UNI-OSNABRUECK.DE
Toni Scharle[*,2] TSCHARLE@ROSEN-GROUP.COM
Martin Atzmueller[1,3] MARTIN.ATZMUELLER@UNI-OSNABRUECK.DE
[1] *Osnabrück University, Semantic Information Systems Group, Osnabrück, Germany*
[2] *ROSEN Technology and Research Center GmbH, Lingen, Germany*
[3] *German Research Center for AI (DFKI), Osnabrück, Germany*
[*] *equal contributions*

Editors: Cecília Coelho, Bernd Zimmering, M. Fernanda P. Costa, Luís L. Ferrás, Oliver Niggemann

Abstract

Physics Informed Neural Networks (PINNs) feature applications to various partial differential equations (PDEs) in physics and engineering. Many real-world problems contain interfaces, i. e., discontinuities in some model parameter, and have to be included in any relevant PDE solver toolkit. These problems do not necessarily admit smooth solutions. Therefore, interfaces cannot be naturally included into classical PINNs, since their learning algorithm uses the strong formulation of the PDE and does not include solutions in the weak sense. The interface information can be incorporated either by an additional flux condition on the interface or by a variational formulation, thus also allowing weak solutions. This paper proposes new approaches to combine either the weak or energy functional formulation with the piece-wise strong formulation, to be able to tackle interface problems. Our new method PINNtegrate can incorporate integrals into the neural network learning algorithm. This novel method cannot only be applied to interface problems but also to other problems that contain an integrand as an optimization objective. We demonstrate PINNtegrate on variational minimal surface and interface problems of linear elliptic PDEs.

Keywords: Partial Differential Equation, PDE, Elliptic PDE, Neural Net, PINN, Calculus of Variations, Weak Solution, Interfaces, Energy Functional, Numerical Integration, Integrals

1. Introduction

Physics Informed Neural Networks as a mean to find approximate solutions to Partial Differential Equations (PDEs) have been the focus of extensive research (Karniadakis et al., 2021; Cuomo et al., 2022). An important class of PDEs in engineering and science are elliptic PDEs: Given a domain $\Omega \subset \mathbb{R}^d$ in d dimensions we want to find $u \in H^1(\Omega, \mathbb{R})$, such that

$$\begin{cases} -\nabla \cdot (\mathbf{A}(\mathbf{x})\nabla u) + \mathbf{B}(\mathbf{x})\nabla u + C(\mathbf{x})u = f(\mathbf{x}) & \text{in } \Omega, \\ u = g(\mathbf{x}) & \text{on } \partial\Omega_D, \\ \partial_{\mathbf{n}} u = \mathbf{h}(x) & \text{on } \partial\Omega_N. \end{cases} \tag{1}$$

with $\mathbf{A}, \mathbf{B}$ and f from suitable function spaces (see Appendix A). For a general $\mathbf{A} \in L^\infty(\Omega, \mathbb{R}^{d\times d})$, those equations have a unique weak solution $u \in H^1(\Omega)$, i.e.

$$\int_\Omega \mathbf{A}(\mathbf{x})\nabla u \cdot \nabla\varphi + (\mathbf{B}(\mathbf{x}) \cdot \nabla u)\varphi + C(\mathbf{x})u\varphi \,\mathrm{d}x = \int_\Omega f(\mathbf{x})\varphi \,\mathrm{d}x \tag{2}$$

for all $\varphi \in H_0^1(\Omega, \mathbb{R})$ or $\varphi \in H^1(\Omega, \mathbb{R})$ for Dirichlet or Neumann boundary conditions, respectively. We will denote this as an interface problem if $\mathbf{A}$ is piece-wise differentiable on disjoint subdomains Ω_i for $i = 1, 2$ with interface $\Gamma = \partial\Omega_1 \cap \partial\Omega_2$. Thus, the weak solution u is a piece-wise strong solution by elliptic regularity. Note that this is equivalent to enforcing the flux conditions $u_1 = u_2$ and $\partial_n \mathbf{A}_1 u_1 = \partial_n \mathbf{A}_2 u_2$ at the interface, where the indices $1, 2$ represent the limits from the left or the right of the interface, respectively. However, the task of computing normal derivatives along the interface is not necessary, if we interpret the interface problem as finding a weak solution on the entire domain Ω. Another way of approaching this problem is interpreting it as an optimization problem. In their most general form those problems are given by finding the unique minimizer of the functional

$$J[u] \quad = \int_\Omega F(\mathbf{x}, u, \nabla u, \nabla^2 u, \ldots, \nabla^n u)\, d\mathbf{x}. \tag{3}$$

In particular, this includes minimal surface problems that we will investigate in Section 4.1. For $\mathbf{B} = 0$, we note that a weak solution to Equation 1 is the unique minimizer of the energy functional of Equation 3 with $F = \frac{1}{2}\mathbf{A}(\mathbf{x})\nabla u \cdot \nabla u + \frac{1}{2}C(\mathbf{x})u^2 - f(\mathbf{x})u$ for $u : \Omega \to \mathbb{R}$ in a suitable function space and F a suitably regular function. All of those formulations have in common that we have to evaluate some integral and optimize the integrand in order to solve them within a PINN framework. However, it turns out that the standard Monte Carlo integration method often leads to unstable training in such cases. In this paper, we introduce our method PINNtegrate to tackle this problem in the form of a PDE that can be modeled via a hard-constraint PINN on its own.

Our contributions in this paper are three-fold:

1. We propose a new method called PINNtegrate, that can incorporate integral-expressions into a deep learning algorithm.
2. We present two new approaches to tackle PDE interface problems in a PINN setup. For this, we combine the strong formulation with the energy-functional or the weak formulation and call these methods *semi-variational* PINNs and *semi-weak* PINNs.
3. We demonstrate our method implementing those approaches using several examples: we consider a minimal surface problem, showcasing the efficacy of the PINNtegrate method in a stand-alone setting and three interface problems for linear elliptic PDEs.

2. Related Work

The capability of artificial neural networks as universal function approximators (Cybenko, 1989) has been used to approximate solutions for PDEs as early as in Lagaris et al. (1997). This has been resumed by Raissi et al. (2017) and Berg and Nyström (2018) and has become popular under the name *Physics Informed Neural Networks* (PINNs), c. f., (Karniadakis et al., 2021; Cuomo et al., 2022) for an overview.

A multitude of research has been published, either improving different components of the architecture and learning algorithm (e. g., sampling strategies (Wu et al., 2023) or adaptive activation functions (Jagtap et al., 2020a)), or approaches which used PINNs as an alternative to a variety of classical numerical solvers for forward and inverse problems (e. g., full waveform inversion in Rasht-Behesht et al. (2021)). A substantial part of this publication is dedicated to deep learning algorithms to find approximate solutions to linear elliptic interface problems. Since classical PINNs make use of the strong formulation of

PDEs, these interface conditions - i.e. a jump in some equation parameter - need to be handled separately. This can be accomplished by some additional loss, introducing an interface condition into the learning algorithm (e.g. some flux conditions for elliptic PDEs or hyperbolic conservation laws). PINNs pursuing this approach can be found in (Jagtap et al., 2020b; Cao et al., 2023; Li et al., 2020; Wu et al., 2022). A continuity condition of the PDE residuals on the interfaces as described in (Jagtap and Karniadakis, 2020) does not suffice for interface problems. Another way of accomplishing this, is by a formulation that not only covers solutions in the strong sense but also in the weak sense. In (Weinan and Yu, 2017; Wang and Zhang, 2020; Nabian and Meidani, 2018) the energy functional formulation gets incorporated into the learning algorithm. Specifically, Wang and Zhang (2020) tackle PDE interface problems with a loss solely posed in the energy functional form and incorporating this into the learning algorithm by summing over all collocation points. In our experiments we observed, that this is not necessarily a well-posed optimization target for all variational problems. In (Ryck et al., 2024; Kharazmi et al., 2019; Zang et al., 2019) the weak formulation gets incorporated into the learning algorithm. In (Ryck et al., 2024) this is used to obtain entropy solutions of scalar conservation laws and in (Zang et al., 2019) it is used to obtain approximate solutions to high-dimensional PDEs.

