LEARNING-GUIDED KANSA COLLOCATION FOR FORWARD AND INVERSE PDES BEYOND LINEARITY

Zheyuan Hu[1], **Weitao Chen**[2], **Cengiz Öztireli**[1], **Chenliang Zhou**[1*], **Fangcheng Zhong**[1*]
[1] Department of Computer Science and Technology,
[2] Department of Applied Mathematics and Theoretical Physics,
University of Cambridge, UK. *Co-corresponding authors.
{zh369, wc358}@cam.ac.uk, {chenliang.zhou, fangcheng.zhong}@cst.cam.ac.uk

ABSTRACT

Partial Differential Equations are precise in modelling the physical, biological and graphical phenomena. However, the numerical methods suffer from the curse of dimensionality, high computation costs and domain-specific discretization. We aim to explore pros and cons of different PDE solvers, and apply them to specific scientific simulation problems, including forwarding solution, inverse problems and equations discovery. In particular, we extend the recent Zhong et al. (2023) framework solver to coupled and non-linear settings, together with downstream applications. The outcomes include implementation of selected methods, self-tuning techniques, evaluation on benchmark problems and a comprehensive survey of neural PDE solvers and scientific simulation applications.

1 INTRODUCTION

PDEs are useful in different domains of scientific computing, including physics, graphics and biology. Zhong et al. (2023) proposed extension to Kansa method, which is a mesh-free Radial Basis Functions (RBFs) PDE solver. They introduced auto-tuning of the shape parameters of RBFs. However, their work focuses only on **single-variable linear PDEs**. Therefore, this paper extends CNFs backend solver to **multiple unknown functions** u **and nonlinear PDEs**, and apply the framework to specific scientific simulation problems, including forward computation and inverse problems.

It's unknown how (extended) Constrained Neural Fields (CNF) Zhong et al. (2023) compared with other classical and neural PDE solvers. Hence, we also implement and evaluate selected prior methods on the benchmarks on their **effectiveness** with different quality metrics (e.g. L1, L2, errors) against ground truth solutions, **efficiency**, computation resource, convergence speed, **method complexity**, and finally **utility in research**, i.e. their scientific simulation applications or integration with other methods, e.g. differentiable rendering to solve inverse physics-related problems in Graphics.

2 RELATED WORK

PDE benchmarks. We identified several representative equations Takamoto et al. (2022) in Table 12. They are different in linearity of the operator and solution u dimensionality.

PDE solvers. Numerical methods, e.g. Finite Difference Method (FDM) and Finite Element Method (FEM), are widely used to solve PDEs. However, they suffer from the curse of dimensionality, high computation costs and domain-specific discretization. Recently, neural network based solvers have shown promising results in addressing these issues. For example, Physics-Informed Neural Networks (PINNs) Raissi et al. (2019) and Fourier Neural Operators (FNOs) Li et al. (2020) have demonstrated the ability to generalize to unseen scenarios and handle high dimension effectively.

Inverse problem, i.e. estimating unknown parameters or inputs of a variable x from given solution observations u, is crucial. However, it's unclear how CNFs can be applied to these problems, including connecting with differentiable rendering pipelines Spielberg et al. (2023) in Visual Computing.

3 METHODOLOGY

3.1 GENERAL FORM OF PDES

With spatial domain $\Omega \subset \mathbb{R}^d$, where its dimension is d, and the unknown field $u(x,t) \in \mathcal{U} : \mathbb{D} \to \mathbb{R}$ defined on the spatio-temporal domain $\mathbb{D} = \Omega \times [t_0, t_f] \subset \mathbb{R}^{d+1}$, the general form of PDEs is,

$$\begin{cases} \mathcal{D}[u] = f, & x \in \Omega, t \in [t_0, t_f], \\ \mathcal{B}_i[u] = g_i, & x \in \partial\Omega_i, t \in [t_0, t_f]. \end{cases} \Leftrightarrow \begin{cases} \mathcal{D}[u](x,t) = f(x,t), & x \in \Omega, t \in [t_0, t_f], \\ \mathcal{B}_i[u](x,t) = g_i(x,t) & x \in \partial\Omega_i, t \in [t_0, t_f]. \end{cases} \tag{1}$$

where $\mathcal{D} : \mathcal{U} \to \mathcal{Y}$ is the differential operator and $f \in \mathcal{Y} : \mathbb{D} \to \mathbb{R}^m$ is source function, e.g. the external force in dynamics, with m being the output dimension of f[1]. The differential operators $\mathcal{D}$ include the gradient ∇, Laplace Δ, divergence $\nabla\cdot$, etc. For boundary conditions, $\mathcal{B}_i : \mathcal{U} \to \mathcal{Z}_i$ is each boundary operator with $g_i \in \mathcal{Z}_i : \partial\Omega_i \times [t_0, t_f] \to \mathbb{R}^{n_i}$ and n_i as the output dimension of g_i.

3.2 KANSA COLLOCATION

Kernel functions. Radial Basis Function (RBF) relates the distance r between the input $\mathbf{x}$ and a fixed origin point $\mathbf{c}$ to the output value.

$$\psi_{\mathbf{c}}(r) = \psi_{\mathbf{c}}(||\mathbf{x} - \mathbf{c}||). \tag{2}$$

There are various infinitely smooth RBFs, among which we choose Gaussian RBF for its effectiveness in approximating smooth functions,

$$\psi_{\mathbf{c}}(r) = \begin{cases} e^{-(\epsilon r)^2}, & \text{Gaussian}, \\ \frac{1}{1+(\epsilon r)^2}, & \text{Inverse quadratic}, \\ \sqrt{1+(\epsilon r)^2}, & \text{Multiquadrics}, \end{cases} \tag{3}$$

where Gaussian shape parameter $\epsilon = \frac{1}{\sqrt{2}\sigma}$, and σ is the standard deviation.

Kansa method Kansa (1990) approximates the solution $u(x,t)$ with a linear combination of kernel functions $\psi_k(||\mathbf{x}_i - \mathbf{x}_k||) \in \mathbb{R}^d \to \mathbb{R}$ centered at each collocation point $\{\mathbf{x}_i \in \mathbb{D}\}_{i=1}^N$,

$$u(x_i, t_i) \approx \hat{u}(\mathbf{x}_i) = \sum_{k=1}^{N} \alpha_k \cdot \psi_k(||\mathbf{x}_i - \mathbf{x}_k||), \quad \mathbf{x} \in \mathbb{D}, \tag{4}$$

where $\alpha_i \in \mathbb{R}$ are the coefficients to be solved. The time dimension t is omitted, which can be treated as an additional spatial dimension here. Equation equation 4 is expressed as, by rewriting the kernel functions into matrix form,

$$\underbrace{\begin{bmatrix} \hat{u}(\mathbf{x}_1) \\ \hat{u}(\mathbf{x}_2) \\ \vdots \\ \hat{u}(\mathbf{x}_N) \end{bmatrix}}_{\mathbf{u} \in \mathbb{R}^N} = \underbrace{\begin{bmatrix} \psi_1(||\mathbf{x}_1 - \mathbf{x}_1||) & \psi_2(||\mathbf{x}_1 - \mathbf{x}_2||) & \cdots & \psi_N(||\mathbf{x}_1 - \mathbf{x}_N||) \\ \psi_1(||\mathbf{x}_2 - \mathbf{x}_1||) & \psi_2(||\mathbf{x}_2 - \mathbf{x}_2||) & \cdots & \psi_N(||\mathbf{x}_2 - \mathbf{x}_N||) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_1(||\mathbf{x}_N - \mathbf{x}_1||) & \psi_2(||\mathbf{x}_N - \mathbf{x}_2||) & \cdots & \psi_N(||\mathbf{x}_N - \mathbf{x}_N||) \end{bmatrix}}_{\text{kernel matrix } \mathbf{K} \in \mathbb{R}^{N \times N}} \cdot \underbrace{\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{bmatrix}}_{\mathbf{a} \in \mathbb{R}^N}. \tag{5}$$

The PDE general form equation 1 can be summarized as a single equation,

$$\mathcal{F}[\hat{u}](\mathbf{x}_i) = h(\mathbf{x}_i), \quad \mathbf{x}_i \in \mathbb{D}, \tag{6}$$

where the operator $\mathcal{F} = \{\mathcal{D}, \mathcal{B}_i\}$ and $h = \{f, g_i\}$ represent both the initial and boundary conditions.

3.2.1 LINEAR OPERATOR CASE

By plugging in the approximation of u equation 4 and assuming the operator $\mathcal{F}$ is *linear*[2], the PDE equation 6 can be simplified as,

$$\mathcal{F}[\hat{u}](\mathbf{x}_i) = \mathcal{F}[\sum_{k=1}^{N} \alpha_k \cdot \psi_k](\mathbf{x}_i) = \sum_{k=1}^{N} \alpha_k \cdot \mathcal{F}[\psi_k](\mathbf{x}_i) = h(\mathbf{x}_i). \tag{7}$$

[1]Note that $\mathcal{U}$ and $\mathcal{Y}$ are two function spaces, and we require they are Banach spaces.

[2]For linear operators, $\mathcal{F}[\alpha \cdot \psi] = \alpha \cdot \mathcal{F}[\psi]$, as defined in § A.

By expanding in matrix form, the above equation is,

$$\underbrace{\begin{bmatrix} \mathcal{F}[\psi_1](\mathbf{x}_1) & \mathcal{F}[\psi_2](\mathbf{x}_1) & \cdots & \mathcal{F}[\psi_N](\mathbf{x}_1) \\ \mathcal{F}[\psi_1](\mathbf{x}_2) & \mathcal{F}[\psi_2](\mathbf{x}_2) & \cdots & \mathcal{F}[\psi_N](\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{F}[\psi_1](\mathbf{x}_N) & \mathcal{F}[\psi_2](\mathbf{x}_N) & \cdots & \mathcal{F}[\psi_N](\mathbf{x}_N) \end{bmatrix}}_{\text{operator-evaluated kernel matrix } \mathbf{F}\in\mathbb{R}^{N\times N}} \cdot \underbrace{\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{bmatrix}}_{\mathbf{a}\in\mathbb{R}^N} = \underbrace{\begin{bmatrix} h(\mathbf{x}_1) \\ h(\mathbf{x}_2) \\ \vdots \\ h(\mathbf{x}_N) \end{bmatrix}}_{\text{constraint values } \mathbf{h}\in\mathbb{R}^N} . \tag{8}$$

Concretely, when the kernel function is Gaussian RBF defined in equation 2, and $\mathbf{x}_i = (x_i, t_i)$,

$$\psi_k = e^{-\frac{r_k^2}{2\sigma^2}}, \quad r_k^2 = (x_k - x_i)^2 + (t_k - t_i)^2. \tag{9}$$

Take $\mathcal{F} = \frac{\partial}{\partial t}$ [3] and by the chain rule, the element $\mathbf{F}_{k,i}$ in the matrix equation 8 is thus,

$$\mathcal{F}[\psi_k](\mathbf{x}_i) = \frac{\partial \psi_k(\mathbf{x}_i)}{\partial t} = \frac{\partial \psi_k(\mathbf{x}_i)}{\partial r_k^2} \cdot \frac{\partial r_k^2}{\partial t} = -\frac{1}{2\sigma^2} e^{-\frac{r_k^2}{2\sigma^2}} \cdot 2(t_k - t_i) = -\frac{t_k - t_i}{\sigma^2} e^{-\frac{r_k^2}{2\sigma^2}}. \tag{10}$$

Simultaneous equations. When there are N_{eq} equations of different constraints to be satisfied, the collocation points $\{\mathbf{x}_i \in \mathbb{D}\}_{i=1}^{N_{\text{total}}}$ are distributed among all equations[4], where $N_{\text{total}} = \sum_{j=1}^{N_{eq}} N_j$.

$$\mathcal{F}_j[\hat{u}](\mathbf{x}_i) = h_j(\mathbf{x}_i), \quad \forall j \in \{1, \ldots, N_{eq}\}, \mathbf{x}_i \in \mathbb{D}. \tag{11}$$

Equation equation 8 can be extended by stacking each matrix $\mathbf{F}^{(j)} \in \mathbb{R}^{N_{\text{total}} \times N_{\text{total}}}$ and constraint vector $\mathbf{h}^{(j)} \in \mathbb{R}^{N_{\text{total}}}$ vertically for all equations Zhong et al. (2023). The block matrix form is,

$$\underbrace{\begin{bmatrix} \mathbf{F}^{(1)} \\ \vdots \\ \mathbf{F}^{(N_{eq})} \end{bmatrix}}_{\text{stacked } \mathbf{F}\in\mathbb{R}^{(N_{eq}\cdot N_{\text{total}})\times N_{\text{total}}}} \cdot \underbrace{\begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_{N_{\text{total}}} \end{bmatrix}}_{\mathbf{a}\in\mathbb{R}^{N_{\text{total}}}} = \underbrace{\begin{bmatrix} \mathbf{h}^{(1)} \\ \vdots \\ \mathbf{h}^{(N_{eq})} \end{bmatrix}}_{\text{stacked } \mathbf{h}\in\mathbb{R}^{N_{eq}\cdot N_{\text{total}}}} . \tag{12}$$

The solution of u equation 4 depends on the coefficients $\mathbf{a} = [\alpha_1, \alpha_2, \ldots, \alpha_N]$, which can be solved by the linear system $\mathbf{Fa} = \mathbf{h}$. The general form is given by the least squares approximation, i.e. minimizing the norm of the error vector and setting the gradient to zero,

$$\begin{aligned} \mathbf{a}^{\text{opt}} &= \min_{\mathbf{a}} (||\mathbf{Fa} - \mathbf{h}||)^2, \\ \nabla_{\mathbf{a}} (\mathbf{Fa} - \mathbf{h})^T (\mathbf{Fa} - \mathbf{h}) = 0 &\implies (\mathbf{F}^T\mathbf{F})\mathbf{a}^{\text{opt}} = \mathbf{F}^T\mathbf{h}. \end{aligned} \tag{13}$$

If matrix $\mathbf{F}$ is full rank, $\mathbf{F}^T\mathbf{F}$ is invertible, thus one can derive $\mathbf{a}^{\text{opt}} = (\mathbf{F}^T\mathbf{F})^{-1}\mathbf{F}^T\mathbf{h}$. Should the matrix $\mathbf{F}$ be square and invertible, equation 13 can be further simplified as $\mathbf{a}^{\text{opt}} = \mathbf{F}^{-1}\mathbf{h}$. Whichever conditions occurs, the final solution for u is approximated by plugging in the optimal coefficients $\mathbf{a}^{\text{opt}}$ into equation 5, i.e. $\hat{u}(\mathbf{x}) = \mathbf{K} \cdot \mathbf{a}^{\text{opt}}$.

