DeepXDE: A deep learning library for solving differential equations

Lu Lu, Xuhui Meng, Zhiping Mao, George Karniadakis

Division of Applied Mathematics, Brown University, Providence, RI

`lu_lu_1@brown.edu`

Abstract

Physics-informed neural networks (PINNs) for solving partial differential equations (PDEs):

- *embed a PDE into the loss of the neural network,*
- *mesh-free,*
- *a unified framework: PDE, integro-differential equations [1], fractional PDEs [2], and stochastic PDEs [3],*
- *solve inverse problems as easily as forward problems.*

DeepXDE, a Python library for PINNs:

- *solve multi-physics problems;*
- *solves time-dependent PDEs as easily as steady states;*
- *supports complex-geometry domains;*
- *enables the user code to be compact, resembling closely the mathematical formulation.*

1. PINNs for solving PDEs

1.1 PINN Algorithm

Consider the PDE parameterized by $\boldsymbol{\lambda}$ for the solution $u(\mathbf{x})$ with $\mathbf{x} = (x_1, \dots, x_d)$ defined on a domain $\Omega \subset \mathbb{R}^d$:

$$f\left(\mathbf{x}; \frac{\partial u}{\partial x_1}, \dots, \frac{\partial u}{\partial x_d}; \frac{\partial^2 u}{\partial x_1 \partial x_1}, \dots, \frac{\partial^2 u}{\partial x_1 \partial x_d}; \dots; \boldsymbol{\lambda}\right) = 0, \quad \mathbf{x} \in \Omega,$$

with boundary conditions (BC) $\mathcal{B}(u, \mathbf{x}) = 0$ on $\partial\Omega$.
We consider time t as a special component of $\mathbf{x}$, and Ω contains the temporal domain. The initial condition (IC) can be simply treated as a special type of Dirichlet boundary condition on the spatio-temporal domain.

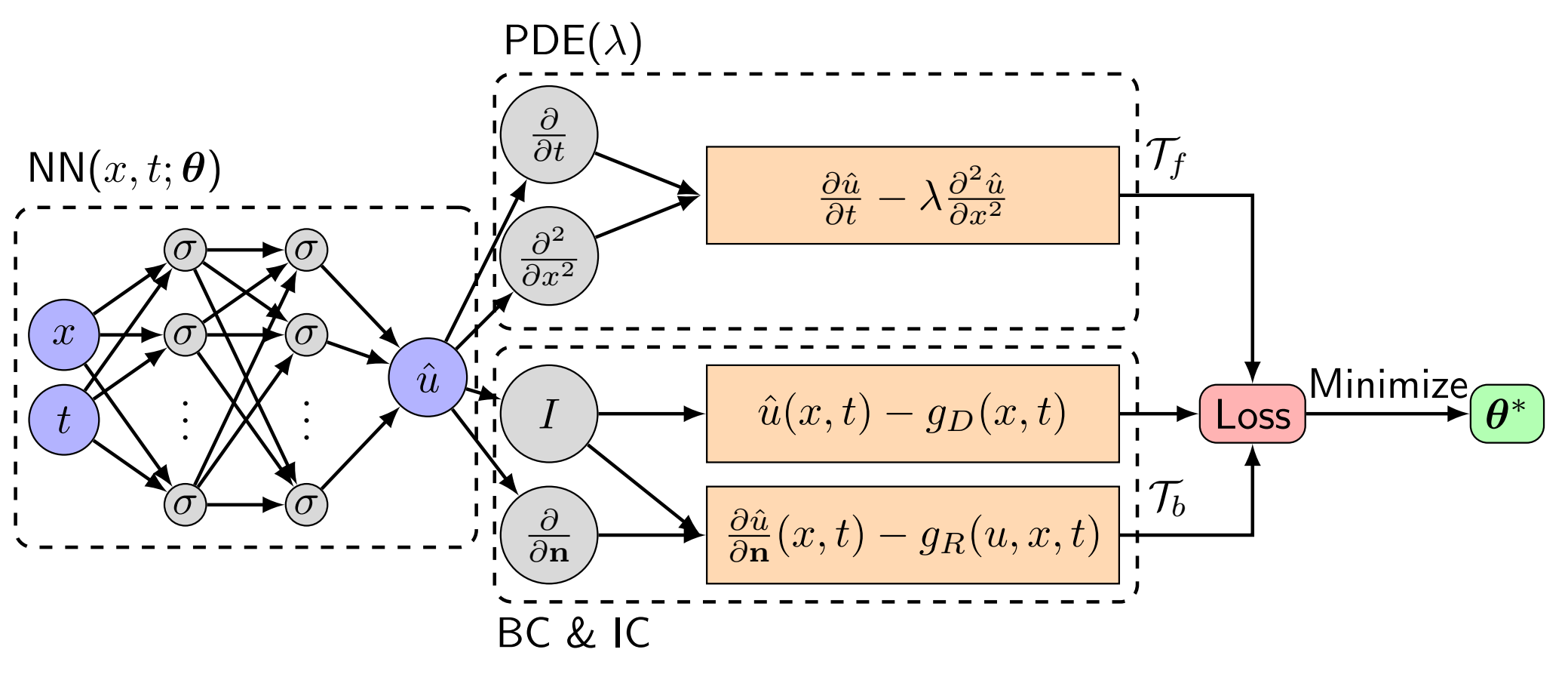
PDE(λ)
NN($x, t; \theta$)
x
t
σ
$\hat{u}$
$\frac{\partial}{\partial t}$
$\frac{\partial^2}{\partial x^2}$
$\frac{\partial \hat{u}}{\partial t} - \lambda \frac{\partial^2 \hat{u}}{\partial x^2}$
$\mathcal{T}_f$
I
$\frac{\partial}{\partial \mathbf{n}}$
$\hat{u}(x,t) - g_D(x,t)$
$\frac{\partial \hat{u}}{\partial \mathbf{n}}(x,t) - g_R(u,x,t)$
$\mathcal{T}_b$
Loss
Minimize
θ^*
BC & IC

