Technical Report

UCAM-CL-TR-1012
ISSN 1476-2986

Number 1012



UNIVERSITY OF CAMBRIDGE

Computer Laboratory

Neural solver for Partial Differential Equations in scientific simulation

Zheyuan Hu

November 2026

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

https://www.cl.cam.ac.uk/

© 2026 Zheyuan Hu

This technical report is based on a dissertation submitted June 2026 by the author for the degree of Master of Engineering (Computer Science Tripos) to the University of Cambridge, Magdalene College.

The author was supervised by Dr Chenliang Zhou, with partial inspiration for this work provided by Dr Fangcheng Zhong.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

https://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986
DOI *https://doi.org/10.48456/tr-1012*

Abstract

Partial Differential Equations offer a mathematically rigorous framework for graphical, biological, and physical phenomena. However, the numerical approaches are hindered by the restrictions of linearity, dimensionality, and domain-specific tuning. I present a generalized framework to solve coupled and nonlinear PDEs with auto tuning, alongside its application to both forward and inverse scientific simulation problems. With an empirical framework for stability proposed, I mitigate solver sensitivity to input perturbations by introducing regularization strategies. Through a comprehensive survey of PDEs with their applications, different types of state-of-the-art baseline methods, including FDM, PINN, FNO, GNN-based solvers, and the learning theory behind them, I systematically evaluate the proposed method across multiple critical dimensions, i.e. effectiveness quantified via error metrics, efficiency characterized by computational cost or convergence rate, and stability. The results contrast each solver strengths and limitations, highlighting the high precision, flexible kernel design, and utility in various scientific simulation contexts of the proposed method.

Keywords: PDE solver, scientific computing, forward simulation, inverse problems, neural networks, self-tuning, stability, accuracy, learning theory.

Acknowledgements

I'm deeply indebted to Dr Chenliang Zhou and Dr Fangcheng Zhong, for mentoring and introducing me into the exciting field of scientific computing. Their insightful guidance and consistent feedback have been instrumental in shaping my research journey.
My sincere thanks go to my University Teaching Officer (UTO), Professor Cengiz Öztireli, for all his support in past years, as well as to Mr Weitao Chen, for his invaluable discussions and suggestions on the mathematical and theoretical aspects of this work. Deep gratitude is owed to my Director of Studies, Dr John Fawcett, for his invaluable insights throughout my undergraduate studies.
Without valuable feedback from the anonymous reviewers during earlier versions of this work submitted to workshops and conferences, this thesis would not have reached its current form. I am deeply grateful to them. Finally, special thanks to my familiy members, fellow students, and friends for their unwavering support and encouragement throughout this journey. Their belief in me has been a continuous source of motivation.

Contents

Contents **5**
List of Figures . . . 7
List of Tables . . . 9

1 Introduction **11**
1.1 Motivation . . . 11
1.2 Research scope and objectives . . . 11
1.3 Related work . . . 12

2 Background **14**
2.1 Foundational theories . . . 14
2.1.1 Linear operator . . . 14
2.1.2 Functional analysis . . . 14
2.1.3 Stability analysis . . . 14
2.2 Partial differential equations . . . 15
2.2.1 General PDEs formulation . . . 15
2.2.2 Initial and boundary conditions . . . 16
2.2.3 Advection equation . . . 16
2.2.4 Predator-prey Lotka-Volterra equations . . . 17
2.2.5 Maxwell's equations . . . 17
2.2.6 Viscous Burgers' equation . . . 19
2.3 PDE solvers . . . 21
2.3.1 Finite difference method (FDM) . . . 21
2.3.2 Physics-informed neural network (PINN) . . . 22
2.3.3 Neural operator (NO) . . . 23
2.3.4 Graph Neural Network (GNN) . . . 25
2.3.5 Kansa method (KM) . . . 26
2.3.6 Summary of PDE solvers . . . 27
2.4 Types of simulation . . . 27
2.5 Community contributions . . . 28

3 Implementation **29**
3.1 PDE domain and dataset design . . . 29
3.2 PDE solvers architecture design details . . . 29
3.3 Kansa algorithm . . . 30
3.3.1 Linear operator case . . . 31
3.3.2 Extension I: stability via Tikhonov regularization . . . 32
3.3.3 Extension II: coupled PDEs solution fields . . . 32

3.3.4 Extension III: nonlinear operator case . . . 33
3.3.5 Kernel parameter auto tuning . . . 35
3.4 Infinite-width ReLU networks kernel . . . 35
3.5 Inverse problem solutions for PDEs . . . 36
3.6 Applications of proposed methods . . . 36
3.6.1 Deployment techniques . . . 36
3.6.2 Sparse matrix operators . . . 36
3.6.3 Application I: Gray-Scott pattern formation . . . 37
3.6.4 Application II: Physics simulation . . . 38

4 Evaluation **40**
4.1 Quantitative metrics . . . 40
4.2 Experimental configuration . . . 40
4.3 Assessment of solving advection equation . . . 41
4.3.1 Forward solution . . . 41
4.3.2 Inverse estimation . . . 42
4.4 Extension I: accuracy-stability trade-off . . . 42
4.5 Extension II: coupled PDEs . . . 43
4.5.1 Forward solution . . . 44
4.5.2 Inverse estimation . . . 44
4.6 Extension III: nonlinear PDEs . . . 45
4.6.1 Forward solution . . . 45
4.6.2 Inverse estimation . . . 46
4.7 Applications of proposed methods . . . 47

5 Conclusions **49**
5.1 Summary and contributions . . . 49
5.2 Lessons learned and reflections . . . 49
5.3 Future work . . . 50

Bibliography **51**

A Linear operator examples **58**

B Learning theory **59**
B.1 Functional analysis addendum . . . 59
B.2 Approximation theory of neural networks . . . 59
B.3 Error analysis . . . 60
B.4 PINN learning theory . . . 60

Acronyms **62**

List of Figures

1.1 PDE formulation, solvers, and applications in scientific or visual computing. . . 11
1.2 Dependency analysis of the project scope and objectives. 12

2.1 An example of PDE computational domain $\Omega \subset \mathbb{R}^2$ with internal, boundary and near-boundary collocation points, i.e. $x \in \Omega$, $x \in \partial\Omega$ and $x \in \Omega_{\text{near-boundary}}$ respectively. The boundary $\partial\Omega$ consists of multiple segments $\{\partial\Omega_i\}_{i=1}^{n}$. Here, I assume uniform sampling in the 2D domain for collocation points, but can be generalized to non-uniform one. 15
2.2 Graphical representations of the 1D advection equation analytical results in one-, two-, and three-dimensional plots. 16
2.3 Dynamical and phase-space behaviours of predator-prey Lotka-Volterra equations. 17
2.4 Graphical representations of Maxwell's equations analytical results in one-, two-, and three-dimensional plots. 19
2.5 Graphical representations of the Burgers' equation analytical results in one-, two-, and three-dimensional plots. 20
2.6 PINN architecture, where parameters θ are optimized by reducing the total training loss $\mathcal{L}_{\text{PINN}}$ of the physics-informed loss $\mathcal{L}_R$, with optional $\mathcal{L}_{\text{BCs}}$, $\mathcal{L}_{\text{ICs}}$ and supervised loss $\mathcal{L}_{\text{data}}$. 23
2.7 The architecture diagrams of FNO (left) and MP-GNN (right). 25

3.1 Left: Illustration of PDE, domains, and boundaries. Middle: High-level flowchart of the baseline methods, including PINN, NO and GNN. Right: The proposed approach with its extensions and features. 29
3.2 UML diagram of PDE data domain and dataset. 30

4.1 Training history of Kansa method with auto tuning of shape parameter ϵ. 43
4.2 Stability and accuracy of Kansa method on 1D advection equation. 43
4.3 Gray-Scott patterns [Tur52, WK91], drawn concurrently using the proposed Kansa framework of truncated kernels (2.65) and sparse matrix solver (§ 3.6.2). 47
4.4 Simulation results of a mass-spring grid, where Burgers' equation is the acceleration, solved by the proposed method and without interaction of material or light. 47
4.5 Rendering of a mass-spring grid, where Burgers' equation is the acceleration, computed by the proposed method and with interaction of material and light. . 47
4.6 Time-varying visualization of the heat equation on a bunny manifolds solved by the proposed method. The yellow-to-orange color mapping represents high to low temperature. 47

4.7 Particle-based simulation with a diffusion-reaction temperature field, governing the motion by buoyancy, stochastic turbulence and damping forces, computed by the proposed method. The yellow-to-orange color mapping represents from high to low temperature. 48
4.8 The same particle-based model as above, but with a wider initial particle density distribution along the vertical axis. These figures illustrate the frames after 2306 simulation steps, showcasing the effect of the asynchronous respawn logic (§ 3.6.4). 48
4.9 The same particle-based model as above, but with the widest initial particle density distribution along the vertical axis. These figures illustrate every five frames after 14475 simulation steps, showcasing the effect of the asynchronous respawn logic (§ 3.6.4). 48

List of Tables

1.1 Classification of PDEs (Eq.) based on their name, application areas, differential operators (op.) linearity, and the dimensionality of the solution u. 12

2.1 Summary of different PDE solvers, where Auto tuning means that the method can automatically adjust its parameters during training, Label-free refers that unsupervised training is possible without labeled data, and Exact enforcement denotes whether the method enforces hard constraints (equality or inequality forms) or soft constraints (penalty functions). 27

2.2 Comparison of my proposed two-stage framework with the prior art, considering whether equations are **M**ultivariable (more than one independent variables), **C**oupled (more than one dependent variables) or **N**onlinear, and if **S**tability analysis is included. 28

3.1 Overview of main features among nonlinear Kansa solvers proposed, where N_t and N_x represent the number of spatio-temporal collocation points, and Δt denotes the time step interval. Fully nonlinear excels in accuracy order but is computationally expensive, while time stepping approaches are the exact opposite. 34

4.1 Benchmark configuration for the 1D advection equation. 41

4.2 Models correctness $\hat{\mathcal{R}}_{\text{relative } L_2} \times 10^{-3}$ ($\downarrow$) for the 1D advection with respect to the computational grid resolution. 42

4.3 Solvers stability S_{L_∞} ($\downarrow$) for the 1D advection with respect to the computational grid resolution. 42

4.4 Estimation of β in the advection equation, for which reference is $\beta = 0.4$. 42

4.5 Benchmark configuration for the Lotka-Volterra equations. 44

4.6 Benchmark configuration for the 1D Maxwell's equations. 44

4.7 $\hat{\mathcal{R}}_{\text{relative } L_2}$ error ($\downarrow$) of the Kansa framework applied to Lotka-Volterra and Maxwell's equations, with respect to the type of basis functions and computational grid resolution. 44

4.8 Estimation of α, β, δ and γ in the Lotka-Volterra solved by the proposed method. 45

4.9 Benchmark configuration for the 1D Burgers' equation. 45

4.10 Models $\hat{\mathcal{R}}_{\text{relative } L_2} \times 10^{-2}$ error ($\downarrow$) on Burgers' equation. Fully nonlinear approach outperforms other time discretization schemes, e.g. Forward, Backward Euler, IMEX and C-N. 45

4.11 Train or infer cost (seconds) for the Burgers' equation across all solvers, where Fw. and Bw. stand for Forward and Backward Euler, respectively. Time stepping schemes have lower computational complexity, thus less training time but more inference time than fully nonlinear approach. The latter reuses the coefficients from training phase for inference. 46

4.12 Investigation of stability on the Burgers' equation for the forward Euler scheme. 46
4.13 Estimation of ν in the Burgers' equation solved by the proposed methods, for which the reference value is $\nu = 0.5$. 46

1 Introduction

Galileo Galilei — *"Mathematics is the language in which God has written the universe."*

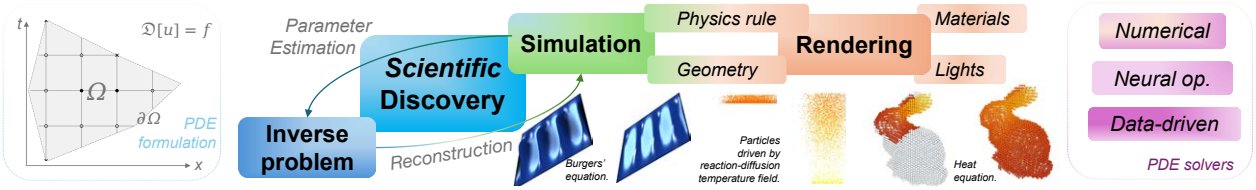


Figure 1.1: PDE formulation, solvers, and applications in scientific or visual computing.

§ 1.1 Motivation

PDEs are fundamental in scientific computing, arising from physics, graphics, and biology. They take the general form below, and together with the initial and boundary conditions,

$$\mathcal{D}[u](x) = f(x), \tag{1.1}$$

where $\mathcal{D}$ is a differential operator acting on the unknown dependent variable $u(x)$, and $f(x)$ is a given function representing sources or sinks in a system. PDEs model how measurable variables evolve across spatial and temporal domains, thereby serving as a cornerstone for electromagnetic, hydrodynamic, heat, and biological phenomena. To solve PDEs, numerous methods have been proposed, including classical numerical, spectral, data-driven, and hybrid methods (Table 2.1). In this work, I research one of the spectral methods, i.e. Kansa Method, with evaluation alongside other state-of-the-art methods for comparison.

Numerical approaches, e.g. Finite Difference Method (FDM) [Ise08] and Finite Element Method (FEM) [BS18], have been extensively employed in the solution of PDEs. Constrained Neural Fields (CNF) [ZFH+23] introduced a broadened version of the Kansa method [Kan90], which uses Radial Basis Functions (RBFs) within a mesh-free PDE solution framework. They facilitate the self-tuning of the RBFs kernel shape. That said, their work addresses solely on *linear* PDEs with multiple independent variables but a *single* dependent variable. In addition, the *stability* of the solver remains largely underexplored, which has been addressed in this work.

Data-driven PDE solvers, e.g. Physics-Informed Neural Network (PINN) [RPK19], Neural Operator (NO) [LKA+20, LJP+21, LKA+21] and Graph Neural Network (GNN) [BWW22], have shown promising results in addressing the issues of classical numerical methods, e.g. application-tailored discretisation, and elevated computational expense. However, they either require a paired dataset or retraining for new instances, which limits their generalization and scalability.

§ 1.2 Research scope and objectives

Solver extensions. Motivated by the above limitations, this project solves *coupled* and *non-linear* PDEs beyond linearity, alongside applications to specific scientific simulation, i.e. *forward* computation and *inverse* problems, while preserving the properties of hard constraints.

To reduce sensitivity to extreme perturbations, I augment solvers with regularization from the *stability analysis*, and suggest a framework to empirically evaluate model or solver stability. Overall, I propose a *unified* Kansa method framework with the support of flexible kernel designs, auto tuning, which are applicable to various types of PDEs.

Evaluation and applications. It is unknown how CNF [ZFH+23] compares against alternative classical and neural PDE methods. Inspired by this, I implement four different types of baseline methods, i.e. FDM, PINN, FNO, GNN-based, and evaluate the proposed method against them on the benchmarks for (i) *effectiveness* using different quality metrics relative to the reference, (ii) *computational efficiency*, (iii) *stability*, and lastly (iv) *utility* in scientific simulation contexts or hybrid pipelines incorporating techniques such as differentiable rendering for inverse problems in physics-based visual computing, and forward synthesis thereafter.

Theoretical foundations and underlying learning theory. I additionally review the learning theory behind neural PDE solvers, for instance error bound, convergence analysis and generalization analysis. This helps to understand the strengths and limitations of different methods, practical guidelines for their application, and potential areas for improvement.

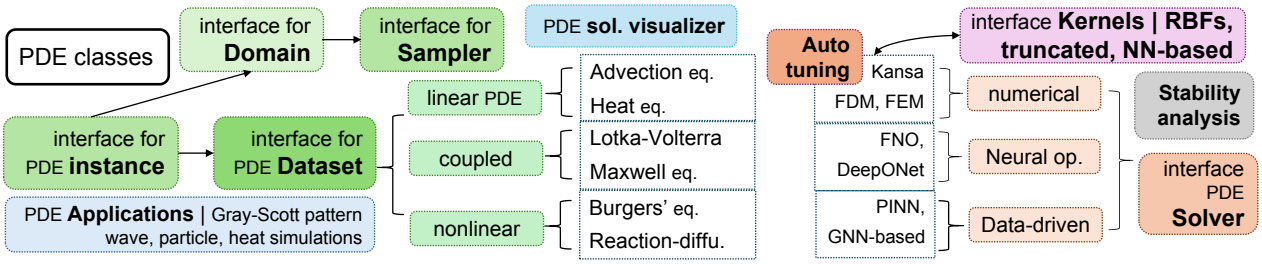


Figure 1.2: Dependency analysis of the project scope and objectives.

Contributions. Part of this work has been published in peer-reviewed International Conference on Learning Representations (ICLR) 2026 AI-PDE workshop, as detailed in § 2.5.

§ 1.3 Related work

PDE benchmarks. From existing PDE surveys [TPL+22], I identified several representative equations [TPL+22] in § 2.2 and Table 1.1. They vary with the linearity of differential operators and the dimensionality of the solution, also termed the independent variable, u in the equation.

Table 1.1: Classification of PDEs (Eq.) based on their name, application areas, differential operators (op.) linearity, and the dimensionality of the solution u.

Name	Areas	Is op. linear?	Dim. of u
Advection Eq.	Visual, Fluid	✓	1
Wave Eq.	Visual, Hydro	✓	1
Lotka-Volterra	Population	✓	**2**
Maxwell Eqs.	Electromagnetics	✓	**2**
Burgers Eq.	Visual, Shock	×	1

PDE solvers overview. A summary of the most relevant and representative PDE methods (§ 2.3) is shown in Table 2.1. The detailed methodology for each one of them is described in the later chapters. I include a concise summary below by categorizing them into classical numerical, data-driven and neural operator methods.

Classical PDE solvers. Numerical methods, e.g. Finite Difference Method (FDM) [Ise08] and Finite Element Method (FEM) [BS18] have been broadly adopted for solving PDEs. Kansa Method (KM) [Kan90] is a mesh-free FEM that uses RBFs as basis functions. However, it requires careful tuning of the shape parameters of RBFs, which is non-trivial and can lead to instability [CFC14]. The restriction of linearity limits its applicability.

Data-driven PDE solvers. Prior works begin by utilizing statistical learning methods, e.g. sparse regression [RBPK17], or gradient descent, e.g. Physics-Informed Neural Network (PINN) [RPK19] to minimize the residual of a single PDE instance by training on a set of collocation points. They model the solution as either combinations of nonlinear or partial derivative terms [RBPK17] or as a neural network [RPK19]. Alternative approaches include predicting the solution field auto-regressively in time steps via a Graph Neural Network (GNN) [GSR+17]. However, the single-instance training paradigm lacks generalization to unseen scenarios and thus requires retraining for each new PDE instance [BWW22].

Neural operators. To address the generalization issue, operator learning methods are established to infer a dependency from the PDE input, i.e. the initial condition, boundary condition, parameters, or source term, to the solution. Such operators are either finite-dimensional, e.g. [GLI16, KLY21, ZZ18], and thus mesh-dependent, or infinite-dimensional, e.g. DeepONet [LJP+21], Fourier Neural Operator (FNO) [LKA+21], which are mesh-independent and thus more flexible. They excel in generalization to unseen scenarios without retraining, which is a significant advantage over data-driven solvers above. Nevertheless, a paired dataset is a must due to its supervised learning nature, which is expensive to obtain, store, and scale up.

Inverse problems. One of the most important applications is to estimate unknown parameters or input conditions of variable x from given solution observations u [RCLV11, WLW22, WW24]. In addition, in the field of Visual Computing, differentiable rendering has been used to solve inverse physics-related problems, e.g. estimating geometry or material properties from images [ZHS+26, Hu26], or videos [DBC+17]. Nonetheless, the applicability of CNFs to these problems is not yet evident. In this work, I demonstrate how to apply it to solve inverse problems, paving the way for integration with hybrid pipelines to solve physics-driven visual problems [SZRea23], with forward synthesis as a downstream application.

Stability. Solvers are expected to perform well under source measurement errors or perturbations [Ise08]. Modern neural-network-based PDE solvers are not thoroughly evaluated for stability. Recent work utilized uncertainty quantification [ZZ18, PMZ+23] to assess reliability, but is different from stability analysis. In this work, I suggest a framework to measure model or solver stability empirically, and augment solvers with regularization.

2 Background

Albert Einstein — *"It is the theory which decides what we can observe."*

This chapter introduces the foundational theories behind linear operators and functional analysis (§ 2.1). Subsequently, I define the general PDE formulation (§ 2.2), followed by concrete instances (Table 1.1), with derived analytical solutions, which is adopted for benchmarking the proposed method against other PDE solvers. I explore a series of PDE solvers in § 2.3, with implementation details and experimental results (§ 3.2, 4), and the field of scientific simulation tasks (§ 2.4) that benefit from PDE solvers. Community contributions are discussed in § 2.5.

§ 2.1 Foundational theories

§ 2.1.1 Linear operator

The **linear operator** $\mathcal{F} : V \to W$ is a mapping of vector spaces $V, W \in \mathbb{R}$ or between itself[1], that satisfies *scalar multiplication*, otherwise called *homogeneity*, and *vector addition* [Ise08]. Consequently, taking any scalar c and arbitrary vectors $\mathbf{u_i} \in V$, we have properties as follows,

$$\begin{aligned} \mathcal{F}(c \cdot \mathbf{u}) &= c \cdot \mathcal{F}(\mathbf{u}), \quad \text{homogeneity}, \\ \mathcal{F}(\sum_{i=1}^{n} \mathbf{u}_i) &= \sum_{i=1}^{n} \mathcal{F}(\mathbf{u}_i), \quad \text{additivity}. \end{aligned} \tag{2.1}$$

They are central to matrix processing in Linear Algebra, Artificial Intelligence, Quantum Computing, and many more. Examples are given in Appendix § A. The theoretical basis of the linear Kansa method is based on the properties of linear operators.

§ 2.1.2 Functional analysis

I recall fundamental concepts from functional analysis relevant to PDE solvers and subsequent learning theory in Appendix § B. I define the p-**norm** for the function $f : \Omega \subseteq \mathbb{R}^d \to \mathbb{R}$ below,

$$||f||_p = (\int_\Omega |f(x)|^p \, dx)^{\frac{1}{p}}, \text{for } p \in [1, \infty), \tag{2.2}$$

When $p = \infty$, the p-norm is thus the maximum absolute value of the function,

$$||f||_\infty = \sup_{x \in \Omega} |f(x)|, \tag{2.3}$$

where supremum is the least upper bound. When this upper bound is reached, the sup is reduced to max. A function $f \in L^p(\Omega)$ is called p-**integrable** when,

$$(||f||_p)^p = \int_\Omega |f(x)|^p \, dx < \infty. \tag{2.4}$$

For completeness, please refer to Appendix § B.1 for concepts related to the learning theory.

§ 2.1.3 Stability analysis

Here, I discuss the PDE solvers numerical stability [Ise08], not to be confused with the properties of PDEs themselves, under noisy or perturbed initial conditions. It is important in practical settings in which measurements are affected by systematic or random errors.

[1] If the domain and codomain coincide $W = V$, the operator F is called an endomorphism on V.

In numerical analysis [Fau16], the **condition number** of a function measures asymptotically worst case change of its output values with respect to a tiny input perturbation.