When testing on unseen data points $\{\mathbf{x}_j^\star \in \mathbb{D}\}_{j=1}^M$, the kernel functions are constructed between the test points and the collocation points $\{\mathbf{x}_i \in \mathbb{D}\}_{i=1}^N$. The solution is thus,

$$u(x,t) \approx \hat{u}(\mathbf{x}^\star) = \sum_{k=1}^{N} \alpha_k \cdot \psi_k(||\mathbf{x}^\star - \mathbf{x}_k||), \quad \mathbf{x} \in \mathbb{D}, \tag{14}$$

Test-time solution equation 14 can be formulated to matrix form,

$$\underbrace{\begin{bmatrix} \hat{u}(\mathbf{x}_1^\star) \\ \hat{u}(\mathbf{x}_2^\star) \\ \vdots \\ \hat{u}(\mathbf{x}_M^\star) \end{bmatrix}}_{\mathbf{u}\in\mathbb{R}^M} = \underbrace{\begin{bmatrix} \psi_1(||\mathbf{x}_1^\star - \mathbf{x}_1||) & \psi_2(||\mathbf{x}_1^\star - \mathbf{x}_2||) & \cdots & \psi_N(||\mathbf{x}_1^\star - \mathbf{x}_N||) \\ \psi_1(||\mathbf{x}_2^\star - \mathbf{x}_1||) & \psi_2(||\mathbf{x}_2^\star - \mathbf{x}_2||) & \cdots & \psi_N(||\mathbf{x}_2^\star - \mathbf{x}_N||) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_1(||\mathbf{x}_M^\star - \mathbf{x}_1||) & \psi_2(||\mathbf{x}_M^\star - \mathbf{x}_2||) & \cdots & \psi_N(||\mathbf{x}_M^\star - \mathbf{x}_N||) \end{bmatrix}}_{\text{kernel matrix } \mathbf{K}^\star\in\mathbb{R}^{M\times N}} \cdot \underbrace{\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{bmatrix}}_{\mathbf{a}\in\mathbb{R}^N} . \tag{15}$$

[3] $\frac{\partial \psi_k}{\partial t}$: `phi_t = torch.autograd.grad(phi, t, create_graph=True)[0]`

[4] Note that repeated collocation points are forced to be repeated here for distinct constraints.

3.2.2 EXTENSION 1: COUPLED SOLUTION FIELDS OF PDES

Coupled multi-dimensional PDE solution fields. Assuming there are N_D solution dimensions, i.e. $\mathbf{u} = [u_1, u_2, \ldots, u_{N_D}]$, the Kansa approximation equation 4 for each dimension is,

$$\hat{u}_d(\mathbf{x}) = \sum_{k=1}^{N} \alpha_k^{(d)} \cdot \psi_k^{(d)}(||\mathbf{x} - \mathbf{x}_k||), \quad \forall d \in \{1, \ldots, N_D\}. \tag{16}$$

The coupled PDE equation is thus formulated as applying the coupling, or governing operator $\mathcal{G}$ on all dimensions of solution, which each has its own operator $\mathcal{F}^{(d)}$,

$$\mathcal{G}\left(\mathcal{F}^{(1)}[\hat{u}_1], \ldots, \mathcal{F}^{(N_D)}[\hat{u}_{N_D}]\right)(\mathbf{x}_i) = h(\mathbf{x}_i), \quad \mathbf{x}_i \in \mathbb{D}. \tag{17}$$

Here we assume the coupling operator $\mathcal{G}$ is linear with each dimension of solution,

$$\mathcal{G}\left(\hat{v}_1, \ldots, \hat{v}_{N_D}\right)(\mathbf{x}) = \sum_{d=1}^{N_D} \beta_d \cdot \hat{v}_d(\mathbf{x}), \tag{18}$$

where $\beta_d \in \mathbb{R}$ is the per-dimension weight. Equation equation 8 can be extended by stacking each matrix $\mathbf{F}^{(d)} \in \mathbb{R}^{N \times N}$ horizontally for all dimensions of solution. The block matrix form is,

$$\underbrace{\begin{bmatrix}\beta_1 \mathbf{I}_N & \cdots & \beta_{N_D}\mathbf{I}_N\end{bmatrix}}_{\boldsymbol{\beta} \in \mathbb{R}^{N \times (N_D \cdot N)}} \circ \underbrace{\begin{bmatrix}\mathbf{F}^{(1)} & \cdots & \mathbf{F}^{(N_D)}\end{bmatrix}}_{\text{coupling } \mathbf{F} \in \mathbb{R}^{N \times (N_D \cdot N)}} \cdot \underbrace{\begin{bmatrix}\mathbf{a}^{(1)} \\ \vdots \\ \mathbf{a}^{(N_D)}\end{bmatrix}}_{\mathbf{a} \in \mathbb{R}^{(N_D \cdot N)}} = \underbrace{\begin{bmatrix}h(\mathbf{x}_1) \\ \vdots \\ h(\mathbf{x}_N)\end{bmatrix}}_{\text{stacked } \mathbf{h} \in \mathbb{R}^{N}}, \tag{19}$$

where $\circ$ is element-wise or Hadamard product and $\mathbf{I}_N$ is the identity matrix of size N. For simultaneous coupled PDE equations, similar to equation 11, they are indexed by $j \in \{1, \ldots, N_{eq}\}$,

$$\mathcal{G}_j\left(\mathcal{F}_j^{(1)}[\hat{u}_1], \ldots, \mathcal{F}_j^{(N_D)}[\hat{u}_{N_D}]\right)(\mathbf{x}_i) = h_j(\mathbf{x}_i), \quad \forall j \in \{1, \ldots, N_{eq}\}, \mathbf{x}_i \in \mathbb{D}. \tag{20}$$

With N_{total} defined as in equation 12, the block matrix form is,

$$\underbrace{\begin{bmatrix}\boldsymbol{\beta}^{(1)} \\ \vdots \\ \boldsymbol{\beta}^{(N_{eq})}\end{bmatrix}}_{\boldsymbol{\beta} \in \mathbb{R}^{(N_{eq} \cdot N_{\text{total}}) \times (N_D \cdot N_{\text{total}})}} \circ \underbrace{\begin{bmatrix}\mathbf{F}^{(1,1)} & \cdots & \mathbf{F}^{(1,N_D)} \\ \vdots & \mathbf{F}^{(j,d)} & \vdots \\ \mathbf{F}^{(N_{eq},1)} & \cdots & \mathbf{F}^{(N_{eq},N_D)}\end{bmatrix}}_{\text{coupling } \mathbf{F} \in \mathbb{R}^{(N_{eq} \cdot N_{\text{total}}) \times (N_D \cdot N_{\text{total}})}} \cdot \underbrace{\begin{bmatrix}\mathbf{a}^{(1)} \\ \vdots \\ \mathbf{a}^{(N_D)}\end{bmatrix}}_{\mathbf{a} \in \mathbb{R}^{(N_D \cdot N_{\text{total}})}} = \underbrace{\begin{bmatrix}\mathbf{h}^{(1)} \\ \vdots \\ \mathbf{h}^{(N_{eq})}\end{bmatrix}}_{\text{stacked } \mathbf{h} \in \mathbb{R}^{(N_{eq} \cdot N_{\text{total}})}}. \tag{21}$$

3.2.3 EXTENSION 2: NONLINEAR OPERATOR CASE

When the operator $\mathcal{F}$ is *nonlinear*, we can no longer simplify equation 6 as in equation 7. However, we can still derive the relation between the solution u and its linear transformed version as below.

Differentiable matrix helps decompose the general non-linear operator $\mathcal{F}$ into a series of linear operators. Take any linear operator, e.g. $\frac{\partial}{\partial x}$, it relates the relation between unknown $\mathbf{u}$ and its derivative $\mathbf{u}' = \mathbf{D}_x \cdot \mathbf{u}$. We derive $\mathbf{D}_x$ from Kansa equation 4, by linearity and equation 7,

$$\frac{\partial}{\partial x} u(x) = \sum_{k=1}^{N} \alpha_k \cdot \frac{\partial}{\partial x} \psi_k(||x - x_k||). \tag{22}$$

In matrix form, we have $\mathbf{u}' = \mathbf{K}_\mathbf{x} \cdot \mathbf{a}$, where the matrix $\mathbf{K}_\mathbf{x} \in \mathbb{R}^{N \times N}$ is constructed by evaluating $\frac{\partial}{\partial x}\psi_k(||x - x_k||)|_{x=x_i}$ for all $i, k \in \{1, \ldots, N\}$ as row and column indices.

By inverting equation 5, $\mathbf{a} = \mathbf{K}^{-1} \cdot \mathbf{u}$, assuming $\mathbf{K}$ invertibility from independent basis. By substituting $\mathbf{a}$ into $\mathbf{u}' = \mathbf{K}_\mathbf{x} \cdot \mathbf{a}$, one gets $\mathbf{u}' = \mathbf{K}_\mathbf{x} \cdot \mathbf{K}^{-1} \cdot \mathbf{u}$. The differentiable matrix is thus,

$$\mathbf{D}_x = \mathbf{K}_\mathbf{x} \cdot \mathbf{K}^{-1} \in \mathbb{R}^{N \times N}. \tag{23}$$

For viscous Burgers' equation equation 63 $\mathcal{F}[u] = \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2}$. Its differentiable matrix form, which follows the same formulation as in equation 23 by replacing the operator accordingly, is,

$$\mathcal{F}[u] = \mathbf{D}_t \cdot \mathbf{u} + \mathbf{u} \circ (\mathbf{D}_x \cdot \mathbf{u}) - \nu(\mathbf{D}_{xx} \cdot \mathbf{u}). \tag{24}$$

Here we present **two** categories of Kansa approaches (Table 1). The first consists of **four** time-stepping schemes, including two per-step *linear* and another two *nonlinear* systems. The second employs **a fully nonlinear solver** on the PDE residuals, without explicit time discretization.

Table 1: Summary of different non-linear Kansa solver features, Δt is the time step size, N_x and N_t are the number of collocation points in spatial and temporal dimensions respectively.

Features	forward	IMEX	backward	Crank–Nicolson	**fully non-linear**
Time-step	explicit	semi-explicit	implicit	implicit	$\times$
Error	$O(\Delta t)$	$O(\Delta t)$	$O(\Delta t)$	$O(\Delta t^2)$	$O(1)$
Stability	unstable	stable	stable	stable	N/A
Memory	$O(N_x^2)$	$O(N_x^2)$	$O(N_x^2)$	$O(N_x^2)$	$O(N_x^2 N_t^2)$

Time-stepping approach with linear system. We can remove the non-linearity by discretizing the time derivative via finite difference method, for a special case of time-dependent PDEs. One solution is to use the (1) *explicit forward* Euler scheme,

$$\frac{\partial u}{\partial t} + \mathcal{D}[u] = 0 \implies \frac{u^{n+1} - u^n}{\Delta t} + O(\Delta t) + \mathcal{D}[u^n] = 0, \tag{25}$$

where $\mathcal{D}$ is the spatial operator. A more *stable* solution is to use the (2) *implicit-explicit (IMEX)* scheme, which splits the stiff and non-stiff parts of the operator $\mathcal{D} = \mathcal{I}_{\text{stiff}} + \mathcal{E}_{\text{non-stiff}}$, which stiffness means the numerical instability incurred by the operator and needs to be treated implicitly,

$$\frac{u^{n+1} - u^n}{\Delta t} + O(\Delta t) + \mathcal{I}_{\text{stiff}}[u^{n+1}] + \mathcal{E}_{\text{non-stiff}}[u^n] = 0. \tag{26}$$

Despite the non-linear spatial operator $\mathcal{D}$ or $\mathcal{E}_{\text{non-stiff}}$, we already know the solution u^n at time step n. Thus, with differentiable matrices, one can evaluate $\mathcal{D}[u^n]$ or $\mathcal{E}_{\text{non-stiff}}[u^n]$ directly, so as to derive the solution u^{n+1} at the next time step $n+1$.