Figure 1: *Schematic of a PINN for solving the diffusion equation* $\frac{\partial u}{\partial t} = \lambda \frac{\partial^2 u}{\partial x^2}$ *with mixed BC* $u(x,t) = g_D(x,t)$ *on* $\Gamma_D \subset \partial\Omega$ *and* $\frac{\partial u}{\partial \mathbf{n}}(x,t) = g_R(u,x,t)$ *on* $\Gamma_R \subset \partial\Omega$.

Procedure 1: *The PINN algorithm for solving differential equations.*

1. Construct a neural network $\hat{u}(\mathbf{x}; \boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$ as a surrogate of the solution $u(\mathbf{x})$.
2. Specify the two sets of "residual points": $\mathcal{T}_f \subset \Omega$ and $\mathcal{T}_b \subset \partial\Omega$ for the equation and boundary/initial conditions.
3. Specify a loss function by summing the weighted L^2 norm of both the PDE equation and boundary condition residuals.
4. Train the neural network to find the best parameters $\boldsymbol{\theta}^*$ by minimizing the loss function $\mathcal{L}(\boldsymbol{\theta}; \mathcal{T})$.

To measure the discrepancy between the neural network $\hat{u}$ and the PDE constraints, we consider the loss function:

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{T}) = w_f \mathcal{L}_f(\boldsymbol{\theta}; \mathcal{T}_f) + w_b \mathcal{L}_b(\boldsymbol{\theta}; \mathcal{T}_b),$$

where

$$\mathcal{L}_f(\boldsymbol{\theta}; \mathcal{T}_f) = \frac{1}{|\mathcal{T}_f|} \sum_{\mathbf{x} \in \mathcal{T}_f} \left\| f\left(\mathbf{x}; \frac{\partial \hat{u}}{\partial x_1}, \dots; \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_1}, \dots; \dots; \boldsymbol{\lambda}\right) \right\|_2^2,$$

$$\mathcal{L}_b(\boldsymbol{\theta}; \mathcal{T}_b) = \frac{1}{|\mathcal{T}_b|} \sum_{\mathbf{x} \in \mathcal{T}_b} \|\mathcal{B}(\hat{u}, \mathbf{x})\|_2^2,$$

and w_f and w_b are the weights.