In contrast to existing approaches, we therefore stress two major foci of our work: While our PINNtegrate method provides a (1) general approach to incorporate integral-expressions into a deep learning algorithm, we (2) specifically provide a reliable methodology to calculate approximate solutions to PDE interface problems.

3. Method

In Section 3.1 we present our method PINNtegrate that can incorporate a definite integral over the domain $\Omega \subset \mathbb{R}^d$ into a neural network learning algorithm. We will see that while one can learn the integral I for provided integrand $i(\mathbf{x})$ with the help of automatic differentiation, our method's intention is for the integrand $i(\mathbf{x})$ to also be dynamically learned. This can either be done for a static target-value of the definite integral, but also by making the target-value learnable (e.g. minimizing it).

In the following notation, vectors of the independent variables will be set in boldface and their elements indexed by a subscript (i.e. $\mathbf{x} = (x_1, x_2, \dots, x_d)^T$). Sets of n sampling points (e.g. a batch of scalars or vectors) will be denoted as the set $\{x^i\}_{i \in I}$ (or $\{\mathbf{x}^i\}_{i \in I_n}$) with index set $I_n = \{1, 2, \dots, n\}$. The indices will be set as superscript in this case. For the sake of brevity we will denote these as $\{x^i\}$ (or $\{\mathbf{x}^i\}$). For the assignments of these sets to specific subsets as collocation points (c), Dirichlet boundary points (bD), Neumann boundary points (bN) or interface points (inter), we also add a corresponding superscript (e.g. $\{\mathbf{x}^{\mathrm{c},i}\}$).

3.1. PINNtegrate

We will first introduce the definite integral as a solution to a partial differential equation (PDE), and then show how a neural network can be used to learn an approximate solution to this PDE. In one dimension, the definite integral $I(x) = \int_{x_s}^{x} i(\xi)\, \mathrm{d}\xi$ can also be expressed as the initial value problem $\frac{dI}{dx}(x) - i(x) = 0$ with $I(x_s) = 0$. We want to be able to learn this integral in the general d-dimensional case with the help of a Neural Network $\mathfrak{N}_I(\mathbf{x}, \boldsymbol{\eta})$,

with model parameters $\boldsymbol{\eta}$ in a hard-constraint setup (see Appendix B.1):

$$\begin{aligned} g(\mathbf{x}, I_{\text{target}}) &= \left[\prod_{i=1}^{d}(x_i - x_{i,s})\right]\left[\frac{I_{\text{target}}}{\prod_{j=1}^{d}(x_{j,e} - x_{j,s})}\right] \\ \ell(\mathbf{x}) &= \left[\prod_{i=1}^{d}(x_i - x_{i,s})\right]\left[\sum_{j=1}^{d}(x_j - x_{j,e})\right] \\ I_{\boldsymbol{\eta}}(\mathbf{x}, I_{\text{target}}) &= \begin{cases} \ell(\mathbf{x})\mathfrak{N}_I(\mathbf{x}, \boldsymbol{\eta}), & \text{if } I_{\text{target}} = 0 \\ \ell(\mathbf{x})\mathfrak{N}_I(\mathbf{x}, \boldsymbol{\eta}) + g(\mathbf{x}, I_{\text{target}}), & \text{if } I_{\text{target}} \neq 0. \end{cases} \end{aligned} \tag{4}$$

This neural form $I_{\boldsymbol{\eta}}(\mathbf{x}, I_{\text{target}})$ can then be used to learn the integral for the provided integrand $i(\mathbf{x})$. Let us now consider, that the integrand $i(x)$ is of the general form as provided in the functional in Equation 3 $i(\mathbf{x}) \equiv F(\cdot, u, \nabla u, \nabla^2 u, \ldots, \nabla^m u)(\mathbf{x})$, for some given suitably regular function F and an unknown function u from a suitable function space. The objective is now either to find a u for a given integration target I_{target} or to find an u that minimizes I_{target}. For this, an approximation to u is learned by a neural network with learnable parameters $\boldsymbol{\theta}$ denoted as $u_{\boldsymbol{\theta}}(\mathbf{x}) := \mathfrak{N}_u(\mathbf{x}, \boldsymbol{\theta})$. This leads to a PINNtegrate loss term of

$$\begin{aligned} \mathcal{L}^P_{\boldsymbol{\eta},\boldsymbol{\theta}}(\{\mathbf{x}^{c,i}\}, I_{\text{target}}) = \frac{1}{N_c}\sum_{i=1}^{N_c}\Bigg[&\frac{\partial^d I_{\boldsymbol{\eta}}}{\partial x_1 \partial x_2 \ldots \partial x_d}(\mathbf{x}^{c,i}, I_{\text{target}}) \\ &- \mathbb{I}_\Omega(\mathbf{x}^{c,i}) F(\cdot, u_{\boldsymbol{\theta}}, \nabla u_{\boldsymbol{\theta}}, \nabla^2 u_{\boldsymbol{\theta}}, \ldots, \nabla^m u_{\boldsymbol{\theta}})(\mathbf{x}^{c,i})\Bigg]^2 \end{aligned} \tag{5}$$

In this formulation, the target value is fixed. To make it learnable, one has to also incorporate it into the learning algorithm. We express this by moving it to the subscript. Additionally, we introduce I_{target} as a loss term weighted with a small hyperparameter parameter α. While this additional term is not necessarily needed, it improves the rate of convergence.

$$\mathcal{L}^P_{\boldsymbol{\eta},\boldsymbol{\theta},I_{\text{target}}}(\mathbf{x}, \alpha) = \mathcal{L}_{\boldsymbol{\eta},\boldsymbol{\theta}}(\mathbf{x}, I_{\text{target}}) + \alpha\, I_{\text{target}} \tag{6}$$

Additionally, u needs to have its boundary conditions incorporated into the learning algorithm (see Appendix B.1). Section 4.1 shows that using Monte Carlo integration for training is unstable, while PINNtegrate leads to stable training.

3.2. Semi-Variational PINNs

While the strong formulation of Equation 1 does not suffice for interface problems, the formulation as energy functional of Equation 3 can completely describe systems with interfaces. The integrand of the energy functional can be combined with our PINNtegrate method to approximate solutions of u. In our experiments we have observed that a formulation solely by the energy functional does not provide a reliable learning algorithm. While for very simple problems (e.g. no interface at all or highly symmetric solutions) the energy functional may suffice, problems of reasonable complexity are likely to not converge to a reasonable approximation. Thus, additionally to the loss provided by the energy functional

in the PINNTEGRATE formulation, we include the strong formulation of the PDE into the loss term for each subdomain and coin this method *semi-variational*. We surmise that due to the localised nature of the strong formulation a loss term using this is numerically more stable, since every sample point of a learning batch must fulfill the condition individually. In contrast to this the energy functional formulation is global (non-localized) and a complete ensemble of points is needed for formulating the optimization target.

