Physics-Informed Neural Networks for Nonsmooth PDE-Constrained Optimization Problems

Yongcun Song

August 4, 2023

Chair for Dynamics, Control, Machine Learning and Numerics-Alexander von Humboldt-Professorship
Department of Mathematics, Friedrich-Alexander-Universität Erlangen-Nürnberg

Background and Motivation

PDEs

- Partial differential equations (PDEs) model various physical phenomena such as diffusion, heat flow, fluid flow, elastic deformation, wave propagation, etc.

PDEs

- Partial differential equations (PDEs) model various physical phenomena such as diffusion, heat flow, fluid flow, elastic deformation, wave propagation, etc.
- For some applications, we need not only to model certain physical process, but also control or optimize the considered process to optimally meet certain objectives.

PDEs

- Partial differential equations (PDEs) model various physical phenomena such as diffusion, heat flow, fluid flow, elastic deformation, wave propagation, etc.
- For some applications, we need not only to model certain physical process, but also control or optimize the considered process to optimally meet certain objectives.
- A given objective functional has to be minimized subject to a PDE or a system of coupled PDEs, usually with other additional constraints.

PDEs

- Partial differential equations (PDEs) model various physical phenomena such as diffusion, heat flow, fluid flow, elastic deformation, wave propagation, etc.
- For some applications, we need not only to model certain physical process, but also control or optimize the considered process to optimally meet certain objectives.
- A given objective functional has to be minimized subject to a PDE or a system of coupled PDEs, usually with other additional constraints.
- PDE-constrained optimization problems arise.

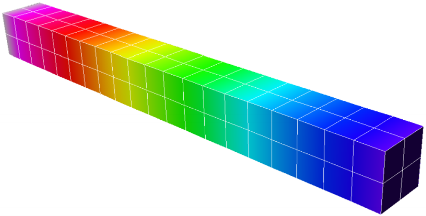


Figure 1: Control the heat distribution of a metal bar

PDE-Constrained Optimization

- A PDE-constrained optimization problem can be abstractly represented as

$$\min_{u \in U, y \in Y} \mathcal{J}(u, y), \quad \text{s.t.} \quad e(u, y) = 0, u \in U_{ad}, y \in Y_{ad},$$

PDE-Constrained Optimization

- A PDE-constrained optimization problem can be abstractly represented as

$$\min_{u \in U, y \in Y} \mathcal{J}(u, y), \quad \text{s.t.} \quad e(u, y) = 0, u \in U_{ad}, y \in Y_{ad},$$

  - U and Y are Banach spaces, $U_{ad} \subset U$ and $Y_{ad} \subset Y$ are closed convex sets;

PDE-Constrained Optimization

- A PDE-constrained optimization problem can be abstractly represented as

$$\min_{u \in U, y \in Y} \mathcal{J}(u, y), \quad \text{s.t.} \quad e(u, y) = 0, u \in U_{ad}, y \in Y_{ad},$$

  - U and Y are Banach spaces, $U_{ad} \subset U$ and $Y_{ad} \subset Y$ are closed convex sets;
  - $\mathcal{J} : U \times Y \to \mathbb{R}$ is the objective functional;

PDE-Constrained Optimization

- A PDE-constrained optimization problem can be abstractly represented as

$$\min_{u \in U, y \in Y} \mathcal{J}(u, y), \quad \text{s.t.} \quad e(u, y) = 0, u \in U_{ad}, y \in Y_{ad},$$

  - U and Y are Banach spaces, $U_{ad} \subset U$ and $Y_{ad} \subset Y$ are closed convex sets;
  - $\mathcal{J} : U \times Y \to \mathbb{R}$ is the objective functional;
  - $e(u, y) = 0$ represents a PDE or a system of coupled PDEs;

PDE-Constrained Optimization

- A PDE-constrained optimization problem can be abstractly represented as

$$\min_{u \in U, y \in Y} \mathcal{J}(u, y), \quad \text{s.t.} \quad e(u, y) = 0, u \in U_{ad}, y \in Y_{ad},$$

  - U and Y are Banach spaces, $U_{ad} \subset U$ and $Y_{ad} \subset Y$ are closed convex sets;
  - $\mathcal{J} : U \times Y \to \mathbb{R}$ is the objective functional;
  - $e(u, y) = 0$ represents a PDE or a system of coupled PDEs;
  - the state variable $y \in Y$ describes the state (e.g., temperature distribution) of the considered system modeled by $e(u, y) = 0$;

PDE-Constrained Optimization

- A PDE-constrained optimization problem can be abstractly represented as

$$\min_{u \in U, y \in Y} \mathcal{J}(u, y), \quad \text{s.t.} \quad e(u, y) = 0, u \in U_{ad}, y \in Y_{ad},$$

  - U and Y are Banach spaces, $U_{ad} \subset U$ and $Y_{ad} \subset Y$ are closed convex sets;
  - $\mathcal{J} : U \times Y \to \mathbb{R}$ is the objective functional;
  - $e(u, y) = 0$ represents a PDE or a system of coupled PDEs;
  - the state variable $y \in Y$ describes the state (e.g., temperature distribution) of the considered system modeled by $e(u, y) = 0$;
  - the control variable $u \in U$ is a parameter (e.g., source term) that shall be adapted in an optimal way;

PDE-Constrained Optimization

- A PDE-constrained optimization problem can be abstractly represented as

$$\min_{u \in U, y \in Y} \mathcal{J}(u, y), \quad \text{s.t.} \quad e(u, y) = 0, u \in U_{ad}, y \in Y_{ad},$$

  - U and Y are Banach spaces, $U_{ad} \subset U$ and $Y_{ad} \subset Y$ are closed convex sets;
  - $\mathcal{J} : U \times Y \to \mathbb{R}$ is the objective functional;
  - $e(u, y) = 0$ represents a PDE or a system of coupled PDEs;
  - the state variable $y \in Y$ describes the state (e.g., temperature distribution) of the considered system modeled by $e(u, y) = 0$;
  - the control variable $u \in U$ is a parameter (e.g., source term) that shall be adapted in an optimal way;
  - the control constraint $u \in U_{ad}$ and the state constraint $y \in Y_{ad}$ describe some physical restrictions and realistic requirements.

Nonsmooth PDE-Constrained Optimization-I

- Nonsmooth objective functional: $\mathcal{J}(y,u) = J(y,u) + R(u)$

Nonsmooth PDE-Constrained Optimization-I

- Nonsmooth objective functional: $\mathcal{J}(y, u) = J(y, u) + R(u)$
  - The functional $J : Y \times U \to \mathbb{R}$ consists of a data-fidelity term and a possible smooth regularization.

Nonsmooth PDE-Constrained Optimization-I

- Nonsmooth objective functional: $\mathcal{J}(y,u) = J(y,u) + R(u)$
  - The functional $J : Y \times U \to \mathbb{R}$ consists of a data-fidelity term and a possible smooth regularization.
  - The nonsmooth functional $R : U \to \mathbb{R}$ is employed to capture some prior information on u, such as sparsity, discontinuity, and lower and/or upper bounds.

Nonsmooth PDE-Constrained Optimization-I

- Nonsmooth objective functional: $\mathcal{J}(y,u) = J(y,u) + R(u)$
  - The functional $J : Y \times U \to \mathbb{R}$ consists of a data-fidelity term and a possible smooth regularization.
  - The nonsmooth functional $R : U \to \mathbb{R}$ is employed to capture some prior information on u, such as sparsity, discontinuity, and lower and/or upper bounds.
- A parabolic sparse optimal control problem:

$$\min_{y,u \in L^2(Q)} \frac{1}{2}\|y - y_d\|_{L^2(Q)}^2 + \frac{\alpha}{2}\|u\|_{L^2(Q)}^2 + \rho\|u\|_{L^1(Q)} + I_{U_{ad}}(u),$$

subject to

$$\frac{\partial y}{\partial t} - \nu \Delta y + c_0 y = u + f \text{ in } \Omega \times (0,T), \; y = 0 \text{ on } \partial\Omega \times (0,T), \; y(0) = \varphi.$$

Nonsmooth PDE-Constrained Optimization-I

- Nonsmooth objective functional: $\mathcal{J}(y,u) = J(y,u) + R(u)$
  - The functional $J : Y \times U \to \mathbb{R}$ consists of a data-fidelity term and a possible smooth regularization.
  - The nonsmooth functional $R : U \to \mathbb{R}$ is employed to capture some prior information on u, such as sparsity, discontinuity, and lower and/or upper bounds.
- A parabolic sparse optimal control problem:

$$\min_{y,u\in L^2(Q)} \frac{1}{2}\|y-y_d\|^2_{L^2(Q)} + \frac{\alpha}{2}\|u\|^2_{L^2(Q)} + \rho\|u\|_{L^1(Q)} + I_{U_{ad}}(u),$$

subject to

$$\frac{\partial y}{\partial t} - \nu\Delta y + c_0 y = u + f \text{ in } \Omega\times(0,T),\ \ y = 0 \text{ on } \partial\Omega\times(0,T),\ \ y(0) = \varphi.$$

  - Above, Ω is a bounded domain in $\mathbb{R}^d$ with $d \geq 1$ and $\partial\Omega$ is its boundary; $Q = \Omega\times(0,T)$ with $0 < T < +\infty$; $y_d \in L^2(Q)$ and $\phi \in L^2(\Omega), f \in L^2(Q)$ are given.
  - The regularization parameters $\alpha > 0$, $\rho > 0$ and the coefficients $\nu > 0$, $c_0 \geq 0$ are constant.
  - $I_{U_{ad}}(\cdot)$ the indicator function of $U_{ad} := \{u \in L^\infty(\Omega) | a \leq u(x,t) \leq b, \text{ a.e. in } Q\} \subset L^2(Q)$, where $a, b \in L^2(\Omega)$ with $a < 0 < b$ almost everywhere.

- Nonsmooth PDE:

Nonsmooth PDE-Constrained Optimization-II

- Nonsmooth PDE: interface problems

Nonsmooth PDE-Constrained Optimization-II

- Nonsmooth PDE: interface problems —— piecewise-defined PDEs in different regions coupled together with interface conditions, e.g., jumps in solution and flux across the interface,

Nonsmooth PDE-Constrained Optimization-II

- Nonsmooth PDE: interface problems —— piecewise-defined PDEs in different regions coupled together with interface conditions, e.g., jumps in solution and flux across the interface, hence nonsmooth or even discontinuous solutions.

Nonsmooth PDE-Constrained Optimization-II

- Nonsmooth PDE: interface problems —— piecewise-defined PDEs in different regions coupled together with interface conditions, e.g., jumps in solution and flux across the interface, hence nonsmooth or even discontinuous solutions.