1.2 Approximation theory

Whether there exists a neural network that can simultaneously and uniformly approximate a function and its partial derivatives?
For $\mathbf{m} = (m_1, \dots, m_d) \in \mathbb{Z}_+^d$, we set $|\mathbf{m}| := m_1 + \dots + m_d$, and $D^{\mathbf{m}} := \frac{\partial^{|\mathbf{m}|}}{\partial x_1^{m_1} \dots \partial x_d^{m_d}}$.

Theorem 1 (Pinkus, 1999) *Let* $\mathbf{m}^i \in \mathbb{Z}_+^d$, $i = 1, \dots, s$, *and set* $m = \max_{i=1,\dots,s} |\mathbf{m}^i|$. *Assume* $\sigma \in C^m(\mathbb{R})$ *and* σ *is not a polynomial. Then the space of single hidden layer neural nets*

$$\mathcal{M}(\sigma) := span\{\sigma(\mathbf{w} \cdot \mathbf{x} + b) : \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}\}$$

is dense in $C^{\mathbf{m}^1,\dots,\mathbf{m}^s}(\mathbb{R}^d) := \cap_{i=1}^s C^{\mathbf{m}^i}(\mathbb{R}^d)$, *i.e., for any* $f \in C^{\mathbf{m}^1,\dots,\mathbf{m}^s}(\mathbb{R}^d)$, *any compact* $K \subset \mathbb{R}^d$, *and any* $\varepsilon > 0$, *there exists a* $g \in \mathcal{M}(\sigma)$ *satisfying* $\max_{\mathbf{x} \in K} |D^{\mathbf{k}} f(\mathbf{x}) - D^{\mathbf{k}} g(\mathbf{x})| < \varepsilon$, *for all* $\mathbf{k} \in \mathbb{Z}_+^d$ *for which* $\mathbf{k} \leq \mathbf{m}^i$ *for some* i.

1.3 Learning mode

Recent studies show that for function approximation, neural networks learn target functions from low to high frequencies, but we show that the learning mode of PINNs is different due to the existence of high-order derivatives.