We again use a neural network $u_{\boldsymbol{\theta}}$ to approximate the solution to our problem. However, each subdomain needs its own neural network denoted as $u_{\boldsymbol{\theta}_j}$ for each subdomain Ω_j. By just writing $u_{\boldsymbol{\theta}}$ here, we indicate the usage of the correct neural network for the respective collocation point. With this definition we have a variational loss for a given set of collocation points $\{\mathbf{x}^{\mathrm{c},i}\}$ in the domain Ω_{H} of

$$\begin{aligned}\mathcal{L}^{\text{variational}}_{\boldsymbol{\eta},\boldsymbol{\theta},I_{\text{target}}}(\{\mathbf{x}^{\mathrm{c},i}\},\alpha) = \frac{1}{N_{\mathrm{c}}}\sum_{i=1}^{N_{\mathrm{c}}}\Bigg[&\frac{\partial^d I_{\boldsymbol{\eta}}}{\partial x_1 \partial x_2 \dots \partial x_d}(\mathbf{x}^{\mathrm{c},i}, I_{\text{target}}) \\ &- \mathbb{I}_{\Omega}(\mathbf{x}^{\mathrm{c},i})\Big(\frac{1}{2}\mathbf{A}(\mathbf{x}^{\mathrm{c},i})\nabla u(\mathbf{x}^{\mathrm{c},i})\cdot\nabla u(\mathbf{x}^{\mathrm{c},i}) + \frac{1}{2}C(\mathbf{x}^{\mathrm{c},i})u(\mathbf{x}^{\mathrm{c},i})^2 - f(\mathbf{x}^{\mathrm{c},i})u(\mathbf{x}^{\mathrm{c},i})\Big)\Bigg]^2 + \alpha I_{\text{target}}.\end{aligned} \tag{7}$$

Note, that this loss is formulated for the complete domain of the encasing hyperbox Ω_{H}. For the strong formulation loss we have to take a loss term for each subdomain Ω_j. Thus, the strong formulation for each subdomain

$$-\nabla\cdot(\mathbf{A}_j(\mathbf{x})\nabla u) + \mathbf{B}_j(\mathbf{x})\cdot\nabla u + C_j(\mathbf{x})u = f_j(\mathbf{x}) \quad \text{in } \Omega_j \tag{8}$$

for all $j = 1, \dots k$ leads to a strong formulation loss of

$$\begin{aligned}\mathcal{L}^{\text{strong}}_{\boldsymbol{\theta}}(\{\mathbf{x}^{c\Omega,i}\}) = \frac{1}{\sum_{j=1}^{k} N_j}\sum_{j=1}^{k}\sum_{i=1}^{N_j}\Bigg[&-\nabla\cdot(\mathbf{A}_j(\mathbf{x}^{c\Omega_j,i})\nabla u_{\boldsymbol{\theta}_j}(\mathbf{x}^{c\Omega_j,i})) \\ &+ \mathbf{B}_j(\mathbf{x}^{c\Omega_j,i})\cdot\nabla u_{\boldsymbol{\theta}_j}(\mathbf{x}^{c\Omega_j,i}) + C_j(\mathbf{x}^{c\Omega_j,i})u_{\boldsymbol{\theta}_j}(\mathbf{x}^{c\Omega_j,i}) - f_j(\mathbf{x}^{c\Omega_j,i})\Bigg]^2\end{aligned} \tag{9}$$

with $\{\mathbf{x}^{c\Omega,i}\}$ as the set of all collocation points in Ω and $\{\mathbf{x}^{c\Omega_j,i}\}$ as the set of collocation points in subdomain Ω_j. Putting together the strong formulation, the weak formulation via our PINNTEGRATE method, the interface condition and the Dirichlet boundary condition we arrive at Algorithm 2 (see Appendix B.2).

3.3. Semi-Weak PINNs

As in the previous section, we combine the strong formulation with a variational formulation. Here, we will use the weak formulation of the linear elliptic PDE (Equation 2).

To parameterize u, we again use a neural network $u_{\boldsymbol{\theta}}(\mathbf{x}) := \mathfrak{N}_u(\mathbf{x},\boldsymbol{\theta})$ with trainable parameters $\boldsymbol{\theta}$. The test-function φ also gets parameterized by a neural network $\mathfrak{N}_\varphi$ with parameters $\boldsymbol{\zeta}$ and can be formulated as

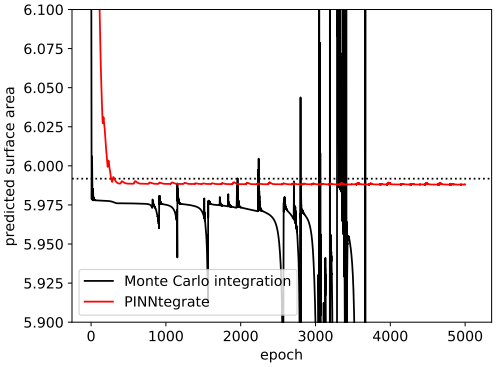
$$\begin{cases}\varphi_{\boldsymbol{\zeta}}(\mathbf{x}) := \ell_\varphi(\mathbf{x})\mathfrak{N}_\varphi(\mathbf{x},\boldsymbol{\zeta}) \\ \ell_\varphi(\mathbf{x}) \begin{cases} = 0, & \forall \mathbf{x} \text{ on } (\partial\Omega)_{\text{Dirichlet}} \\ > 0, & \text{else.}\end{cases}\end{cases} \tag{10}$$

In this setting, we arrive at a loss term for the weak formulation

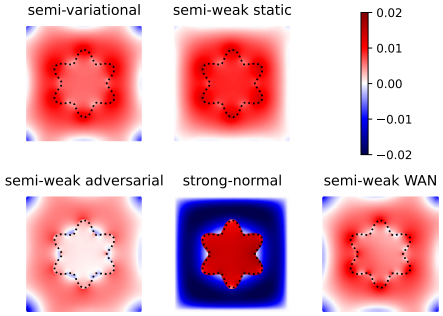
$$\mathcal{L}^{\text{weak}}_{\boldsymbol{\eta},\boldsymbol{\theta},\boldsymbol{\zeta}}(\{\mathbf{x}^{\text{c},i}\}) = \frac{1}{N_\text{c}} \sum_{i=1}^{N_\text{c}} \Bigg[\frac{\partial^d I_{\boldsymbol{\eta}}}{\partial x_1 \partial x_2 \dots \partial x_d}(\mathbf{x}^{c,i}, I_\text{target} = 0) - \mathbb{I}_\Omega(\mathbf{x}^{\text{c},i}) \Bigg(\mathbf{A}(\mathbf{x}^{\text{c},i}) \nabla u_{\boldsymbol{\theta}}(\mathbf{x}^{\text{c},i}) \cdot \nabla \varphi_{\boldsymbol{\zeta}}(\mathbf{x}^{\text{c},i}) + \mathbf{B}(\mathbf{x}^{\text{c},i}) \cdot \nabla u_{\boldsymbol{\theta}}(\mathbf{x}^{\text{c},i})\, \varphi_{\boldsymbol{\zeta}}(\mathbf{x}^{\text{c},i}) + C(\mathbf{x}^{\text{c},i}) u_{\boldsymbol{\theta}}(\mathbf{x}^{\text{c},i})\, \varphi_{\boldsymbol{\zeta}}(\mathbf{x}^{\text{c},i}) - f(\mathbf{x}^{\text{c},i})\, \varphi_{\boldsymbol{\zeta}}(\mathbf{x}^{\text{c},i}) \Bigg) \Bigg]^2 . \tag{11}$$

Since Equation 2 must hold for all suitable φ we must introduce an adversarial training step where we vary $\boldsymbol{\zeta}$ to maximize $\mathcal{L}^{\text{weak}}_{\boldsymbol{\eta},\boldsymbol{\theta},\boldsymbol{\zeta}}$. Using the strong formulation as posed in Equations 8 and 9 and boundary loss terms as posed in Equations 17 and 18, we can put everything together to form Algorithm 3.