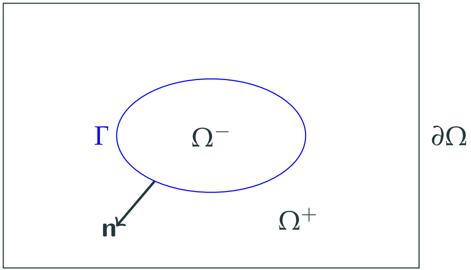


Figure 2: The geometry of an interface problem: an illustration

Nonsmooth PDE-Constrained Optimization-II

- An elliptic interface optimal control problem:

$$\min_{y\in L^2(\Omega),u\in L^2(\Omega)} J(y,u) := \frac{1}{2}\|y-y_d\|^2_{L^2(\Omega)} + \frac{\alpha}{2}\|u\|^2_{L^2(\Omega)},$$

subject to

$$\begin{cases} -\nabla\cdot(\beta\nabla y) = u + f & \text{in } \Omega\backslash\Gamma, \\ [y]_\Gamma = g_0,\ [\beta\partial_{\boldsymbol{n}} y]_\Gamma = g_1 & \text{on } \Gamma, \\ y = h_0 & \text{on } \partial\Omega, \end{cases}$$

Nonsmooth PDE-Constrained Optimization-II

- An elliptic interface optimal control problem:

$$\min_{y\in L^2(\Omega),u\in L^2(\Omega)} J(y,u) := \frac{1}{2}\|y-y_d\|^2_{L^2(\Omega)} + \frac{\alpha}{2}\|u\|^2_{L^2(\Omega)},$$

subject to

$$\begin{cases} -\nabla\cdot(\beta\nabla y) = u+f & \text{in } \Omega\backslash\Gamma, \\ [y]_\Gamma = g_0,\ [\beta\partial_{\boldsymbol{n}} y]_\Gamma = g_1 & \text{on } \Gamma, \\ y = h_0 & \text{on } \partial\Omega, \end{cases}$$

- The functions $f\in L^2(\Omega), g_0\in H^{\frac{1}{2}}(\Gamma), g_1\in L^2(\Gamma)$, and $h_0\in H^{\frac{1}{2}}(\partial\Omega)$ are given, and β is a nonzero piecewise-constant in $\Omega\backslash\Gamma$ such that $\beta=\beta^-$ in Ω^- and $\beta=\beta^+$ in Ω^+.
- The jump discontinuity across Γ: $[y]_\Gamma(x) := \lim_{\tilde{x}\to x \text{ in } \Omega^+} y(\tilde{x}) - \lim_{\tilde{x}\to x \text{ in } \Omega^-} y(\tilde{x}), \forall x\in\Gamma$.
- The operator $\partial_{\boldsymbol{n}}$ stands for the normal derivative on Γ, i.e. $\partial_{\boldsymbol{n}} y(x) = \boldsymbol{n}\cdot\nabla y(x)$ with $\boldsymbol{n}\in\mathbb{R}^d$ the outward unit normal vector of Γ. In particular, we have

$$[\beta\partial_{\boldsymbol{n}} y]_\Gamma(x) := \boldsymbol{n}\cdot(\beta^+ \lim_{\tilde{x}\to x \text{ in } \Omega^+} \nabla y(\tilde{x}) - \beta^- \lim_{\tilde{x}\to x \text{ in } \Omega^+} \nabla y(\tilde{x})), \quad \forall x\in\Gamma.$$

Traditional Numerical Methods

Traditional Numerical Methods

- Optimization algorithms + Numerical discretization

Traditional Numerical Methods

- Optimization algorithms + Numerical discretization
  - Optimization algorithms: semismooth Newton methods, primal-dual active set methods, alternating direction method of multipliers, primal-dual methods, etc.

Traditional Numerical Methods

- Optimization algorithms + Numerical discretization
  - Optimization algorithms: semismooth Newton methods, primal-dual active set methods, alternating direction method of multipliers, primal-dual methods, etc.
  - Numerical discretization: finite difference methods, finite element methods, etc.

Traditional Numerical Methods

- Optimization algorithms + Numerical discretization
  - Optimization algorithms: semismooth Newton methods, primal-dual active set methods, alternating direction method of multipliers, primal-dual methods, etc.
  - Numerical discretization: finite difference methods, finite element methods, etc.
- The implementation of these methods are usually based on the so-called adjoint methodology.

Traditional Numerical Methods

- Optimization algorithms + Numerical discretization
  - Optimization algorithms: semismooth Newton methods, primal-dual active set methods, alternating direction method of multipliers, primal-dual methods, etc.
  - Numerical discretization: finite difference methods, finite element methods, etc.
- The implementation of these methods are usually based on the so-called adjoint methodology.
- Two PDEs (the state equation and its adjoint system) or a saddle point problem (first-order optimality system) are usually required to be solved repeatedly.

Traditional Numerical Methods

- Optimization algorithms + Numerical discretization
  - Optimization algorithms: semismooth Newton methods, primal-dual active set methods, alternating direction method of multipliers, primal-dual methods, etc.
  - Numerical discretization: finite difference methods, finite element methods, etc.
- The implementation of these methods are usually based on the so-called adjoint methodology.
- Two PDEs (the state equation and its adjoint system) or a saddle point problem (first-order optimality system) are usually required to be solved repeatedly.
- After some proper numerical discretization, the resulting systems are large-scale and ill-conditioned, and the computation cost for solving the PDEs or the saddle point problem repeatedly could be extremely high in practice, especially for high-dimensional PDEs.

Traditional Numerical Methods

- Optimization algorithms + Numerical discretization
  - Optimization algorithms: semismooth Newton methods, primal-dual active set methods, alternating direction method of multipliers, primal-dual methods, etc.
  - Numerical discretization: finite difference methods, finite element methods, etc.
- The implementation of these methods are usually based on the so-called adjoint methodology.
- Two PDEs (the state equation and its adjoint system) or a saddle point problem (first-order optimality system) are usually required to be solved repeatedly.
- After some proper numerical discretization, the resulting systems are large-scale and ill-conditioned, and the computation cost for solving the PDEs or the saddle point problem repeatedly could be extremely high in practice, especially for high-dimensional PDEs.
- Moreover, these methods are strongly problem-dependent, e.g., different types of PDEs entail different tailored numerical discretization schemes.

Deep Learning for PDEs

Deep Learning for PDEs

- Thanks to the universal approximation property and great expressibility of deep neural networks (DNNs), deep learning has recently emerged as a new powerful tool for solving PDEs.

Deep Learning for PDEs

- Thanks to the universal approximation property and great expressibility of deep neural networks (DNNs), deep learning has recently emerged as a new powerful tool for solving PDEs.
- Some representative deep learning methods include:

Deep Learning for PDEs

- Thanks to the universal approximation property and great expressibility of deep neural networks (DNNs), deep learning has recently emerged as a new powerful tool for solving PDEs.
- Some representative deep learning methods include:
  - Deep Ritz method [E and Yu, 2018]
  - Deep Galerkin method [Sirignano and Spiliopoulos, 2018]
  - Physic-informed neural networks [Raissi, Perdikaris, and Karniadakis, 2019]
  - Random feature method [Chen, Chi, E, and Yang, 2023]

Deep Learning for PDEs

- Thanks to the universal approximation property and great expressibility of deep neural networks (DNNs), deep learning has recently emerged as a new powerful tool for solving PDEs.
- Some representative deep learning methods include:
  - Deep Ritz method [E and Yu, 2018]
  - Deep Galerkin method [Sirignano and Spiliopoulos, 2018]
  - Physic-informed neural networks [Raissi, Perdikaris, and Karniadakis, 2019]
  - Random feature method [Chen, Chi, E, and Yang, 2023]
  - Operator learning methods: DeepONets [Lu, Jin, Pang, Zhang, and Karniadakis, 2021], Fourier Neural Operator, Graph Neural Operator, [Li, Kovachki, Azizzadenesheli, Liu, Bhattacharya, Stuart, and Anandkumar,2020] and Laplace Neural Operator [Cao, Goswami, and Karniadakis, 2023], etc.
  - ... ...

PINNs

- Physics-informed neural networks (PINNs) can be viewed as a scientific machine learning [1,2] technique used to solve differential equations (e.g., ODEs and PDEs).

[1] https://www.osti.gov/servlets/purl/1478744
[2] https://docs.sciml.ai/Overview/stable/

PINNs

- Physics-informed neural networks (PINNs) can be viewed as a scientific machine learning [1,2] technique used to solve differential equations (e.g., ODEs and PDEs).
- PINNs approximate the PDE solution by a neural network and then train the neural network by minimizing a loss function consisting of the residuals of the PDE and the boundary/initial conditions at selected points in the domain (called residual points).

[1] https://www.osti.gov/servlets/purl/1478744
[2] https://docs.sciml.ai/Overview/stable/

PINNs

- Physics-informed neural networks (PINNs) can be viewed as a scientific machine learning [1,2] technique used to solve differential equations (e.g., ODEs and PDEs).
- PINNs approximate the PDE solution by a neural network and then train the neural network by minimizing a loss function consisting of the residuals of the PDE and the boundary/initial conditions at selected points in the domain (called residual points).
- Given an input point in the domain, PINNs produce an approximate solution in that point of a PDE after training.

[1] https://www.osti.gov/servlets/purl/1478744
[2] https://docs.sciml.ai/Overview/stable/

PINNs for Solving PDEs

PINNs for Solving PDEs

The PINNs algorithm [Raissi, Perdikaris and Karniadakis 2019] for solving $\mathcal{E}(y, u) = 0$ in Ω, $\mathcal{B}(y, u) = 0$ on $\partial\Omega$ with u a know function.

PINNs for Solving PDEs

The PINNs algorithm [Raissi, Perdikaris and Karniadakis 2019] for solving $\mathcal{E}(y, u) = 0$ in Ω, $\mathcal{B}(y, u) = 0$ on $\partial\Omega$ with u a know function.

1. Construct a neural network $\hat{y}(x; \boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$.

PINNs for Solving PDEs

The PINNs algorithm [Raissi, Perdikaris and Karniadakis 2019] for solving $\mathcal{E}(y, u) = 0$ in Ω, $\mathcal{B}(y, u) = 0$ on $\partial\Omega$ with u a know function.

1. Construct a neural network $\hat{y}(x; \boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$.
2. Specify the residual points$\mathcal{T}_i \subset \Omega$ and $\mathcal{T}_b \subset \partial\Omega$.

PINNs for Solving PDEs

The PINNs algorithm [Raissi, Perdikaris and Karniadakis 2019] for solving $\mathcal{E}(y, u) = 0$ in Ω, $\mathcal{B}(y, u) = 0$ on $\partial\Omega$ with u a know function.

1. Construct a neural network $\hat{y}(x; \boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$.
2. Specify the residual points$\mathcal{T}_i \subset \Omega$ and $\mathcal{T}_b \subset \partial\Omega$.
3. Specify a loss function by summing the residuals of the PDE and the boundary condition:

$$\mathcal{L}_{PDE}(\boldsymbol{\theta}, \mathcal{T}) = w_i \mathcal{L}_i(\boldsymbol{\theta}, \mathcal{T}_i) + w_b \mathcal{L}_b(\boldsymbol{\theta}, \mathcal{T}_b),$$

where $\mathcal{L}_i(\boldsymbol{\theta}, \mathcal{T}_i) = \frac{1}{|\mathcal{T}_i|} \sum_{x \in \mathcal{T}_i} \|\mathcal{E}(\hat{y}(x; \boldsymbol{\theta}), u(x))\|^2$, $\mathcal{L}_b(\boldsymbol{\theta}, \mathcal{T}_b) = \frac{1}{|\mathcal{T}_b|} \sum_{x \in \mathcal{T}_b} \|\mathcal{B}(\hat{y}(x; \boldsymbol{\theta}), u(x))\|^2$, and w_i and w_b are the weights.