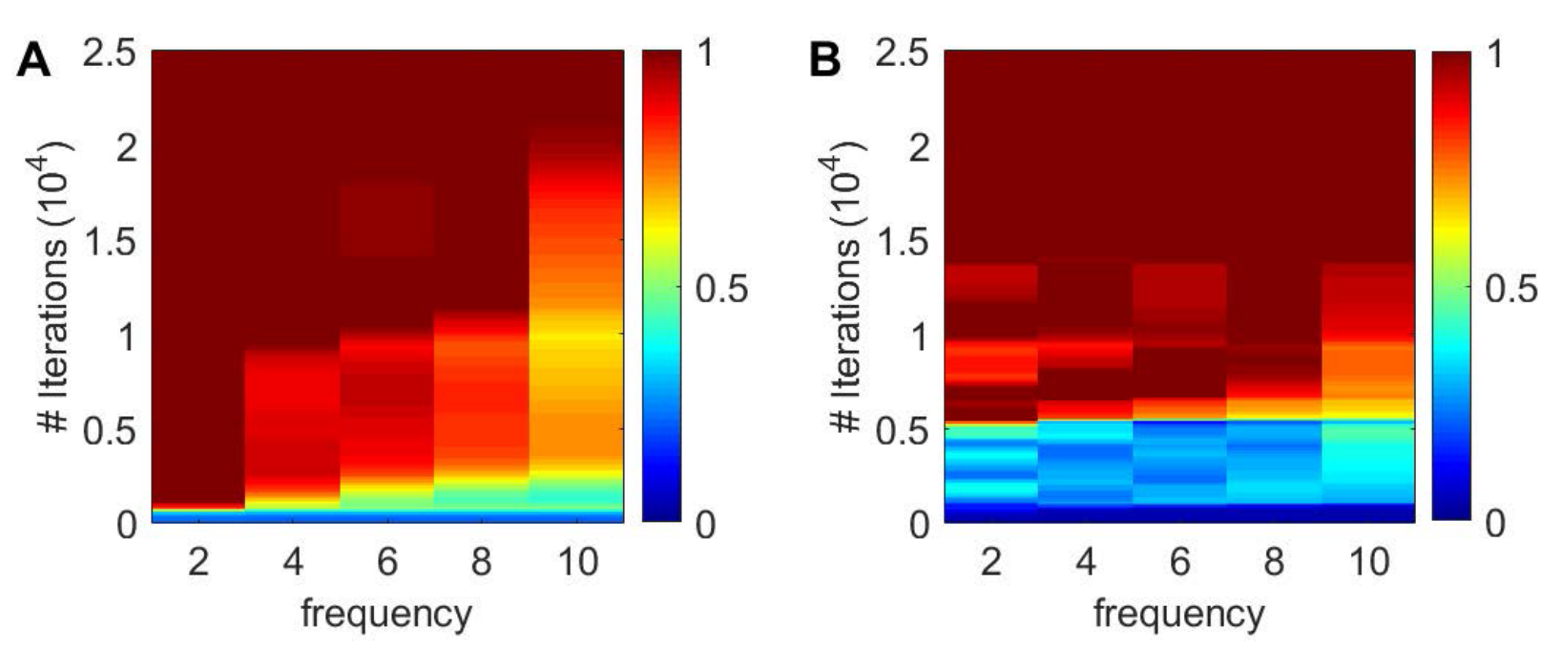

Figure 2: *Convergence of the amplitude for each frequency during the training process.* (***A***) *A neural network is trained to approximate the function* $f(x) = \sum_{k=1}^{5} \sin(2kx)/(2k)$. *The color represents amplitude values with the maximum amplitude for each frequency normalized to 1.* (***B***) *A PINN is used to solve the Poisson equation* $-f_{xx} = \sum_{k=1}^{5} 2k \sin(2kx)$ *with zero boundary conditions.*

1.4 Residual-based adaptive refinement (RAR)

The mean residual

$$\mathcal{E}_r = \frac{1}{V} \int_\Omega \left\| f\left(\mathbf{x}; \frac{\partial \hat{u}}{\partial x_1}, \dots; \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_1}, \dots; \dots; \boldsymbol{\lambda}\right) \right\| d\mathbf{x}$$

Procedure 2: *RAR for improving the distribution of residual points for training.*

1. Select the initial residual points $\mathcal{T}$, and train the neural network for a limited number of iterations.
2. Estimate the mean PDE residual $\mathcal{E}_r$ by Monte Carlo integration, i.e., by the average of values at a set of randomly sampled locations $\mathcal{S} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{|\mathcal{S}|}\}$:
$$\mathcal{E}_r \approx \frac{1}{|\mathcal{S}|} \sum_{\mathbf{x} \in \mathcal{S}} \left\| f\left(\mathbf{x}; \frac{\partial \hat{u}}{\partial x_1}, \dots; \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_1}, \dots; \dots; \boldsymbol{\lambda}\right) \right\|.$$
3. Stop if $\mathcal{E}_r < \mathcal{E}_0$. Otherwise, add m new points with the largest residuals in $\mathcal{S}$ to $\mathcal{T}$, and go to Step 2.

2. DeepXDE (https://github.com/lululxvi/deepxde)

2.1 Usage

Solving differential equations in DeepXDE is no more than specifying the problem using the build-in modules, including computational domain (geometry and time), PDE equations, BC/IC, constraints, training data, network architecture, and training hyperparameters.