Derivation. For linear equation systems $\mathbf{Fa} = \mathbf{h}$, we are given input $\mathbf{h}$ and aim to solve for output $\mathbf{a}$, i.e. the matrix $\mathbf{F}^{-1}$ is the linear operator mapping $\mathbf{h}$ to $\mathbf{a}$. When given a perturbation $\delta\mathbf{h}$ on the right-hand side, the exact solution satisfies $\mathbf{Fa}(1+\delta) = \mathbf{h}(1+\delta)$. The difference between them yields $\delta\mathbf{a} = \mathbf{F}^{-1}\delta\mathbf{h}$. The last equation and original linear equation under the p-norm are, and by multiplying the two inequalities together,

$$\begin{cases} ||\mathbf{F}^{-1}||_p \cdot ||\delta\mathbf{h}||_p & \geq ||\mathbf{F}^{-1}\delta\mathbf{h}||_p = ||\delta\mathbf{a}||_p, \\ ||\mathbf{F}||_p \cdot ||\mathbf{a}||_p & \geq ||\mathbf{Fa}||_p = ||\mathbf{h}||_p. \end{cases} \implies \underbrace{||\mathbf{F}||_p \cdot ||\mathbf{F}^{-1}||_p}_{\kappa_p(\mathbf{F})=\kappa_p(\mathbf{F}^{-1})} \cdot \underbrace{\frac{||\delta\mathbf{h}||_p}{||\mathbf{h}||_p}}_{\delta_{\text{input}}} \geq \underbrace{\frac{||\delta\mathbf{a}||_p}{||\mathbf{a}||_p}}_{\delta_{\text{output}}}. \tag{2.5}$$

A low condition number bounds the output change δ_{output} and thus indicates **well-conditioned** and stable, otherwise it indicates that the system is **ill-conditioned** and unstable. For a chosen norm $p = 2$ as an example, the matrix condition number $\kappa_p(\mathbf{F})$, where the subscript for $p = 2$ is omitted for brevity in later chapters, is as follows,

$$\kappa_2(\mathbf{F}) = ||\mathbf{F}||_2 \cdot ||\mathbf{F}^{-1}||_2 = \frac{\sigma_{\max}(\mathbf{F})}{\sigma_{\min}(\mathbf{F})}, \tag{2.6}$$

where $\delta_{\max}(\mathbf{F}), \sigma_{\min}(\mathbf{F})$ are the largest and smallest singular values of $\mathbf{F}$, and defined as,

$$\sigma_i(\mathbf{F}) = \sqrt{\lambda_i(\mathbf{F}^T\mathbf{F})}, \quad i = 1, \ldots, n, \tag{2.7}$$

where $\lambda_i(\mathbf{F}^T\mathbf{F})$ are the eigenvalues of $\mathbf{F}^T\mathbf{F}$. When the matrix $\mathbf{F}$ is normal, i.e. $\mathbf{F}^T\mathbf{F} = \mathbf{F}\mathbf{F}^T$, one can prove that $\lambda_i(\mathbf{F}^T\mathbf{F}) = |\lambda_i(\mathbf{F})|^2$. Thus the singular values (2.7) are the absolute values of the eigenvalues, i.e. $\sigma_i(\mathbf{F}) = |\lambda_i(\mathbf{F})|$. Thus, the condition number (2.6) is expressed as,

$$\kappa_2(\mathbf{F}) = \frac{|\lambda_{\max}(\mathbf{F})|}{|\lambda_{\min}(\mathbf{F})|}. \tag{2.8}$$

§ 2.2 Partial differential equations

Given the computational domain Ω, also termed region or grid, with d being the spatial dimension, I illustrate three types of collocation points $x \in \Omega \subset \mathbb{R}^d$ and their relationship with the domain boundary $\partial\Omega$ in Figure 2.1 and its caption.

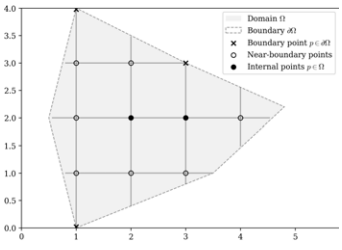


Figure 2.1: An example of PDE computational domain $\Omega \subset \mathbb{R}^2$ with internal, boundary and near-boundary collocation points, i.e. $x \in \Omega$, $x \in \partial\Omega$ and $x \in \Omega_{\text{near-boundary}}$ respectively. The boundary $\partial\Omega$ consists of multiple segments $\{\partial\Omega_i\}_{i=1}^n$. Here, I assume uniform sampling in the 2D domain for collocation points, but can be generalized to non-uniform one.

§ 2.2.1 General PDEs formulation

Define the spatial domain to be $\Omega \subset \mathbb{R}^d$, with d denoting the dimensionality. The scalar-valued independent variable $u(x,t) \in \mathcal{U} : \mathbb{D} \to \mathbb{R}$ over the spatio-temporal domain $\mathbb{D} = \Omega \times [t_0, t_f] \subset \mathbb{R}^{d+1}$ is solution to the PDEs of the general formulation defined below,

$$\begin{cases} \mathcal{D}[u] = f, & x \in \Omega, t \in [t_0, t_f], \\ \mathcal{B}_i[u] = g_i, & x \in \partial\Omega_i, t \in [t_0, t_f]. \end{cases} \Leftrightarrow \begin{cases} \mathcal{D}[u](x,t) = f(x,t), & x \in \Omega, t \in [t_0, t_f], \\ \mathcal{B}_i[u](x,t) = g_i(x,t) & x \in \partial\Omega_i, t \in [t_0, t_f]. \end{cases} \tag{2.9}$$

namely differential operators $\mathcal{D} : \mathcal{U} \to \mathcal{Y}$ and source function $f \in \mathcal{Y} : \mathbb{D} \to \mathbb{R}^m$, such as the input force in physics, where m denotes f output dimensionality[2]. Representative differential operators $\mathcal{D}$ are gradient ∇, divergence $\nabla\cdot$, Laplacian Δ. For BCs, per boundary operator is $\mathcal{B}_i : \mathcal{U} \to \mathcal{Z}_i$ and $g_i \in \mathcal{Z}_i : \partial\Omega_i \times [t_0, t_f] \to \mathbb{R}^{n_i}$, where n_i is g_i output dimensionality.

Coupled PDEs are defined as a system of equations that involve multiple unknown fields, e.g. $u_1, u_2, \ldots, u_k$, which are interdependent and interact with each other.

§ 2.2.2 Initial and boundary conditions

Consider that PDE solutions are determined up to integration constants, which means uniqueness is not guaranteed. To derive a unique solution, one needs to specify additional constraints. Initial Conditions (ICs) initialize function u at the $t = 0$ time slice, whereas Boundary Conditions (BCs) prescribe constraints over the domain boundary $\partial\Omega$, from the spatial perspective. As per the general form of PDEs defined in (2.9), I introduce a few widely adopted types of BCs, defined over the 1D boundary $\Omega = [x_0, x_f]$, and given closed-form functions $\{g_i\}_{i=1}^4$,

$$\begin{aligned} \text{Constant (Zero) BCs:} \quad & u(x_0, t) = 0, \; u(x_f, t) = 0, \\ \text{Dirichlet BCs:} \quad & u(x_0, t) = g_1(t), \; u(x_f, t) = g_2(t), \\ \text{von Neumann BCs:} \quad & \frac{\partial u}{\partial x}(x_0, t) = g_3(t), \; \frac{\partial u}{\partial x}(x_f, t) = g_4(t). \end{aligned} \tag{2.10}$$

§ 2.2.3 Advection equation

The 1D advection equation [TPL+22] governs the linear propagation of a scalar field $u(x, t)$, capturing its variation in space x and time t,

$$\begin{cases} \frac{\partial u(x,t)}{\partial t} + \beta \frac{\partial u(x,t)}{\partial x} = 0, & x \in [x_0, x_f], t \in [t_0, t_f], \\ u(x, 0) = u_0(x), & x \in [x_0, x_f], \quad \text{initial condition}, \end{cases} \tag{2.11}$$

with the real-valued $\beta \in \mathbb{R}$ being the transport speed, and $u_0(x)$ defines the starting state at $t = 0$. The exact analytical result to (2.11) takes the form of,

$$u(x, t) = u_0(x - \beta t). \tag{2.12}$$

Wave propagation direction is determined by the sign of β. In relation (2.12), positive value of $\beta > 0$ induces a rightward wave propagation, reciprocally. Figure 2.2 depicts the solution for the initial condition $u_0(x) = \sin(2\pi x)$, $x \in [0, 1]$.

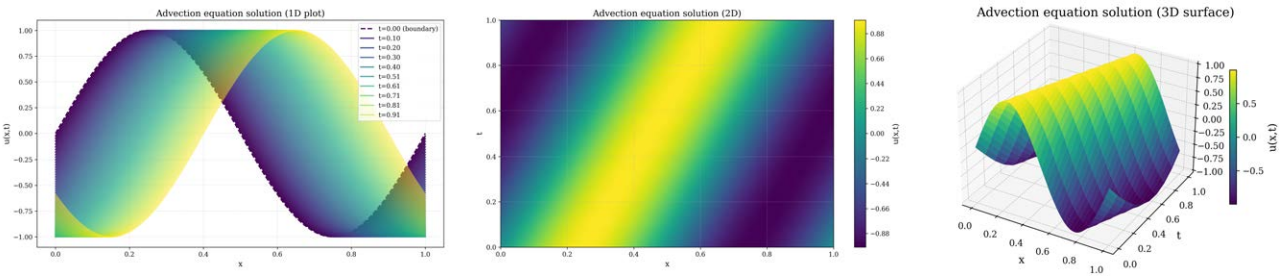


Figure 2.2: Graphical representations of the 1D advection equation analytical results in one-, two-, and three-dimensional plots.

[2] I assume that $\mathcal{U}$ and $\mathcal{Y}$ lie in two function spaces that are Banach spaces.

§ 2.2.4 Predator-prey Lotka-Volterra equations

Predator-prey Lotka-Volterra equations [Bac11] couple the prey population $x(t)$ and predators $y(t)$ within a time-evolving ecological framework by coupled PDEs. The model extends to other areas, such as the association between wage growth and unemployment rate dynamics [OS21],

$$\begin{cases} x'(t) := \frac{dx(t)}{dt} = \alpha x(t) - \beta x(t) \cdot y(t), \\ y'(t) := \frac{dy(t)}{dt} = \delta x(t) \cdot y(t) - \gamma y(t). \end{cases}, \quad t \in [t_0, t_f]. \tag{2.13}$$

in which α, β act as the prey growth and predation rates, respectively. δ denotes the fraction of consumed prey that converts into new predators, while γ represents the death proportion of predators. The assumption of unbounded prey population growth leads to the super-polynomial term $\alpha x(t)$ in the first differential equation, while the interaction term $x(t) \cdot y(t)$ statistically captures the chance of prey caught by predators. Despite the absence of a closed form solution, I derive the implicit representation of the solution by a change of variables,

$$x^*(t) = \frac{\delta}{\gamma} x(t), \quad y^*(t) = \frac{\beta}{\alpha} y(t), \quad \tau = \alpha t, \tag{2.14}$$

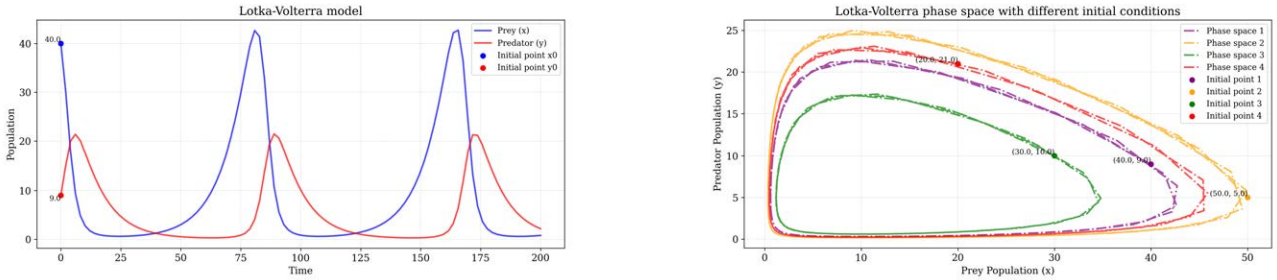
Upon division of the first and second equations in (2.13) and applying the new variables,

$$\frac{dy^*}{dx^*} = \frac{\gamma}{\alpha} \cdot \frac{y^*(x^* - 1)}{x^*(y^* - 1)}, \tag{2.15}$$

Using separation of variables and integrating both sides, I obtain the implicit solution, with $C_{\text{L-V}} \in \mathbb{R}$ as the associated integration constant determined by the initial conditions $x(0)$ and $y(0)$,

$$\ln(y^*) - y^* = \frac{\gamma}{\alpha}[\ln(x^*) - x^*] + C_{\text{L-V}}. \tag{2.16}$$

The phase-space trajectory, i.e. closed curves, of Lotka-Volterra equations is in Figure 2.3.



(a) Time-evolving prey and predator solutions

(b) Phase plane orbit

Figure 2.3: Dynamical and phase-space behaviours of predator-prey Lotka-Volterra equations.

§ 2.2.5 Maxwell's equations

Maxwell's equations [Gre98] establish the electromagnetic relationship between the electric $\mathbf{E}(\mathbf{r}, t)$ and magnetic $\mathbf{B}(\mathbf{r}, t)$ fields, both functions of spatial coordinates $\mathbf{r} \in \mathbb{R}^3$ and time t. $\rho(\mathbf{r}, t) \in \mathbb{R}$ and $\mathbf{J}(\mathbf{r}, t) \in \mathbb{R}^3$ denote the electric charge and current density, respectively.

$$\begin{cases} \nabla \cdot \mathbf{E} = \frac{\rho}{\epsilon_0}, & \text{Gauss's law,} \\ \nabla \cdot \mathbf{B} = 0, & \text{Gauss's law for magnetism,} \\ \nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}, & \text{Faraday's law of induction,} \\ \nabla \times \mathbf{B} = \mu_0 \mathbf{J} + \mu_0 \epsilon_0 \frac{\partial \mathbf{E}}{\partial t}, & \text{Ampère-Maxwell law,} \end{cases} \tag{2.17}$$

where $\epsilon_0, \mu_0 \in \mathbb{R}^+$ represent the permittivity and permeability of free space. The product of these constants equals the reciprocal of the squared vacuum light speed $c \approx 3 \times 10^8$ m $\cdot$ s^{-1},

$$\mu_0 \epsilon_0 = \frac{1}{c^2}. \tag{2.18}$$

Gauss's laws for electricity and magnetism indicate that electric charges act as sources of the **E** field, whereas the magnetic monopole is absent. The remaining equations describe the dynamic coupling between **E** and **B** fields, i.e. how one field induces another and the other way around. **J** denotes the current density. In terms of the differential operators, all equations are linear with regards to **E** and **B**.

Applying the curl operator $\nabla\times$ to the last two equations of Maxwell's equations (2.17) yields,

$$\begin{cases} \nabla \times (\nabla \times \mathbf{E}) = -\frac{\partial}{\partial t}(\nabla \times \mathbf{B}), \\ \nabla \times (\nabla \times \mathbf{B}) = \mu_0(\nabla \times \mathbf{J}) + \mu_0\epsilon_0 \frac{\partial}{\partial t}(\nabla \times \mathbf{E}). \end{cases} \tag{2.19}$$

Employing the identity $\nabla \times (\nabla \times \mathbf{E}) = \nabla(\nabla \cdot \mathbf{E}) - \Delta\mathbf{E}$ from Vector Calculus, and inserting the Ampère-Maxwell law into the expression on the right side in the preceding equation of (2.19),

$$\begin{cases} \nabla(\nabla \cdot \mathbf{E}) - \Delta\mathbf{E} = -\mu_0 \frac{\partial \mathbf{J}}{\partial t} - \mu_0\epsilon_0 \frac{\partial^2 \mathbf{E}}{\partial t^2}, \\ \nabla(\nabla \cdot \mathbf{B}) - \Delta\mathbf{B} = \mu_0(\nabla \times \mathbf{J}) + \mu_0\epsilon_0 \frac{\partial}{\partial t}(\nabla \times \mathbf{E}). \end{cases} \tag{2.20}$$

Upon replacing $\nabla \cdot \mathbf{E}$ with Gauss's law for the first formulation, $\nabla \cdot \mathbf{B}$ with Gauss's law for magnetism, $\nabla \times \mathbf{E}$ with Faraday's law, and subsequently rearranging, both equations reduce to **inhomogeneous wave equations**, of the form $c^2\Delta u - u_{tt} = \mathbf{F}$, with source terms $\mathbf{F}(\mathbf{r}, t)$,

$$\begin{cases} \Delta\mathbf{E} - \mu_0\epsilon_0 \frac{\partial^2 \mathbf{E}}{\partial t^2} = \nabla(\frac{\rho}{\epsilon_0}) + \mu_0 \frac{\partial \mathbf{J}}{\partial t}, \\ \Delta\mathbf{B} - \mu_0\epsilon_0 \frac{\partial^2 \mathbf{B}}{\partial t^2} = -\mu_0(\nabla \times \mathbf{J}). \end{cases} \tag{2.21}$$

To reduce complexity, I consider an electromagnetic wave propagating in the x-direction in the absence of sources, where $\rho = 0$ and $\mathbf{J} = 0$, in the 1D case. The electric and magnetic fields are taken as $\mathbf{E}(\mathbf{r}, t) = (0, 0, E_z(x, t))$ aligned with the z-direction, and $\mathbf{B}(\mathbf{r}, t) = (0, B_y(x, t), 0)$ aligned with the y-direction, correspondingly. Applying the definition of the curl operator $\nabla\times$ and eliminating the zero terms, one can derive the solution for the first equation in (2.19),

$$\nabla \times \mathbf{E} = \left(\frac{\partial E_z}{\partial y} - \frac{\partial E_y}{\partial z}, \frac{\partial E_x}{\partial z} - \frac{\partial E_z}{\partial x}, \frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y} \right) = \left(0, -\frac{\partial E_z}{\partial x}, 0 \right). \tag{2.22}$$

Likewise for simplifying **B**, I reduce the system of equations in (2.17) as follows,

$$\begin{cases} \frac{\partial E_z}{\partial x} = -\frac{\partial B_y}{\partial t}, \\ \frac{\partial B_y}{\partial x} = -\mu_0\epsilon_0 \frac{\partial E_z}{\partial t}. \end{cases} \tag{2.23}$$

Taking derivatives with respect to x and t, and rearranging, one obtains the 1D wave equations[3],

$$\begin{cases} \frac{\partial^2 E_z}{\partial x^2} - \mu_0\epsilon_0 \frac{\partial^2 E_z}{\partial t^2} = 0, \\ \frac{\partial^2 B_y}{\partial x^2} - \mu_0\epsilon_0 \frac{\partial^2 B_y}{\partial t^2} = 0. \end{cases} \tag{2.24}$$

At $t = 0$, I set the initial conditions for E_z and B_y taking the below form,

$$E_z(x, 0) = f(x), \quad B_y(x, 0) = g(x), \quad x \in [x_0, x_f]. \tag{2.25}$$

Consider variables $u = E_z + B_y$, $v = E_z - B_y$, accompanying by the transformation $x_t = x_{t=0} \pm ct$

[3]The 1D wave equation is in line with the canonical inhomogeneous wave equation (2.21), with $\mathbf{F} = 0$.

with the vacuum light speed c defined in (2.18), d'Alembert's formula [lRD47] yields,

$$\begin{cases} u(x,t) = u(x-ct,0) = f(x-ct) + g(x-ct), \\ v(x,t) = v(x+ct,0) = f(x+ct) - g(x+ct). \end{cases} \tag{2.26}$$

The analytical solution to (2.24) follows by reverting the variable transformation,

$$\begin{cases} E_z = \frac{1}{2}(u+v) = \frac{1}{2}[f(x-ct) + f(x+ct)] + \frac{1}{2}[g(x-ct) - g(x+ct)], \\ B_y = \frac{1}{2}(u-v) = \frac{1}{2}[f(x-ct) - f(x+ct)] + \frac{1}{2}[g(x-ct) + g(x+ct)]. \end{cases} \tag{2.27}$$

The results are depicted in Figure 2.4, which is initialized as per (2.25), and specifically,

$$f(x) = \sin(2\pi x) + 0.5\sin(4\pi x),\ g(x) = \cos(2\pi x) + 0.5\cos(4\pi x) \quad x \in [0,1], t \in [0,0.5]. \tag{2.28}$$

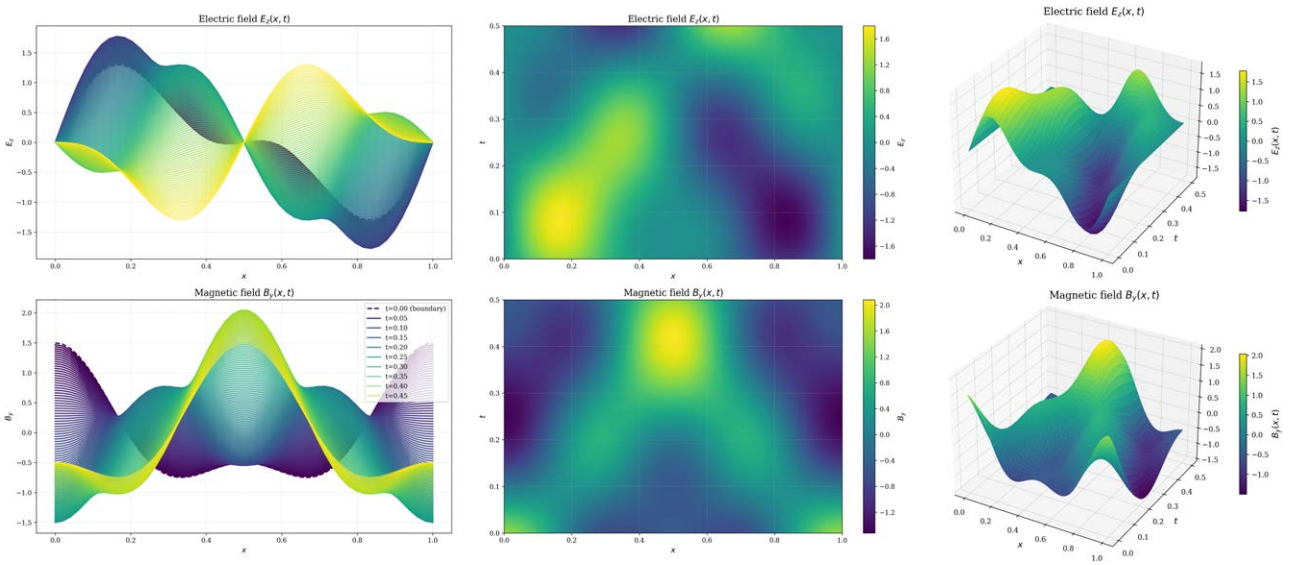


Figure 2.4: Graphical representations of Maxwell's equations analytical results in one-, two-, and three-dimensional plots.