4. Results



(*a*) Results for Section 4.1. The training with the Monte-Carlo approach diverges, while PINNTEGRATE converges to a good approximation.



(*b*) Error plots for the different approaches for the example given in Section 4.2

Figure 1: Results for Sections 4.1 and 4.2

We first present a variational problem to demonstrate our PINNTEGRATE method in a setting without strong formulation loss terms and compare the results to the analytical solution. We then present an experiment for the linear elliptic PDE of Equation 1 with an interface (two further examples with different domains and parameters can be found in Appendix C.1 and C.2). There, we use our *semi-variational* method and our *semi-weak* method. For the *semi-weak* method we also ran a simplified version with one static test function. While this does not guarantee a unique minimization target in general, our calculations showed acceptable results. The weak adversarial networks approach (WAN)

of (Zang et al., 2019) did not converge for our interface problems. For the example in Section 4.2, we added a combination of the WAN method with an additional strong loss term to show that in principle the information to obtain the correct solution is available. As a benchmark for the interface problems, we compared our results to implementations with flux-continuity conditions, i.e. a strong formulation loss together with a loss term using the interface condition in the normal direction of the surface as proposed in (Cao et al., 2023). We call this approach *strong-normal* in the following. While all our interfaces were given analytically, note that in practice, an interface would be most likely given as a set of points. To obtain normal vectors in this case, parameterization by arclength is needed (for the example in Section 4.2 we achieved this by a B-spline fit).

It is well-established, that fully connected Neural Networks within a range of number of layers and neurons, and activation functions are widely used for PINNs (see (Cuomo et al., 2022)). Here, all Neural Networks we used in our experiments have 5 layers with 50 neurons each and tanh activation functions. Collocation points were either sampled on a grid or at random. In all experiments, Adam optimizers were used (L-BFGS optimizers did not further improve the approximations). For the sake of readability, we omit axis labeling in some plots where the units are arbitrary.

The results for the linear elliptic PDE problems can be found in Table 1 (for more details see Table 2 in Appendix D). In these tables, the root mean square error (RMSE) to a high-resolution FEM ground truth and the weights for the loss terms and number of epochs are shown. The specific weights were determined empirically.

4.1. Variational Problem: 1D Minimal Surface Problem

To showcase our PINNtegrate method in a rather comprehensible setting, we first consider a variational problem with only one independent variable. We want to find the C^1-function $y^* : [a, b] \to \mathbb{R}$ with $y^*(x_a) = y_a$ and $y^*(x_b) = y_b$ with $x_a, x_b \in \mathbb{R}$, that minimizes the surface of revolution, i.e. the surface that is obtained by rotating the graph of y^* around the x-axis. The functional for this surface area is given by $I[y] = 2\pi \int_{x_a}^{x_b} y\sqrt{1 + y'^2}\,\mathrm{d}x$ and the optimal y can be obtained by finding $y^* = \underset{y}{\operatorname{argmin}}\, I[y]$. This expression has an analytical solution of the form $y^*(x) = c_1 \cosh(\frac{x}{c_1} + c_2)$ with $c_1, c_2 \in \mathbb{R}$ (see (Bronstein et al., 2015)). For our example we set $x_a = 0, y_a = 1, x_b = 1, y_b = 1$ (and obtain $c_1 \approx 0.848$ and $c_2 \approx -0.589$). We set the prediction of y as a hard-constrained neural form $y_{\boldsymbol{\theta}} = \mathfrak{N}_y(x, \boldsymbol{\theta})\, x\,(x - 1) + 1$, fulfilling the boundary conditions directly (and thus omitting the boundary loss term) and set up the integral network as described in Equation 4, to obtain a loss function as described in Equation 6. For this example, we provide this explicitly written out as

$$\mathcal{L}_{\boldsymbol{\eta},\boldsymbol{\theta},I_{\text{target}}}(\{x^{\text{c},i}\}) = \alpha I_{\text{target}} + \frac{1}{N_\text{c}} \sum_{i=1}^{N_\text{c}} \left[\frac{\partial I_{\boldsymbol{\eta}}}{\partial x}(x^{\text{c},i}, I_{\text{target}}) - 2\pi y_{\boldsymbol{\theta}}(x^{\text{c},i}) \sqrt{1 + \frac{\partial y_{\boldsymbol{\theta}}}{\partial x}(x^{\text{c},i})} \right]^2 . \tag{12}$$

This loss is then used in a training algorithm as described in Algorithm 1 with $N_{\text{epochs}} = 5000$ epochs, $N_\text{c} = 100$ equidistant sampled collocation points and an Adam optimizer with learning rate of $lr = 0.001$. The surface area starting value is set as $I_{\text{target,start}} = 2\pi$ since this is the surface area of the respective cylinder with given boundary conditions and seems

to be a good heuristic. Figure 1(*a*) shows the approximation of the learned minimal surface value during training for the cases of PINNtegrate and Monte Carlo integration. Our method successfully finds a good approximation of the analytical solution (dashed line). In the case of Monte Carlo integration, the learning algorithm becomes unstable and diverges at some point - even in this relatively easy example with one independent variable. The RMSE of our approximation to the analytical solution (based on the sampling points) is $\text{RMSE}_{\text{ex4.1}} = 2.5 \times 10^{-3}$

4.2. Elliptic PDE 'flower': Interface and Dirichlet Boundaries

In this example, we calculate an approximate solution to an interface problem of the linear elliptic PDE (Equation 1) in two dimensions. We set $f(\mathbf{x}) = 2$, $g(\mathbf{x}) = 0$, $\mathbf{B} = \mathbf{0}$ and $C = 0$. The interface Γ is described by the closed curve $\mathbf{x}_{\text{interface}} = (\sin(t)\,(\cos(Nt) + 5)\,, \cos(t)\,(\cos(Nt) + 5))^T\,, t \in [0, 2\pi]$ and $N = 5$. We sample this interface with $N_{\text{inter}} = 200$ points $\{\mathbf{x}^{\text{inter},i}\}$. We call the inner subdomain Ω_{inner} and set $a(\mathbf{x}) = 1$ for $x \in \Omega_{\text{inner}}$ and the outer subdomain Ω_{outer} and set $a(\mathbf{x}) = 6$ for $x \in \Omega_{\text{outer}}$. The outer boundary is the boundary of the square $[-1, 1] \times [-1, 1]$ and we sampled this with $N_{\text{bD}} = 400$ points ($\{\mathbf{x}^{\text{bD},i}\}$). We took $N_{\text{c}} = 10^4$ randomly sampled collocation points ($\{\mathbf{x}^{\text{c},i}\}$) on the complete domain. We ran both our two methods described in sections 3.2 and 3.3 to calculate approximate solutions to this problem. For our *semi-weak* method we also ran an experiment with only one static test function $\varphi_{\text{static}} = (x_1 - 1)(x_1 + 1)(x_2 - 1)(x_2 + 1)$. For the adversarial training, we used a hard-constraint neural network as described in Equation 10: $\varphi_{\boldsymbol{\zeta}}(\mathbf{x}) := (x_1 - 1)(x_1 + 1)(x_2 - 1)(x_2 + 1)\mathfrak{N}_{\varphi}(\mathbf{x}, \boldsymbol{\zeta})$. Additionally, we calculated approximations with the *strong-normal* method and the WAN method. However, the WAN method training only converged in a scenario where we also included the piece-wise strong formulation into the loss.