Time-stepping approach with nonlinear solver. If we discretize the time derivative via the (3) *backward* Euler scheme, the non-linearity remains in the formulation,

$$\frac{\partial u}{\partial t} + \mathcal{D}[u] = 0 \implies \frac{u^{n+1} - u^n}{\Delta t} + O(\Delta t) + \mathcal{D}[u^{n+1}] = 0. \tag{27}$$

We can directly replace linear system solver by a non-linear system solver, e.g. Newton-Raphson method Ypma (1995), to minimize the residual vector and derive the unknown solution at next time step,

$$u^{n+1} = \arg\min_{u^{n+1}} \mathbf{r}^{n+1}, \text{ where } \mathbf{r}^{n+1} = u^{n+1} - u^n + \Delta t \cdot \mathcal{D}[u^{n+1}]. \tag{28}$$

Alternatively, (4) *Crank-Nicolson* scheme can be used to discretize second-order accurate in time,

$$\frac{\partial u}{\partial t} + \mathcal{D}[u] = 0 \implies \frac{u^{n+1} - u^n}{\Delta t} + \frac{1}{2}\left(\mathcal{D}[u^{n+1}] + \mathcal{D}[u^n]\right) + O(\Delta t^2) = 0. \tag{29}$$

Similar with equation 28, the unknown solution at next time step is derived by minimizing the residual vector as stated in equation 29.

Fully nonlinear solver without time-stepping. This approach directly minimizes the PDE residuals equation 6 over all collocation points, without explicit time discretization. After plugging in the differentiable matrix form of the non-linear operator $\mathcal{F}$, the objective function is therefore,

$$\alpha = \arg\min_{\alpha} \sum_{i=1}^{N} \left(\mathcal{F}[\hat{u}](\mathbf{x}_i) - h(\mathbf{x}_i)\right)^2. \tag{30}$$

By plugging in Kansa approximation equation 4, we derive unknown solution u over entire domain.

3.2.4 AUTO-TUNING OF KANSA HYPERPARAMETERS

To tune the key Kansa method hyperparameter, kernel shape parameter ϵ in equation 3, Zhong et al. (2023) proposed one of the self-tuning methods for ϵ by minimizing the variation of the solution field u over all collocation points, and the condition number of operator-evaluated kernel matrix $\mathbf{F}$,

$$\epsilon^\star = \arg\min_{\epsilon} \omega_1 \cdot \text{cond}(\mathbf{F}) + \omega_2 \cdot \int_{\mathbb{D}} ||\nabla u(\mathbf{x})||^2 d\mathbf{x}, \tag{31}$$

where $\text{cond}(\mathbf{F})$ is the condition number of matrix $\mathbf{F}$ defined in equation 8. The integral term can be approximated by summing over all collocation points by Monte Carlo integration. This approach works for linear, including *coupled* and multi-dimensional, PDEs.

For **non-linear** operator case, the solution u depends on ϵ implicitly via the coefficients α_i. The matrix $\mathbf{F}$ no longer exists explicitly. Here, we propose to directly minimize the PDE residuals over all collocation points, the total variation of the solution field u, and the training L2 loss between the predicted solution u and the ground truth solution u^{gt} if training data are available,

$$\epsilon^\star = \arg\min_{\epsilon} \omega_1 \cdot \sum_{i=1}^{N} \left(\mathcal{F}[\hat{u}](\mathbf{x}_i) - h(\mathbf{x}_i)\right)^2 + \omega_2 \cdot \int_{\mathbb{D}} ||\nabla u(\mathbf{x})||^2 d\mathbf{x} + \omega_3 \cdot ||u - u^{gt}||^2, \tag{32}$$

where ω_1, ω_2, and ω_3 are the penalty weights. Grid search is used as an optimizer.

3.3 SOLUTIONS OF INVERSE PDE PROBLEMS

Inverse PDE problems. When given observations of solution field u^{obs}, we infer the unknown PDE parameters $\boldsymbol{\pi}$ that minimize the discrepancy $\mathcal{L}$ between the predicted $u^{\text{pred}}(\boldsymbol{\pi})$ and u^{obs},

$$\boldsymbol{\pi}^\star = \arg\min_{\boldsymbol{\pi}} \mathcal{L}(u^{\text{obs}}, u^{\text{pred}}(\boldsymbol{\pi})). \tag{33}$$

We adopt the `SciPy` implementation of the least squares and root finding algorithms, which are either gradient-based or gradient-free, detailed in the evaluation section.

4 EVALUATION

4.1 PERFORMANCE METRICS

Accuracy. Given the numerical solution $\hat{u}_i$ from PDE solvers, and the ground truth u_i, the L_2 risk $\mathcal{R}_{L_2}$ is the average discretized error over all N_{test} test points on the spatial-temporal domain,

$$\hat{\mathcal{R}}_{L_2} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} ||\hat{u}_i - u_i||_2, \quad \hat{\mathcal{R}}_{\text{relative } L_2} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \frac{||\hat{u}_i - u_i||_2}{||u_i||_2}. \tag{34}$$

The relative L_2 risk is computed from $\mathcal{R}_{L_2}$ and normalized by the ground truth u_i L_2 norm,

4.2 EVALUATION OF SOLVERS FOR THE ADVECTION EQUATION

For the 1D **advection** equation defined in equation 45, we set the number of domain quadrature points $N_{\mathcal{R}} = 100 \times 10$, i.e. initial condition (IC) points $N_d = 10$ and the boundary condition (BC) points $N_{\mathcal{B}} = 100 \times 2$. The advection equation is initialized as per Table 2.

Table 2: 1D advection equation experimental setup.

domain	time range	parameter	IC	BC
$x_0 = 0, x_f = 1$	$t_0 = 0, t_f = 1$	$\beta = 0.4$	$u_0(x) = \sin(2\pi x)$	per equation 46

FNO requires multiple instances of PDEs for training. Hence, we generate $N_{pde} = 100$ instances by varying only the initial condition as, given $c_k \sim \mathcal{N}(0, 1)$,

$$u_0(x) := \frac{u_0(x)}{\max_x |u_0(x)|}, \text{ where } u_0(x) = \sum_{k=1}^{5} c_k \sin(2\pi k x). \tag{35}$$

For training, PINN and FNO are trained via learning rate $\eta = 10^{-3}$ until convergence, i.e. with epoch iterations $N_{\text{iter}} = 3000$ for PINN and $N_{\text{iter}} = 100$ for FNO. For evaluation, the test points $N_{\text{test}} = 64 \times 8$. The error is measured by relative L_2 risk $\hat{\mathcal{R}}_{\text{relative } L_2}$ equation 34.

4.2.1 FORWARD PROBLEM

Since FNO is trained on $N_{pde} = 100$ instances of PDEs, we compensate more training data for single-instance solvers for a fair comparison. The adjustment factor is defined as $C_{\text{scale}} \in [1, N_{pde}] \subset \mathbb{R}^+$. Hence, the domain points is $N'_{\mathcal{R}} = C_{\text{scale}} \times N_{\mathcal{R}}$ and methods denoted as FDM$^{C_{\text{scale}}}$ and PINN$^{C_{\text{scale}}}$. The test-time results are summarized in Table 3.

Table 3: Models accuracy $\hat{\mathcal{R}}_{\text{relative } L_2} \times 10^{-3}$, on 1D advection relative to the data domain resolution.

C_{scale}	**FDM**	**PINN**	**FNO**	**KM**
1	36.63	300.2	744.3	1.918
2^2	17.05	20.68	58.71	0.0028
4^2	7.478	8.654	37.68	N/A
$N_{pde} = 10^2$	3.228	6.457	13.37	N/A
average	16.10 ± 12.87	83.99 ± 124.9	213.5 ± 306.9	0.9603 ± 0.9576

From Table 3, we conclude that all solvers are sensitive to the number of training data, where larger C_{scale} leads to better precision on test points. Kansa outperforms other methods in both accuracy and convergence speed, achieving the least error (up to 10^{-6}) with only $C_{\text{scale}} = 4^2$. However, due to the increasing computational cost above $C_{\text{scale}} = 10^2$, memory limit was exceeded.

4.2.2 INVERSE PROBLEM

For the 1D **advection** equation equation 45 initialized in Table 2, we set up the inverse PDE problem to infer the initial parameter β from the observation data u^{obs} at all time steps. All methods are evaluated at their best performance from the forward problem. The results are summarized in Table 4, with the initial parameter β_0 set.

Table 4: Inverse predictions of β on advection equation, where the ground truth $\beta = 0.4$.

β_0	**FDM**	**PINN**	**FNO**	**KM**	β_0	**FDM**	**PINN**	**FNO**	**KM**
0.2	0.402	0.39987	0.3985	0.402	1.0	1.000	0.40446	1.2267	0.402

For local optimization methods when searching for the optimal parameter, they stuck at different local minima depending on the initial guess β_0. With different runs of initial guesses, they give more precise predictions with more computational cost.

4.3 EXTENSION 1: KANSA METHOD FOR COUPLED PDES

The Lotka-Volterra equations equation 47 are initialized as per Table 5, where the number of domain quadrature points $N_{\mathcal{R}} = 100 \times 1$, and initial condition points $N_d = 1$. For evaluation, the test points $N_{\text{test}} = 64$. The results from Kansa method are summarized in Table 6, where the Gaussian RBF shape parameters, as defined in equation 3, are set as $\epsilon = 0.2$ for both $x(t)$ and $y(t)$.

Table 5: 1D Lotka-Volterra equations experimental setup.

time range	**parameter**	**initial conditions**
$t_0 = 0, t_f = 200$	$\alpha = 0.1, \beta = 0.02, \delta = 0.01, \gamma = 0.1$	$x(0) = 40, y(0) = 9$

The 1D **Maxwell's** equations as defined in equation 58 are initialized per Table 14, where the speed of time propagation $c = 1$, the number of domain quadrature points $N_{\mathcal{R}} = 12 \times 12$, and initial condition points $N_d = 24$. For evaluation, the test points $N_{\text{test}} = 10 \times 10$. The shape parameter of Gaussian RBF, as defined in equation 3, is set as $\epsilon_x = 0.21$ and $\epsilon_y = 0.2$ for Lotka-Volterra equations and $\epsilon_E = 16$ and $\epsilon_B = 16$ for Maxwell's equations, respectively.

4.3.1 FORWARD PROBLEM

Table 6: $\hat{\mathcal{R}}_{\text{relative } L_2}$ error of Lotka-Volterra and Maxwell's equations using Kansa method.

C_{scale}	$x(t)$	$y(t)$	$E_z(x,t)$	$B_y(x,t)$
1	0.1279353	0.055667494	0.8049189	0.5894967
4	0.04539858	0.06230465	0.4383743	0.3830594

Accuracy. The results from Kansa method are summarized in Table 6. Both errors converge with increasing C_{scale} as defined above. **Efficiency**. The training time and inference time of Kansa method on Lotka-Volterra equations are 0.4034 and 0.0001 seconds, respectively. The training time and inference time of Kansa method on Maxwell's equations are 0.4486 and 0.0005 seconds.

4.3.2 INVERSE PROBLEM

For the Lotka-Volterra defined in equation 47 initialized in Table 5, we set up the inverse problem to infer the initial parameter α, β, δ and γ from observation $x^{\text{obs}}(t)$ and $y^{\text{obs}}(t)$ at all time steps.

Table 7: Inverse predictions of α, β, δ and γ on Lotka-Volterra equations.

	α	β	δ	γ		α	β	δ	γ
reference	0.1	0.02	0.01	0.1	prediction	0.102	0.0207	0.0100	0.0994

With the initial guess all set to 1, the results are summarized in Table 7. Despite the four-dimensional search space, the optimization algorithm `SciPy` Powell method successfully infers the parameters with high accuracy and decent computational cost.

4.4 EXTENSION 2: KANSA METHOD FOR NONLINEAR PDES

The **Burgers'** equation defined in equation 63 is initialized as per Table 8, where the number of domain quadrature points $N_{\mathcal{R}} = 64 \times 16$, i.e. initial condition (IC) points $N_d = 64$ and the boundary condition (BC) points $N_{\mathcal{B}} = 16 \times 2$. For evaluation, the test points $N_{\text{test}} = 48 \times 12$. The Gaussian RBF shape parameter, as defined in equation 3, is set as $\epsilon = 0.9$.