§ 2.2.6 Viscous Burgers' equation

Viscous Burgers' equation [TPL+22] describes both diffusion and nonlinear advection phenomena, where the latter commonly refers to convection, in dynamical systems,

$$\begin{cases} \frac{\partial u(x,t)}{\partial t} + u(x,t)\frac{\partial u(x,t)}{\partial x} = \nu \frac{\partial^2 u(x,t)}{\partial x^2}, \quad x \in [x_0, x_f], t \in [t_0, t_f], \\ u(x,0) = u_0(x), \quad x \in [x_0, x_f], \quad \text{the initial condition}, \end{cases} \tag{2.29}$$

in which a positive constant viscosity coefficient $\nu \in \mathbb{R}^+$, and the initial condition $u_0(x)$ for $t = 0$ are specified. Cole-Hopf [Hop50] transforms $u(x,t)$ to $\phi(x,t)$ as follows,

$$u(x,t) = -2\nu \frac{\partial}{\partial x} \ln \phi(x,t) = -2\nu \frac{1}{\phi(x,t)} \frac{\partial \phi(x,t)}{\partial x} \equiv -2\nu \frac{\phi_x}{\phi}. \tag{2.30}$$

Invoking quotient rule together with chain rule in Calculus, I derive the derivatives of $u(x,t)$ of first and second order with respect to x and t as follows,

$$\begin{aligned} \frac{\partial u(x,t)}{\partial x} &= 2\nu \left(\frac{\phi_x^2}{\phi^2} - \frac{\phi_{xx}}{\phi} \right), \quad \frac{\partial u(x,t)}{\partial t} = 2\nu \left(\frac{\phi_x \phi_t}{\phi^2} - \frac{\phi_{xt}}{\phi} \right), \\ \frac{\partial^2 u(x,t)}{\partial x^2} &= 2\nu \left(\frac{3\phi_x \phi_{xx}}{\phi^2} - \frac{2\phi_x^3}{\phi^3} - \frac{\phi_{xxx}}{\phi} \right). \end{aligned} \tag{2.31}$$

Substituting (2.31) into (2.29) and following algebraic manipulation, the resulting equation is,

$$2\nu\left(\frac{\phi_x\phi_t}{\phi^2}-\frac{\phi_{xt}}{\phi}-\nu\frac{\phi_x\phi_{xx}}{\phi^2}+\nu\frac{\phi_{xxx}}{\phi}\right)=0,\quad x\in[x_0,x_f],t\in[t_0,t_f], \tag{2.32}$$

Rewriting (2.32) with the reverse form of the quotient rule, I obtain,

$$2\nu\frac{\partial}{\partial x}\left(\frac{\nu\phi_{xx}-\phi_t}{\phi}\right)=0,\quad x\in[x_0,x_f],t\in[t_0,t_f]. \tag{2.33}$$

Upon integration of (2.33) in x and incorporating an arbitrary function of integration $f(t)$,

$$\frac{\nu\phi_{xx}-\phi_t}{\phi}=f(t),\quad x\in[x_0,x_f],t\in[t_0,t_f]. \tag{2.34}$$

With the introduction of $f(t)=\frac{dF(t)}{dt}$ and $\tilde{\phi}=\phi\cdot e^{F(t)}$, the $\tilde{\phi}$ derivatives are expressed as,

$$\frac{\partial\tilde{\phi}}{\partial t}=e^{F(t)}\left(\phi_t+\phi\frac{dF(t)}{dt}\right),\quad \frac{\partial^2\tilde{\phi}}{\partial x^2}=e^{F(t)}\phi_{xx}, \tag{2.35}$$

by plugging them into (2.34), I reduce the problem to a conventional form of the heat equation,

$$\nu\frac{\partial^2\tilde{\phi}(x,t)}{\partial x^2}-\frac{\partial\tilde{\phi}(x,t)}{\partial t}=0,\quad x\in[x_0,x_f],t\in[t_0,t_f]. \tag{2.36}$$

Equation (2.36) admits a solution given by the convolution of the initial condition $\tilde{\phi}_0(x)=\tilde{\phi}(x,0)$ and the heat kernel $\Phi(x,t)$ [Eva10],

$$\tilde{\phi}(x,t)=\int_{-\infty}^{\infty}\Phi(x-x',t)\,\tilde{\phi}_0(x')\,dx',\quad \text{with } \Phi(x,t)=\frac{1}{\sqrt{4\pi\nu t}}e^{-\frac{x^2}{4\nu t}}. \tag{2.37}$$

It is worth noting that the mapping of ϕ to $\tilde{\phi}$ preserves the Cole-Hopf formulation (2.30), as the extra multiplicative factor $e^{F(t)}$ does not depend on x,

$$u(x,t)=-2\nu\frac{\partial}{\partial x}\ln\phi(x,t)=-2\nu\frac{\partial}{\partial x}\ln\tilde{\phi}(x,t). \tag{2.38}$$

Evaluating Cole-Hopf (2.38) at $t=0$ and integrating yield the $\tilde{\phi}(x,0)$ initial condition, namely,

$$\tilde{\phi}_0(x)=-\frac{1}{2\nu}\int_0^x u_0(x')\,dx'. \tag{2.39}$$

The analytical solution of (2.29) is thus combined by substituting (2.39) into (2.37) and (2.38). In this work, I consider when $u_0(-\infty)$ and $u_0(\infty)$ exist and under the condition that $\forall x\in\mathbb{R}.\ u_0'(x)<0$, the explicit solution [BAT15] corresponds to a propagating wave of the form below,

$$u(x,t)=c-\Delta u_0\tanh\left(\frac{\Delta u_0}{2\nu}(x-ct)\right),\quad \text{where } c=\frac{u_0(-\infty)+u_0(\infty)}{2},\Delta u_0=\frac{u_0(-\infty)-u_0(\infty)}{2}. \tag{2.40}$$

Figure 2.5 presents the solution associated with the initial condition (2.40), namely $u_0(-\infty)=1$, $u_0(\infty)=0$, and $\nu=0.5$.

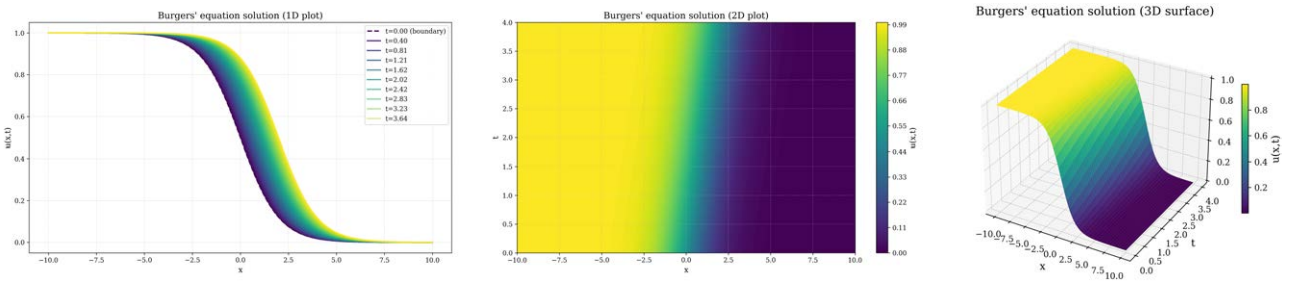


Figure 2.5: Graphical representations of the Burgers' equation analytical results in one-, two-, and three-dimensional plots.

§ 2.3 PDE solvers

A variety of approaches have been proposed for solving PDE unknown field u, broadly categorized as numerical analysis, data-driven, and neural operators methods.

§ 2.3.1 Finite difference method (FDM)

Let $\mathbf{u} \in \mathbb{R}^{N\times M}$ denotes a discrete sampling over a continuous field $u(x,t)$ of (2.11), where M and N represent the sampling size over the time and space dimensions. Finite difference operators [Ise08] for the i-th sampling point take the following forms,

$$(\Delta\mathbf{u})_i = \begin{cases} (\Delta^+\mathbf{u})_i = u_{i+1}^j - u_i^j, & \text{forward operator.} \\ (\Delta^-\mathbf{u})_i = u_i^j - u_{i-1}^j, & \text{backward operator.} \\ (\Delta^0\mathbf{u})_i = \dfrac{u_{i+1}^j - u_{i-1}^j}{2}, & \text{central operator.} \end{cases}, \tag{2.41}$$

where *time indices* are $0 \le j \in \mathbb{Z} < M$ and *space indices* are $0 \le i \in \mathbb{Z} < N$.

PDEs contain spatial and temporal partial derivatives of the unknown field u, which are to be rewritten as difference operators (2.41) in FDM [BS18]. Expanding $u(x \pm \Delta x, t)$ centered at $u(x,t)$ with the Taylor formula and neglecting any higher-order terms beyond the first order, I approximate the spatial derivative for the advection equation (2.11) as follows,

$$\begin{aligned} \frac{\partial u_i^j(x,t)}{\partial x} &= \begin{cases} \frac{(\Delta^-\mathbf{u})_i}{\Delta x} + O(\Delta x) \approx \frac{(\Delta^-\mathbf{u})_i}{\Delta x} = \frac{u_i^j - u_{i-1}^j}{\Delta x} & \text{if } \beta > 0, \\ \frac{(\Delta^+\mathbf{u})_i}{\Delta x} + O(\Delta x) \approx \frac{(\Delta^+\mathbf{u})_i}{\Delta x} = \frac{u_{i+1}^j - u_i^j}{\Delta x} & \text{if } \beta < 0. \end{cases}, \quad \text{upwind scheme.} \\ &= \frac{(\Delta^0\mathbf{u})_i}{\Delta x} + O(\Delta x^2) \approx \frac{(\Delta^0\mathbf{u})_i}{\Delta x} = \frac{u_{i+1}^j - u_{i-1}^j}{2\Delta x}, \quad \text{central difference.} \end{aligned} \tag{2.42}$$

with space interval Δx and indices i, j as aforementioned. (2.11) reduces to an Ordinary Differential Equation (ODE) in the upwind scheme,

$$\frac{\partial u(x,t)}{\partial t} + \beta[\mathbb{I}_{\beta>0}\frac{(\Delta^-\mathbf{u})_i}{\Delta x} + \mathbb{I}_{\beta<0}\frac{(\Delta^+\mathbf{u})_i}{\Delta x}] = 0, \tag{2.43}$$

in which indicator $\mathbb{I}_{\beta>0} = \begin{cases} 1 & \text{if } \beta > 0, \\ 0 & \text{otherwise.} \end{cases}$ splits scenarios based on the value of β[4]. The forward or backward difference operator incorporates the direction of transport. For positive $\beta > 0$, the information flows towards the right, leading to u_i^j being derived from u_{i-1}^j, and the other way around for negative $\beta < 0$ [Pat80]. Taking the forward scheme and turning the time derivative into a forward difference operator, (2.11) takes the following form, where Δt denotes the time interval,

$$\frac{u_i^{j+1} - u_i^j}{\Delta t} + \beta[\mathbb{I}_{\beta>0}\frac{(\Delta^-\mathbf{u})_i}{\Delta x} + \mathbb{I}_{\beta<0}\frac{(\Delta^+\mathbf{u})_i}{\Delta x}] = 0. \tag{2.44}$$

After rearranging, the per-step progression for advection equation (2.11) is expressed as,

$$u_i^{j+1} = u_i^j - \frac{\beta\Delta t}{\Delta x}[\mathbb{I}_{\beta>0}(\Delta^-\mathbf{u})_i + \mathbb{I}_{\beta<0}(\Delta^+\mathbf{u})_i]. \tag{2.45}$$

Stability analysis. The backward difference operators are one of the implicit schemes, which are stable with no condition. On the contrary, the forward difference operators belong to the explicit schemes, stability is not guaranteed and thus remains a concern. To ensure the stability of the latter case, I describe the requirements to be met below.

[4] Alternatively, $\max(\beta, 0)$ and $\min(\beta, 0)$ are the same as $\beta\mathbb{I}_{\beta>0}$ and $\beta\mathbb{I}_{\beta<0}$.

The CFL stability criteria $C \in \mathbb{R}$, commonly referred to as the Courant number, represents the fraction of the distance information transported by the wave speed β in a single time step Δt and the space interval Δx. CFL condition [CFL28] constrains C to meet the below inequality,

$$C = \frac{|\beta|\Delta t}{\Delta x} \leq C_{max}, \tag{2.46}$$

with constant C_{max} that varies for each setting. This condition yields the upper bound of time step Δt to avoid instability, when the space interval Δx is fixed.

§ 2.3.2 Physics-informed neural network (PINN)

Physics-Informed Neural Network (PINN) [RPK19] constitutes a unsupervised framework to approximate PDE unknowns (2.9) in a data-driven manner, with optional supervised dataset for training. PINNs adopt a neural network $\hat{u}_\theta(x,t) \in \mathcal{U}$ to represent the solution field $u(x,t) \in \mathcal{U}$: $\hat{u}_\theta(x,t) \approx u(x,t)$, where $\theta \in \Theta$ denotes trainable parameters of the network.

By automatic differentiation [BPRS18][5] for $\hat{u}_\theta$, one can minimize the PDEs *residual* $\mathcal{R}_\theta$,

$$\mathcal{R}_\theta(x,t) \in \mathcal{Y} = \mathcal{D}[\hat{u}_\theta](x,t) - f(x,t), \quad x \in \Omega, t \in [t_0, t_f]. \tag{2.47}$$

The physics-informed loss, technically termed as the residual loss L_R[6], is given by the residual $\mathcal{R}_\theta$ (2.47) p-norm. For the training process, I numerically compute the integration by discretization and sample $N_\mathcal{R}$ quadrature points with assigned ω_k as the weight for the k-th sample point (x_k, t_k), with $1 \leq k \in \mathbb{Z} < N_\mathcal{R}$. The discrepancy is captured by training error $\mathcal{E}_T(\theta) = L_R - \mathcal{L}_R$,

$$\begin{aligned} L_R := (||\mathcal{R}_\theta||_p)^p := \underbrace{[(\int_\mathbb{D} |\mathcal{R}_\theta|^p \, dx \, dt)^{\frac{1}{p}}]^p}_{\text{integral } L_R} = \int_\mathbb{D} |\mathcal{R}_\theta|^p \, dx \, dt \\ = \underbrace{\sum_{k=1}^{N_\mathcal{R}} \omega_k |\mathcal{R}_\theta(x_k, t_k)|^p}_{\text{discretized } \mathcal{L}_R} + \mathcal{E}_T(\theta) \approx \mathcal{L}_R, \quad \textit{by numerical integration.} \end{aligned} \tag{2.48}$$

Incorporating boundary conditions (2.9), the i-th residual of them $\mathcal{R}_\theta^{\mathcal{B}_i}$ takes the following formulation, yielding the $\mathcal{L}_{\text{BCs}}$ loss that follows the above procedure,

$$\mathcal{R}_\theta^{\mathcal{B}_i}(x,t) \in \mathcal{Z}_i = \mathcal{B}_i[\hat{u}_\theta](x,t) - g_i(x,t), \quad x \in \partial\Omega_i, t \in [t_0, t_f]. \tag{2.49}$$

The *total error* (B.8) arising from adopting the neural network $\hat{u}_\theta$ to approximate the solution u, applying (B.6), is thus,

$$\mathcal{E}_{\text{PINN}}(\theta) = (||\hat{u}_\theta - u||_p)^p. \tag{2.50}$$

Optionally with a supervised dataset $\{(x_n, t_n), u(x_n, t_n)\}_{n=1}^{N_d}$ of labeled solutions for every sample point, one can optimize the neural network by minimizing an extra supervised loss $\mathcal{L}_{\text{data}}$[7], which is the discretized version of the total error (2.50) defined above,

$$\mathcal{L}_{\text{data}} = \frac{1}{N_d} \sum_{n=1}^{N_d} (|\hat{u}_\theta(x_n, t_n) - u(x_n, t_n)|^p). \tag{2.51}$$

Training. Figure 2.6 and Algorithm 1 depict the PINN training architecture and pseudocode, that compute the solution with the neural network $\hat{u}_\theta = u_{\theta^{\text{opt}}}(x,t)$. To counteract overfit-

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

[6] L_R corresponds to $\mathcal{R}$ of (B.6). Rather than for the solution u, here the risk is defined for the residual $\mathcal{R}_\theta$.

[7] It is worth noting that if the supervised data are selected from the boundary, the supervised loss collapses to the boundary condition loss.

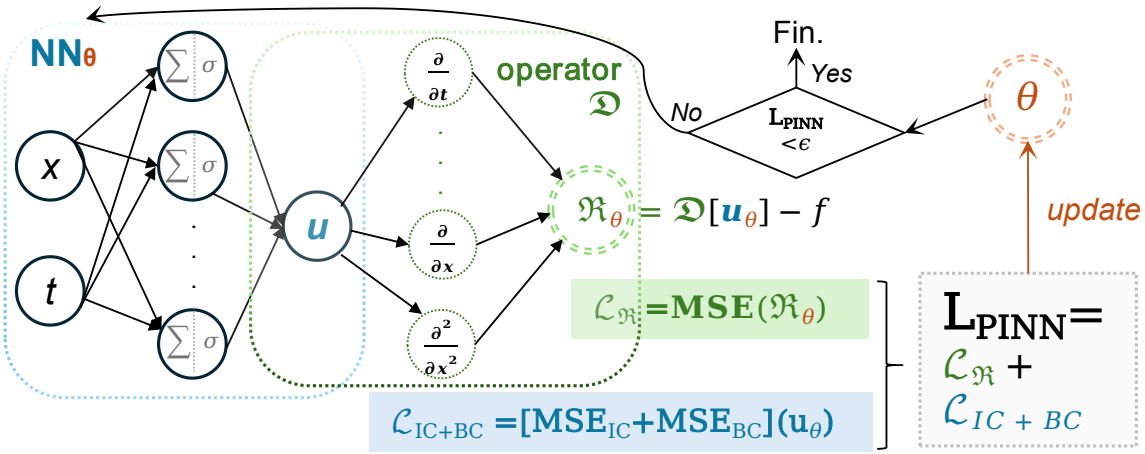


Figure 2.6: PINN architecture, where parameters θ are optimized by reducing the total training loss $\mathcal{L}_{\text{PINN}}$ of the physics-informed loss $\mathcal{L}_R$, with optional $\mathcal{L}_{\text{BCs}}$, $\mathcal{L}_{\text{ICs}}$ and supervised loss $\mathcal{L}_{\text{data}}$.

ting towards the optional supervised dataset, the target of training is to reduce the unsupervised residual error $\mathcal{L}_{\mathcal{R}}$ (2.48). By combining the boundary condition residual (2.49) with the supervised loss (2.51), the parameter θ is optimized by minimizing the total training loss: $\theta^{\text{opt}} \approx \arg\min_{\theta\in\Theta} \mathcal{L}_{\text{PINN}}$,

$$\mathcal{L}_{\text{PINN}} = \underbrace{\sum_{k=1}^{N_{\mathcal{R}}} \omega_k |\mathcal{R}_\theta(x_k, t_k)|^p}_{\text{Discretized residual loss } \mathcal{L}_{\mathcal{R}}} + \lambda_1 \underbrace{\frac{1}{N_d}\sum_{n=1}^{N_d} (|\hat{u}_\theta(x_n, t_n) - u(x_n, t_n)|^p)}_{\text{Supervised loss } \mathcal{L}_{\text{data}}} + \lambda_2 \underbrace{\sum_i \sum_{b=1}^{N_{\mathcal{B}_i}} \omega_b^{\mathcal{B}_i} |\mathcal{R}_\theta^{\mathcal{B}_i}(x_b, t_b)|^p}_{\text{BCs loss } \mathcal{L}_{\text{BCs}}}, \tag{2.52}$$

where the weights $\omega_b^{\mathcal{B}_i}$ represent the b-th sample point and the corresponding boundary condition with index i. To combine three loss terms, I take two regularization hyperparameters $\lambda_1, \lambda_2 > 0$.

Algorithm 1 Pseudocode of PINN optimization procedures.

Hyperparameters: λ_1, λ_2, training iterations N_{iter}, learning rate η, batch size.
Input: Randomly initialized neural network parameters $\theta \in \Theta$.
Output: The optimal parameters θ^{opt} for $\hat{u}_\theta$.
for $n_{\text{iter}} = 1$ to N_{iter} **do**
    Obtain PDE samples $x_k \in \Omega, t_k \in [t_0, t_f]$ and select boundary $x_b \in \partial\Omega_i, t_b \in [t_0, t_f]$.
    Infer the network $\hat{u}_\theta$ and compute derivatives $\mathcal{D}[\hat{u}_\theta]$ and boundary $\mathcal{B}_i[\hat{u}_\theta]$.
    As per (2.52), derive loss $\mathcal{L}_{\text{PINN}} = \mathcal{L}_{\mathcal{R}} + \mathcal{L}_{\text{data}} + \mathcal{L}_{\text{BCs}}$.
    Update $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}$ by backpropagation.
end for

§ 2.3.3 Neural operator (NO)

Following the PDEs general formulation (2.9), here a denotes the PDE parameter with the equivalent dimensionality to that of the unknown field u, and the derivative operator is updated accordingly as $\mathcal{D}_a$. The source function f, and the initial conditions remain unchanged. The resulting PDE formulation is $\mathcal{D}_a[u] = f$.

Operator learning. The intuition behind [LKA+20] is the map between function space $\mathcal{A}$ and $\mathcal{U}$ of *infinite* dimensionality. To put it more concretely, it learns how to map from the input $a \in \mathcal{A} : \mathbb{D} \to \mathbb{R}^{d_a}$ into unknown field $u \in \mathcal{U} : \mathbb{D} \to \mathbb{R}^{d_u}$. To learn such relationship, one needs to

curate a supervised dataset $\{(a_i^j, f_i^j), u_i^j | i = 1, ..., N_{\mathcal{R}}\}_{j=1}^{N_{pde}}$ of size N_{pde} representing the unique number of PDE instances, of which each having $N_{\mathcal{R}}$ samples over its domain. The *challenge* resides in how to adapt *finite* dimensional neural networks for this infinite-dimensional operator learning. Below, I present different directions to address this.

Direction I: finite-dimensional operators [GLI16, KLY21, ZZ18]. With the assumption of a being measurable, $a = a(\mu)$, in the finite dimensional space $\mu \in \mathbb{R}^{d_\mu}$. One can parametrize the PDE as $u = u(t, x, \mu)$, which is the technique widely used in aircraft design and manufacturing [AUC09], image processing [PGB03]. The *training* procedure reduces the p-norm supervised loss $\mathcal{L}_{data}$, that quantifies the discrepancy, between the prediction $\mathcal{G}_\theta(a_i, f_i)$ and the reference solution u_i, aggregated over each i-th sample point and j-th PDE instance,

$$\mathcal{L}_{data} = \frac{1}{N_{pde} \times N_{\mathcal{R}}} \sum_{j=1}^{N_{pde}} \sum_{i=1}^{N_{\mathcal{R}}} (||\mathcal{G}_\theta(a_i^j, f_i^j) - u_i^j||_p)^p. \tag{2.53}$$

The resulting solution is written as $\hat{u}_\theta = \mathcal{G}_{\theta^{\text{opt}}}(a, f)$, with $\theta^{\text{opt}} \approx \arg\min_{\theta \in \Theta} \mathcal{L}_{data}$. The PINN PDE residual loss $\mathcal{L}_{\mathcal{R}}$ (2.47) is absent for the conventional operator learning, but it can be incorporated for **Physics-Informed Neural Operator (PINO)**, which considers both the residual loss $\mathcal{L}_{\mathcal{R}}$ and the supervised loss $\mathcal{L}_{data}$ during training,

$$\mathcal{L}_{\text{PINO}} = \mathcal{L}_{data} + \lambda \underbrace{\frac{1}{N_{pde} \times N_{\mathcal{R}}} \sum_{j=1}^{N_{pde}} \sum_{i=1}^{N_{\mathcal{R}}} (||\mathcal{D}_a[\mathcal{G}_\theta(a_i^j, f_i^j)] - f_i^j(x_i, t_i)||_p)^p}_{\text{Residual loss } \mathcal{L}_{\mathcal{R}}}, \tag{2.54}$$

Learning theory. From the curse of dimensionality (B.5), we need an efficient sampling strategy for dataset. Otherwise, given a fixed total error threshold ϵ, the required dataset size N grows exponentially. Examples are low discrepancy sequences, e.g. Sobol, Halton, Hammersley, etc. Although the underlying idea is simple, it is hindered by the generalization to novel a, non-uniqueness parameterization, and effective sampling strategies.

Direction II: solution fields interpolation. One can directly interpolate the solution from the discrete grid. The most naive approach is via linear or cubic spline interpolation [AMMIR14], while another option might be a neural network based interpolator. However, these interpolation methods are for discrete solution functions and thus lead to *mismatch* across the continuous and discrete solution fields.

Direction III: infinite-dimensional operators. Neural networks admit a generalization from discrete structures to continuous function space, such as DeepONet [LJP+21]. **Fourier Neural Operator (FNO)** [LKA+21] first takes a neural network $P : \mathbb{R}^{d_a} \to \mathbb{R}^{d_v}$ to project the input $a(x) \in \mathbb{R}^{d_a}$ into a latent variable $v_0(x) \in \mathbb{R}^{d_v}$, followed by an iterative update $v_0 \xrightarrow{\mathcal{N}^1} v_1 \ ... \xrightarrow{\mathcal{N}^l} v_l$ defined below, before the output is generated by another neural network $Q : \mathbb{R}^{d_v} \to \mathbb{R}^{d_u}$ as $\hat{u} = Q(v_l)$,

$$(\mathcal{N}^l v_{t+1})(x) = \sigma^l [A^l v_t(x) + \underbrace{\int_{\mathbb{D}} k_\theta^l (x, y, a(x), a(y))\, v(y)\, dy}_{(\mathcal{K}(a)v_t)(x)}], \quad x \in \mathbb{D}, \tag{2.55}$$

where $A^l : \mathbb{R}^{d_v} \to \mathbb{R}^{d_v}$ is a linear transformation, $k^l : \mathbb{R}^{2\times d+2\times d_a} \to \mathbb{R}^{d_v \times d_v}$ is the kernel neural network with parameters θ, and $\sigma^l : \mathbb{R} \to \mathbb{R}$ is the nonlinear activation function. The kernel integral operator $\mathcal{K}(a)$ relates each latent feature at location x to the features at all other

locations y, with a complexity $O(N^2)$ for N discrete points. The Fourier kernel is as below,

$$(\mathcal{K}(a)v_t)(x) \in \mathbb{R}^{d_v} = \mathcal{F}^{-1}(B_\phi \cdot \mathcal{F}(v_t))(x), \quad x \in \mathbb{D}, \tag{2.56}$$

with $\mathcal{F}(v_t)(\xi) \in \mathbb{C}^{d_v}$ for each frequency mode $\xi \in \mathbb{D}$, whose k-th term is given by,

$$\forall k \in \{1, ..., d_v\}. \quad \mathcal{F}(v_t)^k(\xi) \in \mathbb{C} = \mathcal{F}\left(v_t^k(\xi)\right) = \int_{\mathbb{D}} v_t^k(x) e^{-i\omega x} dx, \tag{2.57}$$

where $\omega = 2\pi\xi$. $B_\phi(x) \in \mathbb{C}^{d_v \times d_v}$ is the Fourier transform of a parametric periodic function $p_\phi(x) \in \mathbb{R}^{d_v \times d_v}$, yielding an immediate output $o_t = B_\phi \cdot \mathcal{F}(v_t)(x)$. Lastly, $\mathcal{F}^{-1}$ follows,

$$\forall k \in \{1, ..., d_v\}. \quad \mathcal{F}^{-1}(o_t)^k(x) \in \mathbb{R} = \mathcal{F}^{-1}\left(o_t^k(x)\right) = \int_{\mathbb{D}} o_t^k(\xi) e^{-i\omega x} d\xi. \tag{2.58}$$

A fast implementation for (inverse) Fourier transform of functions v_t and o_t is by Fast Fourier Transform (FFT) [LKA+21], with a total complexity of $O(N \log N)$.