Procedure 3: *Usage of DeepXDE for solving differential equations.*

1. Specify the computational domain using the **`geometry`** module.
2. Specify the PDE using the grammar of **`TensorFlow`**.
3. Specify the boundary and initial conditions.
4. Combine the geometry, PDE and boundary/initial conditions together into **`data.PDE`** or **`data.TimePDE`** for time-independent problems or for time-dependent problems, respectively.
   To specify training data, we can either set the specific point locations, or only set the number of points and then DeepXDE will sample the required number of points on a grid or randomly.
5. Construct a neural network using the **`maps`** module.
6. Define a **`Model`** by combining the PDE problem in Step 4 and the neural net in Step 5.
7. Call **`Model.compile`** to set the optimization hyperparameters, such as optimizer and learning rate. The weights in the loss can be set here by **`loss_weights`**.
8. Call **`Model.train`** to train the network from random initialization or a pre-trained model using the argument **`model_restore_path`**. It is extremely flexible to monitor and modify the training behavior using **`callbacks`**.
9. Call **`Model.predict`** to predict the PDE solution at different locations.

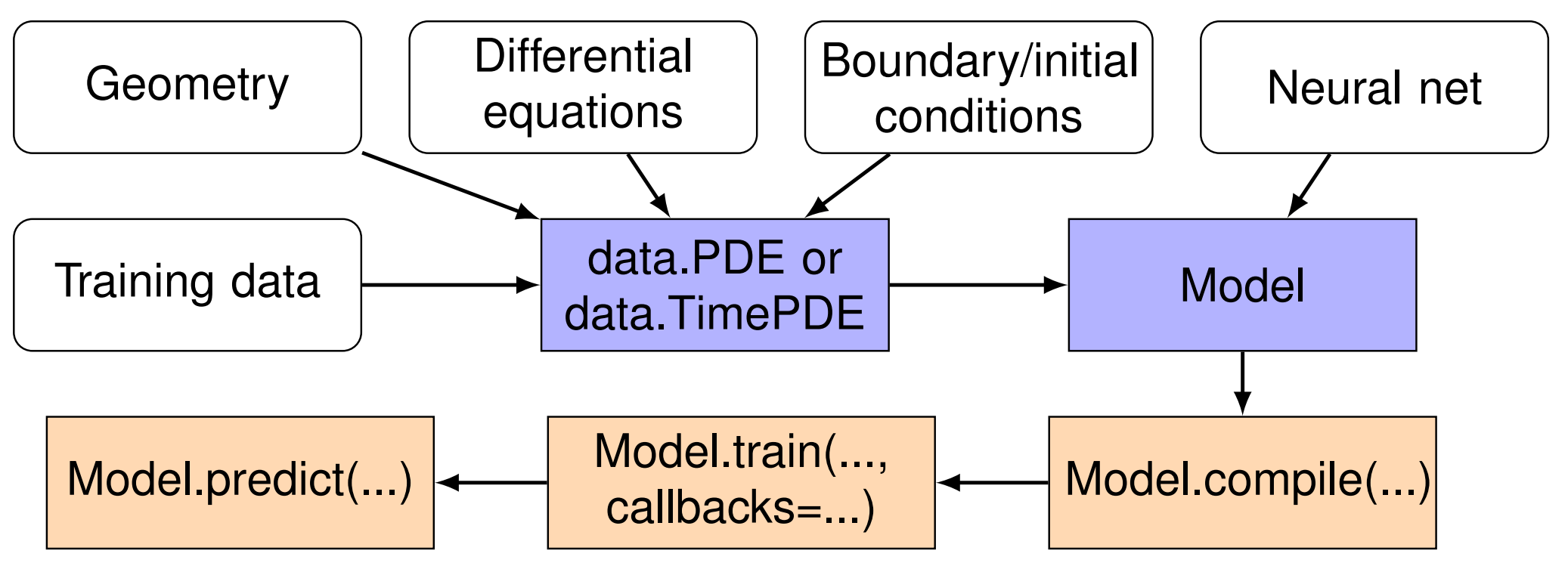

Figure 3: *Flowchart of DeepXDE. White boxes: the PDE problem and hyperparameters. Blue boxes combine white boxes. Orange boxes: the three steps to solve the PDE.*

Primitive geometries: **`interval`**, **`triangle`**, **`rectangle`**, **`polygon`**, **`disk`**, **`cuboid`**, **`sphere`**.