Table 8: Burgers' equation experimental setup.

domain	**time span**	**param.**	**ICs**	**BCs**
$x_0 = -10, x_f = 10$	$t_0 = 0, t_f = 4$	$\nu = 0.5$	per equation 74	$u(x_0) = 1, u(x_f) = 0$

4.4.1 FORWARD PROBLEM

Accuracy. From Table 9, we observe that fully non-linear approach outperforms other time-stepping schemes. It's hard to determine whether IMEX or backward Euler is more accurate theoretically. However, Crank-Nicolson scheme is definitely more accurate than both IMEX and backward Euler, since it's second-order accurate in time while the other two are only first-order accurate.

Table 9: $\hat{\mathcal{R}}_{\text{relative } L_2} \times 10^{-2}$ error of Burgers' equation using Kansa methods.

forward	IMEX	backward	Crank–Nicolson	**fully non-linear**
3.74×10^{31}	1.68	1.33	1.29	0.012

Computational efficiency. We measure the training and inference time of different Kansa methods on Burgers' equation in Table 10. The non-linear solver used is the `SciPy` least-squares.

Table 10: Train or infer time of Burgers' equation using Kansa methods (in seconds).

	forward	IMEX	backward	Crank–Nicolson	*fully non-linear*
Training	0.34	0.47	2.33	1.66	99.2
Inference	0.436	0.272	1.053	1.109	0.005

Training time for fully nonlinear approach is longer because each step involves heavier computation with substantial memory (Table 1), further compounded by nonlinear solvers. Four time-stepping schemes have much less training time. The forward Euler is unstable when the stability condition is not satisfied. **Inference** time of fully non-linear approach is significantly reduced, due to the reuse of coefficient from the training phase. Despite a full test-time recomputation from scratch, the inference time of four time-stepping schemes remains acceptable for most practical applications.

4.4.2 Inverse problem

For the Burgers' equation defined in equation 63 initialized in Table 8, we set up the inverse PDE problem to infer the initial parameter ν from the observation data u^{obs} at all time steps. The results are summarized in Table 11, with the initial parameter ν_0 set as 0.1.

Table 11: Inverse predictions of ν on Burgers' equation, where the ground truth $\nu = 0.5$.

forward	IMEX	backward	**Crank–Nicolson**	***fully non-linear***
0.388	0.535	0.467	0.502	0.500

Accuracy. Under same optimizer and initial guess, the Crank-Nicolson scheme confirms its theoretical advantage (Table 1) over both IMEX and backward Euler, which stuck at local minima. **Computational efficiency**. Fully non-linear approach requires retraining for each new parameter, which is computationally expensive (Table 10). To speed up the per-run training time, it is trained with a maximum iteration. **Stability**. The forward Euler scheme is unstable when given large Δt.

5 Conclusions

This paper extends Zhong et al. (2023) RBF framework solver beyond original scope of linear PDEs. In particular, we generalize its PDE solver to handle **coupled** and **nonlinear** PDEs, addressing the loss property of linear reordering. These broaden the applicability of CNF-driven self-tuning (Appendix 3.2.4) mesh-free solvers to both forward modeling and inverse problem formulations.

In addition, this work contributes a systematic empirical study of how CNF solvers compare with established classical and neural PDE solvers. By implementing representative prior methods and evaluating them across benchmark problems, we assess their relative performance in terms of solution accuracy, efficiency, convergence and complexity. Such comparisons clarify the strengths and limitations of CNF-based approaches within the broader landscape of PDE solvers.

Overall, this paper demonstrates that learning-guided Kansa solvers can serve as a promising and flexible tool for coupled or nonlinear PDE systems. **Future work** includes theoretical analysis of error and convergence properties, application to neural field in computing, and integration with differentiable pipelines in scientific domains.

REFERENCES

Robert A. Adams and John J. F. Fournier. *Sobolev spaces*, volume 140 of *Pure and Applied Mathematics*. Academic Press, Boston, MA, 2 edition, 2003. ISBN 978-0-12-044143-3. Originally published in 1975.

Michael Athanasopoulos, Hassan Ugail, and Gabriela González Castro. Parametric design of aircraft geometry using partial differential equations. *Advances in Engineering Software*, 40(7):479–486, 2009. ISSN 0965-9978. doi: https://doi.org/10.1016/j.advengsoft.2008.08.001. URL https://www.sciencedirect.com/science/article/pii/S0965997808001531.

Nicolas Bacaër. Lotka, Volterra and the predator–prey system (1920–1926). In *A Short History of Mathematical Population Dynamics*, pp. 71–76. Springer, London, 2011. doi: 10.1007/978-0-85729-115-8_13. URL https://doi.org/10.1007/978-0-85729-115-8_13.

Adam W. Bargteil and Tamar Shinar. An Introduction to Physics-based Animation. *ACM SIGGRAPH 2018 Courses*, 1(1):1–57, August 2018. doi: 10.1145/3214834.3214849.

HARRY BATEMAN. Some recent researches on the motion of fluids. *Monthly Weather Review*, 43(4):163 – 170, 1915. doi: 10.1175/1520-0493(1915)43⟨163:SRROTM⟩2.0.CO;2. URL https://journals.ametsoc.org/view/journals/mwre/43/4/1520-0493_1915_43_163_srrotm_2_0_co_2.xml.

Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in Machine Learning: a survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018. URL http://jmlr.org/papers/v18/17-468.html.

Richard E. Bellman. *Dynamic programming*. Princeton University Press, Princeton, NJ, 1957. ISBN 978-0-691-07951-6. Prepared for the Rand Corporation.

Richard Courant, Kurt Friedrichs, and Hans Lewy. Über die partiellen Differenzengleichungen der mathematischen Physik. *Mathematische Annalen*, 100(1):32–74, 1928. doi: 10.1007/BF01448839.

L.C. Evans. *Partial Differential Equations*. Graduate studies in mathematics. American Mathematical Society, 2010. ISBN 9780821849743. URL https://books.google.co.uk/books?id=Xnu0o_EJrCQC.

Walter Greiner. *Maxwell's equations*, pp. 250–275. Springer New York, New York, NY, 1998. ISBN 978-1-4612-0587-6. doi: 10.1007/978-1-4612-0587-6_13. URL https://doi.org/10.1007/978-1-4612-0587-6_13.

Eberhard Hopf. The partial differential equation $u_t + u\,u_x = \mu\,u_{xx}$. *Communications on Pure and Applied Mathematics*, 3(3):201–230, 1950. doi: https://doi.org/10.1002/cpa.3160030302. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cpa.3160030302.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, January 1989. doi: 10.1016/0893-6080(89)90020-8.

Arieh Iserles. *A first course in the Numerical Analysis of Differential Equations*. Cambridge University Press, Cambridge, 2 edition, 2008. ISBN 978-0-521-73490-5. URL http://www.cambridge.org/9780521734905.

E. J. Kansa. Multiquadrics—a scattered data approximation scheme with applications to computational fluid dynamics—II solutions to parabolic, hyperbolic and elliptic partial differential equations. *Computers & Mathematics with Applications*, 19(8–9):147–161, 1990. doi: 10.1016/0898-1221(90)90271-K.

Gitta Kutyniok. The Mathematics of Artificial Intelligence, 2022. URL https://arxiv.org/abs/2203.08890.

Jean le Rond D'Alembert. Recherches sur la courbe que forme une corde tenduë mise en vibration. *Histoire de l'académie royale des sciences et belles lettres de Berlin*, 3:214–219, 1747.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural Operator: Graph Kernel Network for Partial Differential Equations. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations (ODE/PDE+DL)*, 2020. URL https://arxiv.org/abs/2003.03485. Poster presentation.

Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-Informed Neural Operator for learning Partial Differential Equations. *ACM / IMS J. Data Sci.*, 1(3), May 2024. doi: 10.1145/3648506. URL https://doi.org/10.1145/3648506.

Giuseppe Orlando and Mario Sportelli. Growth and cycles as a struggle: Lotka–Volterra, Goodwin and Phillips. In Giuseppe Orlando, Alexander N. Pisarchik, and Ruedi Stoop (eds.), *Nonlinearities in Economics: An Interdisciplinary Approach to Economic Dynamics, Growth and Cycles*, pp. 191–208. Springer International Publishing, Cham, 2021. doi: 10.1007/978-3-030-64234-0_10. URL https://doi.org/10.1007/978-3-030-64234-0_10.

S. V. Patankar. *Numerical heat transfer and fluid flow*. Taylor & Francis, 1980. ISBN 978-0-89116-522-4.

Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. In *ACM SIGGRAPH 2003 Papers*, pp. 313–318, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1-58113-709-5. doi: 10.1145/1201775.882269. URL https://doi.org/10.1145/1201775.882269.

M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.10.045. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

Walter Rudin. *Principles of Mathematical Analysis*. McGraw-Hill, New York, 3 edition, 1976. ISBN 978-0-07-054235-8.

Tim De Ryck and Siddhartha Mishra. Error analysis for physics-informed neural networks (PINNs) approximating Kolmogorov PDEs. *Advances in Computational Mathematics*, 48(6):79, 2022. ISSN 1572-9044. doi: 10.1007/s10444-022-09985-9. URL https://doi.org/10.1007/s10444-022-09985-9.

Andrew Spielberg, Fangcheng Zhong, Kwang Moo Rematas, and et al. Differentiable visual computing for inverse problems and machine learning. *Nature Machine Intelligence*, 5:1189–1199, 2023. doi: 10.1038/s42256-023-00743-0. URL https://doi.org/10.1038/s42256-023-00743-0.

Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Dan MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. PDEBench: an extensive benchmark for scientific machine learning. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Sifan Wang, Hanwen Wang, and Paris Perdikaris. On the eigenvector bias of fourier feature networks: From regression to solving multi-scale pdes with physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 384:113938, 2021. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2021.113938. URL https://www.sciencedirect.com/science/article/pii/S0045782521002759.

Dmitry Yarotsky. Optimal approximation of continuous functions by very deep ReLU networks. In Sebastien Bubeck, Vianney Perchet, and Philippe Rigollet (eds.), *Proceedings of the 31st Annual Conference on Learning Theory*, volume 75 of *Proceedings of Machine Learning Research*, pp. 1–11, Stockholm, Sweden, 2018. PMLR. URL https://proceedings.mlr.press/v75/yarotsky18a.html.

Tjalling J. Ypma. Historical development of the newton–raphson method. *SIAM Review*, 37(4): 531–551, 1995. doi: 10.1137/1037125. URL https://doi.org/10.1137/1037125.

Fangcheng Zhong, Kyle Fogarty, Param Hanji, Tianhao Wu, Alejandro Sztrajman, Andrew Spielberg, Andrea Tagliasacchi, Petra Bosilj, and Cengiz Oztireli. Neural fields with hard constraints of arbitrary differential order. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

A LINEAR OPERATOR

A **linear operator** Iserles (2008) is a function $\mathcal{F} : V \to W$ that maps one vector space $V \in \mathbb{R}$ to another, or itself[5], $W \in \mathbb{R}$, and preserving the operations of **vector addition** and **scalar multiplication**, also known as **homogeneity**. Thus, for all vectors $\mathbf{u_i} \in V$ and all scalars c, the following features hold:

$$\begin{aligned} \mathcal{F}(\sum_{i=1}^{n} \mathbf{u}_i) &= \sum_{i=1}^{n} \mathcal{F}(\mathbf{u}_i), \quad \text{vector additivity,} \\ \mathcal{F}(c \cdot \mathbf{u}) &= c \cdot \mathcal{F}(\mathbf{u}), \quad \text{scalar multiplication.} \end{aligned} \tag{36}$$

Linear operators are fundamental in Linear Algebra for processing matrices, Quantum Mechanics for observables, Machine Learning, and Signal Processing. This forms the basis for Kansa method for linear PDEs.

Here are several commonly used examples of linear operators below, among which some are used in this work for PDE solver algorithms.

- **Matrix multiplication**: For a matrix A, the function $A : \mathbb{R}^n \to \mathbb{R}^m$ is a linear operator,

$$\mathcal{F}(\mathbf{x}) = A\mathbf{x}. \tag{37}$$

- **Integral operator**: The operator that integrates a function over a fixed interval $[a, b]$ is a linear operator,

$$I(f) = \int_a^b f(x)\, dx. \tag{38}$$

- **Differentiation**: The operator taking the derivative in a function space is a linear operator, because differentiation preserves addition and scalar multiplication,

$$D_x(f) = \frac{\partial f}{\partial x}. \tag{39}$$

- **Gradient operator**: In multivariable calculus, the gradient operator ∇ is a linear operator that maps a scalar field to a vector field,

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right). \tag{40}$$

- **Divergence operator**: In vector calculus, the divergence operator $\nabla\cdot$ is a linear operator that maps a vector field to a scalar field,

$$\nabla \cdot \mathbf{F} = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y} + \frac{\partial F_z}{\partial z}. \tag{41}$$

- **Laplace operator**: In the context of partial differential equations, the Laplace operator Δ is a linear operator that maps a scalar field to another scalar field,

$$\Delta f = \nabla \cdot (\nabla f) = \nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2}. \tag{42}$$

- **Curl operator**: In vector calculus, the curl operator $\nabla\times$ is a linear operator that maps a vector field to another vector field,

$$\nabla \times \mathbf{F} = \left(\frac{\partial F_z}{\partial y} - \frac{\partial F_y}{\partial z}, \frac{\partial F_x}{\partial z} - \frac{\partial F_z}{\partial x}, \frac{\partial F_y}{\partial x} - \frac{\partial F_x}{\partial y} \right) \quad = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \\ F_x & F_y & F_z \end{vmatrix}. \tag{43}$$

[5]If the domain and codomain are the same vector space, i.e., $\mathcal{F} : V \to V$, it's called a linear transformation or operator on V.