We tested our prediction against high resolution (66049 vertices) FEM calculation. The error plots for this can be seen in Figure 1(*b*) and RMSE values (using the set of FEM vertices) in Table 1. All methods yielded acceptable results, with our *semi-weak* method exhibiting the smallest RMSE.

Table 1: RMSEs of our experiments compared to high-resolution FEM calculations of all the examples for linear elliptic PDEs.

	flower	ellipse	mixed
semi-variational	6.4e-3	2.0e-3	-
semi-weak-static	6.9e-3	2.9e-3	2.1e-2
semi-weak-advers	3.7e-3	9.2e-4	1.6e-2
strong-normal	1.4e-2	9.4e-4	1.1e-2
semi-weak-WAN	5.3e-3	-	-

5. Conclusion

In this paper, we presented our novel method PINNtegrate: it can incorporate integrals into a neural network learning algorithm in such a way, that its integrand can be optimized. While this can be used for problems in calculus of variations, it can also be extended to be used for PDE interface problems: We combined the variational and the strong formulation of the PDE and implemented the variational formulation with our PINNtegrate method – calling these methods *semi-weak* and *semi-variational*. For demonstrating their successful application, in our experimentation we first tackled variational minimal surface problems for our PINNtegrate method in a stand-alone setting. Not only did this conclusively showcase our method, but we also demonstrated that posing such a problem with Monte Carlo integration may lead to unstable training. Furthermore, we solved interface problems of linear elliptic PDEs with our *semi-weak* and *semi-variational* method and compared the predictions to a high resolution FEM ground truth. Additionally, we used a method employing a flux condition for comparison. All methods yielded good and valid approximations, where our *semi-weak* method with adversarial training produced the best results in two experiments. Also, it does not require the calculation of normal vectors, which is a considerable advantage.

Future research directions include extensions to interface problems of other PDEs, if they can be posed as minimization-functional or in the weak formulation. Furthermore, we aim to investigate whether our PINNtegrate method could also be used for other problems where an integrand has to be optimized. This can, for example, be optimal control problems where a continuous-time cost functional needs to be minimized.

References

J. Berg and K. Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28–41, 2018.

I. N. Bronstein, K. A. Semendyayev, G. Musiol, and H. Muehlig. *Handbook of Mathematics*. Springer, 6th edition, 2015.

F. Cao, X. Guo, F. Gao, and D. Yuan. Deep learning nonhomogeneous elliptic interface problems by soft constraint physics-informed neural networks. *Mathematics*, 11(8), 2023.

S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli. Scientific machine learning through physics-informed neural networks: Where we are and what's next. *arXiv preprint arXiv:2201.05624*, 2022.

G. V. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.

A. D. Jagtap and G. E. 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*, 2020.

A. D. Jagtap, K. Kawaguchi, and G. E. Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, 404:109136, 2020a.

A. D. Jagtap, E. Kharazmi, and G. E. 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, 2020b.

G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

E. Kharazmi, Z. Zhang, and G. E. Karniadakis. Variational physics-informed neural networks for solving partial differential equations. *arXiv preprint arXiv:1912.00873*, 2019.

P. L. Lagari, L. H. Tsoukalas, S. Safarkhani, and I. E. Lagaris. Systematic construction of neural forms for solving partial differential equations inside rectangular domains, subject to initial, boundary and interface conditions. *Int. J. Artif. Intell. Tools*, 29:2050009:1–2050009:12, 2020.

I. E. Lagaris, A. C. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans. Neural Networks*, 9 5:987–1000, 1997.

W. Li, X. Xiang, and Y. Xu. Deep domain decomposition method: Elliptic problems. In *Mathematical and Scientific Machine Learning*, pages 269–286. PMLR, 2020.

M. A. Nabian and H. Meidani. A deep neural network surrogate for high-dimensional random partial differential equations. *arXiv preprint arXiv:1806.02957*, 2018.

M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv*, abs/1711.10561, 2017.

M. Rasht-Behesht, C. Huber, K. Shukla, and G. E. Karniadakis. Physics-informed neural networks (pinns) for wave propagation and full waveform inversions. *Journal of Geophysical Research: Solid Earth*, 127, 2021.

T. D. Ryck, S. Mishra, and R. Molinaro. wpinns: Weak physics informed neural networks for approximating entropy solutions of hyperbolic conservation laws. *SIAM J. Numer. Anal.*, 62:811–841, 2024.

Z. Wang and Z. Zhang. A mesh-free method for interface problems using the deep learning approach. *Journal of Computational Physics*, 400:108963, 2020.

E. Weinan and T. Yu. The deep ritz method: A deep learning-based numerical algorithm for solving variational problems. *Commun. Math. Stat.*, 6:1 – 12, 2017.

C. Wu, M. Zhu, Q. Tan, Y. Kartha, and L. Lu. A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks. *Comput. Methods Appl. Mech. Eng.*, 403:115671, 2023.

S. Wu, A. Zhu, Y. Tang, and B. Lu. On convergence of neural network methods for solving elliptic interface problems. *arXiv preprint arXiv:2203.03407*, 2022.

Y. Zang, G. Bao, X. Ye, and H. Zhou. Weak adversarial networks for high-dimensional partial differential equations. *J. Comput. Phys.*, 411:109409, 2019.

Appendix A. Introduction

The *suitable function spaces* of Section 1 are a uniformly elliptic matrix-valued function $\mathbf{A} \in L^\infty(\Omega, \mathbb{R}^{d\times d})$, $\mathbf{B} \in L^\infty(\Omega, \mathbb{R}^d)$, $\mathbf{A} \in L^\infty(\Omega, \mathbb{R})$, $f \in L^2(\Omega, \mathbb{R})$. In the isotropic case, we write $\mathbf{A}(\mathbf{x}) = a(\mathbf{x})\mathbb{1}_d$ for a scalar-valued function a with $0 < c \leq a(\mathbf{x}) \leq C < \infty$ for some positive constants c, C. In electrostatics, u describes the electric scalar potential and $\mathbf{A}$ represents the electric permittivity. In thermodynamics, the steady state of the temperature distribution in a body can be described by an equation of this form, where $\mathbf{A}$ represents the thermal diffusivity of the material. Note that in both cases $\mathbf{A}$ is not necessarily smooth, but can have jumps. This can occur, for example, at an interface between two materials. Therefore, a solution in the strong sense does not necessarily exist for an arbitrary $\mathbf{A}$, even for seemingly trivial problems. In such a case, an additional (physical / flux) interface condition is required, to have a well-posed problem. Figure 2 depicts such a problem setup.