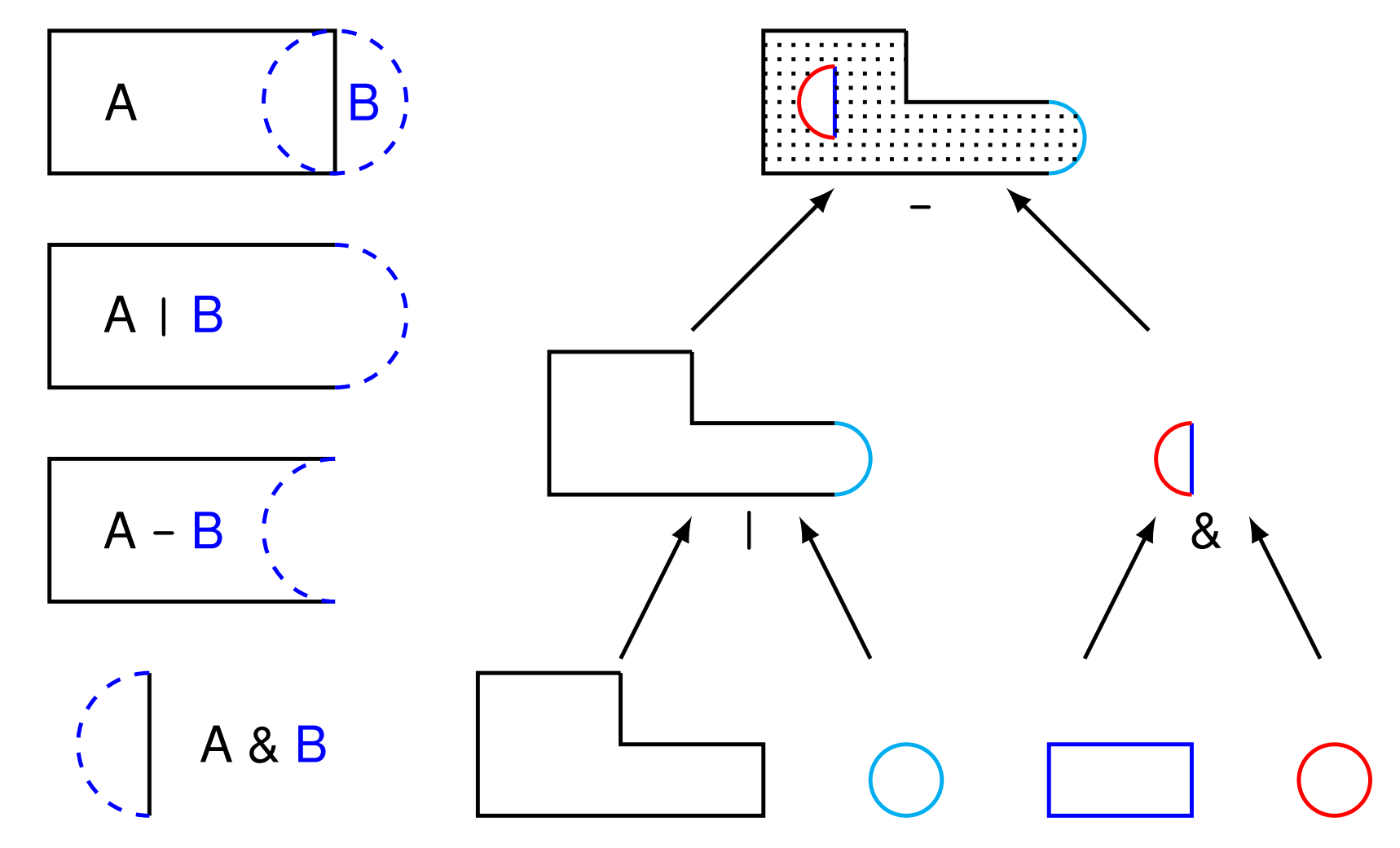

Figure 4: *CSG examples.* (***left***) *Union* $A|B$, *difference* $A - B$, *and intersection* $A\&B$. (***right***) *A complex geometry is constructed from primitive geometries.*