PINNs for Solving PDEs

The PINNs algorithm [Raissi, Perdikaris and Karniadakis 2019] for solving $\mathcal{E}(y, u) = 0$ in Ω, $\mathcal{B}(y, u) = 0$ on $\partial\Omega$ with u a know function.

1. Construct a neural network $\hat{y}(x; \boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$.
2. Specify the residual points$\mathcal{T}_i \subset \Omega$ and $\mathcal{T}_b \subset \partial\Omega$.
3. Specify a loss function by summing the residuals of the PDE and the boundary condition:

$$\mathcal{L}_{PDE}(\boldsymbol{\theta}, \mathcal{T}) = w_i \mathcal{L}_i(\boldsymbol{\theta}, \mathcal{T}_i) + w_b \mathcal{L}_b(\boldsymbol{\theta}, \mathcal{T}_b),$$

   where $\mathcal{L}_i(\boldsymbol{\theta}, \mathcal{T}_i) = \frac{1}{|\mathcal{T}_i|} \sum_{x \in \mathcal{T}_i} \|\mathcal{E}(\hat{y}(x; \boldsymbol{\theta}), u(x))\|^2$, $\mathcal{L}_b(\boldsymbol{\theta}, \mathcal{T}_b) = \frac{1}{|\mathcal{T}_b|} \sum_{x \in \mathcal{T}_b} \|\mathcal{B}(\hat{y}(x; \boldsymbol{\theta}), u(x))\|^2$, and w_i and w_b are the weights.
4. Train the neural network $\hat{y}(x; \boldsymbol{\theta})$ to find the optimal parameters $\boldsymbol{\theta}^*$ by minimizing the loss function $\mathcal{L}_{PDE}(\boldsymbol{\theta}, \mathcal{T})$. At the end of the training procedure, the trained neural network $\hat{y}(x, \boldsymbol{\theta}^*)$ approximately solves the PDE.

PINNs

- Pros of PINNs: little or even no labeled data is required, easy to implement, mesh-free, and flexible to different problem settings.

PINNs

- Pros of PINNs: little or even no labeled data is required, easy to implement, mesh-free, and flexible to different problem settings.
- More applications and discussions about PINNs can be referred to the review papers:
  - Cuomo, S., Di Cola, V. S., Giampaolo, F., Rozza, G., Raissi, M., and Piccialli, F. *Scientific Machine Learning Through Physics–Informed Neural Networks: Where we are and What's Next.* J Sci Comput 92, 88 (2022).
  - Faroughi, S. A., Pawar, N., Fernandes, C., Das, S., Kalantari, N. K., and Mahjour, S. K. *Physics-Guided, Physics-Informed, and Physics-Encoded Neural Networks in Scientific Computing.* arXiv preprint, arXiv:2211.07377, (2022).
  - Hao, Z., Liu, S., Zhang, Y., Ying, C., Feng, Y., Su, H., and Zhu, J. *Physics-Informed Machine Learning: A Survey on Problems, Methods and Applications*. arXiv preprint arXiv:2211.08064, (2022).
  - Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., and Yang, L. *Physics-informed machine learning.* Nature Reviews Physics, 3(6) (2021) 422-440.
  - Lu, L., Meng, X., Mao, Z., and Karniadakis, G.E. *DeepXDE: A deep learning library for solving differential equations.* SIAM Rev. 63 (1) (2021) 208-228.

PINNs for Smooth PDE-Constrained Optimization

- PINNs for smooth PDE-Constrained Optimization, see e.g., [Mowlavi and Nabi, 2023, Barry-Straume, Sarshar, Popov, and Sandu, 2022]

PINNs for Smooth PDE-Constrained Optimization

- PINNs for smooth PDE-Constrained Optimization, see e.g., [Mowlavi and Nabi, 2023, Barry-Straume, Sarshar, Popov, and Sandu, 2022]
- Consider the smooth PDE-constrained optimization problems modeled by

$$\min \mathcal{J}(u, y), \quad \text{s.t.} \quad e(u, y) = 0.$$

- Let $\hat{y}(x; \boldsymbol{\theta}_y)$ parameterized by $\boldsymbol{\theta}_y$ and $\hat{u}(x; \boldsymbol{\theta}_u)$ parameterized by $\boldsymbol{\theta}_u$ be two neural networks to approximate y and u, respectively.
- Specify the residual points $\mathcal{T} \subset \Omega \cup \partial\Omega$ and a loss function by summing the PDE's residual and the objective functional:

$$\mathcal{L}_{total}(\boldsymbol{\theta}_y, \boldsymbol{\theta}_u, \mathcal{T}) = w_o \mathcal{J}(\boldsymbol{\theta}_y, \boldsymbol{\theta}_u, \mathcal{T}) + w_p \mathcal{L}_{PDE}(\boldsymbol{\theta}_y, \boldsymbol{\theta}_u, \mathcal{T}),$$

  where w_o and w_p are the weights.
- Train the neural networks $\hat{y}(x; \boldsymbol{\theta}_y)$ and $\hat{u}(x; \boldsymbol{\theta}_u)$ by minimizing the loss function $\mathcal{L}_{total}(\boldsymbol{\theta}_y, \boldsymbol{\theta}_u, \mathcal{T})$. At the end of the training procedure, we obtain the solution $\hat{y}(x, \boldsymbol{\theta}_y^*)$ and $\hat{u}(x, \boldsymbol{\theta}_u^*)$

Challenges of PINNs for Solving Nonsmooth PDE-Constrained Optimization

Challenges of PINNs for Solving Nonsmooth PDE-Constrained Optimization

- For the parabolic sparse optimal control problem:

Challenges of PINNs for Solving Nonsmooth PDE-Constrained Optimization

- For the parabolic sparse optimal control problem:
  - the nonsmooth term $\rho\|\hat{u}(x;\boldsymbol{\theta}_u)\|_{L^1(Q)} + I_{U_{ad}}(\hat{u}(x;\boldsymbol{\theta}_u))$ appears in the loss function.

Challenges of PINNs for Solving Nonsmooth PDE-Constrained Optimization

- For the parabolic sparse optimal control problem:
  - the nonsmooth term $\rho\|\hat{u}(x;\boldsymbol{\theta}_u)\|_{L^1(Q)} + I_{U_{ad}}(\hat{u}(x;\boldsymbol{\theta}_u))$ appears in the loss function.
  - commonly used neural network training technologies (e.g., back-propagation and stochastic gradient methods) cannot be applied directly.

Challenges of PINNs for Solving Nonsmooth PDE-Constrained Optimization

- For the parabolic sparse optimal control problem:
  - the nonsmooth term $\rho\|\hat{u}(x;\boldsymbol{\theta}_u)\|_{L^1(Q)} + I_{U_{ad}}(\hat{u}(x;\boldsymbol{\theta}_u))$ appears in the loss function.
  - commonly used neural network training technologies (e.g., back-propagation and stochastic gradient methods) cannot be applied directly.
- For the elliptic interface optimal control problem:

Challenges of PINNs for Solving Nonsmooth PDE-Constrained Optimization

- For the parabolic sparse optimal control problem:
  - the nonsmooth term $\rho\|\hat{u}(x;\boldsymbol{\theta}_u)\|_{L^1(Q)} + I_{U_{ad}}(\hat{u}(x;\boldsymbol{\theta}_u))$ appears in the loss function.
  - commonly used neural network training technologies (e.g., back-propagation and stochastic gradient methods) cannot be applied directly.
- For the elliptic interface optimal control problem:
  - the discontinuity or nonsmoothness of y cannot be well captured by the neural network $\hat{y}(x;\boldsymbol{\theta}_y)$ because the activation functions used in a DNN are in general smooth (e.g., the sigmoid function) or at least continuous (e.g., the rectified linear unit (ReLU) function).

The ADMM-PINNs for Parabolic Sparse Optimal Control Problems

Parabolic Sparse Optimal Control: Revisit

- We first recall the parabolic sparse optimal control problem under investigation:

$$\min_{y,u\in L^2(Q)} \frac{1}{2}\|y-y_d\|^2_{L^2(Q)} + \frac{\alpha}{2}\|u\|^2_{L^2(Q)} + \rho\|u\|_{L^1(Q)} + I_{U_{ad}}(u),$$

subject to

$$\frac{\partial y}{\partial t} - \nu\Delta y + c_0 y = u + f \text{ in } \Omega\times(0,T),\ \ y=0 \text{ on } \partial\Omega\times(0,T),\ \ y(0)=\varphi.$$

  - Above, Ω is a bounded domain in $\mathbb{R}^d$ with $d\geq 1$ and $\partial\Omega$ is its boundary; $Q=\Omega\times(0,T)$ with $0<T<+\infty$; $y_d\in L^2(Q)$ and $\phi\in L^2(\Omega), f\in L^2(Q)$ are given.
  - The regularization parameters $\alpha>0$, $\rho>0$ and the coefficients $\nu>0$, $c_0\leq 0$ are assumed to be constant.
  - $I_{U_{ad}}(\cdot)$ the indicator function of $U_{ad}:=\{u\in L^\infty(\Omega)|a\leq u(x,t)\leq b,\ \text{a.e. in } Q\}\subset L^2(Q)$, where $a,b\in L^2(\Omega)$ with $a<0<b$ almost everywhere.

Main Ideas

Main Ideas

- Operator splitting.

Main Ideas

- Operator splitting.
- The PDE constraint and the nonsmooth regularization are treated individually.

Main Ideas

- Operator splitting.
- The PDE constraint and the nonsmooth regularization are treated individually.
- The resulting subproblems admit closed-form solution or can be solved directly by some well-developed computational techniques (e.g., PINNs).

ADMM

ADMM

- The alternating direction method of multipliers (ADMM) is a representative operator splitting method introduced by Glowinski and Marroco in 1975 for nonlinear elliptic problems.