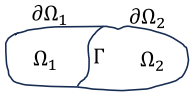


Figure 2: Ω is split into two subdomains Ω_1, Ω_2 via interface Γ.

Appendix B. Method

B.1. PINNtegrate

In the general d-dimensional case

$$I(\mathbf{x}) = \int_{x_{d,s}}^{x_d} \cdots \int_{x_{1,s}}^{x_1} i(\xi_1, \xi_2, \ldots, \xi_d)\, \mathrm{d}\xi_1 \cdots \mathrm{d}\xi_d \tag{13}$$

with $\mathbf{x}_s = (x_{1,s}, x_{2,s}, \ldots, x_{d,s})^T$ as the vector of lower bounds of the integral, this transforms to

$$\begin{cases} \frac{\partial^d I}{\partial x_1 \partial x_2 \ldots \partial x_d}(\mathbf{x}) - i(\mathbf{x}) & = 0 \\ I(\mathbf{x}) = 0 \quad \text{if } x_i = x_{i,s} & \text{for some } i \in \{1, \ldots, d\}. \end{cases} \tag{14}$$

This integrates over the domain of the hyperbox which is spanned by $\Omega_\mathrm{H} = [x_{1,s}, x_{1,e}] \times [x_{2,s}, x_{2,e}] \times \cdots \times [x_{n,s}, x_{n,e}]$ with upper bounds $\mathbf{x}_e = (x_{1,e}, x_{2,e}, \ldots, x_{n,e})^T$. However, one can extend the integrand from an arbitrary domain Ω to a hyperbox $\Omega_\mathrm{H} \supset \Omega$ by setting $i(x) = 0$ on $\Omega_\mathrm{H} \setminus \Omega$.

$$\frac{\partial^d I}{\partial x_1 \partial x_2 \ldots \partial x_d}(\mathbf{x}) := \mathbb{I}_\Omega(x)\, i(\mathbf{x}) = \begin{cases} i(\mathbf{x}) \text{ for } x \in \Omega \\ 0 \text{ for } x \in \Omega_\mathrm{H} \setminus \Omega. \end{cases} \tag{15}$$

Let us now define a neural network $\mathfrak{N}_I(\mathbf{x}, \boldsymbol{\eta})$, with model parameters $\boldsymbol{\eta}$ that will help us to learn I. While we could use this neural network directly to learn I, we still need to incorporate the boundary conditions. This could be done by introducing a term in the loss

function that penalizes the deviation of the neural network from the boundary conditions for a set of sampling points. However, a more elegant way is to use a hard-constraint (e.g. see (Lagari et al., 2020)) setup. For this, we need to constrain the boundary conditions and the integration target $I_{\text{target}} := I(\mathbf{x}_e)$:

$$\begin{aligned} g(\mathbf{x}, I_{\text{target}}) &= \left[\prod_{i=1}^{d}(x_i - x_{i,s})\right]\left[\frac{I_{\text{target}}}{\prod_{j=1}^{d}(x_{j,e} - x_{j,s})}\right] \\ \ell(\mathbf{x}) &= \left[\prod_{i=1}^{d}(x_i - x_{i,s})\right]\left[\sum_{j=1}^{d}(x_j - x_{j,e})\right] \\ I_{\boldsymbol{\eta}}(\mathbf{x}, I_{\text{target}}) &= \begin{cases} \ell(\mathbf{x})\mathfrak{N}_I(\mathbf{x}, \boldsymbol{\eta}), & \text{if } I_{\text{target}} = 0 \\ \ell(\mathbf{x})\mathfrak{N}_I(\mathbf{x}, \boldsymbol{\eta}) + g(\mathbf{x}, I_{\text{target}}), & \text{if } I_{\text{target}} \neq 0. \end{cases} \end{aligned} \tag{16}$$

For boundaries, we distinguish between Dirichlet and Neumann conditions as posed in Equation 1. The established way to formulate the respective loss terms for sets of Dirichlet sampling points $\{x^{\text{bD},i}\}$ is

$$\mathcal{L}_{\boldsymbol{\theta}}^{D}(\{x^{\text{bD},i}\}) = \frac{1}{N_{\text{bD}}}\sum_{i=1}^{N_{\text{bD}}}\left|u_{\boldsymbol{\theta}}(\mathbf{x}^{\text{bD},i}) - g(\mathbf{x}^{\text{bD},i})\right|^2 \tag{17}$$

and sets of Neumann sampling points $\{x^{\text{bN},i}\}$ it is

$$\mathcal{L}_{\boldsymbol{\theta}}^{N}(\{x^{\text{bN},i}\}) = \frac{1}{N_{\text{bN}}}\sum_{i=1}^{N_{\text{bN}}}\left|\mathbf{A}(\mathbf{x}^{\text{bN},i})\partial_{\mathbf{n}}u(\mathbf{x}^{\text{bN},i}) - h(\mathbf{x}^{\text{bN},i})\right|^2. \tag{18}$$

Putting everything together, we arrive at Algorithm 1.

B.2. Semi-Variational PINNs

The complete algorithm for the Semi-Variational PINN method is shown in Algorithm 2.

B.3. Semi-Weak PINNs

The complete Algorithm for the Semi Weak PINN method is shown in Algorithm 3

Appendix C. Results

C.1. Elliptic PDE 'ellipse': Interface and Dirichlet Boundaries

A linear elliptic PDE interface problem with additional complexity is obtained by altering the domain to an ellipse that is divided by an interface. The ellipse has its center at $(0.5, 0.5)$ with $a = 0.4$ the radius of the semi-major axis and $b = 0.25$ the radius of the semi-minor axis. Here, we indeed need to use an indicator function for our PINNtegrate method. We set $g(\mathbf{x}) = 0.1x_1$, $f(\mathbf{x}) = 10$, $\mathbf{B} = \mathbf{0}$ and $C(\mathbf{x}) = 50$. The interface is described by the function