B PARTIAL DIFFERENTIAL EQUATIONS

B.1 BOUNDARY AND INITIAL CONDITIONS (BCS AND ICS)

Since solution to differential equations contain integration constants, which is non-unique, additional conditions are required to enforce uniqueness. The boundary conditions (BCs) specify the function u behavior on the domain boundary $\partial\Omega$, whereas the initial conditions (ICs) from time scale perspective are given at $t = 0$. The formulation is defined in equation 1.

There are some common boundary conditions, defined over the boundary $\Omega = [x_0, x_f]$ in 1D space, where $\{g_i\}_{i=1}^4$ are given closed-form functions,

$$\begin{aligned} \text{Zero BC:} \quad & u(x_0, t) = 0, \ u(x_f, t) = 0, \\ \text{Dirichlet BC:} \quad & u(x_0, t) = g_1(t), \ u(x_f, t) = g_2(t), \\ \text{von Neumann BC:} \quad & \frac{\partial u}{\partial x}(x_0, t) = g_3(t), \ \frac{\partial u}{\partial x}(x_f, t) = g_4(t). \end{aligned} \tag{44}$$

B.2 SUMMARY OF PDES

Table 12: Summary of PDEs with different characteristics.

Equation	**Domains**	**Linearity**	**Solution dim.**
Advection	Physics, Graphics	Linear	1
Wave	Physics, Graphics	Linear	1
Lotka-Volterra	Biology	Linear	**2**
Maxwell	Physics	Linear	**2**
Burgers	Physics, Graphics	**Nonlinear**	1

B.3 1D ADVECTION EQUATION

The advection equation Takamoto et al. (2022) models the linear transport of a scalar quantity $u(x, t)$, which is changed over time t and space x, as follows:

$$\begin{cases} \frac{\partial u(x,t)}{\partial t} + \beta \frac{\partial u(x,t)}{\partial x} = 0, & x \in [x_0, x_f], t \in [t_0, t_f], \\ u(x, 0) = u_0(x), & x \in [x_0, x_f], \quad \text{initial condition}, \end{cases} \tag{45}$$

where parameter $\beta \in \mathbb{R}$ is the advection velocity, and $u_0(x)$ is the initial condition given at $t = 0$. The analytical solution of equation 45 is,

$$u(x, t) = u_0(x - \beta t). \tag{46}$$

The positivity of parameter β indicates the direction of wave propagation. From equation 46, when $\beta > 0$, the wave propagates rightwards, and vice versa. The solution is visualized in Figure 1, with initial condition $u_0(x) = \sin(2\pi x)$, $x \in [0, 1]$.

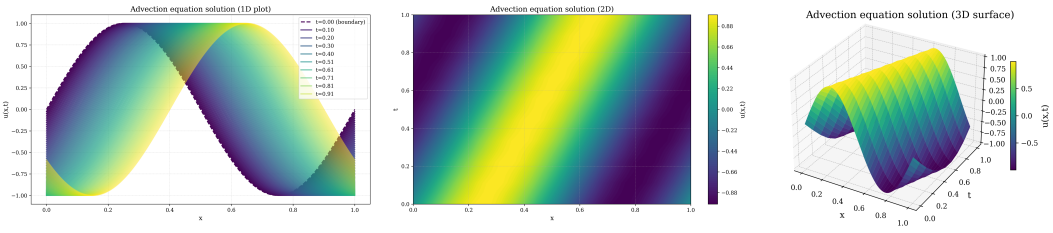


Figure 1: Advection equation solution visualization in 1D, 2D and 3D.

B.4 LOTKA-VOLTERRA PREDATOR-PREY MODEL

Lotka-Volterra predator-prey model Bacaër (2011) relates the populations of prey $x(t)$ and predators $y(t)$ at time t in a dynamic biological system via coupled differential equations, also applicable to other fields, e.g. the unemployment rate with respect to wage growth Orlando & Sportelli (2021) and many more,

$$\begin{cases} x'(t) := \frac{dx(t)}{dt} = \alpha x(t) - \beta x(t) \cdot y(t), \\ y'(t) := \frac{dy(t)}{dt} = \delta x(t) \cdot y(t) - \gamma y(t). \end{cases} \quad , t \in [t_0, t_f]. \tag{47}$$

where α is the prey growth rate, β is the predation rate, δ is the ratio of neonate predators to eaten prey, and γ is the predator death rate. It assumes that there would be unlimited food supply for the prey, and thus exponential growth $\alpha x(t)$. The multiplicative term $x(t) \cdot y(t)$ represents the encounters between prey and predators statistically.

The system has no explicit analytical solution, but the implicit solution exists. After scaling of variables,

$$x^*(t) = \frac{\delta}{\gamma} x(t), \quad y^*(t) = \frac{\beta}{\alpha} y(t), \quad \tau = \alpha t, \tag{48}$$

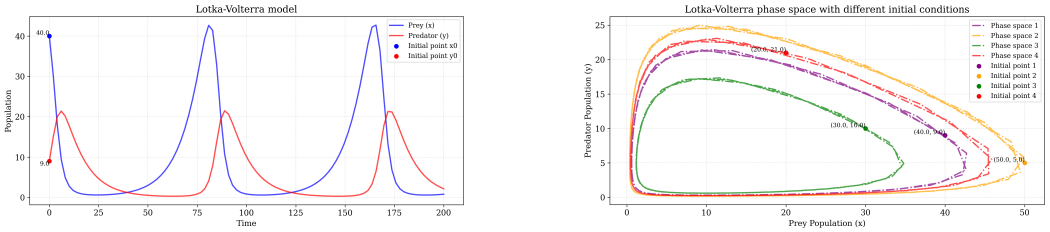
By plugging into equation 47, and dividing the first equation by the second,

$$\frac{dy^*}{dx^*} = \frac{\gamma}{\alpha} \cdot \frac{y^*(x^* - 1)}{x^*(y^* - 1)}, \tag{49}$$

The implicit solution is given by integration separation of variables, for which $C_{\text{L-V}} \in \mathbb{R}$ is the integration constant,

$$\ln(y^*) - y^* - \frac{\gamma}{\alpha}[\ln(x^*) - x^*] = C_{\text{L-V}}. \tag{50}$$

Figure 2 shows the solution with phase space given by the above implicit solution, which depends on the initial conditions $x(0)$ and $y(0)$.



(a) Prey and predator solution populations

(b) Phase space trajectory

Figure 2: Lotka-Volterra predator-prey model solution and phase space.

B.5 MAXWELL'S EQUATIONS

In electromagnetism, Maxwell's equations Greiner (1998) relate the electric field $\mathbf{E}(\mathbf{r}, t)$ and magnetic field $\mathbf{B}(\mathbf{r}, t)$ with spatial position $\mathbf{r} \in \mathbb{R}^3$ and time t, to the electric charge density $\rho(\mathbf{r}, t) \in \mathbb{R}$ and current density $\mathbf{J}(\mathbf{r}, t) \in \mathbb{R}^3$. The differential form is as follows,

$$\begin{cases} \nabla \cdot \mathbf{E} = \frac{\rho}{\epsilon_0}, & \text{Gauss's law,} \\ \nabla \cdot \mathbf{B} = 0, & \text{Gauss's law for magnetism,} \\ \nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}, & \text{Faraday's law of induction,} \\ \nabla \times \mathbf{B} = \mu_0 \mathbf{J} + \mu_0 \epsilon_0 \frac{\partial \mathbf{E}}{\partial t}, & \text{Ampère-Maxwell law,} \end{cases} \tag{51}$$

where constants $\mu_0, \epsilon_0 \in \mathbb{R}^+$ are the vacuum permeability and permittivity respectively. Their product is the reciprocal of the square of the speed of light $c \approx 3 \times 10^8 \, \mathrm{m\,s^{-1}}$ in vacuum,

$$\mu_0 \epsilon_0 = \frac{1}{c^2}. \tag{52}$$

The first two equations state that the electric field $\mathbf{E}$ sourced by electric charges, and no magnetic monopoles exist. The last two equations depict how a time-varying magnetic field $\mathbf{B}$ induces an electric field $\mathbf{E}$, and vice versa with the addition of current density $\mathbf{J}$. In general, Maxwell's equations are *linear* with respect to $\mathbf{E}$ and $\mathbf{B}$.

Taking the curl of Faraday's law and Ampère-Maxwell law respectively,

$$\begin{cases} \nabla \times (\nabla \times \mathbf{E}) = -\frac{\partial}{\partial t}(\nabla \times \mathbf{B}), \\ \nabla \times (\nabla \times \mathbf{B}) = \mu_0(\nabla \times \mathbf{J}) + \mu_0\epsilon_0 \frac{\partial}{\partial t}(\nabla \times \mathbf{E}). \end{cases} \tag{53}$$

By vector calculus identity of $\nabla \times (\nabla \times \mathbf{E}) = \nabla(\nabla \cdot \mathbf{E}) - \Delta\mathbf{E}$, and plugging in Ampère-Maxwell law on the right-hand side of the first equation,

$$\begin{cases} \nabla(\nabla \cdot \mathbf{E}) - \Delta\mathbf{E} = -\mu_0 \frac{\partial \mathbf{J}}{\partial t} - \mu_0\epsilon_0 \frac{\partial^2 \mathbf{E}}{\partial t^2}, \\ \nabla(\nabla \cdot \mathbf{B}) - \Delta\mathbf{B} = \mu_0(\nabla \times \mathbf{J}) + \mu_0\epsilon_0 \frac{\partial}{\partial t}(\nabla \times \mathbf{E}). \end{cases} \tag{54}$$

By substituting Gauss's law for $\nabla \cdot \mathbf{E}$ in the first equation, Gauss's law for magnetism for $\nabla \cdot \mathbf{B}$ and Faraday's law for $\nabla \times \mathbf{E}$ in the second equation, the two equations after rearrangement are inhomogeneous, i.e. including source terms $\mathbf{F}(\mathbf{r}, t)$, wave equations, taking the forms of $c^2\Delta u - u_{tt} = \mathbf{F}$,

$$\begin{cases} \Delta\mathbf{E} - \mu_0\epsilon_0 \frac{\partial^2 \mathbf{E}}{\partial t^2} = \nabla(\frac{\rho}{\epsilon_0}) + \mu_0 \frac{\partial \mathbf{J}}{\partial t}, \\ \Delta\mathbf{B} - \mu_0\epsilon_0 \frac{\partial^2 \mathbf{B}}{\partial t^2} = -\mu_0(\nabla \times \mathbf{J}). \end{cases} \tag{55}$$

To simplify the problem, we take the one-dimensional (1D) electromagnetic wave propagating along x-axis without sources, i.e. $\rho = 0$ and $\mathbf{J} = 0$, with the electric field $\mathbf{E}(\mathbf{r}, t) = (0, 0, E_z(x, t))$ along z-axis and magnetic field $\mathbf{B}(\mathbf{r}, t) = (0, B_y(x, t), 0)$ along y-axis respectively. By expanding the defintion of curl $\nabla\times$ operators, and removing the zero terms,

$$\nabla \times \mathbf{E} = \left(\frac{\partial E_z}{\partial y} - \frac{\partial E_y}{\partial z}, \frac{\partial E_x}{\partial z} - \frac{\partial E_z}{\partial x}, \frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y} \right) = \left(0, -\frac{\partial E_z}{\partial x}, 0 \right), \tag{56}$$

thus the reduced last two equations of Maxwell's equations equation 51 are,

$$\begin{cases} \frac{\partial E_z}{\partial x} = -\frac{\partial B_y}{\partial t}, \\ \frac{\partial B_y}{\partial x} = -\mu_0\epsilon_0 \frac{\partial E_z}{\partial t}. \end{cases} \tag{57}$$

By taking partial derivatives ∂_x and ∂_t, and simplifying, the 1D wave solutions are[6],

$$\begin{cases} \frac{\partial^2 E_z}{\partial x^2} - \mu_0\epsilon_0 \frac{\partial^2 E_z}{\partial t^2} = 0, \\ \frac{\partial^2 B_y}{\partial x^2} - \mu_0\epsilon_0 \frac{\partial^2 B_y}{\partial t^2} = 0. \end{cases} \tag{58}$$

The initial conditions at $t = 0$ are given as follows,

$$E_z(x, 0) = f(x), \quad B_y(x, 0) = g(x), \quad x \in [x_0, x_f]. \tag{59}$$

Let $u = E_z + B_y$ and $v = E_z - B_y$ and with change of variables $x_t = x_{t=0} \pm ct$, where c defined in equation 52 is the speed of light in vacuum. According to d'Alembert's formula le Rond D'Alembert (1747),

$$\begin{cases} u(x, t) = u(x - ct, 0) = f(x - ct) + g(x - ct), \\ v(x, t) = v(x + ct, 0) = f(x + ct) - g(x + ct). \end{cases} \tag{60}$$

By reversing the change of variables, the analytical solutions to equation 58 are,

$$\begin{cases} E_z = \frac{1}{2}(u + v) = \frac{1}{2}[f(x - ct) + f(x + ct)] + \frac{1}{2}[g(x - ct) - g(x + ct)], \\ B_y = \frac{1}{2}(u - v) = \frac{1}{2}[f(x - ct) - f(x + ct)] + \frac{1}{2}[g(x - ct) + g(x + ct)]. \end{cases} \tag{61}$$

The solution is visualized in Figure 3, with initial condition given as,

$$f(x) = \sin(2\pi x) + 0.5\sin(4\pi x),\ g(x) = \cos(2\pi x) + 0.5\cos(4\pi x) \quad x \in [0, 1], t \in [0, 0.5]. \tag{62}$$

[6]The 1D wave equation aligned with the general inhomogeneous wave equation equation 55, with $\mathbf{F} = 0$.