ADMM

- The alternating direction method of multipliers (ADMM) is a representative operator splitting method introduced by Glowinski and Marroco in 1975 for nonlinear elliptic problems.
- Let $y(u)$ be the solution of the parabolic state equation corresponding to u. Introduce $z \in L^2(Q)$ satisfying $u = z$, we then have

$$\min_{u,z \in L^2(Q)} \frac{1}{2}\|y(u) - y_d\|^2_{L^2(Q)} + \frac{\alpha}{2}\|u\|^2_{L^2(Q)} + \rho\|z\|_{L^1(Q)} + I_{U_{ad}}(z), \quad \text{s.t.} \qquad u = z.$$

ADMM

- The alternating direction method of multipliers (ADMM) is a representative operator splitting method introduced by Glowinski and Marroco in 1975 for nonlinear elliptic problems.
- Let $y(u)$ be the solution of the parabolic state equation corresponding to u. Introduce $z \in L^2(Q)$ satisfying $u = z$, we then have

$$\min_{u,z\in L^2(Q)} \frac{1}{2}\|y(u) - y_d\|^2_{L^2(Q)} + \frac{\alpha}{2}\|u\|^2_{L^2(Q)} + \rho\|z\|_{L^1(Q)} + I_{U_{ad}}(z), \quad \text{s.t.} \qquad u = z.$$

- The augmented Lagrangian functional reads as

$$L^{SC}_\beta(u, z; \lambda) = \frac{1}{2}\|y(u) - y_d\|^2_{L^2(Q)} + \frac{\alpha}{2}\|u\|^2_{L^2(Q)} - (\lambda, u - z)_{L^2(Q)} + \frac{\beta}{2}\|u - z\|^2_{L^2(Q)},$$

where $\lambda \in L^2(Q)$ is the Lagrange multiplier associated with $u = z$ and $\beta > 0$ is a penalty parameter.

ADMM-Cont'd

- The ADMM iterative scheme:

$$\begin{cases} u^{k+1} = \arg\min\limits_{u \in L^2(Q)} L_\beta^{SC}(u, z^k; \lambda^k), \\ z^{k+1} = \arg\min\limits_{z \in L^2(Q)} L_\beta^{SC}(u^{k+1}, z; \lambda^k), \\ \lambda^{k+1} = \lambda^k - \beta(u^{k+1} - z^{k+1}). \end{cases}$$

ADMM-Cont'd

- The ADMM iterative scheme:

$$\begin{cases} u^{k+1} = \arg\min\limits_{u\in L^2(Q)} L_\beta^{SC}(u, z^k; \lambda^k), \\ z^{k+1} = \arg\min\limits_{z\in L^2(Q)} L_\beta^{SC}(u^{k+1}, z; \lambda^k), \\ \lambda^{k+1} = \lambda^k - \beta(u^{k+1} - z^{k+1}). \end{cases}$$

- The z-subproblem is

$$z^{k+1} = \arg\min_{z\in L^2(Q)} I_{U_{ad}}(z) + \rho\|z\|_{L^1(Q)} - (\lambda^k, u^{k+1} - z)_{L^2(Q)} + \frac{\beta}{2}\|u^{k+1} - z\|_{L^2(Q)}^2.$$

ADMM-Cont'd

- The ADMM iterative scheme:

$$\begin{cases} u^{k+1} = \arg \min\limits_{u \in L^2(Q)} L_\beta^{SC}(u, z^k; \lambda^k), \\ z^{k+1} = \arg \min\limits_{z \in L^2(Q)} L_\beta^{SC}(u^{k+1}, z; \lambda^k), \\ \lambda^{k+1} = \lambda^k - \beta(u^{k+1} - z^{k+1}). \end{cases}$$

- The z-subproblem is

$$z^{k+1} = \arg \min_{z \in L^2(Q)} I_{U_{ad}}(z) + \rho \|z\|_{L^1(Q)} - (\lambda^k, u^{k+1} - z)_{L^2(Q)} + \frac{\beta}{2} \|u^{k+1} - z\|_{L^2(Q)}^2.$$

- The solution z^{k+1} can be computed by

$$z^{k+1} = \mathbb{P}_{U_{ad}} \left(\mathbb{S}_{\frac{\rho}{\beta}} \left(u^{k+1} - \frac{\lambda^k}{\beta} \right) \right),$$

where $\mathbb{S}_\zeta$ is the Shrinkage operator: $\mathbb{S}_\zeta(v)(x) = \operatorname{sgn}(v(x))(|v(x)| - \zeta)_+, \forall\, \zeta > 0.$

PINNs for the u-Subproblem

- The u-subproblem can be reformulated as

$$\min_{y,u} \mathcal{J}_{SC}^{k}(y,u) := \frac{1}{2}\|y(u)-y_d\|_{L^2(Q)}^2 + \frac{\alpha}{2}\|u\|_{L^2(Q)}^2 - (\lambda^k, u-z^k)_{L^2(Q)} + \frac{\beta}{2}\|u-z^k\|_{L^2(Q)}^2$$

PINNs for the u-Subproblem

- The u-subproblem can be reformulated as

$$\min_{y,u} \mathcal{J}_{SC}^k(y,u) := \frac{1}{2}\|y(u)-y_d\|^2_{L^2(Q)} + \frac{\alpha}{2}\|u\|^2_{L^2(Q)} - (\lambda^k, u - z^k)_{L^2(Q)} + \frac{\beta}{2}\|u-z^k\|^2_{L^2(Q)}$$

- The first-order optimality system reads as

$$\begin{cases} p + (\alpha+\beta)u - \lambda^k - \beta z^k = 0, \\ \dfrac{\partial y}{\partial t} - \nu\Delta y + c_0 y = u + f \text{ in } \Omega\times(0,T), \quad y = 0 \text{ on } \partial\Omega\times(0,T), \quad y(0) = \varphi, \\ -\dfrac{\partial p}{\partial t} - \nu\Delta p + c_0 p = y - y_d \text{ in } \Omega\times(0,T), \quad p = 0 \text{ on } \partial\Omega\times(0,T), \quad p(T) = 0, \end{cases}$$

where p is the corresponding adjoint variable.

PINNs for the u-Subproblem

- The u-subproblem can be reformulated as

$$\min_{y,u} \mathcal{J}_{SC}^k(y,u) := \frac{1}{2}\|y(u)-y_d\|_{L^2(Q)}^2 + \frac{\alpha}{2}\|u\|_{L^2(Q)}^2 - (\lambda^k, u-z^k)_{L^2(Q)} + \frac{\beta}{2}\|u-z^k\|_{L^2(Q)}^2$$

- The first-order optimality system reads as

$$\begin{cases} p + (\alpha+\beta)u - \lambda^k - \beta z^k = 0, \\ \dfrac{\partial y}{\partial t} - \nu\Delta y + c_0 y = u + f \text{ in } \Omega\times(0,T), \quad y = 0 \text{ on } \partial\Omega\times(0,T), \quad y(0) = \varphi, \\ -\dfrac{\partial p}{\partial t} - \nu\Delta p + c_0 p = y - y_d \text{ in } \Omega\times(0,T), \quad p = 0 \text{ on } \partial\Omega\times(0,T), \quad p(T) = 0, \end{cases}$$

where p is the corresponding adjoint variable.

- Eliminate the variable u, and then construct two neural networks $\hat{y}(x;\boldsymbol{\theta}_y)$ parameterized by $\boldsymbol{\theta}_y$ and $\hat{p}(x;\boldsymbol{\theta}_p)$ parameterized by $\boldsymbol{\theta}_p$ to approximate y and p, respectively.

PINNs for the u-Subproblem-Cont'd

- Choose residual points $\mathcal{T}_i \subset \Omega \times (0, T)$, $\mathcal{T}_{b_1} \subset \partial\Omega \times (0, T)$, and $\mathcal{T}_{b_2} \subset \Omega$.
- Specify a loss function by summing the residuals of the first-order optimality system

$$
\begin{aligned}
\mathcal{L}_{OS}(\boldsymbol{\theta}_y, \boldsymbol{\theta}_p) = w_y \Big(\frac{w_i}{|\mathcal{T}_i|} \sum_{\{x,t\} \in \mathcal{T}_i} \Big| & \frac{\partial \hat{y}(x,t;\boldsymbol{\theta}_y)}{\partial t} - \nu \frac{\partial^2 \hat{y}(x,t;\boldsymbol{\theta}_y)}{\partial x^2} + c_0 \hat{y}(x,t;\boldsymbol{\theta}_y) \\
& - \frac{1}{\alpha+\beta}\big(-\hat{p}(x,t;\boldsymbol{\theta}_p) + \lambda^k(x,t) + \beta z^k(x,t)\big) - f(x,t)\Big|^2 \\
& + \frac{w_{b_1}}{|\mathcal{T}_{b_1}|} \sum_{\{x,t\} \in \mathcal{T}_{b_1}} |\hat{y}(x,t;\boldsymbol{\theta}_y)|^2 + \frac{w_{b_2}}{|\mathcal{T}_{b_2}|} \sum_{x \in \mathcal{T}_{b_2}} |\hat{y}(x,0;\boldsymbol{\theta}_y) - \varphi(x)|^2 \Big) \\
+ w_p \Big(\frac{w_i}{|\mathcal{T}_i|} \sum_{\{x,t\} \in \mathcal{T}_i} \Big| - & \frac{\partial \hat{p}(x,t;\boldsymbol{\theta}_p)}{\partial t} - \nu \Delta \hat{p}(x,t;\boldsymbol{\theta}_p) + c_0 \hat{p}(x,t;\boldsymbol{\theta}_p) - \hat{y}(x,t;\boldsymbol{\theta}_y) + y_d(x,t)\Big|^2 \\
& + \frac{w_{b_1}}{|\mathcal{T}_{b_1}|} \sum_{\{x,t\} \in \mathcal{T}_{b_1}} |\hat{p}(x,t;\boldsymbol{\theta}_p)|^2 + \frac{w_{b_2}}{|\mathcal{T}_{b_2}|} \sum_{x \in \mathcal{T}_{b_2}} |\hat{p}(x,0;\boldsymbol{\theta}_p)|^2 \Big)
\end{aligned}
$$

- Train the neural networks to update the parameters $\boldsymbol{\theta}_y^{k+1}$ and $\boldsymbol{\theta}_p^{k+1}$, and update u^{k+1} by $u^{k+1}(x,t) = \frac{1}{\alpha+\beta}(-\hat{p}(x,t;\boldsymbol{\theta}_p^{k+1}) + \lambda^k(x,t) + \beta z^k(x,t))$.

ADMM-PINNs

- **Input:** $\beta > 0$, $z^0, \lambda^0, \boldsymbol{\theta}_y^0, \boldsymbol{\theta}_p^0$.
- For $k \geq 1$
- Update u^{k+1} by the above PINNs.
- Update z^{k+1} by $z^{k+1}(x,t) = \mathbb{P}_{U_{ad}}\left(\mathbb{S}_{\frac{\rho}{\beta}}\left(u^{k+1}(x,t) - \frac{\lambda^k(x,t)}{\beta}\right)\right)$.
- Update $\lambda^{k+1}(x,t) = \lambda^k(x,t) - \beta(u^{k+1}(x,t) - z^{k+1}(x,t))$.
- **Output:** Parameters $(\boldsymbol{\theta}_y^*, \boldsymbol{\theta}_p^*)$ and hence approximate solutions $\hat{y}(x,t;\boldsymbol{\theta}_y^*)$ and $\hat{u}(x,t) = \frac{1}{\alpha+\beta}(-\hat{p}(x,t;\boldsymbol{\theta}_p^*) + \lambda^k(x,t) + \beta z^k(x,t))$.