$$x_2(x_1) = -2.5x_1 + 1.75 \tag{19}$$

Algorithm 1: PINNtegrate algorithm

Initialize network parameters $\boldsymbol{\theta}, \boldsymbol{\eta}$ uniformly
Initialize:$\{\mathbf{x}^{\mathrm{c},i}\}$, $\{\mathbf{x}^{\mathrm{bD},i}\}$, $\{\mathbf{u}^{\mathrm{bD},i}\}$, $\{\mathbf{x}^{\mathrm{bN},i}\}$, $\{\mathbf{u}^{\mathrm{bN},i}\}$, $\mathbb{I}_{\Omega}$
Initialize weights: $w_{\mathrm{P}}, w_{\mathrm{bD}}, w_{\mathrm{bN}}, \alpha$
Set number of N_{epochs}
$i_{\mathrm{epoch}} \leftarrow 0$;
while $i_{epoch} < N_{epochs}$ **do**
  $\tilde{\mathcal{L}}^{D}_{\boldsymbol{\theta}} \leftarrow w_{\mathrm{bD}} \mathcal{L}^{D}_{\boldsymbol{\theta}}(\{\mathbf{x}^{\mathrm{bD},i}\})$;
  $\tilde{\mathcal{L}}^{N}_{\boldsymbol{\theta}} \leftarrow w_{\mathrm{bN}} \mathcal{L}^{N}_{\boldsymbol{\theta}}(\{\mathbf{x}^{\mathrm{bN},i}\})$;
  if *static* I_{target} **then**
    $\tilde{\mathcal{L}}^{P}_{\boldsymbol{\eta},\boldsymbol{\theta}} \leftarrow w_{\mathrm{P}} \mathcal{L}^{P}_{\boldsymbol{\eta},\boldsymbol{\theta}}(\{\mathbf{x}^{\mathrm{c},i}\}, I_{\mathrm{target}})$;
    $\boldsymbol{\theta}, \boldsymbol{\eta} \leftarrow \mathrm{minimize}(\tilde{\mathcal{L}}^{P}_{\boldsymbol{\eta},\boldsymbol{\theta}} + \tilde{\mathcal{L}}^{D}_{\boldsymbol{\theta}} + \tilde{\mathcal{L}}^{N}_{\boldsymbol{\theta}})$;
  else if *learnable* I_{target} **then**
    $\tilde{\mathcal{L}}^{P}_{\boldsymbol{\eta},\boldsymbol{\theta},I_{\mathrm{target}}} \leftarrow w_{\mathrm{P}} \mathcal{L}^{P}_{\boldsymbol{\eta},\boldsymbol{\theta},I_{\mathrm{target}}}(\{\mathbf{x}^{\mathrm{c},i}\}, \alpha)$;
    $\boldsymbol{\theta}, \boldsymbol{\eta}, I_{\mathrm{target}} \leftarrow \mathrm{minimize}(\tilde{\mathcal{L}}^{P}_{\boldsymbol{\eta},\boldsymbol{\theta},I_{\mathrm{target}}} + \tilde{\mathcal{L}}^{D}_{\boldsymbol{\theta}} + \tilde{\mathcal{L}}^{N}_{\boldsymbol{\theta}})$;
  end
  $i_{\mathrm{epoch}} \leftarrow i_{\mathrm{epoch}} + 1$;
end

Algorithm 2: Semi-Variational PINN algorithm

Initialize network parameters $\boldsymbol{\theta}, \boldsymbol{\eta}, I_{\mathrm{target}}$
Initialize: $\{\mathbf{x}^{\mathrm{c},i}\}$, $\{\mathbf{x}^{\mathrm{inter},i}\}$, $\{\mathbf{x}^{\mathrm{bD},i}\}$, $\{\mathbf{u}^{\mathrm{bD},i}\}$, $\mathbb{I}_{\Omega}$
Initialize loss weights: $w_{\mathrm{P}}, w_{\mathrm{strong}}, w_{\mathrm{bD}}, w_{\mathrm{inter}}, \alpha$
$i_{\mathrm{epoch}} \leftarrow 0$;
while $i_{epoch} < N_{epochs}$ **do**
  $\tilde{\mathcal{L}}^{D}_{\boldsymbol{\theta}} \leftarrow w_{\mathrm{bD}} \mathcal{L}^{D}_{\boldsymbol{\theta}}(\{\mathbf{x}^{\mathrm{bD},i}\})$;
  $\tilde{\mathcal{L}}^{\mathrm{inter}}_{\boldsymbol{\theta}} \leftarrow w_{\mathrm{inter}} \mathcal{L}^{\mathrm{inter}}_{\boldsymbol{\theta}}(\{\mathbf{x}^{\mathrm{inter},i}\})$;
  $\tilde{\mathcal{L}}^{\mathrm{strong}}_{\boldsymbol{\theta}} \leftarrow w_{\mathrm{strong}} \mathcal{L}^{\mathrm{strong}}_{\boldsymbol{\theta}}(\{\mathbf{x}^{\mathrm{c}\Omega,i}\})$;
  $\tilde{\mathcal{L}}^{\mathrm{variational}}_{\boldsymbol{\eta},\boldsymbol{\theta},I_{\mathrm{target}}} \leftarrow w_{\mathrm{P}} \mathcal{L}^{\mathrm{variational}}_{\boldsymbol{\eta},\boldsymbol{\theta},I_{\mathrm{target}}}(\{\mathbf{x}^{\mathrm{c},i}\}, \alpha)$;
  $\boldsymbol{\theta}, \boldsymbol{\eta}, I_{\mathrm{target}} \leftarrow \mathrm{minimize}(\tilde{\mathcal{L}}^{\mathrm{variational}}_{\boldsymbol{\eta},\boldsymbol{\theta},I_{\mathrm{target}}} + \tilde{\mathcal{L}}^{D}_{\boldsymbol{\theta}} + \tilde{\mathcal{L}}^{\mathrm{strong}}_{\boldsymbol{\theta}} + \tilde{\mathcal{L}}^{\mathrm{inter}}_{\boldsymbol{\theta}})$;
  $i_{\mathrm{epoch}} \leftarrow i_{\mathrm{epoch}} + 1$;
end

Algorithm 3: Semi-Weak PINN algorithm

Initialize network parameters $\boldsymbol{\theta}, \boldsymbol{\eta}$ uniformly
Initialize $\{\mathbf{x}^{\mathrm{c},i}\}$, $\{\mathbf{x}^{\mathrm{c},i}\}$, $\{\mathbf{x}^{\mathrm{bD},i}\}$, $\{\mathbf{u}^{\mathrm{bD},i}\}$, $\{\mathbf{x}^{\mathrm{bN},i}\}$, $\{\mathbf{u}^{\mathrm{bN},i}\}$, $\mathbb{I}_\Omega$
Initialize loss weights $w_{\mathrm{P}}, w_{\mathrm{bD}}, w_{\mathrm{bN}}, w_{\mathrm{strong}}, w_{\mathrm{inter}}, \alpha$
$i_{\mathrm{epoch}} \leftarrow 0$;
Initialize: $\{\mathbf{x}^{\mathrm{c},i}\}$, $\{\mathbf{x}^{\mathrm{inter},i}\}$, $\{\mathbf{x}^{\mathrm{bD},i}\}$, $\{\mathbf{u}^{\mathrm{bD},i}\}$, $\mathbb{I}_\Omega$
Initialize loss weights: $w_{\mathrm{P}}, w_{\mathrm{strong}}, w_{\mathrm{bD}}, w_{\mathrm{inter}}, \alpha$
$i_{\mathrm{epoch}} \leftarrow 0$;
while $i_{epoch} < N_{epochs}$ **do**
  $\tilde{\mathcal{L}}^D_{\boldsymbol{\theta}} \leftarrow w_{\mathrm{bD}} \mathcal{L}^D_{\boldsymbol{\theta}}(\{\mathbf{x}^{\mathrm{bD},i}\})$;
  $\tilde{\mathcal{L}}^N_{\boldsymbol{\theta}} \leftarrow w_{\mathrm{bN}} \mathcal{L}^N_{\boldsymbol{\theta}}(\{\mathbf{x}^{\mathrm{bN},i}\})$;
  $\tilde{\mathcal{L}}^{\mathrm{strong}}_{\boldsymbol{\theta}} \leftarrow w_{\mathrm{strong}} \mathcal{L}^{\mathrm{strong}}_{\boldsymbol{\theta}}(\{\mathbf{x}^{\mathrm{c}\Omega,i}\})$;
  $\tilde{\mathcal{L}}^{\mathrm{weak}}_{\boldsymbol{\eta},\boldsymbol{\theta},\boldsymbol{\zeta}} \leftarrow w_{\mathrm{P}} \mathcal{L}^{\mathrm{weak}}_{\boldsymbol{\eta},\boldsymbol{\theta},\boldsymbol{\zeta}}(\{\mathbf{x}^{\mathrm{c},i}\})$;
  $\boldsymbol{\theta}, \boldsymbol{\eta}, \boldsymbol{\zeta} \leftarrow \mathrm{minimize}(\tilde{\mathcal{L}}^{\mathrm{weak}}_{\boldsymbol{\eta},\boldsymbol{\theta},\boldsymbol{\zeta}}, + \tilde{\mathcal{L}}^D_{\boldsymbol{\theta}} + \tilde{\mathcal{L}}^N_{\boldsymbol{\theta}} + \tilde{\mathcal{L}}^{\mathrm{strong}}_{\boldsymbol{\theta}} + \tilde{\mathcal{L}}^{\mathrm{inter}}_{\boldsymbol{\theta}})$;
  $\boldsymbol{\zeta} \leftarrow \mathrm{maximize}(\tilde{\mathcal{L}}^{\mathrm{weak}}_{\boldsymbol{\eta},\boldsymbol{\theta},\boldsymbol{\zeta}})$;
  $i_{\mathrm{epoch}} \leftarrow i_{\mathrm{epoch}} + 1$;
end