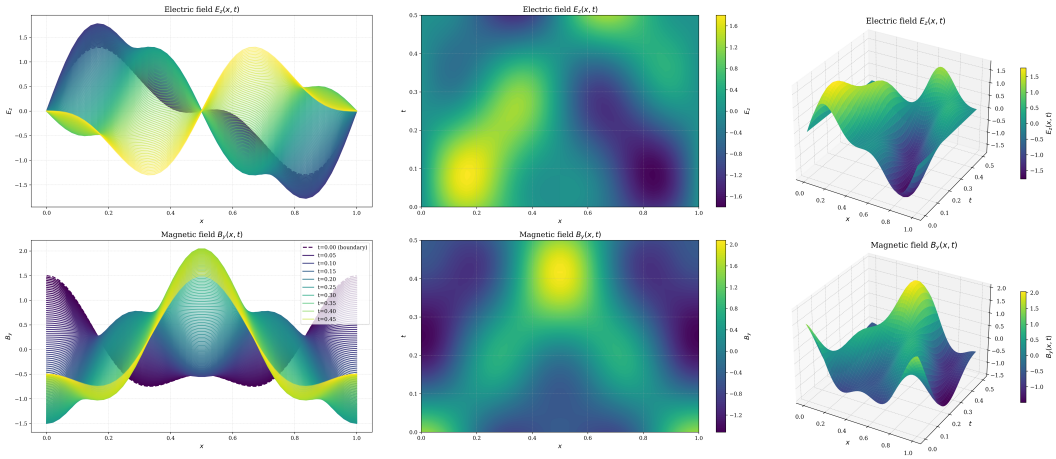


Figure 3: Maxwell's equations solution visualization in 1D, 2D and 3D.

B.6 VISCOUS BURGERS' EQUATION

Viscous Burgers' equation Takamoto et al. (2022) captures both non-linear advection, also known as convection and diffusion phenomena in dynamics,

$$\begin{cases} \frac{\partial u(x,t)}{\partial t} + u(x,t)\frac{\partial u(x,t)}{\partial x} = \nu \frac{\partial^2 u(x,t)}{\partial x^2}, & x \in [x_0, x_f], t \in [t_0, t_f], \\ u(x,0) = u_0(x), & x \in [x_0, x_f], \quad \text{initial condition}, \end{cases} \tag{63}$$

where viscosity $\nu \in \mathbb{R}^+$ is the positive constant, and $u_0(x)$ is the initial condition given at $t = 0$. By Cole-Hopf transformation Hopf (1950), unknown function $u(x,t)$ is converted into $\phi(x,t)$ via,

$$u(x,t) = -2\nu \frac{\partial}{\partial x} \ln \phi(x,t) = -2\nu \frac{1}{\phi(x,t)} \frac{\partial \phi(x,t)}{\partial x} \equiv -2\nu \frac{\phi_x}{\phi}. \tag{64}$$

By chain rule and quotient rule of differentiation, the first-order and second-order spatial or temporal derivatives of $u(x,t)$ are,

$$\begin{aligned} \frac{\partial u(x,t)}{\partial x} &= 2\nu \left(\frac{\phi_x^2}{\phi^2} - \frac{\phi_{xx}}{\phi} \right), \quad \frac{\partial u(x,t)}{\partial t} = 2\nu \left(\frac{\phi_x \phi_t}{\phi^2} - \frac{\phi_{xt}}{\phi} \right), \\ \frac{\partial^2 u(x,t)}{\partial x^2} &= 2\nu \left(\frac{3\phi_x \phi_{xx}}{\phi^2} - \frac{2\phi_x^3}{\phi^3} - \frac{\phi_{xxx}}{\phi} \right). \end{aligned} \tag{65}$$

By plugging equation 65 into equation 63 and simplifying,

$$2\nu \left(\frac{\phi_x \phi_t}{\phi^2} - \frac{\phi_{xt}}{\phi} - \nu \frac{\phi_x \phi_{xx}}{\phi^2} + \nu \frac{\phi_{xxx}}{\phi} \right) = 0, \quad x \in [x_0, x_f], t \in [t_0, t_f], \tag{66}$$

With the inversion of quotient rule, equation 66 is rearranged as,

$$2\nu \frac{\partial}{\partial x} \left(\frac{\nu \phi_{xx} - \phi_t}{\phi} \right) = 0, \quad x \in [x_0, x_f], t \in [t_0, t_f]. \tag{67}$$

By integrating equation 67 with respect to x and introducing an integration function $f(t)$,

$$\frac{\nu \phi_{xx} - \phi_t}{\phi} = f(t), \quad x \in [x_0, x_f], t \in [t_0, t_f]. \tag{68}$$

Now introduce $f(t) = \frac{dF(t)}{dt}$ and $\tilde{\phi} = \phi \cdot e^{F(t)}$, thus the derivatives of $\tilde{\phi}$ are,

$$\frac{\partial \tilde{\phi}}{\partial t} = e^{F(t)} \left(\phi_t + \phi \frac{dF(t)}{dt} \right), \quad \frac{\partial^2 \tilde{\phi}}{\partial x^2} = e^{F(t)} \phi_{xx}, \tag{69}$$

by plugging them into equation 68. The resulting equation is reduced to the standard heat equation,

$$\nu \frac{\partial^2 \tilde{\phi}(x,t)}{\partial x^2} - \frac{\partial \tilde{\phi}(x,t)}{\partial t} = 0, \quad x \in [x_0, x_f], t \in [t_0, t_f]. \tag{70}$$

The solution of equation 70 is formed by heat kernel $\Phi(x,t)$ convolved with the initial condition $\tilde{\phi}_0(x) = \tilde{\phi}(x,0)$ Evans (2010),

$$\tilde{\phi}(x,t) = \int_{-\infty}^{\infty} \Phi(x-x',t)\tilde{\phi}_0(x')\, dx', \quad \text{where} \quad \Phi(x,t) = \frac{1}{\sqrt{4\pi\nu t}} e^{-\frac{x^2}{4\nu t}}. \tag{71}$$

Note that the transformation from ϕ to $\tilde{\phi}$ does not change the Cole-Hopf transformation equation 64, since the additional multiplicative term $e^{F(t)}$ is independent of x,

$$u(x,t) = -2\nu \frac{\partial}{\partial x} \ln \phi(x,t) = -2\nu \frac{\partial}{\partial x} \ln \tilde{\phi}(x,t). \tag{72}$$

From the Cole-Hopf equation 72 at $t = 0$ and via integration, the initial condition for $\tilde{\phi}(x,0)$ is thus,

$$\tilde{\phi}_0(x) = -\frac{1}{2\nu} \int_0^x u_0(x')\, dx'. \tag{73}$$

The analytical solution of equation 63 is thus plugging equation 73 into equation 71 and then into equation 72.

In this work, we consider when $u_0(-\infty)$ and $u_0(\infty)$ exist and $u_0'(x) < 0$ for all $x \in \mathbb{R}$, the explicit expression BATEMAN (1915) is then a steadily propagating wave as below,

$$u(x,t) = c - \Delta u_0 \tanh\left(\frac{\Delta u_0}{2\nu}(x-ct)\right), \quad \text{where } c = \frac{u_0(-\infty)+u_0(\infty)}{2}, \Delta u_0 = \frac{u_0(-\infty)-u_0(\infty)}{2}. \tag{74}$$

The solution is visualized in Figure 4, with initial condition set by equation 74, $u_0(-\infty) = 1$, $u_0(\infty) = 0$, and $\nu = 0.5$.

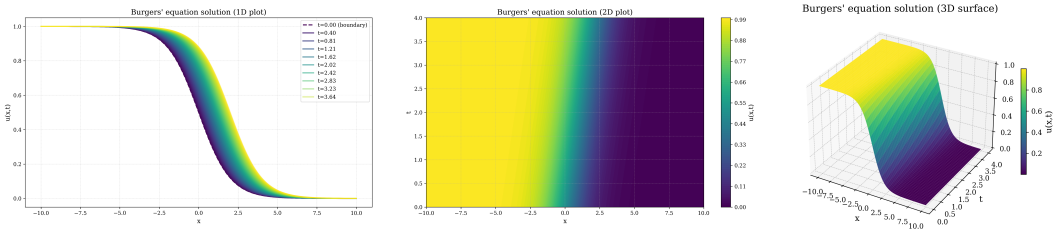


Figure 4: Burgers' equation solution visualization in 1D, 2D and 3D.

B.7 PDE SOLVERS

There are many attempts to solve the PDE solution field u. Among which, we categorize PDE solvers into two types, i.e. numerical analysis methods and neural-based methods.

Constrained optimization. In PDE solvers, constraints are boundary conditions, initial conditions, or PDE residuals. They can be in the form of either soft or hard constraints. The former is the cost functions that are penalised, while the latter is that can not be violated, e.g. (in)equality forms.

Table 13: Summary of different PDE solvers.

method	**motivation**	**training**	**supervised**	**constraint**
FDM	grid-based	×	N/A	hard
PINN	physics-driven	✓	×	soft

method	motivation	training	supervised	constraint
NO	data-driven	✓	✓	soft
- FNO	data-driven	✓	✓	soft
- PINO	hybrid	✓	✓	soft
KM	mesh-free grid	×	N/A	hard
- CNF	KM on neural fields	✓	×	hard

B.7.1 FINITE DIFFERENCE METHOD (FDM)

Considering a discretized sequence $\mathbf{u} \in \mathbb{R}^{N\times M}$ of the continuous function $u(x,t)$ as in equation 45. Along the spatial dimension x and temporal dimension t, there are N and M sampled points respectively. The finite difference operators Iserles (2008) defined on per element, are as follows:

$$(\Delta \mathbf{u})_i = \begin{cases} (\Delta^+ \mathbf{u})_i = u_{i+1}^j - u_i^j, & \text{forward difference.} \\ (\Delta^- \mathbf{u})_i = u_i^j - u_{i-1}^j, & \text{backward difference.} \\ (\Delta^0 \mathbf{u})_i = \dfrac{u_{i+1}^j - u_{i-1}^j}{2}, & \text{central difference.} \end{cases}, \tag{75}$$

for which $i \in \{0, 1, \ldots, N-1\}$ is the *spatial index*, and $j \in \{0, 1, \ldots, M-1\}$ is the *temporal index* of the sequence $\mathbf{u}$.

The partial equations often involve full and/or partial derivatives, where differential operators can be discretized into difference operators equation 75 via finite difference method Bargteil & Shinar (2018). By Taylor expansion of $u(x \pm \Delta x, t)$ around $u(x,t)$ up to the first order error, the corresponding examples for the spatial derivative are,

$$\begin{aligned} \frac{\partial u_i^j(x,t)}{\partial x} &= \begin{cases} \frac{(\Delta^- \mathbf{u})_i}{\Delta x} + O(\Delta x) \approx \frac{(\Delta^- \mathbf{u})_i}{\Delta x} = \frac{u_i^j - u_{i-1}^j}{\Delta x} & \text{if } \beta > 0, \\ \frac{(\Delta^+ \mathbf{u})_i}{\Delta x} + O(\Delta x) \approx \frac{(\Delta^+ \mathbf{u})_i}{\Delta x} = \frac{u_{i+1}^j - u_i^j}{\Delta x} & \text{if } \beta < 0. \end{cases}, && \text{upwind scheme.} \\ &= \frac{(\Delta^0 \mathbf{u})_i}{\Delta x} + O(\Delta x^2) \approx \frac{(\Delta^0 \mathbf{u})_i}{\Delta x} = \frac{u_{i+1}^j - u_{i-1}^j}{2\Delta x}, && \text{central difference.} \end{aligned} \tag{76}$$

where Δx is the spatial spacing, with spatial index i and temporal index j defined above. The **upwind scheme** Patankar (1980) considers where the information comes from, e.g. when $\beta > 0$, the wave propagates rightwards, and thus u_i^j is influenced by u_{i-1}^j, and vice versa for downwind scheme. Under the upwind scheme, the advection equation equation 45 is therefore as the following ODE,

$$\frac{\partial u(x,t)}{\partial t} + \beta[\mathbb{I}_{\beta>0} \frac{(\Delta^- \mathbf{u})_i}{\Delta x} + \mathbb{I}_{\beta<0} \frac{(\Delta^+ \mathbf{u})_i}{\Delta x}] = 0, \tag{77}$$

where the indicator function $\mathbb{I}_{\beta>0} = \begin{cases} 1 & \text{if } \beta > 0, \\ 0 & \text{otherwise.} \end{cases}$ is for controlling different cases of β[7].