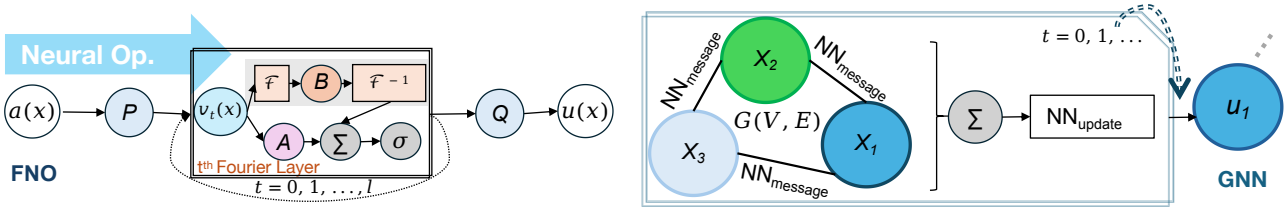


Figure 2.7: The architecture diagrams of FNO (left) and MP-GNN (right).

§ 2.3.4 Graph Neural Network (GNN)

Early work [SS97] starts designing neural networks for structured data, which is later formalized as Graph Neural Network (GNN) [WPC+21]. A graph is denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where sets $\mathcal{V}$ and $\mathcal{E}$ are for vertices $v \in \mathcal{V}$ and edges $e_{ij} = (v_i, v_j) \in \mathcal{E}$ respectively, where e_{ij} represents the directed edge from vertex v_i to vertex v_j. The neighborhood $\mathcal{N}(v)$ of vertex v is defined as $\mathcal{N}(v) = \{u | (u, v) \in \mathcal{E}\}$, which is the set of vertices that have directed edges to v of interest. A graph $\mathcal{G}$ has vertex attributes $\mathbf{X}^v \subset \mathbb{R}^{|\mathcal{V}| \times d_\mathcal{V}}$ and optionally edge attributes $\mathbf{X}^e \subset \mathbb{R}^{|\mathcal{E}| \times d_\mathcal{E}}$, where $d_\mathcal{V}$ and $d_\mathcal{E}$ are the feature dimensions of vertices $\mathcal{V}$ and edges $\mathcal{E}$.

Graph Neural Network (GNN) [SGT+09] considers aggregating neighborhood $\mathcal{N}(v)$ information for each vertex v in the graph $\mathcal{G}$, rather than just considering inputs from the last timestep. The initial hidden state $\mathbf{h}_v^0$ is set randomly or by the vertex attributes $\mathbf{X}^v$. At each timestep t, each vertex hidden state $\mathbf{h}_v^t$ is repeatedly updated until convergence,

$$\mathbf{h}_v^t = \sum_{u \in \mathcal{N}(v)} f_\Theta(\mathbf{x}_v, \mathbf{x}_u, \mathbf{h}_v^{t-1}, \mathbf{h}_u^{t-1}), \tag{2.59}$$

where f_Θ is a differentiable (neural) function, $\mathbf{x}_v, \mathbf{x}_u \in \mathbf{X}^v$ are the vertex attributes of v and u, and $\mathbf{x}_{uv}^e \in \mathbf{X}^e$ is the edge attribute of e_{uv}. For the sake of convergence, the recursive function f_Θ is assumed to be contractive, which bounds the relative latent-space distance by a constant.

Graph Convolutional Network (GCN). Inspired by the successful Convolutional Neural Network (CNN) in Computer Vision, [KW17] generalized the convolution operation for graphs,

$$\mathbf{h}_v^t = \sigma\left(\sum_{u \in \mathcal{N}(v) \cup \{i\}} \frac{1}{\sqrt{|\mathcal{N}(v)|}\sqrt{|\mathcal{N}(u)|}} \mathbf{W}\mathbf{h}_u^{t-1} + \mathbf{b}\right), \tag{2.60}$$

where weight matrix $\mathbf{W}$ and bias $\mathbf{b}$ are learnable parameters, and σ is the nonlinear activation function. The normalization term, i.e. the product of the square root of the degree of vertex v

and its neighbor u, is for numerical stability and avoiding overfitting.

An alternative scheme is Edge Convolution [WSL+19], which is proposed to process graphs or point clouds. They define hidden edge features as $\mathbf{e}_{uv} = \hat{f}_\Theta(\mathbf{h}_v, \mathbf{h}_u) = f_\Theta(\mathbf{h}_v, \mathbf{h}_u - \mathbf{h}_v)$, which is asymmetric and captures both the local and neighborhood information,

$$\mathbf{h}_v^t = \max_{u \in \mathcal{N}(v)} \hat{f}_\Theta(\mathbf{h}_v^{t-1}, \mathbf{h}_u^{t-1} - \mathbf{h}_v^{t-1}), \tag{2.61}$$

MP-GNN [GSR+17] is a general framework for GNNs (Figure 2.7), which consists of two phases: message passing and readout or update phase,

$$\begin{aligned} \text{message passing:} \quad & \mathbf{m}_v^t = \oplus_{u \in \mathcal{N}(v)} M_\Theta(\mathbf{h}_v^{t-1}, \mathbf{h}_u^{t-1}, \mathbf{x}_{uv}^e), \\ \text{readout:} \quad & \mathbf{h}_v^t = U_\Theta(\mathbf{h}_v^{t-1}, \mathbf{m}_v^t), \end{aligned} \tag{2.62}$$

where aggregation $\oplus$ is a differentiable permutation invariant function, e.g. mean $\frac{1}{|\mathcal{N}(v)|}$, $\sum_{u \in \mathcal{N}(v)}$, or $\max_{u \in \mathcal{N}(v)}$, and M_Θ, U_Θ are the message, update differentiable (neural) functions. In this work, I adopt the message-passing GNN framework for its incorporation of static edge attributes, which is important for modeling the interactions between vertices in the graph. In PDE solvers, the vertices are the collocation points, and the edge attributes are set as the distance between the collocation points. For GCN, the edge attributes are implicitly captured by the normalization term. Hence, without the edge attributes, the GCN can not generalize to unseen test-time collocation points theoretically.

§ 2.3.5 Kansa method (KM)

Kernel functions. Radial Basis Function (RBF) expresses the output as a function of the distance r from the reference origin point $\mathbf{c}$ to input $\mathbf{x}$, which acts as if the activation function in a neural network, such as the RBF network introduced below,

$$\psi_\mathbf{c}(r) = \psi_\mathbf{c}(||\mathbf{x} - \mathbf{c}||). \tag{2.63}$$

Denote σ as the standard deviation, and the Gaussian shape parameter $\epsilon = \frac{1}{\sqrt{2}\sigma}$, we have,

$$\psi_\mathbf{c}(r) = \begin{cases} e^{-(\epsilon r)^2}, & \text{Gaussian}, \\ e^{-(\epsilon r)^2} \sin(\epsilon r), & \text{Sinusoidal}, \\ \frac{1}{1+(\epsilon r)^2}, & \text{Inverse quadratic}, \\ \sqrt{1+(\epsilon r)^2}, & \text{Multiquadrics}, \end{cases} \tag{2.64}$$

Numerous smooth RBFs exist, of these I adopt Gaussian and Sinusoidal RBF in their efficient approximation of smooth functions. Besides RBFs, I consider neural network kernels (§ 3.4).

Truncated kernel functions. To compactly represent the kernel functions, one can truncate it when the distance r exceeds a certain threshold τ to trade off accuracy for efficiency,

$$\psi_\mathbf{c}^\tau(r) = \begin{cases} \psi_\mathbf{c}(r), & r \le \tau, \\ 0, & r > \tau, \end{cases} \tag{2.65}$$

Kansa Method (KM) [Kan90] expresses $u(x,t)$ with a RBF network, which is the linear superposition of kernels $\psi_k(||\mathbf{x}_i - \mathbf{x}_k||) \in \mathbb{R}^d \to \mathbb{R}$. Each kernel is associated with every collocation point $\{\mathbf{x}_i \in \mathbb{D}\}_{i=1}^N$ over the domain $\mathbb{D}$, after which these kernels are linearly combined to approximate the PDE solution u as in (2.66),

$$u(x_i, t_i) \approx \hat{u}(\mathbf{x}_i) = \sum_{k=1}^{N} \alpha_k \cdot \psi_k(||\mathbf{x}_i - \mathbf{x}_k||), \quad \mathbf{x} \in \mathbb{D}, \tag{2.66}$$

for which the unknown expansion coefficients $\alpha_i \in \mathbb{R}$ to be determined using algorithms detailed in § 3.3. The temporal coordinate t is treated here as an added spatial variable.

§ 2.3.6 Summary of PDE solvers

Constrained optimization. In optimization of a parametric objective function, constraints are imposed on the parameter space. Within PDE solvers, such constraints typically arise from initial conditions, boundary conditions, or equation residuals. They are commonly formulated as one of hard or soft constraints, where the former involves reducing penalty terms during optimization. In contrast, the latter represents conditions to be strictly satisfied, such as equality or inequality formulations. I categorize all the solvers introduced above in Table 2.1.

Table 2.1: Summary of different PDE solvers, where Auto tuning means that the method can automatically adjust its parameters during training, Label-free refers that unsupervised training is possible without labeled data, and Exact enforcement denotes whether the method enforces hard constraints (equality or inequality forms) or soft constraints (penalty functions).

Method	Motivation	Auto tuning	Label-free	Exact enforcement
FDM	grid-based	×	N/A	✓
PINN	physics-driven	✓	✓	×
NO	operator learning	✓	×	×
– FNO	operator learning	✓	×	×
– PINO	composite	✓	×	×
GNN	graph-based	✓	×	×
KM	mesh-free FEM	×	N/A	✓
– CNF	KM on neural fields	✓	✓	✓

The *summary* of aforementioned PDE methods is in Table 2.1, covering aspects of motivation, self-tuning of parameters, requirements of supervised data with labels, and the type of constraints enforced. It shows the relative advantages of proposed framework.

§ 2.4 Types of simulation

In scientific computing, simulation tasks are formulated as the mapping[8] $F : \mathbb{C}_n \to \mathbb{C}_m$ from input $a \in \mathbb{C}^n$ to output $b \in \mathbb{C}^m$, i.e. a scientific modelling function $b = F(a)$. Examples of such functions are advection or Maxwell's equations in Physics, population models in Biology, Bidirectional Scattering Distribution Function (BSDF) or the rendering equation in Computer Graphics, and many more. For a known function F, there are two types of simulation, i.e. forward and inverse problem, described as follows.

Forward problem $\boldsymbol{b} = F(a)$ aims to solve for the output $\boldsymbol{b}$, given each input a. Due to its resource-intensive nature, one of the key objective is to speed up the computation taken and minimize the memory usage. Depending on the use case, one may favor accuracy over efficiency or vice versa. For example, in safety-critical applications, the model's predictions

[8]Note that the domain consists of complex numbers, as encountered in simulation of Quantum Computing.

must be highly accurate, despite the cost involved. Whereas in real-time demos, lower latency is crucial to maintain high Quality of Service (QoS).

Inverse problem $b = F(\boldsymbol{a})$, in contrast, is selecting the input $\boldsymbol{a}$ that best fits the corresponding output b. It's usually considered as a search problem, where forward simulation will be carried out multiple times. The challenges comprise the high computation cost, an ill-posed problem, i.e. without enough data to secure the unique solution, data sparsity and measurement noise. Whereas, when the function $\boldsymbol{F}$ is unknown, the task is to **discover the underlying equations** via $b = \boldsymbol{F}(a)$ that best capture the input-output pairs. Thereafter, new scientific rules are derived from the discovered equations, which can be adopted for further investigations.

§ 2.5 Community contributions

Open-source and reproducibility. I aim to open-source the codebase under `MIT` license, due to its openness and broad compatibility. The polished paper has been released to `arXiv`, an open-access preprint repository. For reproducibility and further research, all necessary details for training and evaluation are included in the codebase and the paper.

Review feedback and service. Throughout the dissertation period, I have been actively engaging with the AI and PDE communities, where suggestions are both given and received. Reviewers have given constructive feedback, which has been incorporated into the dissertation and codebase. With thanks to nomination and invitation, I have served as a peer reviewer in this topic and provided feedback to other researchers, which has been appreciated.

Contributions. In this dissertation, I extend prior work by introducing a two-phase framework. Table 2.2 presents my proposed approaches (last two rows) evaluated against the prior art.

- **Stage 1** is the *learnable Kansa collocation for forward and inverse PDEs beyond linearity*. It led to my first-author paper [HC +26], which has been **accepted for publication** by the AI-PDE workshop at International Conference on Learning Representations (ICLR), and the contributions have been well-received by reviewers and the community.

- **Stage 2** is the *CNF extension beyond linearity*. I have further developed new kernel formulations, such as learnable kernels with adaptive parameter tuning, and conducted the stability analysis that I proposed within and beyond linearity. The consolidated manuscript has been submitted to SIGGRAPH Asia 2026 and is currently under review.

Table 2.2: Comparison of my proposed two-stage framework with the prior art, considering whether equations are **M**ultivariable (more than one independent variables), **C**oupled (more than one dependent variables) or **N**onlinear, and if **S**tability analysis is included.

Method	Kernel	M	C	N	S	Applications
Kansa [Kan90]	RBF	✓	×	×	✓	PDEs
CNF [ZFH+23]	Neural	✓	×	×	×	General optim.
Data-driven	N/A	✓	✓	✓	×	PDEs
Stage 1	RBF	✓	✓	✓	×	PDEs
Stage 2	Neural	✓	✓	✓	✓	General optim.

3 Implementation

THOMAS EDISON — *"The value of an idea lies in the using of it."*

This chapter begins with a high-level design of PDE instances, and dataset (§ 3.1). As introduced in § 2.3, I formulate the PDE solvers adopted in this project (§ 3.2), covering Kansa Method, PINN, FNO and GNN. With Kansa Method as the main focus, I present stability augmentation (§ 3.3.2), linear, coupled PDEs, nonlinear operator cases (§ 3.3.1, 3.3.3, 3.3.4), auto tuning (§ 3.3.5), and incorporate novel kernels (§ 3.4). Lastly, I employ them in various scientific contexts, covering forward, inverse problems (§ 3.5), and application domains (§ 3.6).

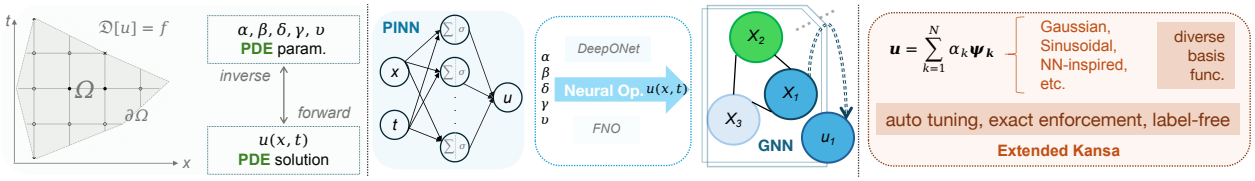


Figure 3.1: Left: Illustration of PDE, domains, and boundaries. Middle: High-level flowchart of the baseline methods, including PINN, NO and GNN. Right: The proposed approach with its extensions and features.

§ 3.1 PDE domain and dataset design

The high-level Unified Modeling Language (UML) is illustrated in Figure 3.1. I model each PDE `Domain` via *factory* design pattern [GHJV94]. This follows the open for extension but closed for modification principle, allowing extension of temporal $[t_0, t_f]$, spatial one Ω or their combinations. Each `Domain` instance has a `Sampler` in the overall data domain $\mathbb{D}$. The symbolic expressions are defined in § 2.2.1. The global configurations, e.g. noise level for perturbation and stability, PDE parameters, ICs and BCs, if static, are shared across solvers for fair comparison.

Sampling methods modelling. I adopt the *strategy* design pattern [GHJV94] for `Sampler` class, where subclasses are implemented for different domain sampling strategies. Examples are uniform, non-uniform samplers such as random one and many more.

Single instance formulation. A PDE `DataSample` contains corresponding `Domain` and `Sampler` for temporal or spatial dimensions, as well as the Initial Conditions (ICs). Its member functions include evaluation and visualization of solver outputs on this PDE instance.

Dataset formulation. For FNO, it requires a dataset of multiple PDE instances. This single instance class is then wrapped into `Dataset` class, which contains a set of `DataSample` instances with different initial conditions. During runtime, the `PyTorch` DataLoader loads the dataset in batches for model training and evaluation.

§ 3.2 PDE solvers architecture design details

For PINN backend network architecture, I employ a multi-scale Multi-Layer Perceptron (MLP) [WWP21] with three latent layers each having 64 neurons, that extends the standard fully

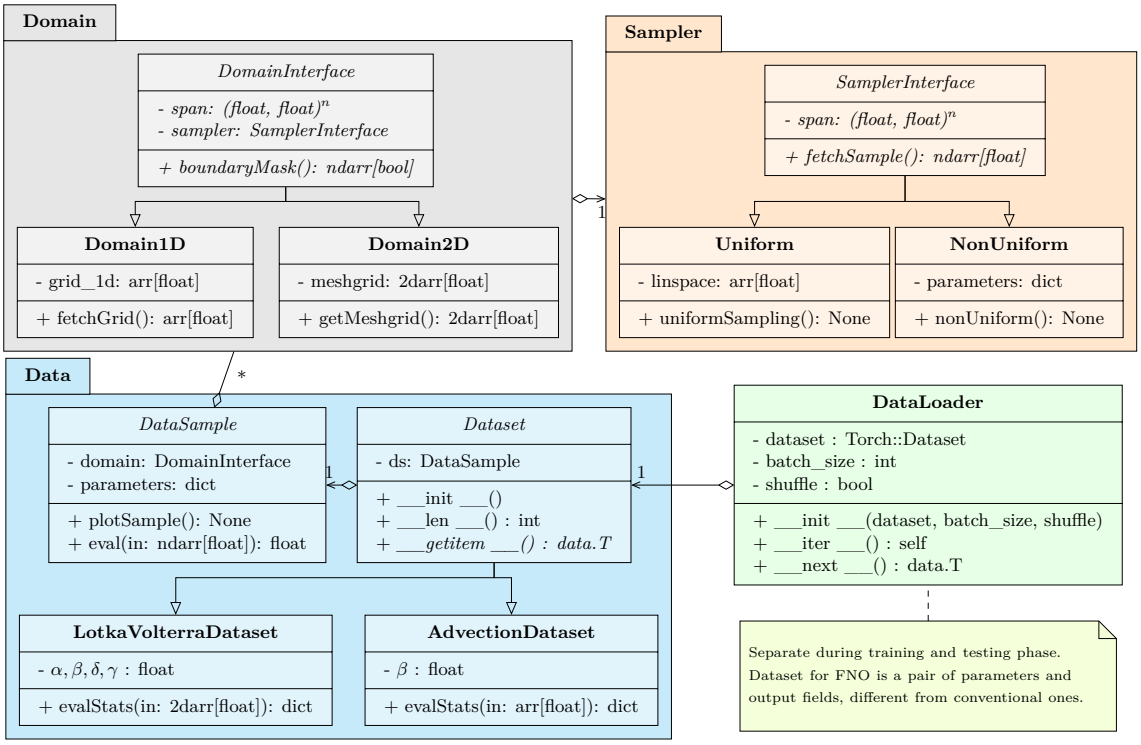


Figure 3.2: UML diagram of PDE data domain and dataset.

connected neural network structure by incorporating multiple frequency scales in the input encoding layers. This better captures the input features with varying frequencies.

For FNO, the spectral convolution layer is implemented by FFT and its inverse (iFFT) with 16 modes retained and 64-dimensional hidden feature space.

For GNN, the domain is represented as a directed graph, where the nodes are the solution fields of every spatial point at the current timestep. The edges are the connections between neighboring timestamps, each with an attribute representing the spatial distance between the connected nodes. The graph is updated at every timestep until the final time t_f. I adopt two message-passing (MP) [BWW22] layers with 32 hidden nodes and an Exponential Linear Unit (ELU) activation function, followed by the final linear layer with 32 neurons, to predict the solution field at the next timestep auto-regressively.

§ 3.3 Kansa algorithm

Equation (2.66) is expressed as, by rewriting the kernel functions into matrix form,

$$\underbrace{\begin{bmatrix}\hat{u}(\mathbf{x}_1)\\ \hat{u}(\mathbf{x}_2)\\ \vdots\\ \hat{u}(\mathbf{x}_N)\end{bmatrix}}_{\mathbf{u}\in\mathbb{R}^N} = \underbrace{\begin{bmatrix}\psi_1(||\mathbf{x}_1-\mathbf{x}_1||) & \psi_2(||\mathbf{x}_1-\mathbf{x}_2||) & \cdots & \psi_N(||\mathbf{x}_1-\mathbf{x}_N||)\\ \psi_1(||\mathbf{x}_2-\mathbf{x}_1||) & \psi_2(||\mathbf{x}_2-\mathbf{x}_2||) & \cdots & \psi_N(||\mathbf{x}_2-\mathbf{x}_N||)\\ \vdots & \vdots & \ddots & \vdots\\ \psi_1(||\mathbf{x}_N-\mathbf{x}_1||) & \psi_2(||\mathbf{x}_N-\mathbf{x}_2||) & \cdots & \psi_N(||\mathbf{x}_N-\mathbf{x}_N||)\end{bmatrix}}_{\text{kernel matrix } \mathbf{K}\in\mathbb{R}^{N\times N}} \cdot \underbrace{\begin{bmatrix}\alpha_1\\ \alpha_2\\ \vdots\\ \alpha_N\end{bmatrix}}_{\mathbf{a}\in\mathbb{R}^N}. \tag{3.1}$$

Each multi-dimensional PDE (2.9) can be summarized as a single equation,

$$\mathcal{F}[\hat{u}](\mathbf{x}_i) = h(\mathbf{x}_i), \quad \mathbf{x}_i \in \mathbb{D}, \tag{3.2}$$

in which $\mathcal{F} = \{\mathcal{D}, \mathcal{B}_i\}$ denotes the differential and boundary operators, $h = \{f, g_i\}$ corresponds to the constraint functions.