and separates the ellipse into the two domains Ω_{left} with $a(\mathbf{x}) = 5$ for $\mathbf{x} \in \Omega_{\mathrm{left}}$ and Ω_{right} with $a(\mathbf{x}) = 1$ for $\mathbf{x} \in \Omega_{\mathrm{right}}$.

We used $N_{\mathrm{c}} = 10^4$ points sampled on a grid. There were 1542 in the left and right domains each and 6916 outside. We again used our methods and the *strong-normal* method. In all cases, we sampled the interface at 523 points. For the *semi-weak* case with one static test function, we set $\varphi(\mathbf{x}) = \frac{(x_1-0.5)^2}{0.4^2} + \frac{(x_2-0.5)^2}{0.25^2} - 1$ and in the adversarial case set $\varphi_{\boldsymbol{\zeta}}(\mathbf{x}) := \left(\frac{(x_1-0.5)^2}{0.4^2} + \frac{(x_2-0.5)^2}{0.25^2} - 1\right) \mathfrak{N}_\varphi(\mathbf{x}, \boldsymbol{\zeta})$.
Again, we tested our predictions against high resolution FEM calculation (prediction on the FEM vertices). The results are depicted in Figure 3 and Table 1. While all methods yielded acceptable results, our *semi-weak* method had the smallest RMSE, followed by the *strong-normal* method.

C.2. Elliptic PDE 'mixed': Interface and Mixed Dirichlet/Neumann Boundaries

For the next example, we use mixed Dirichlet and Neumann boundaries. We consider a unit square domain Ω. The boundaries at $(x = 0)$ and $(x = 1)$ are of Dirichlet type and the boundaries at $(y = 0)$ and $(y = 1)$ are of Neumann type. The Neumann boundaries are set to be 0 everywhere. The left Dirichlet boundary is set to $u^{\mathrm{bD,left}} = 0$ and the right Dirichlet boundary is set to $u^{\mathrm{bD,right}} = 0.1$. In this setup, the test function φ for the semi-weak method needs to be zero on the Dirichlet boundaries and non-zero on the Neumann boundaries. For the case of one test function, we set $\varphi(\mathbf{x}) = x_1(x_1 - 1)$ and in the case of adversarial training to $\varphi_{\boldsymbol{\zeta}}(\mathbf{x}) = (\mathfrak{N}_\varphi(\mathbf{x}, \boldsymbol{\zeta}) x_1(x_1 - 1))^2$. The parameters of the elliptic PDE (Equation 1) in this example are set to $g(\mathbf{x}) = 0.1x_1$, $f(\mathbf{x}) = 10$, $B(\mathbf{x}) = (1, 1)^T$ and $a(\mathbf{x}) = 5$ for $\mathbf{x} \in \Omega_{\mathrm{left}}$ and $a(\mathbf{x}) = 1$ for $\mathbf{x} \in \Omega_{\mathrm{right}}$.

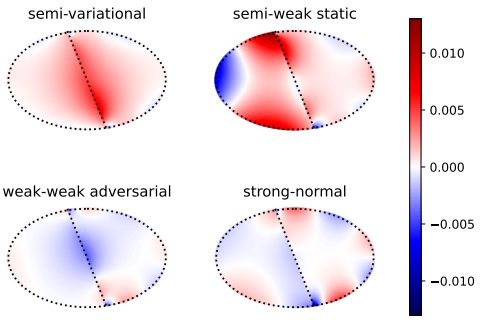


Figure 3: Results for C.1. The plots depict the error comparing each of our methods to high-resolution FEM calculations. The black dotted area shows the position of the interface. All methods find reasonable approximations of the solution (assuming the error of the FEM calculation is reasonably small). Our *semi-weak* method had the smallest RMSE.

The semi-variational approach is omitted, since $B(\mathbf{x}) \neq \mathbf{0}$. The results are shown in Figure 4 and Table 1. The *strong-normal* method yields the best results, closely followed by our *semi-weak* method. The comparison was again performed on a set of vertices of high resolution FEM calculations.

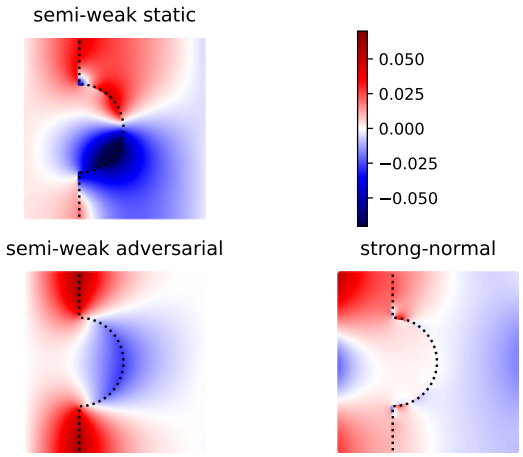


Figure 4: Results for C.2. The plots depict the error comparing each of our methods to high-resolution FEM calculations. All methods find reasonable approximations of the solution. The *strong-normal* method yielded the best results

Appendix D. Weights

Weights and number of epochs for the training of the interface problems are shown in Table 2

Table 2: Epochs and weights of the different loss terms for the examples of linear elliptic PDEs. (#E: number of epochs, PNT: PINNtegrate, intrf: interface, strg: strong)

	#E	PNT	intrf	strg	bD	bN	flux
flower:							
semi-variational	35k	0.2	30	0.05	0.5	-	-
semi-weak-static	90k	1e-3	1.0	1e-3	1.0	-	-
semi-weak-advers	40k	1.0	50	1.0	10	-	-
strong-normal	40k	-	100	1.0	1.0	-	2.0
semi-weak-WAN	30k	-	30	1.0	1.0	-	-
ellipse:							
semi-variational	40k	50	500	10	550	-	-
semi-weak-static	60k	50	500	10	550	-	-
semi-weak-advers	20k	50	500	10	550	-	-
strong-normal	20k	-	500	10	550	-	1.0
puzzel:							
semi-weak-static	50k	1.0	500.0	1.0	10.0	10.0	-
semi-weak-advers	50k	1.0	500.0	1.0	10.0	10.0	-
strong-normal	50k	-	100.0	1.0	1.0	1.0	2.0