Numerical Experiments-Problem Setting

- Set $\Omega = (0,1)^2$, $T = 1, \nu = 1, c_0 = 0, a = -1, b = 2, \bar{y} = 5\sqrt{\rho}t\sin(3\pi x_1)\sin(\pi x_2), \bar{p} = 5\sqrt{\rho}(t-1)\sin(\pi x_1)\sin(\pi x_2)$, and

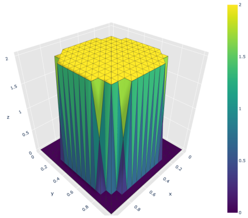
$$\bar{u} = \begin{cases} \max\{\dfrac{-\bar{p}+\rho}{\alpha}, a\} & \text{in } \{(x,t) \in \Omega \times (0,T) : \bar{p}(x,t) > \rho\}, \\ \min\{\dfrac{-\bar{p}-\rho}{\alpha}, b\} & \text{in } \{(x,t) \in \Omega \times (0,T) : \bar{p}(x,t) < -\rho\}, \\ 0 & \text{otherwise.} \end{cases}$$

- We further set $f = \frac{\partial \bar{y}}{\partial t} - \Delta \bar{y} - u$ and $y_d = \bar{y} - (-\frac{\partial \bar{p}}{\partial t} - \Delta \bar{p})$.
- Then it can be shown that $\bar{u}$ is the optimal control and $\bar{y}$ is the corresponding optimal state.

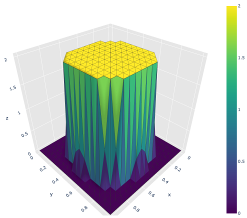
Numerical Experiments- Neural Networks and Training

- We approximate y and p with fully-connected feed-forward neural networks containing 3 hidden layers of 32 neurons each. The hyperbolic tangent activation function is used in all the neural networks.
- We uniformly sample $|\mathcal{T}_i| = 4096$ residual points in the spatial-temporal domain $\Omega \times (0, T)$, and $|\mathcal{T}_{b_1}| = 1024$ points in $\partial\Omega \times (0, T)$ and $|\mathcal{T}_{b_2}| = 256$ points in Ω for the boundary and initial conditions.
- The weights are set as $w_y = w_p = 1$, $w_i = 1$ and $w_{b1} = w_{b2} = 5$.
- To train the neural networks, we first use the Adam optimizer with learning rate $\eta = 10^{-3}$ for 10000 iterations, and then switch to the L-BFGS for 10 iterations.
- We execute 10 ADMM iterations with $\alpha = 0.1, \rho = 0.8, \beta = 0.1$, $z^0 = 0$ and $\lambda^0 = 0$.

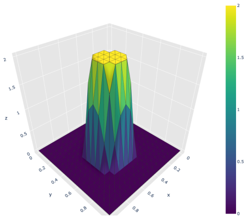
Numerical Results - Spatial Sparsity



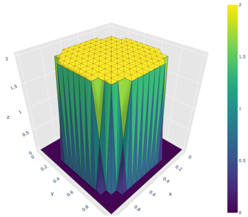
(a) Exact control at $t = 0.25$



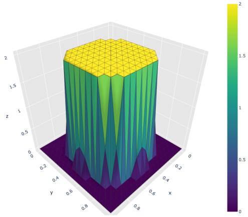
(b) Exact control at $t = 0.5$



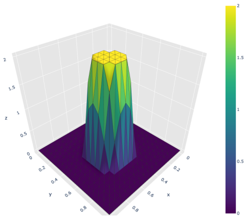
(c) Exact control at $t = 0.75$



(d) Computed control at $t = 0.25$



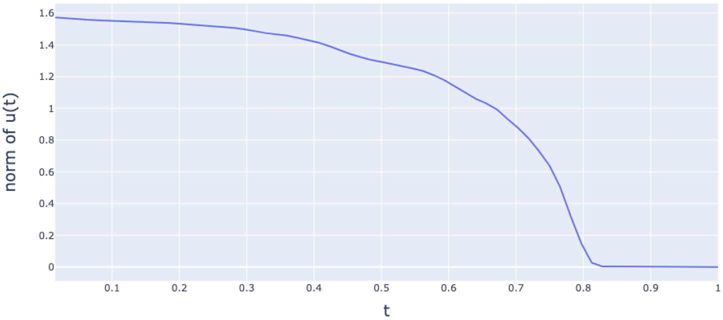
(e) Computed control at $t = 0.5$



(f) Computed control at $t = 0.75$

Numerical Results - Temporal Sparsity

- It is easy to see $\bar{u} = 0$ in $\{(x,t) \in \Omega \times (0,T) : \bar{p}(x,t) < \rho\}$ and we can show that when $t > t^* = 0.8211$, $u(x,t) = 0$ a.e. in Ω.



- The relative error $\frac{\|u^k(x,t) - \bar{u}(x,t)\|_{L^2(Q)}}{\|\bar{u}(x,t)\|_{L^2(Q)}} = 1.45 \times 10^{-2}$.

Extensions of ADMM-PINNs

- More applications of the ADMM-PINNs and numerical results can be found in *Y. Song, Y. Yuan, and H. Yue, "The ADMM-PINNs algorithmic framework for nonsmooth PDE-constrained optimization: a deep learning approach", arXiv preprint arXiv:2302.08309,2023.*

The Hard-Constraint PINNs for Elliptic Interface Optimal Control Problems

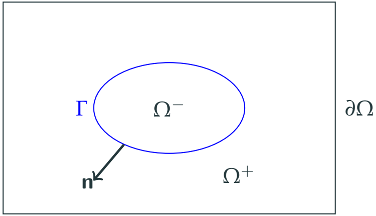
Elliptic Interface Optimal Control-Revisit

- An elliptic interface optimal control problem:

$$\min_{y\in L^2(\Omega), u\in L^2(\Omega)} J(y,u) := \frac{1}{2}\|y-y_d\|^2_{L^2(\Omega)} + \frac{\alpha}{2}\|u\|^2_{L^2(\Omega)},$$

subject to

$$\begin{cases} -\nabla\cdot(\beta\nabla y) = u + f & \text{in } \Omega\backslash\Gamma, \\ [y]_\Gamma = g_0,\ [\beta\partial_{\boldsymbol{n}} y]_\Gamma = g_1 & \text{on } \Gamma, \\ y = h_0 & \text{on } \partial\Omega, \end{cases}$$



First-Order Optimality Systems

- Let $(u^*, y^*)^\top$ be the solution to the elliptic interface optimal control problem and p^* be the corresponding adjoint variable, then the following first-order optimality system holds:

$$u^* = -\frac{1}{\alpha} p^*,$$

$$\begin{cases} -\nabla \cdot (\beta \nabla y^*) = u^* + f & \text{in } \Omega \backslash \Gamma, \\ [y^*]_\Gamma = g_0, \ [\beta \partial_{\boldsymbol{n}} y^*]_\Gamma = g_1 & \text{on } \Gamma, \\ y^* = h_0 & \text{on } \partial\Omega, \end{cases}$$

$$\begin{cases} -\nabla \cdot (\beta \nabla p^*) = y^* - y_d & \text{in } \Omega \backslash \Gamma, \\ [p^*]_\Gamma = 0, \ [\beta \partial_{\boldsymbol{n}} p^*]_\Gamma = 0 & \text{on } \Gamma, \\ p^* = 0 & \text{on } \partial\Omega. \end{cases}$$

Smooth Extension

- First, the function $y : \Omega \to \mathbb{R}$ is only piecewise-smooth, but it can be extended to a $(d+1)$-dimensional function $\tilde{y}(x, z) : \Omega \times \mathbb{R} \to \mathbb{R}$, which is smooth on the domain $\Omega \times \mathbb{R}$ and satisfies

$$y(x) = \begin{cases} \tilde{y}(x, 1), & \text{if } x \in \Omega^+, \\ \tilde{y}(x, -1), & \text{if } x \in \Omega^-, \end{cases}$$

Smooth Extension

- First, the function $y : \Omega \to \mathbb{R}$ is only piecewise-smooth, but it can be extended to a $(d+1)$-dimensional function $\tilde{y}(x, z) : \Omega \times \mathbb{R} \to \mathbb{R}$, which is smooth on the domain $\Omega \times \mathbb{R}$ and satisfies

$$y(x) = \begin{cases} \tilde{y}(x, 1), & \text{if } x \in \Omega^+, \\ \tilde{y}(x, -1), & \text{if } x \in \Omega^-, \end{cases}$$

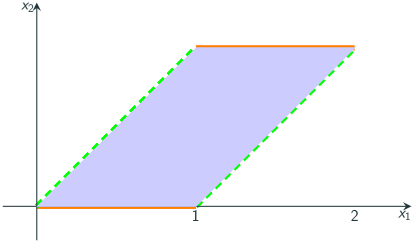
- The additional input $z \in \mathbb{R}$ is the augmented coordinate variable that labels Ω^+ and Ω^-.

Smooth Extension

- First, the function $y : \Omega \to \mathbb{R}$ is only piecewise-smooth, but it can be extended to a $(d+1)$-dimensional function $\tilde{y}(x, z) : \Omega \times \mathbb{R} \to \mathbb{R}$, which is smooth on the domain $\Omega \times \mathbb{R}$ and satisfies

$$y(x) = \begin{cases} \tilde{y}(x, 1), & \text{if } x \in \Omega^+, \\ \tilde{y}(x, -1), & \text{if } x \in \Omega^-, \end{cases}$$

- The additional input $z \in \mathbb{R}$ is the augmented coordinate variable that labels Ω^+ and Ω^-.

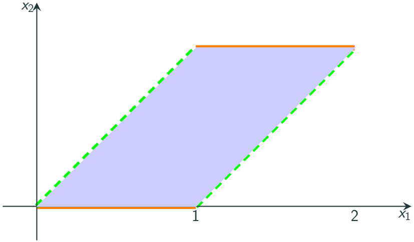


Smooth Extension

- First, the function $y : \Omega \to \mathbb{R}$ is only piecewise-smooth, but it can be extended to a $(d+1)$-dimensional function $\tilde{y}(x, z) : \Omega \times \mathbb{R} \to \mathbb{R}$, which is smooth on the domain $\Omega \times \mathbb{R}$ and satisfies

$$y(x) = \begin{cases} \tilde{y}(x, 1), & \text{if } x \in \Omega^+, \\ \tilde{y}(x, -1), & \text{if } x \in \Omega^-, \end{cases}$$

- The additional input $z \in \mathbb{R}$ is the augmented coordinate variable that labels Ω^+ and Ω^-.



- Similarly, one can extend p to a $(d+1)$-dimensional smooth function $\tilde{p}(x, z)$.