§ 3.3.1 Linear operator case

Replacing u with (2.66), and under the assumption of *linear* operator[1], PDEs (3.2) become,

$$\mathcal{F}[\hat{u}](\mathbf{x}_i) = \mathcal{F}[\sum_{k=1}^{N} \alpha_k \cdot \psi_k](\mathbf{x}_i) = \sum_{k=1}^{N} \alpha_k \cdot \mathcal{F}[\psi_k](\mathbf{x}_i) = h(\mathbf{x}_i). \tag{3.3}$$

Expressing the above equation in matrix form yields,

$$\underbrace{\begin{bmatrix} \mathcal{F}[\psi_1](\mathbf{x}_1) & \mathcal{F}[\psi_2](\mathbf{x}_1) & \cdots & \mathcal{F}[\psi_N](\mathbf{x}_1) \\ \mathcal{F}[\psi_1](\mathbf{x}_2) & \mathcal{F}[\psi_2](\mathbf{x}_2) & \cdots & \mathcal{F}[\psi_N](\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{F}[\psi_1](\mathbf{x}_N) & \mathcal{F}[\psi_2](\mathbf{x}_N) & \cdots & \mathcal{F}[\psi_N](\mathbf{x}_N) \end{bmatrix}}_{\text{operator-evaluated kernel matrix } \mathbf{F} \in \mathbb{R}^{N \times N}} \cdot \underbrace{\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{bmatrix}}_{\mathbf{a} \in \mathbb{R}^N} = \underbrace{\begin{bmatrix} h(\mathbf{x}_1) \\ h(\mathbf{x}_2) \\ \vdots \\ h(\mathbf{x}_N) \end{bmatrix}}_{\text{constraint values } \mathbf{h} \in \mathbb{R}^N} . \tag{3.4}$$

Specifically, for the Gaussian RBF kernel function in (2.63), with $\mathbf{x}_i = (x_i, t_i)$,

$$\psi_k = e^{-\frac{r_k^2}{2\sigma^2}}, \quad r_k^2 = (x_k - x_i)^2 + (t_k - t_i)^2. \tag{3.5}$$

Let $\mathcal{F} = \frac{\partial}{\partial t}$ [2]. Applying the chain rule, the matrix entry $\mathbf{F}_{k,i}$ in (3.4) is given by,

$$\mathcal{F}[\psi_k](\mathbf{x}_i) = \frac{\partial \psi_k(\mathbf{x}_i)}{\partial t} = \frac{\partial \psi_k(\mathbf{x}_i)}{\partial r_k^2} \cdot \frac{\partial r_k^2}{\partial t} = -\frac{1}{2\sigma^2} e^{-\frac{r_k^2}{2\sigma^2}} \cdot 2(t_k - t_i) = -\frac{t_k - t_i}{\sigma^2} e^{-\frac{r_k^2}{2\sigma^2}}. \tag{3.6}$$

Simultaneous equations. Given N_{eq} equations each subject to condition defined on N_j collocation points, indexed by equation j. Overall collocation points are $\{\mathbf{x}_i \in \mathbb{D}\}_{i=1}^{N_{\text{total}}}$, where repetition of collocation points is necessary to handle distinct constraints and $N_{\text{total}} = \sum_{j=1}^{N_{\text{eq}}} N_j$,

$$\mathcal{F}_j[\hat{u}](\mathbf{x}_i) = h_j(\mathbf{x}_i), \quad \forall j \in \{1, \ldots, N_{\text{eq}}\}, \mathbf{x}_i \in \mathbb{D}. \tag{3.7}$$

I extend (3.4) through vertically concatenating the matrix $\mathbf{F}^{(j)} \in \mathbb{R}^{N_{\text{total}} \times N_{\text{total}}}$ and constraint vector $\mathbf{h}^{(j)} \in \mathbb{R}^{N_{\text{total}}}$ over every equation j [ZFH+23], yielding the block matrix formulation,

$$\underbrace{\begin{bmatrix} \mathbf{F}^{(1)} \\ \vdots \\ \mathbf{F}^{(N_{\text{eq}})} \end{bmatrix}}_{\text{stacked } \mathbf{F} \in \mathbb{R}^{(N_{\text{eq}} \cdot N_{\text{total}}) \times N_{\text{total}}}} \cdot \underbrace{\begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_{N_{\text{total}}} \end{bmatrix}}_{\mathbf{a} \in \mathbb{R}^{N_{\text{total}}}} = \underbrace{\begin{bmatrix} \mathbf{h}^{(1)} \\ \vdots \\ \mathbf{h}^{(N_{\text{eq}})} \end{bmatrix}}_{\text{stacked } \mathbf{h} \in \mathbb{R}^{N_{\text{eq}} \cdot N_{\text{total}}}} . \tag{3.8}$$

The problem of deriving u (2.66) is reduced to solving coefficients $\mathbf{a} = [\alpha_1, \alpha_2, \ldots, \alpha_N]$, determined by simultaneous equations $\mathbf{Fa} = \mathbf{h}$. Such linear systems are commonly cast as a least squares problem, by reducing the error norm and thus enforcing a zero gradient,

$$\begin{aligned} \mathbf{a}^{\text{opt}} &= \min_{\mathbf{a}} (||\mathbf{Fa} - \mathbf{h}||)^2, \\ \nabla_{\mathbf{a}} (\mathbf{Fa} - \mathbf{h})^T (\mathbf{Fa} - \mathbf{h}) = 0 &\implies (\mathbf{F}^T \mathbf{F}) \mathbf{a}^{\text{opt}} = \mathbf{F}^T \mathbf{h}. \end{aligned} \tag{3.9}$$

Provided that $\mathbf{F}$ has full column rank, that matrix $\mathbf{F}^T\mathbf{F}$ becomes invertible, yielding the optimal coefficients $\mathbf{a}^{\text{opt}} = (\mathbf{F}^T\mathbf{F})^{-1}\mathbf{F}^T\mathbf{h}$. When $\mathbf{F}$ is additionally invertible and square, (3.9) simplifies to $\mathbf{a}^{\text{opt}} = \mathbf{F}^{-1}\mathbf{h}$. In either situation, the final approximation is obtained by inserting $\mathbf{a}^{\text{opt}}$ in (3.1): $\hat{u}(\mathbf{x}) = \mathbf{K} \cdot \mathbf{a}^{\text{opt}}$.

[1] According to homogeneity of linear operators in § 2.1.1, $\mathcal{F}[\alpha \cdot \psi] = \alpha \cdot \mathcal{F}[\psi]$.

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

Given hold-out evaluation or test points $\{\mathbf{x}_j^\star \in \mathbb{D}\}_{j=1}^M$, the kernel relation is formulated across collocation points $\{\mathbf{x}_i \in \mathbb{D}\}_{i=1}^N$ and test points, resulting in the test-time solution,

$$u(x,t) \approx \hat{u}(\mathbf{x}^\star) = \sum_{k=1}^{N} \alpha_k \cdot \psi_k(||\mathbf{x}^\star - \mathbf{x}_k||), \quad \mathbf{x} \in \mathbb{D}, \tag{3.10}$$

(3.10) is equivalent to the following matrix formulation,

$$\underbrace{\begin{bmatrix} \hat{u}(\mathbf{x}_1^\star) \\ \hat{u}(\mathbf{x}_2^\star) \\ \vdots \\ \hat{u}(\mathbf{x}_M^\star) \end{bmatrix}}_{\mathbf{u}\in\mathbb{R}^M} = \underbrace{\begin{bmatrix} \psi_1(||\mathbf{x}_1^\star - \mathbf{x}_1||) & \psi_2(||\mathbf{x}_1^\star - \mathbf{x}_2||) & \cdots & \psi_N(||\mathbf{x}_1^\star - \mathbf{x}_N||) \\ \psi_1(||\mathbf{x}_2^\star - \mathbf{x}_1||) & \psi_2(||\mathbf{x}_2^\star - \mathbf{x}_2||) & \cdots & \psi_N(||\mathbf{x}_2^\star - \mathbf{x}_N||) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_1(||\mathbf{x}_M^\star - \mathbf{x}_1||) & \psi_2(||\mathbf{x}_M^\star - \mathbf{x}_2||) & \cdots & \psi_N(||\mathbf{x}_M^\star - \mathbf{x}_N||) \end{bmatrix}}_{\text{kernel matrix } \mathbf{K}^\star\in\mathbb{R}^{M\times N}} \cdot \underbrace{\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{bmatrix}}_{\mathbf{a}\in\mathbb{R}^N}. \tag{3.11}$$

§ 3.3.2 Extension I: stability via Tikhonov regularization

I observe that the results from solving the linear system $\mathbf{Fa} = \mathbf{h}$ are unstable when the matrix $\mathbf{F}$ is ill-conditioned, i.e. the condition number $\kappa(\mathbf{F}^T\mathbf{F}) \gg 1$. This will degrade the generalization capability of the solution, especially when the input data is noisy or perturbed. Such issues are common given that the operator-evaluated kernel matrix $\mathbf{F}$ includes BCs and ICs, which are sparse and thus lead to ill-conditioning.

To mitigate the issue, I introduce **Tikhonov regularization**, also known as **Ridge regression** [HK70]. It extends the least squares objective in (3.9) by adding a L_2 regularization,

$$\mathbf{a}^{\text{opt}} = \arg\min_{\mathbf{a}} \left(||\mathbf{Fa} - \mathbf{h}||^2 + \lambda||\mathbf{a}||^2\right) \implies (\mathbf{F}^T\mathbf{F} + \lambda\mathbf{I})\mathbf{a}^{\text{opt}} = \mathbf{F}^T\mathbf{h}. \tag{3.12}$$

Compared with (3.9), the matrix $\mathbf{F}^T\mathbf{F}$ diagonals are augmented with positive values λ, which decreases the condition number (2.6) of the matrix, resulting in a more stable solution $\mathbf{a}^{\text{opt}}$,

$$\kappa(\mathbf{F}^T\mathbf{F} + \lambda\mathbf{I}) = \frac{\sigma_{\max}(\mathbf{F}^T\mathbf{F} + \lambda\mathbf{I})}{\sigma_{\min}(\mathbf{F}^T\mathbf{F} + \lambda\mathbf{I})} = \frac{\sigma_{\max}(\mathbf{F}^T\mathbf{F}) + \lambda}{\sigma_{\min}(\mathbf{F}^T\mathbf{F}) + \lambda}. \tag{3.13}$$

There exists a trade-off between stability (large λ) and accuracy (small λ). It is also known as the bias-variance trade-off [GBD92], i.e. high accuracy yields low bias but high variance, and vice versa. The optimal λ can be determined by cross-validation on the hold-out data.

§ 3.3.3 Extension II: coupled PDEs solution fields

Coupled PDE solution fields. Consider a solution with N_D dimensions of independent variables, $\mathbf{u} = [u_1, u_2, \ldots, u_{N_D}]$, Kansa approximation (2.66) indexed by d is,

$$\hat{u}_d(\mathbf{x}) = \sum_{k=1}^{N} \alpha_k^{(d)} \cdot \psi_k^{(d)}(||\mathbf{x} - \mathbf{x}_k||), \quad \forall d \in \{1, \ldots, N_D\}. \tag{3.14}$$

The coupled PDEs system is therefore established through the governing or coupling operator $\mathcal{G}$ across every solution dimension, each governed by $\mathcal{F}^{(d)}$,

$$\mathcal{G}\left(\mathcal{F}^{(1)}[\hat{u}_1], \ldots, \mathcal{F}^{(N_D)}[\hat{u}_{N_D}]\right)(\mathbf{x}_i) = h(\mathbf{x}_i), \quad \mathbf{x}_i \in \mathbb{D}. \tag{3.15}$$

I assume that $\mathcal{G}$ exhibits linearity along every solution dimension,

$$\mathcal{G}\left(\hat{v}_1, \ldots, \hat{v}_{N_D}\right)(\mathbf{x}) = \sum_{d=1}^{N_D} \beta_d \cdot \hat{v}_d(\mathbf{x}), \tag{3.16}$$

with $\beta_d \in \mathbb{R}$ being the weight associated with each dimension. Resembling (3.8), I extend (3.4) through horizontal concatenation of $\mathbf{F}^{(d)} \in \mathbb{R}^{N\times N}$ across every solution dimension d, yielding

the following block matrix formulation,

$$\underbrace{\begin{bmatrix}\beta_1\mathbf{I}_N & \cdots & \beta_{N_D}\mathbf{I}_N\end{bmatrix}}_{\boldsymbol{\beta}\in\mathbb{R}^{N\times(N_D\cdot N)}} \circ \underbrace{\begin{bmatrix}\mathbf{F}^{(1)} & \cdots & \mathbf{F}^{(N_D)}\end{bmatrix}}_{\text{coupling }\mathbf{F}\in\mathbb{R}^{N\times(N_D\cdot N)}} \cdot \underbrace{\begin{bmatrix}\mathbf{a}^{(1)}\\ \vdots \\ \mathbf{a}^{(N_D)}\end{bmatrix}}_{\mathbf{a}\in\mathbb{R}^{(N_D\cdot N)}} = \underbrace{\begin{bmatrix}h(\mathbf{x}_1)\\ \vdots \\ h(\mathbf{x}_N)\end{bmatrix}}_{\text{stacked }\mathbf{h}\in\mathbb{R}^{N}}, \tag{3.17}$$

for which Hadamard or element-wise product is denoted as $\circ$ and the $N \times N$ identity matrix is written as $\mathbf{I}_N$. Following (3.7) and (3.8), the solution of simultaneous coupled PDEs system is,

$$\forall j \in \{1,\ldots,N_{\text{eq}}\}, \mathbf{x}_i \in \mathbb{D}. \quad \mathcal{G}_j\left(\mathcal{F}_j^{(1)}[\hat{u}_1],\ldots,\mathcal{F}_j^{(N_D)}[\hat{u}_{N_D}]\right)(\mathbf{x}_i) = h_j(\mathbf{x}_i). \tag{3.18}$$

The system admits the following block matrix form, with horizontal and vertical concatenation,

$$\underbrace{\begin{bmatrix}\boldsymbol{\beta}^{(1)}\\ \vdots \\ \boldsymbol{\beta}^{(N_{\text{eq}})}\end{bmatrix}}_{\boldsymbol{\beta}\in\mathbb{R}^{(N_{\text{eq}}\cdot N_{\text{total}})\times(N_D\cdot N_{\text{total}})}} \circ \underbrace{\begin{bmatrix}\mathbf{F}^{(1,1)} & \cdots & \mathbf{F}^{(1,N_D)}\\ \vdots & \mathbf{F}^{(j,d)} & \vdots \\ \mathbf{F}^{(N_{\text{eq}},1)} & \cdots & \mathbf{F}^{(N_{\text{eq}},N_D)}\end{bmatrix}}_{\text{coupling }\mathbf{F}\in\mathbb{R}^{(N_{\text{eq}}\cdot N_{\text{total}})\times(N_D\cdot N_{\text{total}})}} \cdot \underbrace{\begin{bmatrix}\mathbf{a}^{(1)}\\ \vdots \\ \mathbf{a}^{(N_D)}\end{bmatrix}}_{\mathbf{a}\in\mathbb{R}^{(N_D\cdot N_{\text{total}})}} = \underbrace{\begin{bmatrix}\mathbf{h}^{(1)}\\ \vdots \\ \mathbf{h}^{(N_{\text{eq}})}\end{bmatrix}}_{\text{stacked }\mathbf{h}\in\mathbb{R}^{(N_{\text{eq}}\cdot N_{\text{total}})}}. \tag{3.19}$$

§ 3.3.4 Extension III: nonlinear operator case

Differentiable matrices facilitate the decomposition of a general operator $\mathcal{F}$ into a collection of linear and nonlinear operators. For linear operators, such as $\frac{\partial}{\partial x}$, the derivative relationship is written as $\mathbf{u}' = \mathbf{D}_x \cdot \mathbf{u}$. The matrix $\mathbf{D}_x$ is derived from Kansa (2.66), and by linearity (3.3),

$$\frac{\partial}{\partial x}u(x) = \sum_{k=1}^{N} \alpha_k \cdot \frac{\partial}{\partial x}\psi_k(||x - x_k||). \tag{3.20}$$

This yields the matrix form of $\mathbf{u}' = \mathbf{K}_{\mathbf{x}} \cdot \mathbf{a}$. I evaluate $\frac{\partial}{\partial x}\psi_k(||x - x_k||)|_{x=x_i}$ for all rows and columns indexed by $i, k \in \{1,\ldots,N\}$, which constructs the matrix $\mathbf{K}_{\mathbf{x}} \in \mathbb{R}^{N\times N}$.
By inverting (3.1), $\mathbf{a} = \mathbf{K}^{-1} \cdot \mathbf{u}$, assuming $\mathbf{K}$ is invertible owing to independent bases. By substituting $\mathbf{a}$ into $\mathbf{u}' = \mathbf{K}_{\mathbf{x}} \cdot \mathbf{a}$, one obtains $\mathbf{u}' = \mathbf{K}_{\mathbf{x}} \cdot \mathbf{K}^{-1} \cdot \mathbf{u}$. The differentiable matrix is thus,

$$\mathbf{D}_x = \mathbf{K}_{\mathbf{x}} \cdot \mathbf{K}^{-1} \in \mathbb{R}^{N\times N}. \tag{3.21}$$

$\mathcal{F}[u] = \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2}$ represents the operator for Burgers' equation (2.29). The corresponding differentiable matrix form, following the construction in (3.21) for each operator, is given by,

$$\mathcal{F}[u] = \mathbf{D}_t \cdot \mathbf{u} + \mathbf{u} \circ (\mathbf{D}_x \cdot \mathbf{u}) - \nu(\mathbf{D}_{xx} \cdot \mathbf{u}). \tag{3.22}$$

For the *nonlinear* operator $\mathcal{F}$, it is no longer possible to simplify (3.2) as is in (3.3). Hence, I present the corresponding solver below, which is divided into **two** categories, as shown in Table 3.1. I adopt **four** time stepping methods as the first category, covering both *explicit* and *implicit* schemes. The alternative category is a *fully nonlinear* solver that reduces the residuals of the PDE directly. A trade-off is present because the former achieves superior computational speed, whereas the latter delivers higher precision.

Explicit time stepping schemes. For any time-dependent PDE, the nonlinearity is removed through time discretization of the partial derivative with respect to t, retaining only the spatial operator $\mathcal{D}$. This has been used in the FDM. A typical choice is the explicit (1) *forward Euler* method, which entails only rearrangement and eliminates the nonlinearity as well as the

Table 3.1: Overview of main features among nonlinear Kansa solvers proposed, where N_t and N_x represent the number of spatio-temporal collocation points, and Δt denotes the time step interval. Fully nonlinear excels in accuracy order but is computationally expensive, while time stepping approaches are the exact opposite.

	Forward	IMEX	Backward	Crank-Nicolson	Fully nonlinear
Time step	explicit	semi-explicit	implicit	implicit	N/A
Order	$O(\Delta t)$	$O(\Delta t)$	$O(\Delta t)$	$O(\Delta t^2)$	$O(1)$
Memory	$O(N_x)$	$O(N_x^2)$	$O(N_x^2)$	$O(N_x^2)$	$O(N_x^2 N_t^2)$
Stability	unstable	stable	stable	stable	N/A

need for a linear system solver,

$$\frac{\partial u}{\partial t} + \mathcal{D}[u] = 0 \implies \frac{u^{n+1} - u^n}{\Delta t} + O(\Delta t) + \mathcal{D}[u^n] = 0, \tag{3.23}$$

Improved numerical stability can be achieved by the (2) *Implicit-Explicit (IMEX)* scheme instead, separating the operator into stiff and non-stiff components $\mathcal{D} = \mathcal{I}_{\text{stiff}} + \mathcal{E}_{\text{non-stiff}}$. The non-stiff part is intrinsically stable and thus computed explicitly. Conversely, the stiff part corresponds to the numerical unstable behavior of the operator, which requires implicit treatment,

$$\frac{u^{n+1} - u^n}{\Delta t} + O(\Delta t) + \mathcal{I}_{\text{stiff}}[u^{n+1}] + \mathcal{E}_{\text{non-stiff}}[u^n] = 0. \tag{3.24}$$

Although the nonlinearity may exist in $\mathcal{D}$, specifically $\mathcal{E}_{\text{non-stiff}}$, the solution at time step n is already available. Applying the differentiable matrix, $\mathcal{E}_{\text{non-stiff}}[u^n]$ can therefore be evaluated directly as a scalar. Solving the only unknown term u^{n+1} at the next time step yields the result. For nonlinear stiff parts, a nonlinear solver is adopted. Otherwise, a linear system solver works.

Implicit time stepping schemes. The (3) *backward* Euler scheme provides an alternative way to discretize the time derivative, the resulting formulation remains nonlinear but is more stable than the forward Euler counterpart,

$$\frac{\partial u}{\partial t} + \mathcal{D}[u] = 0 \implies \frac{u^{n+1} - u^n}{\Delta t} + O(\Delta t) + \mathcal{D}[u^{n+1}] = 0. \tag{3.25}$$

Accordingly, the linear system solver is replaced with a nonlinear optimization method, such as Newton-Raphson solver [Ypm95]. The goal is to reduce the PDE residual term and compute the next target solution along the time dimension,

$$u^{n+1} = \arg\min_{u^{n+1}} \mathbf{r}^{n+1}, \text{ where } \mathbf{r}^{n+1} = u^{n+1} - u^n + \Delta t \cdot \mathcal{D}[u^{n+1}]. \tag{3.26}$$

For more precision, the (4) *Crank-Nicolson (C-N)* scheme discretizes the time derivative with second-order accuracy,

$$\frac{\partial u}{\partial t} + \mathcal{D}[u] = 0 \implies \frac{u^{n+1} - u^n}{\Delta t} + \frac{1}{2}\left(\mathcal{D}[u^{n+1}] + \mathcal{D}[u^n]\right) + O(\Delta t^2) = 0. \tag{3.27}$$

In line with (3.26), I obtain the subsequent result by reducing the residual term in (3.27).

Fully nonlinear time-stepping-free solver. Extra error is introduced by all time stepping schemes, irrespective of explicit or implicit schemes, due to unnecessary temporal derivative discretization. To alleviate such error, I reduce the PDE residuals (3.2) over the complete

spatio-temporal domain. Applying the differentiable matrices of $\mathcal{F}$, the optimization target is,

$$\alpha = \arg\min_{\alpha} \sum_{i=1}^{N} \left(\mathcal{F}[\hat{u}](\mathbf{x}_i) - h(\mathbf{x}_i)\right)^2. \tag{3.28}$$

Using the Kansa approximation (2.66), the full-domain solution u is obtained.

§ 3.3.5 Kernel parameter auto tuning

An essential hyperparameter in Kansa method is the kernel parameter ϵ in (2.64), which controls the shape of kernels and determines the contribution of each basis function of pairwise distance. Only values that are neither too small nor too large would yield a reasonable solution. Otherwise, if the kernel matrix is ill-conditioned, it leads to failure when solving coefficients α_i from the linear system. For optimizer selection, grid search is a straightforward solution. Gradient-based methods introduced below serve as an alternative option.