DeepXDE supports

1. Dirichlet/Neumann/Robin/periodic/general BC, & IC;
2. feed-forward network, and residual network.

2.2 Customizability

All the components are loosely coupled, and thus DeepXDE is well-structured and highly configurable.

3. Demonstration examples

3.1 Forward problems: Poisson equation

$$-\Delta u(x,y) = 1, \quad (x,y) \in \Omega, \quad u(x,y) = 0, \quad (x,y) \in \partial\Omega.$$

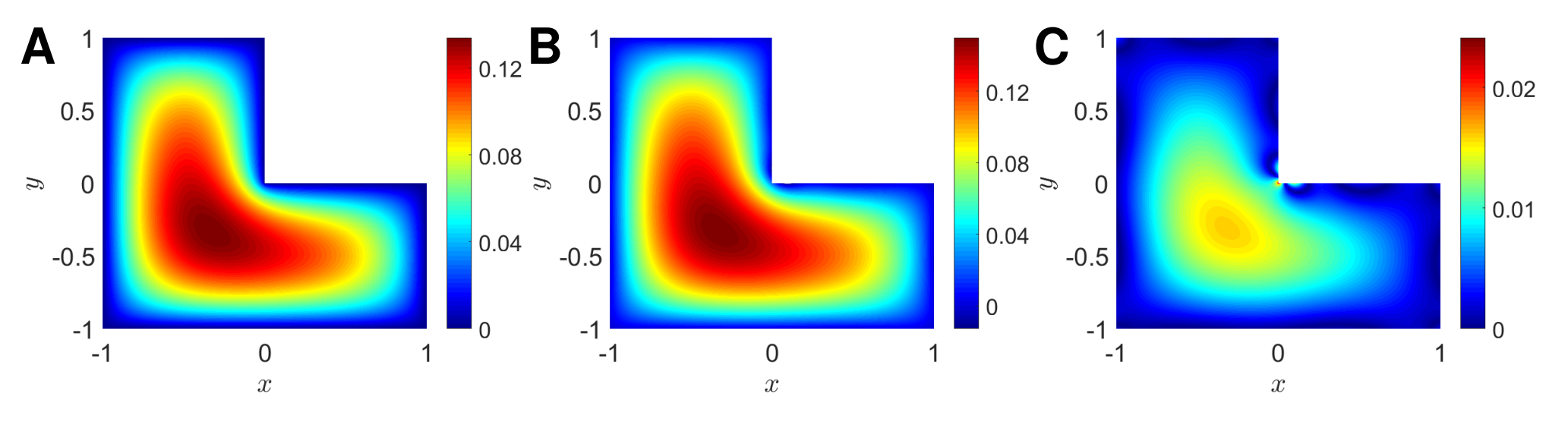

Figure 5: *Comparison of the PINN solution with the solution obtained by using spectral element method (SEM).* (***A***) *SEM solution,* (***B***) *PINN solution,* (***C***) *the absolute error.*

3.2 Inverse problems

The Lorenz system:

$$\frac{dx}{dt} = \rho(y - x), \quad \frac{dy}{dt} = x(\sigma - z) - y, \quad \frac{dz}{dt} = xy - \beta z.$$

A diffusion-reaction system on $x \in [0,1], t \in [0,10]$:

$$\frac{\partial C_A}{\partial t} = D \frac{\partial^2 C_A}{\partial x^2} - k_f C_A C_B^2, \quad \frac{\partial C_B}{\partial t} = D \frac{\partial^2 C_B}{\partial x^2} - 2k_f C_A C_B^2.$$