By forward Euler method for ODEs, the temporal derivative is discretized via the forward difference operator. After which, the advection equation equation 45 is simplified as, with Δt being the temporal spacing,

$$\frac{u_i^{j+1} - u_i^j}{\Delta t} + \beta[\mathbb{I}_{\beta>0} \frac{(\Delta^- \mathbf{u})_i}{\Delta x} + \mathbb{I}_{\beta<0} \frac{(\Delta^+ \mathbf{u})_i}{\Delta x}] = 0. \tag{78}$$

With algebraic reordering, the upwind scheme update rule is thus,

$$u_i^{j+1} = u_i^j - \frac{\beta \Delta t}{\Delta x}[\mathbb{I}_{\beta>0} (\Delta^- \mathbf{u})_i + \mathbb{I}_{\beta<0} (\Delta^+ \mathbf{u})_i]. \tag{79}$$

Stability condition. For implicit numerical schemes, e.g. the backward Euler method, the solution is unconditionally stable. However, for explicit numerical schemes, e.g. the forward Euler method above, stability conditions must be satisfied to avoid numerical instability, which we briefly introduce below.

[7] Alternatively, one may use $\max(\beta, 0)$ and $\min(\beta, 0)$ to replace $\beta \mathbb{I}_{\beta>0}$ and $\beta \mathbb{I}_{\beta<0}$ respectively.

In the 1D space, the scalar Courant number C, also known as the CFL stability criteria, measures the ratio of how far the wave propagates in one time interval Δt to the spatial spacing Δx. The CFL condition Courant et al. (1928) states that C must satisfy,

$$C = \frac{|\beta|\Delta t}{\Delta x} \leq C_{max}, \tag{80}$$

where C_{max} is a problem-dependent constant. It sets the maximum allowable time step Δt for a given Δx, for numerical stability.

B.7.2 PHYSICS-INFORMED NEURAL NETWORK (PINN)

Physics-informed neural network (PINN) Raissi et al. (2019) is a data-driven approach for functional PDE approximation, which requires a large labeled dataset but has the ability to generalize. Consider the general form of PDEs defined in equation 1, PINNs approximate the unknown solution $u(x,t) \in \mathcal{U}$ with a neural network $\hat{u}_\theta(x,t) \in \mathcal{U}$, i.e. $\hat{u}_\theta(x,t) \approx u(x,t)$, parameterized by updatable parameters $\theta \in \Theta$.

Residual $\mathcal{R}_\theta$ of the PDEs is calculated without supervised data for the neural network $\hat{u}_\theta$, which is minimized via automatic differentiation Baydin et al. (2018)[8] during training for generalizability,

$$\mathcal{R}_\theta(x,t) \in \mathcal{Y} = \mathcal{D}[\hat{u}_\theta](x,t) - f(x,t), \quad x \in \Omega, t \in [t_0, t_f]. \tag{81}$$

The residual loss L_R[9], also known as the physics-informed loss, is defined to be the p-norm of the residual $\mathcal{R}_\theta$ in equation 81. During training, $N_\mathcal{R}$ quadrature points are sampled, where the integral loss is approximated by the discretized loss $\mathcal{L}_R$ with weights ω_k at each sample index k and training error $\mathcal{E}_T(\theta)$,

$$\begin{aligned} L_R := (||\mathcal{R}_\theta||_p)^p := \underbrace{[(\int_\mathbb{D} |\mathcal{R}_\theta|^p \, dx \, dt)^{\frac{1}{p}}]^p}_{\text{integral } L_R} = \int_\mathbb{D} |\mathcal{R}_\theta|^p \, dx \, dt \\ \text{By quadrature,} \quad = \underbrace{\sum_{k=1}^{N_\mathcal{R}} \omega_k |\mathcal{R}_\theta(x_k,t_k)|^p}_{\text{discretized } \mathcal{L}_R} + \mathcal{E}_T(\theta) \approx \mathcal{L}_R, \quad \text{where } \mathcal{E}_T(\theta) = L_R - \mathcal{L}_R. \end{aligned} \tag{82}$$

If considering the boundary conditions, the residual for the i-th boundary condition $\mathcal{R}_\theta^{\mathcal{B}_i}$ is calculated via equation 1 as well, after which the boundary condition loss $\mathcal{L}_{\text{BC}}$ is defined accordingly,

$$\mathcal{R}_\theta^{\mathcal{B}_i}(x,t) \in \mathcal{Z}_i = \mathcal{B}_i[\hat{u}_\theta](x,t) - g_i(x,t), \quad x \in \partial\Omega_i, t \in [t_0, t_f]. \tag{83}$$

As defined in equation 99, the *total error* between the optimal solution from the network $\hat{u}_\theta$ and the ground truth u is, by expanding equation 97,

$$\mathcal{E}_{\text{PINN}}(\theta) = (||\hat{u}_\theta - u||_p)^p. \tag{84}$$

During training, the network is optimized on supervised dataset $\{(x_n,t_n), u(x_n,t_n)\}_{n=1}^{N_d}$, with N_d being the total number of data. The supervised loss $\mathcal{L}_{\text{data}}$[10] approximates the total error equation 84,

$$\mathcal{L}_{\text{data}} = \frac{1}{N_d} \sum_{n=1}^{N_d} (|\hat{u}_\theta(x_n,t_n) - u(x_n,t_n)|^p). \tag{85}$$

Training. PINN approximates the solution as $\hat{u}_\theta = u_{\theta^{\text{opt}}}(x,t)$. To avoid overfitting due to the *limited* supervised data, the main goal is to minimize the unsupervised residual error $\mathcal{L}_\mathcal{R}$ equation 82. With the addition of the supervised loss equation 85 and the boundary condition residual

[8]Example of $\frac{\partial u}{\partial x}$: `u_x = torch.autograd.grad(outputs=u, inputs=x, create_graph=True)[0]`

[9]Note that L_R is the same as the risk $\mathcal{R}$ defined in equation 97, but for the residual $\mathcal{R}_\theta$ instead of the solution u.

[10]Note that when the supervised data is only sampled on the boundary, the supervised loss and the boundary condition loss are the same.

equation 83, the optimized theta is $\theta^{\text{opt}} \approx \arg\min_{\theta\in\Theta} \mathcal{L}$, where the total training loss $\mathcal{L}_{\text{PINN}}$ is,

$$\mathcal{L}_{\text{PINN}} = \underbrace{\sum_{k=1}^{N_{\mathcal{R}}} \omega_k |\mathcal{R}_\theta(x_k, t_k)|^p}_{\text{Discretized residual loss } \mathcal{L}_{\mathcal{R}}} + \lambda_1 \underbrace{\frac{1}{N_d}\sum_{n=1}^{N_d} (|\hat{u}_\theta(x_n, t_n) - u(x_n, t_n)|^p)}_{\text{Supervised loss } \mathcal{L}_{\text{data}}} + \lambda_2 \underbrace{\sum_i \sum_{b=1}^{N_{\mathcal{B}_i}} \omega_b^{\mathcal{B}_i} |\mathcal{R}_\theta^{\mathcal{B}_i}(x_b, t_b)|^p}_{\text{BC loss } \mathcal{L}_{\text{BC}}}, \tag{86}$$

with weights $\omega_b^{\mathcal{B}_i}$ at each sample index b for i-th boundary condition and regularization parameters $\lambda_1, \lambda_2 > 0$ for combining different losses.

Algorithm 1 Physics-Informed Neural Network training pseudocode.

1: **Input:** Initial parameters θ for network $\hat{u}_\theta$.
2: **Output:** Optimized parameters θ^{opt} for network $\hat{u}_\theta$.
3: **Hyperparameters:** Learning rate η, number of training iterations N_{iter}.
4: **while** number of iterations $< N_{\text{iter}}$ **do**
5: Sample PDE points $x_k \in \Omega, t_k \in [t_0, t_f]$ and boundary points $x_b \in \partial\Omega_i, t_b \in [t_0, t_f]$.
6: Compute the network outputs$\hat{u}_\theta$ and their derivatives $\mathcal{D}[\hat{u}_\theta]$ and boundary $\mathcal{B}_i[\hat{u}_\theta]$.
7: Compute loss $\mathcal{L}_{\text{PINN}} = \mathcal{L}_{\mathcal{R}} + \mathcal{L}_{\text{data}} + \mathcal{L}_{\text{BC}}$ by equation 86.
8: By gradient descent, update $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}$.
9: **end while**

B.7.3 NEURAL OPERATOR (NO)

Operator learning. From the general form of PDEs equation 1, we assume that within the $\mathcal{D}$ operator, the source function f or the initial conditions, there is a parameter a of the same dimension as the solution u. In this subsection, we denote the differential operator $\mathcal{D}$ as $\mathcal{D}_a$, where the PDE is thus $\mathcal{D}_a[u] = f$.

Given the dataset $\{(a_i^j, f_i^j), u_i^j | i = 1, ..., N_{\mathcal{R}}\}_{j=1}^{N_{pde}}$ with N_{pde} PDE instances each $N_{\mathcal{R}}$ quadrature points, the idea of operator learning Li et al. (2020) is to learn the operator $\mathcal{G}$ mapping input $a \in \mathcal{A} : \mathbb{D} \to \mathbb{R}$ to the solution $u \in \mathcal{U} : \mathbb{D} \to \mathbb{R}$, i.e. $\mathcal{G}(a, f) = u$ connecting two function spaces $\mathcal{A}$ and $\mathcal{U}$ with *infinite* dimensions, which is **challenging** for neural networks since they are for *finite* dimensions instead.

Solution A. To solve this challenge, one solution is to **parameterize** the PDE $u = u(t, x, \mu)$, assuming that a is measureable in finite dimension, i.e. $a = a(\mu), \mu \in \mathbb{R}^{d_y}$. It's a techinique widely used in aircraft design and manufacturing Athanasopoulos et al. (2009), image processing Pérez et al. (2003). The **training** process is therefore to minimize the supervised loss $\mathcal{L}_{data}$ from data with p-norm, which measures how much the predicted solution $\mathcal{G}_\theta(a_i, f_i)$ deviates from the ground truth u_i for each data point i,

$$\mathcal{L}_{data} = \frac{1}{N_{pde} \times N_{\mathcal{R}}} \sum_{j=1}^{N_{pde}} \sum_{i=1}^{N_{\mathcal{R}}} (||\mathcal{G}_\theta(a_i^j, f_i^j) - u_i^j||_p)^p. \tag{87}$$

The approximated solution is therefore $\hat{u}_\theta = \mathcal{G}_{\theta^{\text{opt}}}(a, f)$, where $\theta^{\text{opt}} \approx \arg\min_{\theta\in\Theta} \mathcal{L}_{data}$. There is no addition of the PDE residual loss $\mathcal{L}_{\mathcal{R}}$ equation 81 as in PINNs for basic operator learning. An extension, termed as **physics-informed neural operator (PINO)** Li et al. (2024), combines the supervised loss $\mathcal{L}_{data}$ and the residual loss $\mathcal{L}_{\mathcal{R}}$ as the total training loss,

$$\mathcal{L}_{\text{PINO}} = \mathcal{L}_{data} + \lambda \underbrace{\frac{1}{N_{pde} \times N_{\mathcal{R}}} \sum_{j=1}^{N_{pde}} \sum_{i=1}^{N_{\mathcal{R}}} (||\mathcal{D}_a[\mathcal{G}_\theta(a_i^j, f_i^j)] - f_i^j(x_i, t_i)||_p)^p}_{\text{Residual loss } \mathcal{L}_{\mathcal{R}}}, \tag{88}$$

Despite the simplicity in ideas, the parameterization *suffers from* how to sample from the given space, non-uniqueness, and low generalization to unseen a.

Solution B. Interpolation from the discretized grid, including a neural network based interpolator or traditional methods (linear, cubic, spline, etc). However, it *suffers from* inconsistency between the discretized and continuous functions.

Solution C. Generalize the neural network from discrete to continuous function space.

$$(\mathcal{N}_l v)(x) = \sigma[A_l v(x) + B_l(x) + \int_D K_l(x,y)v(y)\ dy], \quad x \in D. \tag{89}$$

Fast implemenentation via FFT.

C ARCHITECTURE DETAILS

We adopt a multi-scale feed-forward neural network architecture for PINN Wang et al. (2021), which augments the original feed-forward architecture with input encoding layers of multiple frequency scales. There are three hidden layers each with 64 neurons. For FNO, we adopt the architecture with 16 modes retained in the spectral convolution layer, and the latent feature dimension is 64.