[ZFH+23] introduced a ϵ self-tuning strategy. It reduces the operator-evaluated kernel matrix $\mathbf{F}$ condition number, and variation of the solution field u across the spatio-temporal domain,

$$\epsilon^{\star} = \arg\min_{\epsilon} \omega_1 \cdot \text{cond}(\mathbf{F}) + \omega_2 \cdot \int_{\mathbb{D}} ||\nabla u(\mathbf{x})||^2 d\mathbf{x}, \tag{3.29}$$

for which the matrix $\mathbf{F}$ (3.4) condition number is cond($\mathbf{F}$). To compute the integration, I adopt the numerical quadrature method, which approximates the integral with a sum of Monte Carlo samples. This method is effective for multi-dimensional coupled PDEs with *linear* operators. In the case of *nonlinear* operators, ϵ implicitly affects the coefficients α_i, which in turn influence the solution field u. $\mathbf{F}$ is no longer explicitly formed. Hence, I reduce the PDE residuals across the whole domain, together with the solution field u aggregate variation, and an optional supervised L_2 loss of the reference u^{gt} and prediction u, when given a labeled dataset,

$$\epsilon^{\star} = \arg\min_{\epsilon} \omega_1 \cdot \sum_{i=1}^{N} \left(\mathcal{F}[\hat{u}](\mathbf{x}_i) - h(\mathbf{x}_i)\right)^2 + \omega_2 \cdot \int_{\mathbb{D}} ||\nabla u(\mathbf{x})||^2 d\mathbf{x} + \omega_3 \cdot ||u - u^{gt}||^2, \tag{3.30}$$

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

§ 3.4 Infinite-width ReLU networks kernel

Besides Radial Basis Function (RBF), there are many novel kernels proposed, some of which are inspired by neural networks in the machine learning community. One such kernel stems from an infinitely wide network with weights $\mathbf{w}$ and Rectified Linear Unit (ReLU) [FA18], denoted as $\text{ReLU}(z) := [z]_+ = \max(0, z)$,

$$\psi_n(\mathbf{x}, \mathbf{c}) = \int \mathcal{N}(\mathbf{w}; 0, \mathbf{I}) [\mathbf{w}^T \tilde{\mathbf{x}}]_+^n [\mathbf{w}^T \tilde{\mathbf{c}}]_+^n d\mathbf{w}, \quad n \in \mathbb{Z}^+. \tag{3.31}$$

Arc-cosine kernels are given by [CS09], which initially adopts the Heaviside step function $\Theta(z) = \frac{1}{2}(1 + \text{sign}(z))$ as the activation function, and is later extended to ReLU [FA18]. The above kernel function to measure the similarity between two input vectors $\mathbf{x}$ and $\mathbf{c}$ is simplified to a closed-form solution,

$$\psi_n(\mathbf{x}, \mathbf{c}) = \frac{||\tilde{\mathbf{x}}||^n ||\tilde{\mathbf{c}}||^n}{\pi} J_n(\theta), \text{ where } \theta = \measuredangle(\tilde{\mathbf{x}}, \tilde{\mathbf{c}}) = \arccos\left(\frac{\tilde{\mathbf{x}}^T \tilde{\mathbf{c}}}{||\tilde{\mathbf{x}}|| \cdot ||\tilde{\mathbf{c}}||}\right), \tag{3.32}$$

where the angular dependence $J_n(\theta)$ is given by a family of functions, and $\tilde{\mathbf{x}} = [\mathbf{x}^T, 1]^T$, $\tilde{\mathbf{c}} = [\mathbf{c}^T, 1]^T$ are the input and center vectors in homogeneous coordinates, serving as the bias term in the kernel function. When $n = 1$, following Kaiming-He initialization [HZRS15], e.g. Gaussian

or Uniform weight distribution, the angular function is [WGK+22],

$$J_1(\theta) = \sin\theta + 2(\pi - \theta)\cos\theta. \tag{3.33}$$

To compute the angle more uniformly accurate, especially near zero range, I adopt [Kah16],

$$\theta = \measuredangle(\tilde{\mathbf{x}}, \tilde{\mathbf{c}}) = 2\arctan\left(\left|\left|\frac{\frac{\tilde{\mathbf{x}}}{||\tilde{\mathbf{x}}||} - \frac{\tilde{\mathbf{c}}}{||\tilde{\mathbf{c}}||}}{\frac{\tilde{\mathbf{x}}}{||\tilde{\mathbf{x}}||} + \frac{\tilde{\mathbf{c}}}{||\tilde{\mathbf{c}}||}}\right|\right|\right) = 2\arctan\left(\left|\left|\frac{||\tilde{\mathbf{c}}||\tilde{\mathbf{x}} - ||\tilde{\mathbf{x}}||\tilde{\mathbf{c}}}{||\tilde{\mathbf{c}}||\tilde{\mathbf{x}} + ||\tilde{\mathbf{x}}||\tilde{\mathbf{c}}}\right|\right|\right). \tag{3.34}$$

§ 3.5 Inverse problem solutions for PDEs

PDE inverse problems. Provided u^{obs} measurements, the task is to derive the target $\boldsymbol{\pi}$, such as PDE parameters, the initial or boundary conditions. This is achieved by reducing the mismatch $\mathcal{L}$ across the u^{obs} and the prediction $u^{\text{pred}}(\boldsymbol{\pi})$ through the optimization process,

$$\boldsymbol{\pi}^\star = \arg\min_{\boldsymbol{\pi}} \mathcal{L}(u^{\text{obs}}, u^{\text{pred}}(\boldsymbol{\pi})). \tag{3.35}$$

For the underlying optimizer, I employ methods with gradient and those without, such as the least squares, or root finding procedures in `SciPy`, which is detailed in the Evaluation chapter.

§ 3.6 Applications of proposed methods

In this section, I present some applications from different scientific domains with proposed Kansa Method. In addition, I discuss the techniques to deploy them in real-world scenarios, e.g. pre-computation of reusable components, concurrent calculation and sparse matrix operators.

§ 3.6.1 Deployment techniques

Pre-computation. The coefficients α_i in Kansa approximation (2.66) are derived by solving the linear system (3.4), which are reusable for different inference test cases as long as the collocation points and kernel function remain unchanged. Thus, I pre-compute the α values and store them for later different test-time simulation computation, reducing the test-time inference computation and thus latency.

Concurrent calculation. For simulation that shared the same computation procedure but with different initial conditions or kernel function hyperparameters, I compute them concurrently on all available devices, i.e. multi-core CPU or GPU, by parallelizing the matrix operations and linear system solver. This brings significant speedup compared to sequential computation.

§ 3.6.2 Sparse matrix operators

Kansa Method (KM) relies heavily on the matrix representations and operations, which incurs both memory and computation costs. This yields space for optimization with consideration of the matrix properties. For example, the operator-evaluated kernel matrices $\mathbf{F}$ of shape $M \times N$ are sparse due to the presence of both BCs and ICs, which only have non-zero values at the collocation points on the boundary and initial timestep. These matrices are sparse.

Sparse matrix [GVL13] is the one having a large proportion of zero entries. To represent the sparse matrix, there are a few widely used formats, i.e. Coordinate format (COO), Compressed Sparse Row (CSR), Compressed Sparse Column (CSC). I denote the Number of Non-Zero entries (NNZ) in the matrix $\mathbf{F}$ as $\text{NNZ}(\mathbf{F})$ and that for the r-th row or c-th column as $\text{NNZ}(r)$

or $\mathrm{NNZ}(c)$.

Coordinate format format tracks the non-zero entries by storing the row and column indices, forming a list of three-element tuples with length equal to $\mathrm{NNZ}(\mathbf{F})$ as below,

$$\{(\mathrm{row}_k, \mathrm{col}_k, \mathrm{value}_k) = (i, j, \mathbf{F}_{ij}) \mid \mathbf{F}_{ij} \neq 0\}_{k=1}^{\mathrm{NNZ}(\mathbf{F})}. \tag{3.36}$$

CSR format [BFF+09], also known as the Yale format, resembles the COO format but stores each row index no more than once via the following recursive row stride stride_r,

$$\begin{gathered}\{\mathrm{stride}_r\}_{r=1}^{M+1}, \{(\mathrm{col}_k, \mathrm{value}_k) = (j, \mathbf{F}_{ij}) \mid \mathrm{stride}_i \leq k < \mathrm{stride}_{i+1} \wedge \mathbf{F}_{ij} \neq 0\}_{k=1}^{\mathrm{NNZ}(\mathbf{F})}, \\ \text{where } \mathrm{stride}_0 = 0, \text{ and } \mathrm{stride}_r = \mathrm{stride}_{r-1} + \mathrm{NNZ}(r-1), \quad \forall r \in \{1, \ldots, M\}.\end{gathered} \tag{3.37}$$

CSC format is the column-wise version of the above CSR format, which stores the column stride stride_c no more than once as below,

$$\begin{gathered}\{\mathrm{stride}_c\}_{c=1}^{N+1}, \{(\mathrm{row}_k, \mathrm{value}_k) = (i, \mathbf{F}_{ij}) \mid \mathrm{stride}_j \leq k < \mathrm{stride}_{j+1} \wedge \mathbf{F}_{ij} \neq 0\}_{k=1}^{\mathrm{NNZ}(\mathbf{F})}, \\ \text{where } \mathrm{stride}_0 = 0, \text{ and } \mathrm{stride}_c = \mathrm{stride}_{c-1} + \mathrm{NNZ}(c-1), \quad \forall c \in \{1, \ldots, N\}.\end{gathered} \tag{3.38}$$

Hence, CSR and CSC formats are more memory efficient than COO format, with a total of $2 \cdot \mathrm{NNZ}(\mathbf{F}) + M + 1$ and $2 \cdot \mathrm{NNZ}(\mathbf{F}) + N + 1$ entries respectively, compared to $3 \cdot \mathrm{NNZ}(\mathbf{F})$ entries in COO format. As long as the compressed storage is smaller than $M \times N$, the sparse matrix format is more memory efficient than the dense one. In this work, I adopt the `SciPy` library for sparse matrix representation and operations, including sparse matrix-vector multiplication and sparse linear system solver.

§ 3.6.3 Application I: Gray-Scott pattern formation

Reaction-diffusion systems are used to represent textures in Computer Graphics [WK91]. They are similar with the Lotka-Volterra equations (§ 2.2.4), but with addition of diffusion term, and slightly different reaction term. Here I utilize a two-component reaction-diffusion system $u(t), v(t) \in \mathbb{R}^{M \times N}$, which was first proposed by Alan Turing [Tur52],

$$\begin{pmatrix} \frac{\partial u(t)}{\partial t} \\ \frac{\partial v(t)}{\partial t} \end{pmatrix} = \underbrace{\begin{pmatrix} d_u & 0 \\ 0 & d_v \end{pmatrix} \begin{pmatrix} \nabla^2 u(t) \\ \nabla^2 v(t) \end{pmatrix}}_{\text{diffusion terms}} + \underbrace{\begin{pmatrix} P(u,v) \\ Q(u,v) \end{pmatrix}}_{\text{reaction terms}}, \tag{3.39}$$

where coefficients $d_u = 0.04$, $d_v = 0.02$. The local chemical reactions terms are set as,

$$P(u,v) = -uv^2 + f(1-u), \quad Q(u,v) = uv^2 - (f+k)v, \tag{3.40}$$

given that the feed and kill rates $f, k \in \mathbb{R}^+$ control the pattern shapes directly. It can be rewritten as a system of two coupled PDEs,

$$\begin{cases} \frac{\partial u(t)}{\partial t} - d_u \nabla^2 u(t) + u(t)v^2(t) - f + fu(t) = 0, \\ \frac{\partial v(t)}{\partial t} - d_v \nabla^2 v(t) - u(t)v^2(t) + (f+k)v(t) = 0. \end{cases} \tag{3.41}$$

The initial conditions are set as $u(0) = 1$ and $v(0) = 0$ everywhere other than a half-size square in the center with $u(0) = 0.5, v(0) = 0.25$ and optionally with some noise added. This creates a non-zero reaction term $u(t)v^2(t)$ in the center, catalyzing the pattern formation.

For displaying as two-dimensional textures of shape $\mathbb{R}^{M \times N}$, I solve the above system with time discretization for $t \in [0, T] \subset \mathbb{Z}^+$, and final timestep $T = 8000$. The results are shown for $v(t)$ only. To compute efficiently, I adopt the truncated Gaussian kernel with $\tau = 0.45 \max(M, N)$ as defined in § 2.3.5, and run concurrently with 8 threads on 8-core CPU. The kernel matrix

has been precomputed beforehand, which are reused and fetched for per-timestep simulation.

§ 3.6.4 Application II: Physics simulation

Here I present three downstream simulation applications, where all the physics update functions are implemented via the proposed method with either Gaussian kernel or Multiquadrics. For backend physics simulator, I utilize `Taichi` [HLA+19] for its efficiency and differentiability.

Mass-spring mesh grid. I model the mass-spring system as a grid of $M \times N$ identical masses connected by massless springs, which is a common model for cloth, hair, water surface, and other deformable objects in Computer Graphics [P+95]. The governing equation for mass (m, n) position $x_{(m,n)}$ is given by the Newton's second law,

$$m \cdot \frac{d^2 x_{(m,n)}}{dt^2} = F^{\text{ext}}_{(m,n)}, \quad \text{where } F^{\text{ext}}_{(m,n)} = \sum_{(m',n') \in \mathcal{N}(m,n)} F_{(m',n') \to (m,n)}, \tag{3.42}$$

where m is the mass value, and the aggregate external force on mass (m, n) is the sum of forces exerted by its neighboring masses $\mathcal{N}(m, n)$.

Here, I consider the **structural constraints** only, i.e. between masses (m, n) and $(m + 1, n)$ or $(m, n + 1)$, for all $m \in \{1, \ldots, M - 1\}$ and $n \in \{1, \ldots, N - 1\}$. There are different types of the constraints, including Hooke's law as force and Burgers' equation as acceleration below.

(i) Hooke's law as force. Each neighboring force is computed by Hooke's law, where k is the spring stiffness and l_0 is the spring rest length,

$$F_{(m,n)\to(m+1,n)} = k \cdot (||x_{(m+1,n)} - x_{(m,n)}|| - l_0) \cdot \frac{x_{(m+1,n)} - x_{(m,n)}}{||x_{(m+1,n)} - x_{(m,n)}||}. \tag{3.43}$$

(ii) Burgers' equation as acceleration. With the above mass-spring grid setting, I adopt Burgers' equation (2.29) to constrain the acceleration of each mass instead of Hooke's law,

$$\frac{d^2 x_{(m,n)}}{dt^2} = \underbrace{\nu \cdot \nabla^2 x_{(m,n)} - x_{(m,n)} \cdot \nabla x_{(m,n)} - \frac{\partial x_{(m,n)}}{\partial t}}_{\text{Burgers' equation}}. \tag{3.44}$$

When plugging 1D Burgers' equation in the Newton's second law (3.42), we have the position $x_{(m,n)}$ of mass (m, n) as x_z in the z-axis,

$$\underbrace{\nu \cdot \frac{\partial^2 x_z}{\partial z^2} - x_z \cdot \frac{\partial x_z}{\partial z} - \frac{\partial x_z}{\partial t}}_{\text{1D Burgers' equation}} = \frac{F^{\text{ext}}}{m}. \tag{3.45}$$

The external force is set as constant to reduce complexity, or the sinusoidal function of time,

$$F^{\text{ext}}_{(m,n)} = A \cdot \sin\left(\omega(x_{(m,n)} - vt)\right). \tag{3.46}$$

In this work, I consider $A = 2$, $\omega = \frac{\pi}{2}$, and wave velocity $v = 2$.

(iii) Rendering. The mass-spring grid scene is set up with ambient light, point light above the center of the grid. The meshes are rendered with the Fresnel linear interpolation for diffuse sky and water reflection, and the Blinn-Phong for specular highlights.

Heat equation visualisation. I visualize the heat equation (3.47) for temperature field T over time, when given an initial temperature distribution $T(\vec{x}, 0)$ and boundary conditions,

$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T. \tag{3.47}$$

where α is the diffusivity constant. Here, I set $\alpha = 0.5$ and the initial temperature distribution,

$$T(\vec{x}, 0) = \begin{cases} 1, & \text{if } ||\vec{x} - \vec{x}_0|| < r^2 \text{ and } \vec{x} > \vec{x}_1, \\ 0, & \text{otherwise.} \end{cases} \tag{3.48}$$

Specially, the constant $\vec{x}_0 = (0.5, 0, 0.5)$, and $\vec{x}_1 = (0, 0.75, 0)$, representing part of the sphere with radius $r = 0.1$. I build a grid of cells for efficient computation, by merging a few close vertices into one larger cell. To speed up the Laplacian operator, I precompute the neighboring relationship between cells beforehand. The prefix sum array tracks the starting index of each large cell, helping to order the collocation points by the cell they belong to.

Particle systems driven by a temperature field. I simulate the particle system with $N = 4096$ spherical particles with radius $r = 0.003$, each with position, velocity $\vec{x}_i, \vec{v}_i \in \mathbb{R}^2$.
Temperature field $T(\vec{x}, t)$ governing equation is the reaction-diffusion,

$$\frac{\partial T}{\partial t} = \nu \nabla^2 T - \lambda T + S(x), \tag{3.49}$$

where the viscosity is $\nu = 0.1$, and the cooling rate is $\lambda = 1.2$. For the source term $S(\vec{x})$, which is near the bottom of the 2D domain, I set it according to a non-zero decreasing function,

$$S(\vec{x}) = \begin{cases} 2.5(1 - \frac{y}{0.1}), & \text{if } y < 0.1, \\ 0, & \text{otherwise.} \end{cases}, \tag{3.50}$$

where $y \in \mathbb{R}$ denotes the vertical coordinate, i.e. the second dimension of $\vec{x}$.
Dynamics system. Each particle is initialized randomly towards the lower boundary of the 2D domain, and later constrained by the damping force, the vertical buoyancy, and the horizontal turbulence. The latter two are driven by the temperature field $T_i(\vec{x}_i, t) \in \mathbb{R}$ at its position,

$$\begin{cases} \vec{v}_i \leftarrow \vec{v}_i \cdot (1 - \gamma), & \text{velocity damping.} \\ \vec{v}_i \leftarrow \vec{v}_i + \underbrace{\beta\, T_i(\vec{x}_i, t) \cdot \hat{y}}_{\text{buoyancy}} + \underbrace{\eta\, \mathcal{U}(-1, 1) T_i(\vec{x}_i, t) \cdot \hat{x}}_{\text{turbulence}}. \\ \vec{x}_i \leftarrow \vec{x}_i + \vec{v}_i \cdot \Delta t, & \text{position-velocity update.} \end{cases}, \tag{3.51}$$

where $\gamma = 0.1$ corresponds to the damping factor, $\beta = 0.4$ denotes the buoyancy factor, $\eta = 0.1$ is the turbulence factor, and $\mathcal{U}(-1, 1) \in \mathbb{R}$ represents a random scalar value sampled from a uniform distribution between -1 and 1.
Respawn logic. When a particle goes out of the boundary, it is respawned to a random initial position with zero velocity. Due to the asynchronous respawn logic, we can observe a progressive widening of the vertical distribution of particles after a long simulation time.

Qualitative evaluation. The simulation results of the aforementioned applications, i.e. the Gray-Scott pattern formation, and three physics simulations, are shown in § 4.7. These results are providing a visionary demonstration of the proposed methods in various scientific domains.

4 Evaluation

George E. P. Box — *"All models are wrong, but some are useful."*

This chapter explores evaluation metrics (§ 4.1), followed by experimental setup (§ 4.2), and results for proposed method against other PDE solvers, i.e. FDM, PINN, FNO, GNN and Kansa Method. I evaluate them on the advection equation (§ 4.3), with accuracy-stability trade-off and ablation studies (§ 4.4), coupled PDEs (§ 4.5), i.e. the Lotka-Volterra or Maxwell's equations, nonlinear PDEs (§ 4.6), i.e. the Burgers' equation, and various application domains (§ 4.7).

§ 4.1 Quantitative metrics

Correctness. The L_2 risk $\mathcal{R}_{L_2}$ measures the weighted average of difference between ground truth reference u_i and solver results $\hat{u}_i$ at every N_{test} test point on the spatial-temporal domain,

$$\hat{\mathcal{R}}_{L_2} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} ||\hat{u}_i - u_i||_2, \quad \hat{\mathcal{R}}_{\text{relative } L_2} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \frac{||\hat{u}_i - u_i||_2}{||u_i||_2}. \tag{4.1}$$

The relative L_2 risk stems from normalizing $\mathcal{R}_{L_2}$ with the L_2 norm of the reference u_i.

Stability. When the inputs, e.g. initial condition, boundary condition, or PDE parameters, are perturbed by a small noise ϵ, the solver or model should be robust or reliable to such perturbations or uncertainties. Consistent with classical numerical analysis, we define two types of stability metrics $S_{\mathcal{U}}$ here, i.e. model stability $S_{\mathcal{U}}^{\mathcal{M}}$ and solver stability $S_{\mathcal{U}}^{\mathcal{S}}$.
Model stability measures how a fixed model $\mathcal{M}$ predicts u_θ given input perturbation $\delta > 0$. A PDE model is stable if there exist constants $S_{\mathcal{U}}^{\mathcal{M}} > 0$ and $\delta_0 > 0$ such that for all $\|\delta\|_F \leq \delta_0$ and for all input conditions $f \in \mathcal{F}$,

$$\|u_\theta(f + \delta) - u_\theta(f)\|_{\mathcal{U}} \leq S_{\mathcal{U}}^{\mathcal{M}} \|\delta\|_{\mathcal{F}}, \tag{4.2}$$

where subscripts $\mathcal{U}$ and $\mathcal{F}$ are the solution space and input space norms. Neural operators perform model-level stability well, since they don't need to be retrained on perturbation data. *Solver stability*, also termed as algorithm-level stability, is defined likewise, except that the solver $\mathcal{S}$ is not fixed and thus retrained or re-optimized with the perturbed input data,

$$\|\mathcal{S}(f + \delta) - \mathcal{S}(f)\|_{\mathcal{U}} \leq S_{\mathcal{U}}^{\mathcal{S}} \|\delta\|_{\mathcal{F}}. \tag{4.3}$$

When both spaces $\mathcal{U}$ and $\mathcal{F}$ are in the L_∞ norm (2.3), the stability metric S_{L_∞} is defined as the ratio between the output perturbation and the input perturbation. When the input perturbation is fixed, it reflects solely the output perturbation $\|u_\theta(f + \delta) - u_\theta(f)\|_\infty$.
Within the experiments present in this work, I adopt the *solver stability* metric (4.3) for evaluation across all solvers. The underlying reason is that PINN, FDM and Kansa Method require case-by-case retraining, and thus solver-level stability becomes more relevant for them.

§ 4.2 Experimental configuration

I conduct every experiment on the Mac M1, equipped with 16 GB of RAM and eight-core CPUs, each operating at 3.2 GHz. Domain sampling is carried out with uniform distribution

under the same seeding, except stated otherwise.

§ 4.3 Assessment of solving advection equation

As formulated in (2.11) and prescribed by Table 4.1, I evaluate the proposed method against baseline solvers on the 1D *advection* equation. I sample $N_{\mathcal{R}} = 100 \times 10$ collocation points over the spatial-temporal domain, $N_{\mathcal{B}} = 100 \times 2$ for BCs, and $N_d = 10$ for ICs. For hold-out *testing*, $N_{\text{test}} = 64 \times 8$ are sampled, which is disjoint from the training data for fair evaluation. I quantify the *error* using relative L_2 risk (4.1) $\hat{\mathcal{R}}_{\text{relative } L_2}$. Within the Kansa Method, I set the kernel coefficient ϵ (2.64), i.e. the shape of Gaussian RBF, as 0.22, after tuning for optimal performance. I train the PINN, FNO and GNN models until they converge, using a learning

Table 4.1: Benchmark configuration for the 1D advection equation.

Time span	ICs	Region	BCs	Constant
$t_0 = 0, t_f = 1$	$u_0(x) = \sin(2\pi x)$	$x_0 = 0, x_f = 1$	per (2.12)	$\beta = 0.4$

rate $\eta = 10^{-3}$. The resulting number of epochs for PINN is $N_{\text{iter}} = 3000$ iterations, while GNN and FNO are executed for $N_{\text{iter}} = 100$ iterations. For FNO training, a dataset of varying PDE instances is necessary to learn the mapping from the input parameter space to the target solution space. Accordingly, I synthesize $N_{pde} = 100$ PDE samples with the initial condition below, where the random distribution is used to generate $c_k \sim \mathcal{N}(0, 1)$,

$$u_0(x) := \frac{u_0(x)}{\max_x |u_0(x)|}, \text{ where } u_0(x) = \sum_{k=1}^{5} c_k \sin(2\pi k x). \tag{4.4}$$

§ 4.3.1 Forward solution

Given that I trained FNO using $N_{pde} \in [1, 100] \subset \mathbb{R}^+$ PDE instances, additional training samples are introduced for solvers on a single PDE instance, such as FDM, PINN and Kansa Method, to align compute resources and thus ensure a fair comparison. The rescaling coefficient is $C_{\text{scale}} = N_{pde}$. Consequently, $N'_{\mathcal{R}} = C_{\text{scale}} \times N_{\mathcal{R}}$ collocation points are sampled, and the corresponding approaches are termed FDM$^{C_{\text{scale}}}$ and PINN$^{C_{\text{scale}}}$. I present the evaluation outcomes at the inference time in Table 4.2.

For Kansa method, I evaluate both without (w/o) and with (w/) **Tikhonov regularization** (Tikh.), where the regularization strength λ is set to be 10^{-9}. For more detailed ablation studies, please refer to § 4.4 or Figure 4.2.

Correctness. Table 4.2 shows that every solver scales effectively with the computational domain resolution, for which higher C_{scale} yields improved accuracy. Kansa method exceeds baseline methods, with the best performance of up to 10^{-5} error under just $C_{\text{scale}} = 2^2$. Nevertheless, as C_{scale} goes beyond 10^2, the memory limitations are reached.

Stability. I present results for the stability metric (4.3) in Table 4.3, where the Kansa method without Tikh. regularization is the least stable method, as also reflected by its large matrix condition number. With the incorporation of Tikh. regularization, it achieves a stability level even better than the FDM.

Table 4.2: Models correctness $\hat{\mathcal{R}}_{\text{relative } L_2} \times 10^{-3}$ (↓) for the 1D advection with respect to the computational grid resolution.