Smooth Extension -Cont'd

- Substituting $\tilde{y}$ and $\tilde{p}$ into the first-order optimality system, we obtain that

$$
\begin{cases}
-\Delta_x \tilde{y}(x,z) = \begin{cases} \dfrac{1}{\beta^+}\left(f(x) + \left(-\dfrac{1}{\alpha}\tilde{p}(x,z)\right)\right) & \text{if } x \in \Omega^+, z = 1 \\ \dfrac{1}{\beta^-}\left(f(x) + \left(-\dfrac{1}{\alpha}\tilde{p}(x,z)\right)\right) & \text{if } x \in \Omega^-, z = -1 \end{cases}, \\
\tilde{y}(x,1) - \tilde{y}(x,-1) = g_0(x), \quad \boldsymbol{n}\cdot(\beta^+\nabla_x \tilde{y}(x,1) - \beta^-\nabla_x \tilde{y}(x,-1)) = g_1(x), \quad \text{if } x \in \Gamma, \\
-\Delta_x \tilde{p}(x,z) = \begin{cases} \dfrac{1}{\beta^+}(\tilde{y}(x,z) - y_d(x)) & \text{if } x \in \Omega^+, z = 1 \\ \dfrac{1}{\beta^-}(\tilde{y}(x,z) - y_d(x)) & \text{if } x \in \Omega^-, z = -1 \end{cases}, \\
\tilde{p}(x,1) - \tilde{p}(x,-1) = 0, \quad \boldsymbol{n}\cdot(\beta^+\nabla_x \tilde{p}(x,1) - \beta^-\nabla_x \tilde{p}(x,-1)) = 0, \quad \text{if } x \in \Gamma, \\
\tilde{y}(x,1) = h_0(x), \quad \tilde{p}(x,1) = 0 \quad \text{if } x \in \partial\Omega.
\end{cases}
$$

Discontinuity Capturing Shallow Neural Networks

- Since $\tilde{y}$ and $\tilde{p}$ are continuous in $\Omega \times \mathbb{R}$, it follows from the universal approximation theorem [Cybenko,1989] that one can approximate them by two shallow neural networks $\hat{y}(x, z; \theta_y)$ and $\hat{p}(x, z; \theta_p)$.

Discontinuity Capturing Shallow Neural Networks

- Since $\tilde{y}$ and $\tilde{p}$ are continuous in $\Omega \times \mathbb{R}$, it follows from the universal approximation theorem [Cybenko,1989] that one can approximate them by two shallow neural networks $\hat{y}(x, z; \theta_y)$ and $\hat{p}(x, z; \theta_p)$.
- Such neural networks are referred to as the Discontinuity Capturing Shallow Neural Networks (DCSNN) [Hu, Lin, and Lai, 2022].

Discontinuity Capturing Shallow Neural Networks

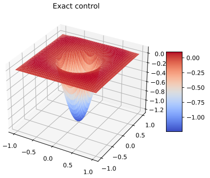
- Since $\tilde{y}$ and $\tilde{p}$ are continuous in $\Omega \times \mathbb{R}$, it follows from the universal approximation theorem [Cybenko,1989] that one can approximate them by two shallow neural networks $\hat{y}(x, z; \theta_y)$ and $\hat{p}(x, z; \theta_p)$.
- Such neural networks are referred to as the Discontinuity Capturing Shallow Neural Networks (DCSNN) [Hu, Lin, and Lai, 2022].
- PINNs can be applied!

PINNs

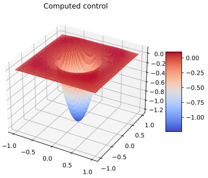
- Sample $\mathcal{T} := \{(x^i, z^i)\}_{i=1}^{M} \subset (\Omega^+ \times \{1\}) \cup (\Omega^- \times \{-1\})$, $\mathcal{T}_B := \{x_B^i\}_{i=1}^{M_B} \subset \partial\Omega$, and $\mathcal{T}_\Gamma := \{x_\Gamma^i\}_{i=1}^{M_\Gamma} \subset \Gamma$.
- Train the neural networks $\hat{y}(x, z; \theta_y)$ and $\hat{p}(x, z; \theta_p)$ by minimizing the loss function:

$$
\begin{aligned}
\mathcal{L}(\theta_y, \theta_p) = & \frac{w_{y,r}}{M} \sum_{i=1}^{M} \left| -\Delta_x \hat{y}(x^i, z^i; \theta_y) - \frac{(-\frac{1}{\alpha}\hat{p}(x^i, z^i; \theta_p)) + f(x^i)}{\beta^\pm} \right|^2 + \frac{w_{y,b}}{M_b} \sum_{i=1}^{M_b} |\hat{y}(x_B^i, 1; \theta_y) - h_0(x_B^i)|^2 \\
& + \frac{w_{y,\Gamma}}{M_\Gamma} \sum_{i=1}^{M_\Gamma} \left| \hat{y}(x_\Gamma^i, 1; \theta_y) - \hat{y}(x_\Gamma^i, -1; \theta_y) - g_0(x_\Gamma^i) \right|^2 \\
& + \frac{w_{y,\Gamma_n}}{M_\Gamma} \sum_{i=1}^{M_\Gamma} \left| \boldsymbol{n} \cdot (\beta^+ \nabla_x \hat{y}(x_i^\Gamma, 1; \theta_y) - \beta^- \nabla_x \hat{y}(x_i^\Gamma, -1; \theta_y)) - g_1(x_\Gamma^i) \right|^2 \\
& + \frac{w_{p,r}}{M} \sum_{i=1}^{M} \left| -\Delta_x \hat{p}(x_i, z_i; \theta_p) - \frac{\hat{y}(x_i, z_i; \theta_y) - y_d(x_i)}{\beta^\pm} \right|^2 + \frac{w_{p,b}}{M_b} \sum_{i=1}^{M_b} |\hat{p}(x_i^b, 1; \theta_p)|^2 \\
& + \frac{w_{p,\Gamma}}{M_\Gamma} \sum_{i=1}^{M_\Gamma} \left| \hat{p}(x_\Gamma^i, 1; \theta_p) - \hat{p}(x_\Gamma^i, -1; \theta_p) \right|^2 + \frac{w_{p,\Gamma_n}}{M_\Gamma} \sum_{i=1}^{M_\Gamma} \left| \boldsymbol{n} \cdot (\beta^+ \nabla_x \hat{p}(x_i^\Gamma, 1; \theta_p) - \beta^- \nabla_x \hat{p}(x_i^\Gamma, -1; \theta_p)) \right|^2
\end{aligned}
$$

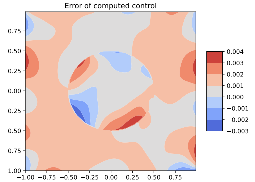
Numerical Results



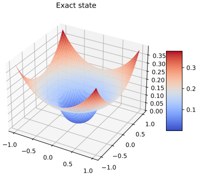
(g) Exact control u.



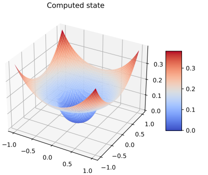
(h) Computed control u.



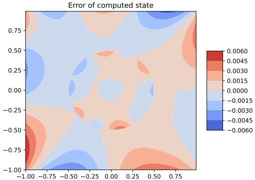
(i) Error of control u.



(j) Exact state y.



(k) Computed state y.



(l) Error of state y.

Hard Constraints-Motivation

- In the above PINNs,

Hard Constraints-Motivation

- In the above PINNs,
  - The boundary and interface conditions are penalized in the loss function with constant penalty parameters.

Hard Constraints-Motivation

- In the above PINNs,
  - The boundary and interface conditions are penalized in the loss function with constant penalty parameters. Hence, these conditions are treated as soft constraints and cannot be satisfied rigorously.

Hard Constraints-Motivation

- In the above PINNs,
    - The boundary and interface conditions are penalized in the loss function with constant penalty parameters. Hence, these conditions are treated as soft constraints and cannot be satisfied rigorously.
    - The boundary and interface conditions, as well as the PDE, are treated together during the training process and its effectiveness strongly depends on the choices of the weights in the loss function.

Hard Constraints-Motivation

- In the above PINNs,
  - The boundary and interface conditions are penalized in the loss function with constant penalty parameters. Hence, these conditions are treated as soft constraints and cannot be satisfied rigorously.
  - The boundary and interface conditions, as well as the PDE, are treated together during the training process and its effectiveness strongly depends on the choices of the weights in the loss function. Manually determining these weights through trial and error is challenging and time-demanding.

Hard Constraints-Motivation

- In the above PINNs,
  - The boundary and interface conditions are penalized in the loss function with constant penalty parameters. Hence, these conditions are treated as soft constraints and cannot be satisfied rigorously.
  - The boundary and interface conditions, as well as the PDE, are treated together during the training process and its effectiveness strongly depends on the choices of the weights in the loss function. Manually determining these weights through trial and error is challenging and time-demanding.
  - The above numerical results show that the numerical errors mainly accumulate on the boundaries and the interfaces.

Hard Constraints-Motivation

- In the above PINNs,
  - The boundary and interface conditions are penalized in the loss function with constant penalty parameters. Hence, these conditions are treated as soft constraints and cannot be satisfied rigorously.
  - The boundary and interface conditions, as well as the PDE, are treated together during the training process and its effectiveness strongly depends on the choices of the weights in the loss function. Manually determining these weights through trial and error is challenging and time-demanding.
  - The above numerical results show that the numerical errors mainly accumulate on the boundaries and the interfaces.
- To tackle the above issues, we consider imposing the boundary and interface conditions as hard constraints so that they are satisfied exactly and can be treated separately from the PDE in the training of the neural networks.

Hard Constraints- Strategies

Hard Constraints- Strategies

- Develop a novel neural network architecture by generalizing the DCSNN to approximate y and p.

Hard Constraints- Strategies

- Develop a novel neural network architecture by generalizing the DCSNN to approximate y and p.
  - Modify the output of the neural network to impose the boundary condition.

Hard Constraints- Strategies

- Develop a novel neural network architecture by generalizing the DCSNN to approximate y and p.
  - Modify the output of the neural network to impose the boundary condition.
  - Construct an auxiliary function for the interface as an additional feature input of the neural network to impose the interface condition.

Hard Constraints - Neural Network Architectures

- Recall that $y = h_0$ on $\partial\Omega$, and $[y]_\Gamma = g_0$.

Hard Constraints - Neural Network Architectures

- Recall that $y = h_0$ on $\partial\Omega$, and $[y]_\Gamma = g_0$.
- We approximate y by

$$\hat{y}(x;\theta_y) = g(x) + h(x)\mathcal{N}_y(x, \phi(x); \theta_y).$$

- The function $g : \overline{\Omega} \to \mathbb{R}$ satisfies

$$g|_{\partial\Omega} = h_0, \quad [g]_\Gamma = g_0, \quad g|_{\Omega^+} \in C^2(\overline{\Omega^+}), \quad g|_{\Omega^-} \in C^2(\overline{\Omega^-}).$$

- The function $h : \overline{\Omega} \to \mathbb{R}$ satisfies

$$h \in C^2(\overline{\Omega}), \quad h(x) = 0 \text{ if and only if } x \in \partial\Omega.$$

- $\phi : \overline{\Omega} \to \mathbb{R}$ is an auxiliary function for the interface Γ and satisfies

$$\phi \in C(\overline{\Omega}), \quad \phi|_{\Omega^+} \in C^2(\overline{\Omega^+}), \quad \phi|_{\Omega^-} \in C^2(\overline{\Omega^-}), [\phi]_\Gamma = 0, \quad [\beta\partial_n\phi]_\Gamma \neq 0 \text{ a.e. on } \Gamma$$

- $\mathcal{N}_y(x, \phi(x); \theta_y)$ is a neural network with smooth activation functions.