D ENVIRONMENT SETUP

All the measurements are conducted on a Mac M1, with a single-core CPU running at 3.2 GHz. In the following sections, the data points are uniformly sampled unless otherwise specified.

Table 14: 1D Maxwell's equations experimental setup.

domain	**time span**	**parameter**	**initial conditions**
$x_0 = 0$, $x_f = 1$	$t_0 = 0$, $t_f = \frac{1}{2}$	$c = 1$	$E_z(x,0) = \sin(2\pi x) + \frac{1}{2}\sin(4\pi x)$ $B_y(x,0) = \cos(2\pi x) + \frac{1}{2}\cos(4\pi x)$

E BURGERS' EQUATION STABILITY EXPERIMENT

Stability. For four time discretization schemes, only forward Euler scheme is unstable (Table 15), where time step Δt exceeds the stability limit when $C^t_{\text{scale}} = 1$ and 2 according to CFL condition.

Table 15: Stability test of forward Euler Kansa method on Burgers' equation.

C^t_{scale}	1	2	4	10
$\hat{\mathcal{R}}_{\text{relative } L_2}$	3.74×10^{29}	NaN	4.31×10^{-3}	3.11×10^{-3}
Stability	*unstable*	*unstable*	**stable**	**stable**

F LEARNING THEORY

F.1 FUNCTIONAL ANALYSIS

We introduce basic functional analysis concepts here for PDE solvers and later learning theory (Appendix § F).

The p-**norm** of a function $f : \Omega \subseteq \mathbb{R}^d \to \mathbb{R}$ is defined as,

$$||f||_p = (\int_\Omega |f(x)|^p\ dx)^{\frac{1}{p}}, \text{for } 1 \leq p < \infty. \tag{90}$$

The p-**integrable** function $f \in L^p(\Omega)$, is defined as

$$(||f||_p)^p = \int_\Omega |f(x)|^p\ dx < \infty. \tag{91}$$

For additional concepts used for learning theory, please refer to Appendix § F.2.

F.2 FUNCTIONAL ANALYSIS ADDENDUM

Smoothness Rudin (1976) of a function is defined to be the number of continuous derivatives it has. The class of function f with smoothness $k \in \mathbb{N}^+$ has at least a k-th derivative, and is denoted as $f \in C^k$,

$$C^k(\Omega) = \{f : \Omega \subseteq \mathbb{R}^d \to \mathbb{R} \mid \forall \alpha \leq k.\ \partial^\alpha f \text{ exists and is continuous}\}. \tag{92}$$

When $k = \infty$, the function is differentiable at all orders. While not every function is not smooth, there is a generalization of smooth functions, i.e. Sobolev functions.

The **weak derivative** f' generalizes to include functions that are not differentiable, but locally integrable on bounded domain $[a, b]$. The f' definition is for all smooth test functions ϕ, with $\phi(a) = \phi(b) = 0$,

$$\begin{aligned} \int_a^b f(x)\phi'(x)\ dx &= [f(x)\phi(x)]_a^b - \int_a^b f'(x)\phi(x)\ dx, \quad \text{by integration by parts,} \\ &= -\int_a^b f'(x)\phi(x)\ dx, \quad \text{as } \phi(a) = \phi(b) = 0. \end{aligned} \tag{93}$$

Sobolev spaces Adams & Fournier (2003) $W^{k,p}(\Omega)$ is a function space where all functions f having weak derivatives up to order k and every derivate is p-integrable via equation 91,

$$W^{k,p}(\Omega) \subset L^p(\Omega) = \{f : \Omega \subseteq \mathbb{R}^d \to \mathbb{R} \mid \forall \alpha \leq k.\ \exists\ \partial^\alpha f \in L^p(\Omega)\}, \tag{94}$$

When $k = 2$, it forms a Hilbert space, i.e. $W^{k,2}(\Omega) = H^k(\Omega)$.

F.3 APPROXIMATION THEORY OF NEURAL NETWORKS

We quote some known bounds for neural networks from theoretical machine learning field here, which are relevant to PDE solvers analysis later.

Universal approximation theorem Hornik et al. (1989) states that neural networks $\hat{u}_\theta$, for which parameters $\theta \in \Theta$, can approximate any continuous functions $u : \mathbb{R}^d \to \mathbb{R}$ with little error $\epsilon > 0$ in the p-norm of function space $\mathcal{U}$, with an extension to their differential operator $\mathcal{D}$,

$$\exists\ \theta \in \Theta.\ ||\hat{u}_\theta - u||_p < \epsilon \implies ||\mathcal{D}[\hat{u}_\theta] - \mathcal{D}[u]||_p < \epsilon. \tag{95}$$

Optimal DNN functions approximation theorem Yarotsky (2018). Assuming a continuous function $u \in W^{s,p}$ as defined in equation 94, where $s \in \mathbb{N}^+$ is the smoothness of u, there exists a neural network $\hat{u}_\theta$ with M parameters, such that the error bound is,

$$||\hat{u}_\theta - u||_p = O(M^{-\frac{s}{d}}), \tag{96}$$

where d is the input dimension of u. It means intuitively that the smoother and lower-dimensional the u is, the easier for it to be approximated by a neural network $\hat{u}_\theta$. For a fixed error ϵ, the required number of parameters is $M = O(\epsilon^{-\frac{d}{s}})$, which suffers from the exponential growth of d, i.e. **the curse of dimensionality** Bellman (1957).

F.4 ERROR ANALYSIS

Error and risk estimation. Define the *risk* of the approximation $\hat{u}_\theta$ against the ground truth function $u : \Omega \to \mathbb{R}$ with p-norm integral,

$$\mathcal{R}(\hat{u}_\theta) = (||\hat{u}_\theta - u||_p)^p := \int_\Omega |\hat{u}_\theta(x) - u(x)|^p\ dx, \tag{97}$$

For discretized computation, quadrature $\hat{\mathcal{R}}$ is used to approximate the integral risk $\mathcal{R}$ with N sample points from the dataset, where ω_k is the weight at each sample index k,

$$\underbrace{\int_\Omega |\hat{u}_\theta(x) - u(x)|^p\ dx}_{\text{integral } \mathcal{R}(\hat{u}_\theta)} = \underbrace{\sum_{k=1}^N \omega_k |\hat{u}_\theta(x_k) - u(x_k)|^p}_{\text{discretized } \hat{\mathcal{R}}(\hat{u}_\theta)} + \mathcal{E}_T(\theta) \approx \hat{\mathcal{R}}(\hat{u}_\theta), \quad \mathcal{E}_T(\theta) := \mathcal{R}(\hat{u}_\theta) - \hat{\mathcal{R}}(\hat{u}_\theta), \tag{98}$$

where the training, also known as generalization or out-of-sample, error $\mathcal{E}_T(\theta)$ measures the difference between the integral risk and the discretized risk due to quadrature.

Error decomposition Kutyniok (2022). When approximating a continuous function u with a neural network $\hat{u}_\theta$, the *total error* $\mathcal{E}(\theta)$[11] is decomposed into three parts, with the risk $\mathcal{R}$ and its quadrature $\hat{\mathcal{R}}$ defined in equation 97 and equation 98 respectively,

$$\mathcal{E}(\theta) := \mathcal{R}(\hat{u}_\theta) \leq \underbrace{\inf_{\theta^\star \in \Theta} \mathcal{R}(u_{\theta^\star})}_{\mathcal{E}_A(\theta)} + \underbrace{\hat{\mathcal{R}}(\hat{u}_\theta) - \inf_{\theta^\star \in \Theta} \mathcal{R}(u_{\theta^\star})}_{\mathcal{E}_O(\theta)} + \underbrace{\mathcal{R}(\hat{u}_\theta) - \hat{\mathcal{R}}(\hat{u}_\theta)}_{\mathcal{E}_T(\theta)}, \tag{99}$$

the *approximation error* $\mathcal{E}_A$ measures the risk between the best network approximation $u_{\theta^\star}$ and ground truth u, *optimization error* $\mathcal{E}_O$ measures the trained network result $\hat{u}_\theta$ deviation from the best network approximation, and *training error* $\mathcal{E}_T$ defined in equation 98.

F.5 PINN LEARNING THEORY

We briefly analyze the PINN error bound[12]. The total error between the optimal solution $\hat{u}_\theta$ and the ground truth u is shown in equation 84. However, during training, the network doesn't have access to the exact ground truth for u. Therefore, we aim to reduce the PDE residual instead.

$$\begin{aligned} \mathcal{E}_{\mathcal{R}}(\theta) = (||\mathcal{R}_\theta||_p)^p &= (||\mathcal{D}[\hat{u}_\theta] - f||_p)^p, \quad \text{by } equation\ 81. \\ &= ||\mathcal{D}[\hat{u}_\theta] - \mathcal{D}[u]||_p, \quad \text{by } equation\ 1. \\ &= ||\hat{f} - f||_p, \quad \text{by the definition of } \hat{f}, \end{aligned} \tag{100}$$

where $\hat{f} = \mathcal{D}[\hat{u}_\theta]$ is the approximated source function. In practice, this integral is approximated via quadrature, with training error defined in equation 82.

From the **theoretical** perspective, the goal is to derive that the total error $\mathcal{E}_{\text{PINN}}$ equation 84 is sufficiently small. To prove this, a sufficient condition is that the total error is bounded by the residual error $\mathcal{E}_{\mathcal{R}}$ equation 100, i.e. we can prove that the smallest residual error ensures the smallest total error.

$$\forall \theta \in \Theta.\ \mathcal{E}_{\text{PINN}}(\theta) \leq C\mathcal{E}_{\mathcal{R}}(\theta), \tag{101}$$

where C is a constant. By expansion of $\mathcal{E}_{\text{PINN}}$ equation 84 and $\mathcal{E}_{\mathcal{R}}$ equation 100, the abovementioned inequality equation 101 is equivalent to the following **coercivity** condition Ryck & Mishra (2022),

$$\forall \theta \in \Theta.\ ||\hat{u}_\theta - u|| \leq C||\hat{f} - f||_p, \tag{102}$$

By quadrature bound Iserles (2008), the smallest practical training error $\mathcal{E}_T$ equation 82 ensures the smallest residual error $\mathcal{E}_R$ equation 100, where C' is a constant,

$$\forall \theta \in \Theta.\ \mathcal{E}_{\mathcal{R}}(\theta) \leq C'[\mathcal{E}_T(\theta) + \mathcal{E}_u(N_{\mathcal{R}})], \tag{103}$$

and the extra term $\mathcal{E}_u(N_{\mathcal{R}})$ converges faster than $\frac{1}{N_{\mathcal{R}}}$, thus can be ignored given the increasing sampled quadrature points $N_{\mathcal{R}}$,

$$\begin{aligned} \mathcal{E}_u(N_{\mathcal{R}}) \sim o(\frac{1}{N_{\mathcal{R}}}) &\implies \frac{\mathcal{E}_u(N_{\mathcal{R}})}{\frac{1}{N_{\mathcal{R}}}} = 0, n \to \infty, \qquad \text{by the definition of little-o notation.} \\ &\implies \lim_{N_{\mathcal{R}} \to \infty} N_{\mathcal{R}} \mathcal{E}_u(N_{\mathcal{R}}) = 0, \qquad \text{by the definition of limit.} \end{aligned} \tag{104}$$

By the above two inequalities equation 101 and equation 103, the total error $\mathcal{E}_{\text{PINN}}$ equation 84 converges as the training error $\mathcal{E}_T$ equation 82 converges,

$$\forall \theta \in \Theta.\ \mathcal{E}_{\text{PINN}}(\theta) \leq CC'[\mathcal{E}_T(\theta) + o(\frac{1}{N_{\mathcal{R}}})]. \tag{105}$$

By Universal approximation theorem equation 95, the smoothness of the solution u ensures that the residual error $\mathcal{E}_{\mathcal{R}}(\theta) < \epsilon$ is sufficiently small. Given sufficient quadrature points $N_{\mathcal{R}}$, and smooth activation functions in the neural network $\hat{u}_\theta$ Iserles (2008),

$$\min_{\theta \in \Theta} \mathcal{E}_T(\theta) \leq \mathcal{E}_{\mathcal{R}}(\theta) + o(\frac{1}{N_{\mathcal{R}}}), \tag{106}$$

[11] $\inf_\theta$ is the infimum over all possible network parameters θ, which might not be attained.

[12] The PDE residual is considered here, whereas boundary and initial conditions are omitted for simplicity.

Hence, the training error $\mathcal{E}_T(\theta) < \epsilon + o(\frac{1}{N_\mathcal{R}})$ is sufficiently small, according to equation 104. So is the total error $\mathcal{E}_{\text{PINN}}(\theta) < CC'[\epsilon + o(\frac{1}{N_\mathcal{R}})]$, by equation 105, which concludes the proof.

From the **practical** perspective, the common failure modes, from the above theoretical analysis, are (1) few quadrature points $N_\mathcal{R}$ leading to large training error $\mathcal{E}_T$ in equation 98, (2) insufficient training resulting in large optimization error $\mathcal{E}_O$ in equation 99, (3) violation of the coercivity condition equation 102 for PDEs, and (4) large constant C in equation 101 or C' in equation 103.