C_{scale}	FDM	PINN	FNO	MP-GNN	KM w/o	KM w/ Tikh.
1	36.63	300.2	744.3	74.85	8.084	**1.133**
2^2	17.05	20.68	58.71	61.87	**0.023**	0.385
4^2	**7.478**	8.654	37.68	44.22	N/A	N/A
$N_{pde} = 10^2$	**3.228**	6.457	13.37	57.59	N/A	N/A
avg.	16.10	83.99	213.5	59.63	4.05	0.759
std.	12.87	124.9	306.9	10.93	5.69	0.53

Table 4.3: Solvers stability S_{L_∞} (↓) for the 1D advection with respect to the computational grid resolution.

C_{scale}	FDM	PINN	FNO	MP-GNN	KM w/o	KM w/ Tikh.
1	0.2037	0.6631	0.5349	0.2992	2.121	**0.0395**
2^2	0.2197	0.0626	0.3585	0.4158	38.74	**0.0269**
4^2	0.2005	**0.0373**	0.3653	0.4559	N/A	N/A
10^2	0.1998	**0.0277**	0.3341	0.3854	N/A	N/A
avg.	0.206	0.198	0.398	0.389	20.43	0.033
std.	0.009	0.311	0.092	0.003	25.89	0.009

§ 4.3.2 Inverse estimation

Continuing with the 1D *advection* equation (2.11) set up in Table 4.1, I aim to estimate β when given observations u^{obs} across the domain and time steps. With optimal solvers from the forward problem, I present their performance of inverse estimation in Table 4.4.

Table 4.4: Estimation of β in the advection equation, for which reference is $\beta = 0.4$.

β_0	FDM	PINN	FNO	MP-GNN	KM
0.2	0.402	0.39987	0.3985	0.402	0.402
1.0	1.000	0.40446	1.2267	0.419	0.402

The proposed approach achieves consistently the best accuracy in the inverse problem. For search procedures adopted to identify optimal coefficient, convergence is often attracted to various suboptimal points that depend on the starting value β_0. With multiple runs, they yield higher fidelity at the cost of extra computation.

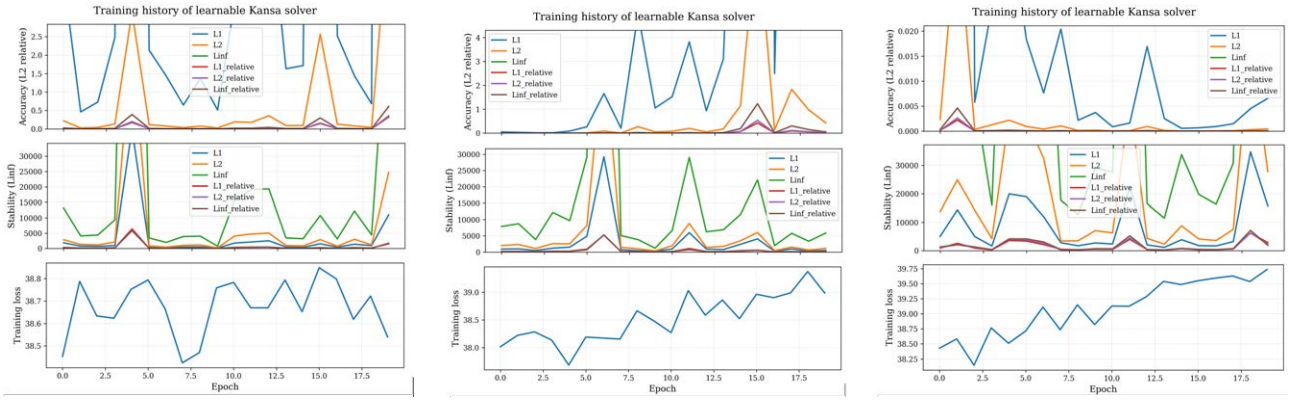
§ 4.4 Extension I: accuracy-stability trade-off

Ablation studies. I follow the experimental setup defined above to investigate the stability-accuracy bi-optimization of Kansa method.

To show the utility of **auto tuning** of Kansa method, I conduct studies showing the training history in Figure 4.1, where the shape parameter ϵ is tuned only by minimizing the condition number $\kappa(A)$ of the coefficient matrix A in Kansa method. I display (relative) error metrics of both accuracy and stability under different p-norm (2.2).

The increasing loss here is expected, given the large learning rate. The small learning rate

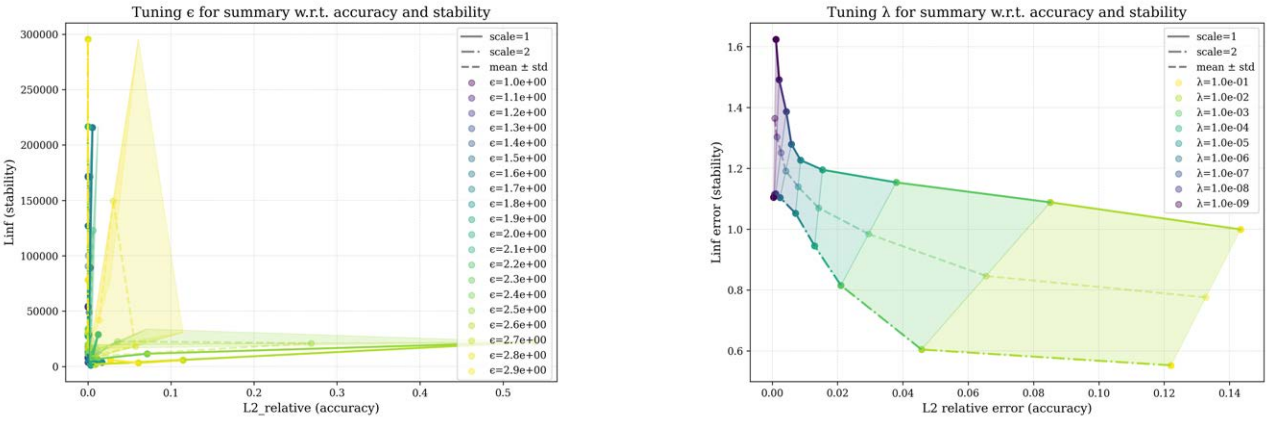
$lr = 10^{-4}$ is used to show the convergence of $\epsilon \in [2, 2.002]$ (Figure 4.1a). The learning rate $lr = 10^{-1}$ is used to show a large range of $\epsilon \in [1, 3]$ and the corresponding stability and accuracy metrics (Figure 4.1b, 4.1c).
The results show that the auto tuning of ϵ correlates with the stability of the solver to a certain extent, but remains large overall without regularization.



(a) $C_{\text{scale}} = 1, lr = 10^{-4}$. (b) $C_{\text{scale}} = 1, lr = 0.1$. (c) $C_{\text{scale}} = 2^2, lr = 0.1$.

Figure 4.1: Training history of Kansa method with auto tuning of shape parameter ϵ.

To investigate the Kansa method with (w/) and without (w/o) **Tikhonov regularization** (Tikh.), I take (4.1) as the accuracy metric and (4.3) as the stability metric. The results are shown in Figure 4.2.



(a) w/o Tikh. with $C_{\text{scale}} = 1, 2^2$. (b) w/ Tikh. with $C_{\text{scale}} = 1, 2^2$.

Figure 4.2: Stability and accuracy of Kansa method on 1D advection equation.

The naive Kansa method without Tikh. is unstable despite the auto tuning of the shape parameter ϵ and the adjustment of C_{scale}, since the minimum $\kappa(A) > 380$, which is much larger than that with Tikh. with $\kappa(A) \in [0.5, 1.5]$. This has shown the significant improvement of stability by Tikhonov regularization, which is consistent with the theoretical analysis in (3.13).

§ 4.5 Extension II: coupled PDEs

I set up *Lotka-Volterra* equations (2.13) according to Table 4.5, for which $N_{\mathcal{R}} = 100 \times 1$ collocation points are sampled over the temporal domain, and $N_d = 1$ for each of the two initial conditions. For *testing*, I sample $N_{\text{test}} = 64$ points and present the Kansa results in Table 4.7.

Table 4.5: Benchmark configuration for the Lotka-Volterra equations.

Time span	ICs	Constants
$t_0 = 0, t_f = 200$	$x(0) = 40, y(0) = 9$	$\alpha = 0.1, \beta = 0.02, \delta = 0.01, \gamma = 0.1$

For 1D *Maxwell's* equations (2.24), I set it up as in Table 4.6, for which $c = 1$ denotes the propagation speed, and sample $N_{\mathcal{R}} = 12 \times 12$ collocation points over the domain, $N_d = 24$ for the initial condition. For *testing*, $N_{\text{test}} = 10 \times 10$ points are sampled.

Within the Kansa framework, I evaluate three different types of kernel basis functions. (i) For Gaussian, I set coefficient ϵ (2.64) as $\epsilon_E = 16$ and $\epsilon_B = 16$ for Maxwell's equations. For Lotka-Volterra equations, $\epsilon_x = 0.21$ and $\epsilon_y = 0.2$ when $C_{\text{scale}} = 1$; $\epsilon_x = 0.3$ and $\epsilon_y = 0.3$ when $C_{\text{scale}} = 2^2$, respectively. (ii) For Sinusoidal (Sin.), the shape parameter is set as $\epsilon_x = 0.001$ and $\epsilon_y = 0.001$ for Lotka-Volterra equations, and $\epsilon_E = 1$ and $\epsilon_B = 1$ when $C_{\text{scale}} = 1$, and $\epsilon_E = 1.2$ and $\epsilon_B = 1.1$ when $C_{\text{scale}} = 2^2$ for Maxwell's equations, respectively. (iii) For ReLU (§ 3.4), the bias term is scaled by 6×10^4 for Lotka-Volterra equations and 2 for Maxwell's equations.

Table 4.6: Benchmark configuration for the 1D Maxwell's equations.

Time span	ICs	Region	Constant
$t_0 = 0,$ $t_f = \frac{1}{2}$	$E_z(x,0) = \sum_{k=1}^{2} \frac{1}{k} \sin(2\pi k x)$ $B_y(x,0) = \sum_{k=1}^{2} \frac{1}{k} \cos(2\pi k x)$	$x_0 = 0,$ $x_f = 1$	$c = 1$

§ 4.5.1 Forward solution

Correctness. I present the Kansa framework performance with three kernel functions in Table 4.7. These two equations yield smaller errors as C_{scale}, introduced above, increases.

Computational efficiency. For training and inference time, solving Lotka-Volterra takes 0.4034 and 0.0001 seconds, whereas for Maxwell's, it takes 0.4486 and 0.0005 seconds.

Table 4.7: $\hat{\mathcal{R}}_{\text{relative } L_2}$ error (↓) of the Kansa framework applied to Lotka-Volterra and Maxwell's equations, with respect to the type of basis functions and computational grid resolution.

Basis	C_{scale}	$x(t)$	$y(t)$	$\frac{x+y}{2}(t)$	$E_z(x,t)$	$B_y(x,t)$	$\frac{E_z+B_y}{2}(x,t)$
Gauss	1	0.1279	0.05667	0.09285	0.8049	0.5895	0.6972
Sin.	1	0.08222	0.02728	0.05475	**0.4421**	0.3589	0.4005
ReLU	1	**0.08108**	**0.02669**	**0.05388**	0.4603	**0.3030**	**0.3816**
Gauss	2^2	0.03577	**0.01744**	0.02660	0.4384	0.3831	0.4107
Sin.	2^2	0.01772	0.03408	0.02590	**0.3587**	**0.3933**	**0.3760**
ReLU	2^2	**0.01084**	0.03597	**0.02340**	0.7978	0.4035	0.6006

§ 4.5.2 Inverse estimation

Building on the Lotka-Volterra equations (2.13) and configuration in Table 4.5, I aim to estimate α, β, δ and γ when given observations $x^{\text{obs}}(t)$ and $y^{\text{obs}}(t)$ across the temporal domain.

I present the results with all parameters initialized at 1 in Table 4.8. Although the search space spans four dimensions, the adopted optimizer `SciPy` Powell performs well, yielding accurate

Table 4.8: Estimation of α, β, δ and γ in the Lotka-Volterra solved by the proposed method.

	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

estimates of the parameters with moderate computational resources.

§ 4.6 Extension III: nonlinear PDEs

As formulated in (2.29) and set up by Table 4.9, I evaluate the proposed method against baseline solvers on the 1D *Burgers'* equation. I sample PDE ν within a set $\mathcal{V} = \{0.2, 0.5, 0.8\}$ for forward computation, $\mathcal{V} = \{0.5\}$ being the target for inverse estimation, $N_{\mathcal{R}} = 64 \times 16$ collocation points over the spatial-temporal domain, $N_{\mathcal{B}} = 16 \times 2$ for BCs, and $N_d = 64$ for ICs. For *testing*, $N_{\text{test}} = 48 \times 12$ are sampled, which is disjoint from the training data for fair evaluation. I quantify the *error* using relative L_2 risk (4.1) $\hat{\mathcal{R}}_{\text{relative } L_2}$. For the Kansa Method, I set the kernel coefficient ϵ (2.64), i.e. the shape of Gaussian RBF, as 0.8, after tuning for optimal performance. I train the PINN, FNO and GNN models until they converge, using a

Table 4.9: Benchmark configuration for the 1D Burgers' equation.

Time span	ICs	Region	BCs	Constant
$t_0 = 0, t_f = 4$	per (2.40)	$x_0 = -10, x_f = 10$	$u(x_0) = 1, u(x_f) = 0$	$\nu \in \mathcal{V}$

learning rate $\eta = 10^{-3}$, and the training dataset FNO is synthesized from randomly assigning PDE parameter $\nu \sim \mathcal{N}(0.1, 1)$. The resulting number of epochs for PINN is $N_{\text{iter}} = 3000$ iterations, while GNN and FNO run for $N_{\text{iter}} = 100$ iterations.

§ 4.6.1 Forward solution

Correctness. Table 4.10 shows that fully nonlinear approach achieves superior performance compared to all time discretization schemes. Theoretically, no clear conclusion can be drawn if IMEX excels backward Euler or vice versa. Nevertheless, Crank-Nicolson (C-N), by theory, is expected to outperform the above two schemes, given a higher order of accuracy in time.

Table 4.10: Models $\hat{\mathcal{R}}_{\text{relative } L_2} \times 10^{-2}$ error (↓) on Burgers' equation. Fully nonlinear approach outperforms other time discretization schemes, e.g. Forward, Backward Euler, IMEX and C-N.

ν	PINN	FNO	MP-GNN	Forward	IMEX	Backward	C-N	Fully nonlinear
0.2	1.47	7.92	0.69	296	3.23	2.10	1.96	**0.0337**
0.5	0.79	1.81	0.35	4.96×10^{15}	1.66	1.31	1.26	**0.00717**
0.8	0.59	5.11	0.95	1.02×10^{56}	1.26	1.05	1.05	**0.136**
avg.	0.95	4.95	0.66	3.40×10^{55}	2.05	1.49	1.42	**0.059**
std.	0.377	2.497	0.246	4.81×10^{55}	0.850	0.447	0.389	0.056

Computational efficiency. The training and inference time across Kansa framework and baselines are shown in Table 4.11. I adopt the `SciPy` least-squares as the nonlinear solver. For *training* stages, the fully nonlinear approach incurs a higher computation cost, due to more

Table 4.11: Train or infer cost (seconds) for the Burgers' equation across all solvers, where Fw. and Bw. stand for Forward and Backward Euler, respectively. Time stepping schemes have lower computational complexity, thus less training time but more inference time than fully nonlinear approach. The latter reuses the coefficients from training phase for inference.

	ν	PINN	FNO	MP-GNN	Fw.	IMEX	Bw.	C-N	Fully nonlinear
Train	0.2	6.63	2.81	2.35	0.29	**0.28**	0.41	0.41	49.5
	0.5	6.54	2.01	2.29	0.36	**0.28**	0.38	0.46	51.5
	0.8	6.30	2.40	2.33	**0.29**	**0.29**	0.37	0.42	52.2
	avg.	6.49	2.41	2.32	0.31	**0.28**	0.39	0.43	51.1
	std.	0.14	0.33	0.025	0.33	0.005	0.017	0.022	1.14
Infer	0.2	0.010	0.76	0.005	0.46	0.47	0.52	0.52	**0.005**
	0.5	0.015	0.69	0.005	0.47	0.46	0.50	0.50	**0.005**
	0.8	0.010	0.92	0.005	0.49	0.46	0.51	0.52	**0.005**
	avg.	0.012	0.79	0.005	0.47	0.46	0.51	0.51	**0.005**
	std.	0.002	0.096	0	0.012	0.005	0.008	0.009	0

expensive memory consumption (Table 3.1) per time step, additionally amplified by nonlinear operators. In contrast, all the time discretization schemes exhibit significantly lower training time. For *inference* stages, the fully nonlinear solver is considerably efficient, because of the coefficients reused from the training stage. Although the time stepping schemes require a complete recomputation, their inference cost remains within the acceptable bound.

Stability. The forward Euler scheme remains unstable (Table 4.12), if the stability condition is violated. By the CFL condition, its time step Δt must not exceed the upper bound. In contrast, all the rest of the time stepping schemes are unconditionally stable (Table 3.1).

Table 4.12: Investigation of stability on the Burgers' equation for the forward Euler scheme.

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**

§ 4.6.2 Inverse estimation

In continuation of the Burgers' equation (2.29) and configuration in Table 4.9, I explore the inverse estimation of the PDE parameter ν when given the observation data $u^{\text{obs}}(x,t)$ across the spatial-temporal domain and present the results in Table 4.13, with the guess of ν at 0.1.

Table 4.13: Estimation of ν in the Burgers' equation solved by the proposed methods, for which the reference value is $\nu = 0.5$.

Forward	IMEX	Backward	C-N	Fully nonlinear
0.388	0.535	0.467	0.502	0.500

Correctness. For a fixed starting value and optimizer, the Crank-Nicolson scheme validates its theoretical superiority (Table 3.1) against both backward Euler and IMEX. The latter two stuck at local minima. **Computational efficiency.** The fully nonlinear formulation necessitates

training from scratch when estimating new PDE parameters, incurring extra computational costs (Table 4.11). I limit the training to a maximum number of iterations, for lightweight computation. **Stability.** The forward Euler scheme remains unstable for a large value of Δt.

§ 4.7 Applications of proposed methods

Here, I present the qualitative visualization of the applications proposed in § 3.6, including the Gray-Scott pattern formation (§ 3.6.3) in Figure 4.3, and three physics simulations (§ 3.6.4) in Figure 4.4, 4.5, 4.6, 4.7 and 4.8. They reveal the potential of utilizing the proposed method in wave, heat, temperature, particle-based simulations, and beyond.

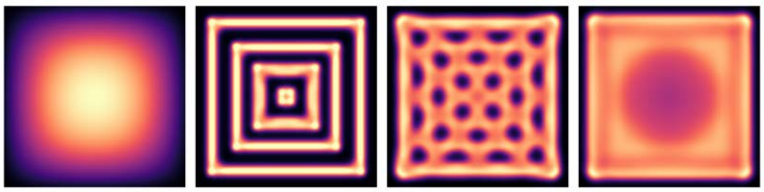


Figure 4.3: Gray-Scott patterns [Tur52, WK91], drawn concurrently using the proposed Kansa framework of truncated kernels (2.65) and sparse matrix solver (§ 3.6.2).

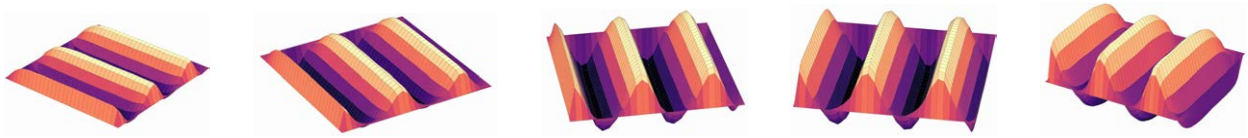


Figure 4.4: Simulation results of a mass-spring grid, where Burgers' equation is the acceleration, solved by the proposed method and without interaction of material or light.

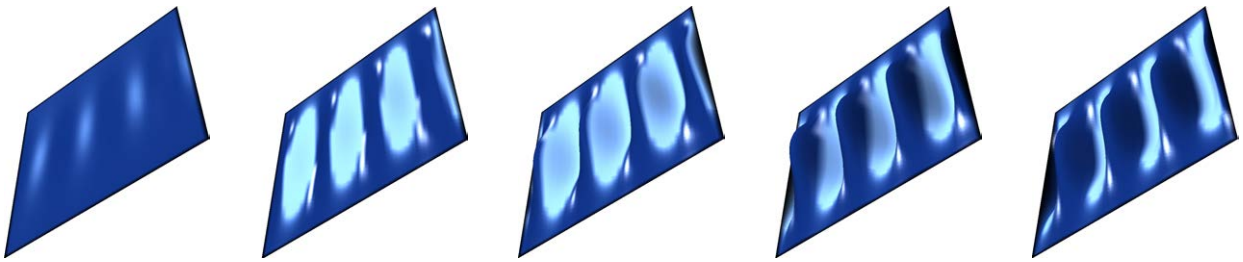


Figure 4.5: Rendering of a mass-spring grid, where Burgers' equation is the acceleration, computed by the proposed method and with interaction of material and light.

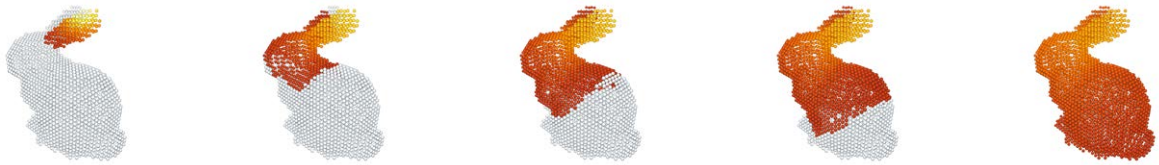


Figure 4.6: Time-varying visualization of the heat equation on a bunny manifolds solved by the proposed method. The yellow-to-orange color mapping represents high to low temperature.



Figure 4.7: Particle-based simulation with a diffusion-reaction temperature field, governing the motion by buoyancy, stochastic turbulence and damping forces, computed by the proposed method. The yellow-to-orange color mapping represents from high to low temperature.



Figure 4.8: The same particle-based model as above, but with a wider initial particle density distribution along the vertical axis. These figures illustrate the frames after 2306 simulation steps, showcasing the effect of the asynchronous respawn logic (§ 3.6.4).



Figure 4.9: The same particle-based model as above, but with the widest initial particle density distribution along the vertical axis. These figures illustrate every five frames after 14475 simulation steps, showcasing the effect of the asynchronous respawn logic (§ 3.6.4).

5 Conclusions

Richard Hamming — *"The purpose of computing is insight, not numbers."*

§ 5.1 Summary and contributions

Solver extensions. I present a unified formulation to solve *coupled PDEs* and *nonlinear* PDEs, beyond [ZFH+23] scope of linear ones. This involves addressing the loss property of linear reordering, trade-off analysis of four different time stepping schemes reducing to nonlinearity or suitable nonlinear solvers, and the development of generalized auto tuning methods to find optimal hyperparameters, filling a long-lasting gap or challenge in the PDE community.

Solver stability. I propose both theoretical stability derivations for linear cases and empirical evaluations for nonlinear cases. To mitigate the solver sensitivity to input perturbations, I introduce regularization strategies informed by stability analysis. After conducting ablation studies, I demonstrate its effectiveness in improving solver *stability*. Furthermore, I explore the trade-off between *accuracy* and *stability*, with automated methods to balance dual objectives.

Evaluation against baselines. I implement four different types of established classical and neural PDE solvers, i.e. FDM, PINN, FNO, GNN-based. I systematically examine CNF solvers in relation to aforementioned baselines over multiple critical dimensions, including (i) *effectiveness* quantified via error metrics, (ii) *efficiency* characterized by computational cost or convergence rate, (iii) *stability*, and finally (iv) *utility* in forward and inverse scientific simulation applications. Such benchmarking and evaluation elucidate the pros and cons between CNF-based approaches and a wider context of PDE solvers, pinpointing the future directions.