Hard Constraints - Verification

Hard Constraints - Verification

- First, $h(x)\mathcal{N}_y(x, \phi(x); \theta_y)$ is a continuous function of x over $\overline{\Omega}$.

Hard Constraints - Verification

- First, $h(x)\mathcal{N}_y(x, \phi(x); \theta_y)$ is a continuous function of x over $\overline{\Omega}$.
- $[\hat{y}]_\Gamma(x) = [g]_\Gamma(x) + [h(\cdot)\mathcal{N}_y(\cdot, \phi(\cdot))]_\Gamma(x) = g_0(x), \ \forall x \in \Gamma$ – The interface condition is satisfied.

Hard Constraints - Verification

- First, $h(x)\mathcal{N}_y(x, \phi(x); \theta_y)$ is a continuous function of x over $\overline{\Omega}$.
- $[\hat{y}]_\Gamma(x) = [g]_\Gamma(x) + [h(\cdot)\mathcal{N}_y(\cdot, \phi(\cdot))]_\Gamma(x) = g_0(x), \ \forall x \in \Gamma$ – The interface condition is satisfied.
- $\hat{y}|_{\partial\Omega}(x) = g|_{\partial\Omega}(x) + h|_{\partial\Omega}(x) \left(\mathcal{N}_y(\cdot, \phi(\cdot))|_{\partial\Omega}\right)(x) = h_0(x), \ \forall x \in \partial\Omega$. – The boundary condition is satisfied.

Hard Constraints - Verification

- First, $h(x)\mathcal{N}_y(x, \phi(x); \theta_y)$ is a continuous function of x over $\overline{\Omega}$.
- $[\hat{y}]_\Gamma(x) = [g]_\Gamma(x) + [h(\cdot)\mathcal{N}_y(\cdot, \phi(\cdot))]_\Gamma(x) = g_0(x), \;\; \forall x \in \Gamma$ – The interface condition is satisfied.
- $\hat{y}|_{\partial\Omega}(x) = g|_{\partial\Omega}(x) + h|_{\partial\Omega}(x)\left(\mathcal{N}_y(\cdot, \phi(\cdot))|_{\partial\Omega}\right)(x) = h_0(x), \;\; \forall x \in \partial\Omega.$ – The boundary condition is satisfied.
- Furthermore, we have

$$
\begin{aligned}
[\beta \partial_{\boldsymbol{n}} \hat{y}]_\Gamma(x) &= [\beta \partial_n g]_\Gamma(x) + [\beta \partial_n \mathcal{N}_y(\cdot, \phi(\cdot))]_\Gamma(x) \\
&= [\beta \partial_{\boldsymbol{n}} g]_\Gamma(x) + (\beta^+ - \beta^-)\left(\mathcal{N}_y(x, \phi(x))(\boldsymbol{n} \cdot \nabla h(x)) + h(x)(\boldsymbol{n} \cdot \nabla_x \mathcal{N}(x, \phi(x)))\right) \\
&\qquad + \frac{\partial \mathcal{N}_y}{\partial \phi}\left(h(x)[\beta \partial_{\boldsymbol{n}} \phi]_\Gamma(x)\right), \quad \forall x \in \Gamma,
\end{aligned}
$$

which implies that the interface-gradient condition $[\beta \partial_{\boldsymbol{n}} y]_\Gamma = g_1$ cannot be exactly satisfied by $\hat{y}(x; \theta_y)$ and should be penalized in the loss function of PINNs.

Choices of Auxiliary Functions

- If the functions g_0 and h_0, the interface Γ, and the boundary $\partial\Omega$ admit analytic forms, it is usually easy to construct g and h with analytic expressions.
- For instance, if $\Omega = (0,1) \times (0,1)$, then we can choose $h = x_1(1 - x_1)x_2(1 - x_2)$.
- More discussions can be found in e.g., [Lagari, Tsoukalas, Safarkhani, and Lagaris, 2020; Lagaris, Likas, and Fotiadis, 1998; Lu, Pestourie, Yao, Wang, Verdugo, and, Johnson, 2021].

Choices of Auxiliary Functions - Cont'd

- The choice of ϕ is more delicate.

Choices of Auxiliary Functions - Cont'd

- The choice of ϕ is more delicate.
- If Γ is the zero level set of a function in $C^k(\Omega)(k \geq 2)$, then ϕ can be easily constructed.

Choices of Auxiliary Functions - Cont'd

- The choice of ϕ is more delicate.
- If Γ is the zero level set of a function in $C^k(\Omega)(k \geq 2)$, then ϕ can be easily constructed.
- (Circle-shaped Interface) Consider a domain $\Omega \subset \mathbb{R}^d$ and the interface $\Gamma \subset \Omega$ is given by the circle $\Gamma = \{x \in \mathbb{R}^{d-1} : \|x\|_2 = r_0\}$, with $r_0 > 0$. The domain Ω is divided into two parts $\Omega^- = \{x \in \mathbb{R}^{d-1} : \|x\|_2 < r_0\}$ and $\Omega^+ = \{x \in \Omega : \|x\|_2 > r_0\}$. In this case, the auxiliary function ϕ can be constructed as

$$\phi(x) = \begin{cases} r_0^2 - \|x\|_2^2, & \text{if } x \in \Omega^- \\ 0, & \text{if } x \in \Omega^+ \cup \Gamma \cup \partial\Omega. \end{cases}$$

Choices of Auxiliary Functions - Cont'd

- The choice of ϕ is more delicate.
- If Γ is the zero level set of a function in $C^k(\Omega)(k \geq 2)$, then ϕ can be easily constructed.
- (Circle-shaped Interface) Consider a domain $\Omega \subset \mathbb{R}^d$ and the interface $\Gamma \subset \Omega$ is given by the circle $\Gamma = \{x \in \mathbb{R}^{d-1} : \|x\|_2 = r_0\}$, with $r_0 > 0$. The domain Ω is divided into two parts $\Omega^- = \{x \in \mathbb{R}^{d-1} : \|x\|_2 < r_0\}$ and $\Omega^+ = \{x \in \Omega : \|x\|_2 > r_0\}$. In this case, the auxiliary function ϕ can be constructed as

$$\phi(x) = \begin{cases} r_0^2 - \|x\|_2^2, & \text{if } x \in \Omega^- \\ 0, & \text{if } x \in \Omega^+ \cup \Gamma \cup \partial\Omega. \end{cases}$$

- (Box-shaped Interface) Consider a domain $\Omega \subset \mathbb{R}^d$ containing the box $B := [a_1, b_1] \times \cdots \times [a_d, b_d] \in \mathbb{R}^d$. Let the interface $\Gamma = \partial B$, which divides Ω into $\Omega^- = (a_1, b_1) \times \cdots \times (a_d, b_d)$ and $\Omega^+ = \Omega \backslash B$. In this case, we define

$$\phi(x) = \begin{cases} \prod_{i=1}^d (x_i - a_i)(b_i - x_i), & \text{if } x \in \Omega^-, \\ 0, & \text{if } x \in \Omega^+ \cup \Gamma \cup \partial\Omega. \end{cases}$$

- Otherwise, we shall show that, if Ω^+, Ω^-, and Γ satisfy the following assumptions, then we can construct an auxiliary function $\phi(x)$ analytically.

Assumptions

- The sub-domain Ω^- is the intersection of the interior of finitely many oriented, smooth, and embedded manifolds $M_1, M_2, \ldots, M_n$, where $M_i \cap M_j$ is of measure zero whenever $i \neq j$ and $i, j \in \{1, \ldots, n\}$.
- There exists an open neighborhood $U \subset \mathbb{R}^d$ of Γ, such that for each M_i ($i \in \{1, \ldots, n\}$), there exists smooth functions $\psi_i : U \to \mathbb{R}$ satisfying $\psi_i \in C^2(\overline{U})$ and

$$\psi_i(x) = 0 \text{ if } x \in \Gamma, \quad \psi_i(x) > 0 \text{ if } x \in U \cap \Omega^-, \quad \partial_{\boldsymbol{n}} \psi_i \neq 0 \text{ on } M_i \cap \Gamma.$$

- There exists constants $c_i > 0$ such that $\psi_i(x) > c_i$ for all $x \in \partial U \cap \overline{\Omega^-}$ and for all $i \in \{1, \ldots, n\}$.

Choices of Auxiliary Functions - Cont'd

Theorem

Suppose the above assumptions hold and we define $\psi : U \to \mathbb{R}$ as $\psi(x) = \prod_{i=1}^{n} \psi_i(x)$. For any constant c such that $0 < c < \prod_{i=1}^{n} c_i$, let

$$L_c := \{x \in U : \psi(x) \geq c\}.$$

Then the function $\phi : \bar{\Omega} \to \mathbb{R}$ given by

$$\phi(x) = \begin{cases} c^3, & \text{if } x \in (\overline{\Omega^-} \backslash U) \cup (\overline{\Omega^-} \cap L_c), \\ c^3 - (c - \psi(x))^3, & \text{if } x \in (U \cap \overline{\Omega^-}) \backslash L_c, \\ 0, & \text{if } x \in \overline{\Omega^+} \end{cases}$$

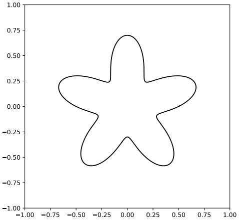
is well-defined and satisfies

$$\phi \in C(\overline{\Omega}), \quad \phi|_{\Omega^+} \in C^2(\overline{\Omega^+}), \quad \phi|_{\Omega^-} \in C^2(\overline{\Omega^-}), [\phi]_\Gamma = 0, \quad [\beta \partial_n \phi]_\Gamma \neq 0 \text{ a.e. on } \Gamma.$$

An Example of ϕ

- Let $\Omega \subset \mathbb{R}^2$ be a bounded domain and the star-shaped interface $\Gamma \subset \mathbb{R}$ be defined by the zero level set of the following function in polar coordinates $\psi(r,\theta) = r - a - b\sin(5\theta)$ with constants $b < a$. The domain Ω is divided into $\Omega^- = \{(r,\theta) \in \mathbb{R}^2 : r < a + b\sin(5\theta)\}$ and $\Omega^+ = \{(r,\theta) \in \Omega : r > a + b\sin(5\theta)\}$.
- Note that $\psi(r,\theta)$ is not differentiable on Ω, since the polar angle is not differentiable at the origin. In this case, we define

$$\phi(r,\theta) = \begin{cases} \left(\dfrac{a-b}{2}\right)^3, & \text{if } a + b\sin(5\theta) - r \geq \dfrac{a-b}{2}, \\ \left(\dfrac{a-b}{2}\right)^3 - \left(\dfrac{a-b}{2} + \psi(r,\theta)\right)^3, & \text{if } 0 < a + b\sin(5\theta) - r < \dfrac{a-b}{2}, \\ 0, & \text{otherwise.} \end{cases}$$



Remarks on the Choices of Auxiliary Functions

- If the functions g, h, and ϕ are difficult to construct analytically, we can construct them by training some neural networks.