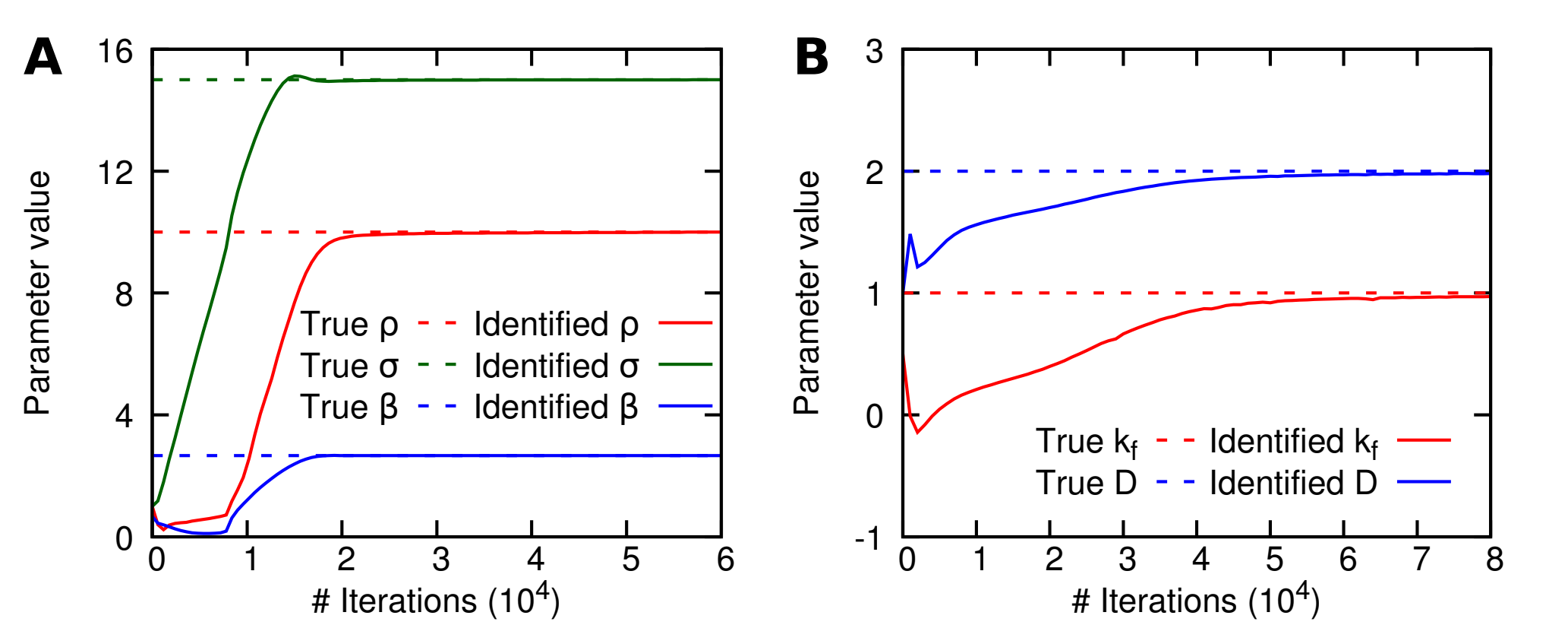

Figure 6: *Identified values of* (***A***) *the Lorenz system and* (***B***) *diffusion-reaction system converge to the true values.*

References

[1] L. Lu, X. Meng, Z. Mao, G. Karniadakis. DeepXDE: A deep learning library for solving differential equations. arXiv preprint arXiv:1907.04502 (2019).

[2] G. Pang*, L. Lu*, G. Karniadakis. fPINNs: Fractional physics-informed neural networks. *SIAM Journal on Scientific Computing* 41.4 (2019): A2603-A2626. (*Contributed equally)

[3] D. Zhang, L. Lu, L. Guo, G. Karniadakis. Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. *Journal of Computational Physics* 397 (2019): 108850.