Remarks on the Choices of Auxiliary Functions

- If the functions g, h, and ϕ are difficult to construct analytically, we can construct them by training some neural networks.
- For instance, we can train a DCSNN $\hat{g}(x, z; \theta_g)$ and a neural network $\hat{h}(x; \theta_h)$ with smooth activation functions by minimizing the following loss functions:

$$\frac{w_{1g}}{M_b} \sum_{i=1}^{M_b} |\hat{g}(x_B^i, 1; \theta_y) - h_0(x_B^i)|^2 + \frac{w_{2g}}{M_\Gamma} \sum_{i=1}^{M_\Gamma} \left| \hat{g}(x_\Gamma^i, 1; \theta_g) - \hat{g}(x_\Gamma^i, -1; \theta_g) - g_0(x_\Gamma^i) \right|^2,$$

and

$$\frac{w_{1h}}{M_b} \sum_{i=1}^{M_b} |\hat{h}(x_B^i; \theta_h)|^2 + \frac{w_{2h}}{M} \sum_{i=1}^{M} |\hat{h}(x^i; \theta_h) - \bar{h}(x^i)|^2,$$

  - w_{1g}, w_{2g}, w_{1h}, and $w_{2h} > 0$ are the weights.
  - $\{x^i\}_{i=1}^{M} \subset \Omega$, $\{x_B^i\}_{i=1}^{M_B} \subset \partial\Omega$, and $\{x_\Gamma^i\}_{i=1}^{M_\Gamma} \subset \Gamma$ are the training points.
  - $\bar{h}(x) \in C^2(\Omega)$ is a known function satisfying $\bar{h}(x) \neq 0$ in Ω, e.g. $\bar{h}(x) = \min_{\hat{x} \in \partial\Omega} \{\|x - \hat{x}\|_2^4\}$.

The Hard-Constraint PINNs

- We approximate y by the neural network $\hat{y}(x;\theta_y) = g(x) + h(x)\mathcal{N}_y(x, \phi(x); \theta_y)$ defined above.

The Hard-Constraint PINNs

- We approximate y by the neural network $\hat{y}(x;\theta_y) = g(x) + h(x)\mathcal{N}_y(x, \phi(x); \theta_y)$ defined above.
- Since the boundary and interface conditions for p are homogeneous, we approximate it by

$$\hat{p}(x;\theta_p) = h(x)\mathcal{N}_p(x, \phi(x); \theta_p),$$

where $\mathcal{N}_p(x, \phi(x); \theta_p)$ is a neural netowrk with smooth activation functions and parameterized by θ_p, the functions h and ϕ are the same as those in $\hat{y}(x;\theta_y)$.

The Hard-Constraint PINNs

- We approximate y by the neural network $\hat{y}(x;\theta_y) = g(x) + h(x)\mathcal{N}_y(x,\phi(x);\theta_y)$ defined above.
- Since the boundary and interface conditions for p are homogeneous, we approximate it by

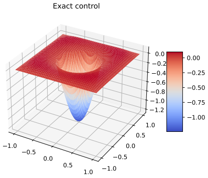
$$\hat{p}(x;\theta_p) = h(x)\mathcal{N}_p(x,\phi(x);\theta_p),$$

where $\mathcal{N}_p(x,\phi(x);\theta_p)$ is a neural netowrk with smooth activation functions and parameterized by θ_p, the functions h and ϕ are the same as those in $\hat{y}(x;\theta_y)$.

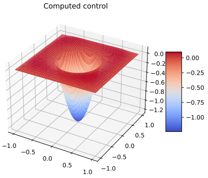
- The neural networks $\hat{y}(x;\theta_y)$ and $\hat{p}(x;\theta_p)$ are trained by minimizing

$$\begin{aligned}\mathcal{L}_{HC}(\theta_y,\theta_p) =& \frac{w_{y,r}}{M}\sum_{i=1}^{M}\left|-\Delta_x\hat{y}(x^i;\theta_y) - \frac{(-\frac{1}{\alpha}\hat{p}(x^i;\theta_p)) + f(x^i)}{\beta^{\pm}}\right|^2 + \frac{w_{y,\Gamma_n}}{M_\Gamma}\sum_{i=1}^{M_\Gamma}\left|[\beta\partial_{\boldsymbol{n}}\hat{y}]_\Gamma(x_\Gamma^i;\theta_y) - g_1(x_\Gamma^i)\right|^2 \\ &+ \frac{w_{p,r}}{M}\sum_{i=1}^{M}\left|-\Delta_x\hat{p}(x_i;\theta_p) - \frac{\hat{y}(x_i;\theta_y) - y_d(x_i)}{\beta^{\pm}}\right|^2 + \frac{w_{p,\Gamma_n}}{M_\Gamma}\sum_{i=1}^{M_\Gamma}\left|[\beta\partial_{\boldsymbol{n}}\hat{p}]_\Gamma(x_i^\Gamma;\theta_p)\right|^2.\end{aligned}$$

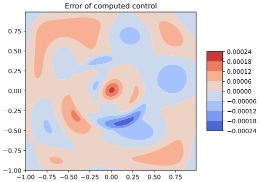
Numerical Results



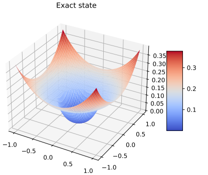
(m) Exact control u.



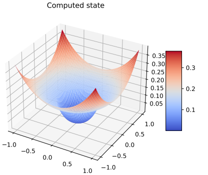
(n) Computed control u.



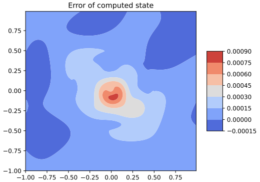
(o) Error of control u.



(p) Exact state y.



(q) Computed state y.



(r) Error of state y.

Numerical Comparisons

- To test the accuracy, we select 256×256 testing points $\{x^i\}_{i=1}^{M_T} \subset \Omega$ following the Latin hypercube sampling.
- Then compute

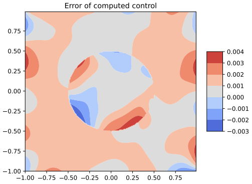
$$\varepsilon_{\text{abs}} = \sqrt{\frac{1}{M_T}\sum_{i=1}^{M_T}(\hat{u}(x^i) - u^*(x^i))^2}, \text{ and } \varepsilon_{rel} = \varepsilon_{abs}\sqrt{A(\Omega)}/||u^*||_{L^2(\Omega)}$$

where $A(\Omega)$ is the area of Ω (i.e. the Lebesgue measure of Ω), and $||u^*||_{L^2(\Omega)}$ is computed using the numerical integration function `dblquad` implemented in the SciPy library of Python.

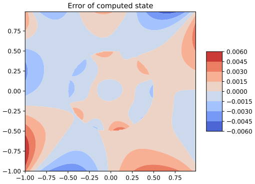
- The soft-constraint PINNs: $\varepsilon_{\text{abs}} = 1.2360 \times 10^{-3}$, $\varepsilon_{\text{rel}} = 4.0461 \times 10^{-3}$
- The hard-constraint PINNs: $\varepsilon_{\text{abs}} = 4.7652 \times 10^{-5}$, $\varepsilon_{\text{rel}} = 1.5599 \times 10^{-4}$
- The hard-constraint PINNs approach improves the accuracy by more than 20x.

Numerical Comparisons

- Numerical errors of the soft-constraint PINNs.

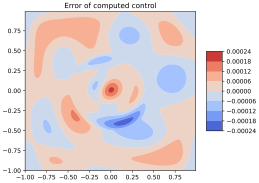


(s) Error of control u.

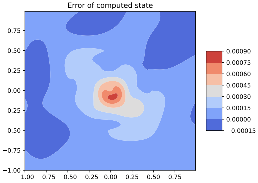


(t) Error of state y.

- Numerical errors of the hard-constraint PINNs.



(u) Error of control u.



(v) Error of state y.

More Results

- More numerical results on the hard-constraint PINNs for solving other interface optimal control problems can be found in
  M. Lai, Y. Song, X. Yuan, H. Yue, and T. Zeng. "The hard-constraint physics-informed neural networks for interface optimal control problems". to appear

Conclusions and Perspectives

Conclusions and Perspectives

Conclusions and Perspectives

- It is challenging to apply the PINNs directly to nonsmooth PDE-constrained optimization problems.

Conclusions and Perspectives

- It is challenging to apply the PINNs directly to nonsmooth PDE-constrained optimization problems.
- Combine PINNs with classic operator splitting optimization techniques (e.g., ADMM) leads to implementable and efficient algorithms for PDE-constrained optimization problems with nonsmooth objective functional.

Conclusions and Perspectives

- It is challenging to apply the PINNs directly to nonsmooth PDE-constrained optimization problems.
- Combine PINNs with classic operator splitting optimization techniques (e.g., ADMM) leads to implementable and efficient algorithms for PDE-constrained optimization problems with nonsmooth objective functional.
- Leveraged by the discontinuity capturing neural networks, PINNs can be applied to solve interface optimal control problem.

Conclusions and Perspectives

- It is challenging to apply the PINNs directly to nonsmooth PDE-constrained optimization problems.
- Combine PINNs with classic operator splitting optimization techniques (e.g., ADMM) leads to implementable and efficient algorithms for PDE-constrained optimization problems with nonsmooth objective functional.
- Leveraged by the discontinuity capturing neural networks, PINNs can be applied to solve interface optimal control problem.
- Imposing the boundary and interface conditions as hard constraints can improves the numerical accuracy and simplifies the training procedure of PINNs.

Conclusions and Perspectives

- It is challenging to apply the PINNs directly to nonsmooth PDE-constrained optimization problems.
- Combine PINNs with classic operator splitting optimization techniques (e.g., ADMM) leads to implementable and efficient algorithms for PDE-constrained optimization problems with nonsmooth objective functional.
- Leveraged by the discontinuity capturing neural networks, PINNs can be applied to solve interface optimal control problem.
- Imposing the boundary and interface conditions as hard constraints can improves the numerical accuracy and simplifies the training procedure of PINNs.
- The validated efficiency of the ADMM-PINNs and the hard-constraint PINNs clearly justifies the necessity to investigate the underlying theoretical issues such as the convergence analysis and the error estimate.


Danke
Merci
Cảm ơn bạn
Teşekkür ederiz
Grazie
谢谢
Obrigado
謝謝
Thank you
شكرا
Gracias
Terima kasih
धन्यवाद
ధన్యవాదాలు
Спасибо
شكرا
நன்றி
धन्यवाद
ありがとう
ขอบคุณ
감사합니다