Unified framework and diverse applications. Overall, I expand the application range of self-tuning (§ 3.3.5) mesh-free solvers across *forward* simulation modeling and *inverse* problems alike. The proposed *unified* framework supports at least three categories of kernel functions, including RBFs of various types (e.g. Gaussian, Sinusoidal, and Multiquadrics), truncated kernels or neural network-based ones. I demonstrate that learning guided Kansa framework constitutes a versatile yet effective solver for multi-dimensional coupled PDEs and nonlinear PDEs, when evaluated against classical and neural baseline solvers, and on a wide of application domains, covering Computer Graphics, Visual Computing, and scientific simulations.

§ 5.2 Lessons learned and reflections

Research process. This project witnessed a complete independent research cycle, from the beginning of identifying the research gap and formulating questions, surveying the literature, designing the framework, to the end of implementation, evaluation, writeup, being reviewed, reviewing others, and accepted for publication at International Conference on Learning Representations (ICLR) AI-PDE workshop 2026. Significant iterative research and development

happened both before and after the acceptance, where suggestions from reviewers and supervisors have been taken seriously and reflected on the final version of the project. For example, the stability and applications sections are introduced with thanks to the reviewers' comments for lack of related explorations. Every step of the process has been a valuable learning experience for me and future ventures, which I will continue to build upon and improve.

Engineering techniques and testing. The development of both the proposed *unified* framework and baseline methods required significant engineering efforts, especially for aligning the PDE formulations, setup, and evaluation metrics across different solvers. I design UML diagrams before actual implementation to clarify the structure and interactions of different components. For testing, I adopt *unit tests* for every PDE classes, *integration tests* to verify the end-to-end correctness of the solvers against reference solutions, and *regression tests* after every major code change, e.g. introducing abstract features or classes for encapsulation after proof-of-concept version, to assert no unintended consequences has introduced.

Personal growth. Despite almost new to the field of PDEs, I have been able to adapt quickly and learn to connect between different domains, e.g. Mathematical Analysis, Machine Learning, and Visual Computing. This interdisciplinary nature of the project has been challenging but rewarding. It requires a broad understanding of concepts and techniques, as well as the ability to synthesize them into a coherent framework. Throughout this journey, I have learnt the value of perseverance, curiosity, openness to unknowns, ability to discuss with a range of colleagues from different backgrounds. Upon completion, I feel confident for challenging myself next time.

§ 5.3 Future work

Theoretical proof. Despite the incorporation of learning theory and stability analysis, the current work primarily focuses on empirical evaluations, especially for the nonlinear cases. This leaves open avenues for further theoretical proof of *error bounds*, *convergence*, and *stability* properties of the proposed framework. For different classes of PDEs and underlying kernel functions, it would be invaluable to establish rigorous guarantees on the performance and stability of the solvers, preferably with interdisciplinary collaborations.

Integration with related pipelines. With a *unified* framework for handling a wide range of PDEs and the ability to solve inverse problems, a natural next step is to integrate the proposed solvers with differentiable rendering pipelines in Computer Graphics [SZRea23]. This brings the possibility of reconstructing physical, material, and geometric properties from visual data, which allows further editing, generation, and simulation of complex phenomena or requirements. Other hybrid pipelines to be constructed with physics and data-driven likewise, e.g. natural language for controllability, action manipulation for robotics, and many more.

Interdisciplinary research. The proposed framework has potential applications across numerous scientific domains, much beyond the already diverse set of applications explored in this work. With additional diverse requirements integrated, the proposed framework admits further verification and enhancement to address domain-specific challenges.

Bibliography

[AF03] Robert A. Adams and John J. F. Fournier. *Sobolev spaces*, volume 140 of *Pure and Applied Mathematics*. Academic Press, Boston, MA, 2 edition, 2003. Originally published in 1975. 59

[AMMIR14] Muhammad Abbas, Ahmad Abd Majid, Ahmad Izani Md. Ismail, and Abdur Rashid. Numerical method using cubic B-spline for a strongly coupled reaction-diffusion system. *PLoS One*, 9(1):e83265, 2014. 24

[AUC09] 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. URL: https://www.sciencedirect.com/science/article/pii/S0965997808001531, doi:10.1016/j.advengsoft.2008.08.001. 24

[Bac11] Nicolas Bacaër. Lotka, Volterra and the predator–prey system (1920–1926). In *A Short History of Mathematical Population Dynamics*, pages 71–76. Springer, London, 2011. doi:10.1007/978-0-85729-115-8_13. 17

[BAT15] HARRY BATEMAN. Some recent researches on the motion of fluids. *Monthly Weather Review*, 43(4):163 – 170, 1915. URL: https://journals.ametsoc.org/view/journals/mwre/43/4/1520-0493_1915_43_163_srrotm_2_0_co_2.xml, doi:10.1175/1520-0493(1915)43<163:SRROTM>2.0.CO;2. 20

[Bel57] Richard E. Bellman. *Dynamic programming*. Princeton University Press, Princeton, NJ, 1957. Prepared for the Rand Corporation. 60

[BFF+09] Aydin Buluç, Jeremy T. Fineman, Matteo Frigo, John R. Gilbert, and Charles E. Leiserson. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In *Proceedings of the Twenty-First Annual Symposium on Parallelism in Algorithms and Architectures*, SPAA '09, page 233–244, New York, NY, USA, 2009. Association for Computing Machinery. doi:10.1145/1583991.1584053. 37

[BPRS18] 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. 22

[BS18] 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. 11, 13, 21

[BWW22] Johannes Brandstetter, Daniel Worrall, and Max Welling. Message passing neural pde solvers. *arXiv preprint arXiv:2202.03376*, 2022. 11, 13, 30

[CFC14] Wen Chen, Zhuo-Jia Fu, and C. S. Chen. *Different Formulations of the Kansa Method: Domain Discretization*, pages 29–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. doi:10.1007/978-3-642-39572-7_3. 13

[CFL28] 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. 22

[CS09] Youngmin Cho and Lawrence Saul. Kernel methods for deep learning. *Advances in neural information processing systems*, 22, 2009. 35

[DBC+17] Abe Davis*, Katherine L. Bouman*, Justin G. Chen, Michael Rubinstein, Oral Büyüköztürk, Frédo Durand, and William T. Freeman. Visual vibrometry: Estimating material properties from small motions in video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):732–745, 2017. doi:10.1109/TPAMI.2016.2622271. 13

[Eva10] L.C. Evans. *Partial Differential Equations*. Graduate studies in mathematics. American Mathematical Society, 2010. URL: http://dx.doi.org/10.1090/gsm/019. 20

[FA18] Abien Fred and M Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, pages 1–6, 2018. 35

[Fau16] A. C. Faul. *A Concise Introduction to Numerical Analysis*. CRC Press, 2016. URL: https://lccn.loc.gov/2015041395. 15

[GBD92] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58, 1992. doi:10.1162/neco.1992.4.1.1. 32

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Softwares*. Addison-Wesley Professional, 1994. 29

[GLI16] Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 481–490, 2016. 13, 24

[Gre98] Walter Greiner. *Maxwell's equations*, pages 250–275. Springer New York, New York, NY, 1998. doi:10.1007/978-1-4612-0587-6_13. 17

[GSR+17] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1263–1272. JMLR.org, 2017. 13, 26

[GVL13] Gene H. Golub and Charles F. Van Loan. *Matrix Computations - 4th Edition*. Johns Hopkins University Press, Philadelphia, PA, 2013. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781421407944, arXiv:https://epubs.siam.org/doi/pdf/10.1137/1.9781421407944, doi:10.1137/1.9781421407944. 36

[HC +26] Zheyuan Hu, Weitao Chen, Cengiz Öztireli, Chenliang Zhou, and Fangcheng Zhong. Learning-guided kansa collocation for forward and inverse PDEs beyond linearity. In *AI&PDE: ICLR 2026 Workshop on AI and Partial Differential Equations*, 2026. Poster presentation. URL: https://openreview.net/forum?id=ZbTGbUzejl, arXiv:2602.07970. 28

[HK70] Arthur E. Hoerl and Robert W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970. doi:10.1080/00401706.1970.10488634. 32

[HLA+19] Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)*, 38(6):201, 2019. 38

[Hop50] 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. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpa.3160030302, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpa.3160030302, doi:10.1002/cpa.3160030302. 19

[HSW89] 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. 59

[Hu26] Zheyuan Hu. Neural representation of Bidirectional Reflectance Distribution Function. Technical Report UCAM-CL-TR-1005, University of Cambridge, Computer Laboratory, March 2026. URL: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-1005.pdf, doi:10.48456/tr-1005. 13

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 35

[Ise08] Arieh Iserles. *A first course in the Numerical Analysis of Differential Equations*. Cambridge University Press, Cambridge, 2 edition, 2008. URL: http://www.cambridge.org/9780521734905. 11, 13, 14, 21, 61

[Kah16] W Kahan. Computing cross-products and rotations, 2016. 36

[Kan90] 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. 11, 13, 26, 28

[KLY21] YUEHAW KHOO, JIANFENG LU, and LEXING YING. Solving parametric PDE problems with artificial neural networks. *European Journal of Applied Mathematics*, 32(3):421–435, 2021. doi:10.1017/S0956792520000182. 13, 24

[Kut22] Gitta Kutyniok. The Mathematics of Artificial Intelligence, 2022. URL: https://arxiv.org/abs/2203.08890, arXiv:2203.08890. 60

[KW17] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL: https://openreview.net/forum?id=SJU4ayYgl. 25

[LJP+21] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature machine intelligence*, 3(3):218–229, 2021. 11, 13, 24

[LKA+20] 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. Poster presentation. URL: https://arxiv.org/abs/2003.03485, arXiv:2003.03485. 11, 23

[LKA+21] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2021. URL: https://arxiv.org/abs/2010.08895, arXiv:2010.08895. 11, 13, 24, 25

[lRD47] Jean le Rond D'Alembert. Recherches sur la courbe que forme une corde tenduëe mise en vibration. *Histoire de l'académie royale des sciences et belles lettres de Berlin*, 3:214–219, 1747. 19

[OS21] 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, editors, *Nonlinearities in Economics: An Interdisciplinary Approach to Economic Dynamics, Growth and Cycles*, pages 191–208. Springer International Publishing, Cham, 2021. doi:10.1007/978-3-030-64234-0_10. 17

[P+95] Xavier Provot et al. Deformation constraints in a mass-spring model to describe rigid cloth behaviour. In *Graphics interface*, pages 147–147. Canadian Information Processing Society, 1995. 38

[Pat80] S. V. Patankar. *Numerical heat transfer and fluid flow*. Taylor & Francis, 1980. 21

[PGB03] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. In *ACM SIGGRAPH 2003 Papers*, pages 313–318, New York, NY, USA, 2003. Association for Computing Machinery. doi:10.1145/1201775.882269. 24

[PMZ+23] Apostolos F. Psaros, Xuhui Meng, Zongren Zou, Ling Guo, and George Em Karniadakis. Uncertainty quantification in scientific machine learning: Methods, metrics, and comparisons. *Journal of Computational Physics*, 477:111902, 2023. URL: https://www.sciencedirect.com/science/article/pii/S0021999122009652, doi:10.1016/j.jcp.2022.111902. 13

[RBPK17] Samuel H. Rudy, Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 3(4):e1602614, 2017. URL: https://www.science.org/doi/abs/10.1126/sciadv.1602614, arXiv:https://www.science.org/doi/pdf/10.1126/sciadv.1602614, doi:10.1126/sciadv.1602614. 13

[RCLV11] Juri Ranieri, Amina Chebira, Yue M. Lu, and Martin Vetterli. Sampling and reconstructing diffusion fields with localized sources. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4016–4019, 2011. doi:10.1109/ICASSP.2011.5947233. 13

[RM22] 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. doi:10.1007/s10444-022-09985-9. 61

[RPK19] 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. URL: https://www.sciencedirect.com/science/article/pii/S0021999118307125, doi:10.1016/j.jcp.2018.10.045. 11, 13, 22

[Rud76] Walter Rudin. *Principles of Mathematical Analysis*. McGraw-Hill, New York, 3 edition, 1976. 59

[SGT+09] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi:10.1109/TNN.2008.2005605. 25

[SS97] A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997. doi:10.1109/72.572108. 25

[SZRea23] 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. 13, 50

[TPL+22] 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. 12, 16, 19

[Tur52] Alan Mathison Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*, 237(641):37–72, 08 1952. arXiv:https://royalsocietypublishing.org/rstb/article-pdf/237/641/37/1331199/rstb.1952.0012.pdf, doi:10.1098/rstb.1952.0012. 7, 37, 47

[WGK+22] Francis Williams, Zan Gojcic, Sameh Khamis, Denis Zorin, Joan Bruna, Sanja Fidler, and Or Litany. Neural fields as learnable kernels for 3d reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18500–18510, 2022. 36

[WK91] Andrew Witkin and Michael Kass. Reaction-diffusion textures. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '91, page 299–308, New York, NY, USA, 1991. Association for Computing Machinery. doi:10.1145/122718.122750. 7, 37, 47

[WLW22] Hongping Wang, Yi Liu, and Shizhao Wang. Dense velocity reconstruction from particle image velocimetry/particle tracking velocimetry using a physics-informed neural network. *Physics of Fluids*, 34(1):017116, January 2022. doi:10.1063/5.0078143. 13

[WPC+21] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021. doi:10.1109/TNNLS.2020.2978386. 25

[WSL+19] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Trans. Graph.*, 38(5), October 2019. doi:10.1145/3326362. 26

[WW24] Tian Wang and Chuang Wang. Latent neural operator for solving forward and inverse PDE problems. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL: https://openreview.net/forum?id=VLw8ZyKfcm. 13

[WWP21] 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. URL: https://www.sciencedirect.com/science/article/pii/S0045782521002759, doi:10.1016/j.cma.2021.113938. 29

[Yar18] Dmitry Yarotsky. Optimal approximation of continuous functions by very deep ReLU networks. In Sebastien Bubeck, Vianney Perchet, and Philippe Rigollet, editors, *Proceedings of the 31st Annual Conference on Learning Theory*, volume 75 of *Proceedings of Machine Learning Research*, pages 1–11, Stockholm, Sweden, 2018. PMLR. URL: https://proceedings.mlr.press/v75/yarotsky18a.html. 59

[Ypm95] Tjalling J. Ypma. Historical development of the newton–raphson method. *SIAM Review*, 37(4):531–551, 1995. arXiv:https://doi.org/10.1137/1037125, doi:10.1137/1037125. 34

[ZFH+23] 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. 11, 12, 28, 31, 35, 49

[ZHS+26] Chenliang Zhou, Zheyuan Hu, Alejandro Sztrajman, Yancheng Cai, Yaru Liu, and Cengiz Oztireli. M3ashy: Multi-modal material synthesis via hyperdiffusion. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(16):13575–13583, Mar. 2026. URL: https://ojs.aaai.org/index.php/AAAI/article/view/38363, doi:10.1609/aaai.v40i16.38363. 13

[ZZ18] Yinhao Zhu and Nicholas Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415–447, 2018. URL: https://www.sciencedirect.com/science/article/pii/S0021999118302341, doi:10.1016/j.jcp.2018.04.018. 13, 24

A Linear operator examples

As defined in § 2.1.1, I provide several commonly used examples of linear operators below, some of which 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,

$$\mathcal{F}(\mathbf{x}) = A\mathbf{x}. \tag{A.1}$$

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

$$I(f) = \int_a^b f(x)\,dx. \tag{A.2}$$

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

$$D_x(f) = \frac{\partial f}{\partial x}. \tag{A.3}$$

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

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\right). \tag{A.4}$$

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

$$\nabla \cdot \mathbf{F} = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y} + \frac{\partial F_z}{\partial z}. \tag{A.5}$$

- **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,

$$\Delta f = \nabla \cdot (\nabla f) = \nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2}. \tag{A.6}$$

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

$$\nabla \times \mathbf{F} = \left(\frac{\partial F_z}{\partial y} - \frac{\partial F_y}{\partial z}, \frac{\partial F_x}{\partial z} - \frac{\partial F_z}{\partial x}, \frac{\partial F_y}{\partial x} - \frac{\partial F_x}{\partial y}\right) = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \\ F_x & F_y & F_z \end{vmatrix}. \tag{A.7}$$

B Learning theory

This chapter explores why neural networks can approximate PDE solutions from learning theory perspective. With theoretical proofs and analysis, I further explain when and how neural PDE solvers work, including potential failure modes. This is valuable for understanding the limitations and guiding the debugging of such solvers in practice.

§ B.1 Functional analysis addendum

Smoothness [Rud76] 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 $f \in C^k$,

$$C^k(\Omega) = \{f : \Omega \subseteq \mathbb{R}^d \to \mathbb{R} \mid \forall \alpha \leq k.\ \partial^\alpha f \text{ exists and is continuous}\}. \tag{B.1}$$

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 $\phi(a) = \phi(b) = 0$,

$$\begin{aligned} \int_a^b f(x)\phi'(x)\,dx &= [f(x)\phi(x)]_a^b - \int_a^b f'(x)\phi(x)\,dx, \quad \text{by integration by parts,} \\ &= -\int_a^b f'(x)\phi(x)\,dx, \quad \text{as } \phi(a) = \phi(b) = 0. \end{aligned} \tag{B.2}$$

Sobolev spaces [AF03] $W^{k,p}(\Omega)$ is a function space where all functions f have weak derivatives up to order k and every derivative is p-integrable via (2.4),

$$W^{k,p}(\Omega) \subset L^p(\Omega) = \{f : \Omega \subseteq \mathbb{R}^d \to \mathbb{R} \mid \forall \alpha \leq k.\ \exists\ \partial^\alpha f \in L^p(\Omega)\}, \tag{B.3}$$

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

§ B.2 Approximation theory of neural networks

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

Universal approximation theorem [HSW89] 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 $\mathcal{D}$,

$$\exists\ \theta \in \Theta.\ ||\hat{u}_\theta - u||_p < \epsilon \implies ||\mathcal{D}[\hat{u}_\theta] - \mathcal{D}[u]||_p < \epsilon. \tag{B.4}$$

Optimal DNN functions approximation theorem [Yar18]. Assuming a continuous function $u \in W^{s,p}$ as defined in (B.3), 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,

$$||\hat{u}_\theta - u||_p = O(M^{-\frac{s}{d}}), \tag{B.5}$$

where d is the input dimensions 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** [Bel57].

§ B.3 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,

$$\mathcal{R}(\hat{u}_\theta) = (||\hat{u}_\theta - u||_p)^p := \int_\Omega |\hat{u}_\theta(x) - u(x)|^p \, dx, \tag{B.6}$$

For discrete 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,

$$\underbrace{\int_\Omega |\hat{u}_\theta(x) - u(x)|^p \, dx}_{\text{integral } \mathcal{R}(\hat{u}_\theta)} = \underbrace{\sum_{k=1}^{N} \omega_k |\hat{u}_\theta(x_k) - u(x_k)|^p}_{\text{discretized } \hat{\mathcal{R}}(\hat{u}_\theta)} + \mathcal{E}_T(\theta), \quad \mathcal{E}_T(\theta) := \mathcal{R}(\hat{u}_\theta) - \hat{\mathcal{R}}(\hat{u}_\theta), \tag{B.7}$$

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 [Kut22]. When approximating a continuous function u with a neural network $\hat{u}_\theta$, the *total error* $\mathcal{E}(\theta)$[1] is decomposed into three parts, with the risk $\mathcal{R}$ and its quadrature $\hat{\mathcal{R}}$ defined in (B.6) and (B.7) respectively,

$$\mathcal{E}(\theta) := \mathcal{R}(\hat{u}_\theta) \leq \underbrace{\inf_{\theta^\star \in \Theta} \mathcal{R}(u_{\theta^\star})}_{\mathcal{E}_A(\theta)} + \underbrace{\hat{\mathcal{R}}(\hat{u}_\theta) - \inf_{\theta^\star \in \Theta} \mathcal{R}(u_{\theta^\star})}_{\mathcal{E}_O(\theta)} + \underbrace{\mathcal{R}(\hat{u}_\theta) - \hat{\mathcal{R}}(\hat{u}_\theta)}_{\mathcal{E}_T(\theta)}, \tag{B.8}$$

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 (B.7).

§ B.4 PINN learning theory

I briefly analyze the PINN error bound[2]. The total error between the optimal solution $\hat{u}_\theta$ and the ground truth u is shown in (2.50). However, during training, the network doesn't have access to the exact ground truth for u. Therefore, I aim to reduce the PDE residual instead.

$$\begin{aligned} \mathcal{E}_\mathcal{R}(\theta) = (||\mathcal{R}_\theta||_p)^p &= (||\mathcal{D}[\hat{u}_\theta] - f||_p)^p, \quad \text{by (2.47).} \\ &= ||\mathcal{D}[\hat{u}_\theta] - \mathcal{D}[u]||_p, \quad \text{by (2.9).} \\ &= ||\hat{f} - f||_p, \quad \text{by the definition of } \hat{f}, \end{aligned} \tag{B.9}$$

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 (2.48).

From the **theoretical** perspective, the goal is to derive that the total error $\mathcal{E}_{\text{PINN}}$ (2.50) is sufficiently small. To prove this, a sufficient condition is that the total error is bounded by the residual error $\mathcal{E}_\mathcal{R}$ (B.9), i.e. I can prove that the smallest residual error ensures the smallest total error.

$$\forall \theta \in \Theta. \; \mathcal{E}_{\text{PINN}}(\theta) \leq C \mathcal{E}_\mathcal{R}(\theta), \tag{B.10}$$

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

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

where C is a constant. By expansion of $\mathcal{E}_{\text{PINN}}$ (2.50) and $\mathcal{E}_{\mathcal{R}}$ (B.9), the abovementioned inequality (B.10) is equivalent to the following **coercivity** condition [RM22],

$$\forall \theta \in \Theta.\ ||\hat{u}_\theta - u|| \leq C||\hat{f} - f||_p, \tag{B.11}$$

By quadrature bound [Ise08], the smallest practical training error $\mathcal{E}_T$ (2.48) ensures the smallest residual error $\mathcal{E}_R$ (B.9), where C' is a constant,

$$\forall \theta \in \Theta.\ \mathcal{E}_{\mathcal{R}}(\theta) \leq C'[\mathcal{E}_T(\theta) + \mathcal{E}_u(N_{\mathcal{R}})], \tag{B.12}$$

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 $N_{\mathcal{R}}$,

$$\begin{aligned} \mathcal{E}_u(N_{\mathcal{R}}) \sim o(\frac{1}{N_{\mathcal{R}}}) &\implies \frac{\mathcal{E}_u(N_{\mathcal{R}})}{\frac{1}{N_{\mathcal{R}}}} = 0, n \to \infty, && \text{by the definition of little-o notation.} \\ &\implies \lim_{N_{\mathcal{R}} \to \infty} N_{\mathcal{R}} \mathcal{E}_u(N_{\mathcal{R}}) = 0, && \text{by the definition of limit.} \end{aligned} \tag{B.13}$$

By the above two inequalities (B.10) and (B.12), the total error $\mathcal{E}_{\text{PINN}}$ (2.50) converges as the training error $\mathcal{E}_T$ (2.48) converges,

$$\forall \theta \in \Theta.\ \mathcal{E}_{\text{PINN}}(\theta) \leq CC'[\mathcal{E}_T(\theta) + o(\frac{1}{N_{\mathcal{R}}})]. \tag{B.14}$$

By Universal approximation theorem (B.4), 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 $\hat{u}_\theta$ [Ise08],

$$\min_{\theta \in \Theta} \mathcal{E}_T(\theta) \leq \mathcal{E}_{\mathcal{R}}(\theta) + o(\frac{1}{N_{\mathcal{R}}}), \tag{B.15}$$

Hence, the training error $\mathcal{E}_T(\theta) < \epsilon + o(\frac{1}{N_{\mathcal{R}}})$ is sufficiently small, according to (B.13). So is the total error $\mathcal{E}_{\text{PINN}}(\theta) < CC'[\epsilon + o(\frac{1}{N_{\mathcal{R}}})]$, by (B.14), 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 (B.7), (2) insufficient training resulting in large optimization error $\mathcal{E}_O$ in (B.8), (3) violation of the coercivity condition (B.11) for PDEs, and (4) large constant C in (B.10) or C' in (B.12).

Acronyms

avg. average. 42, 45, 46

BCs Boundary Conditions. 7, 16, 22, 23, 29, 32, 36, 41, 45

BSDF Bidirectional Scattering Distribution Function. 27

C-N Crank-Nicolson. 9, 34, 45, 46

CNF Constrained Neural Fields. 11, 12, 27, 28, 49

CNN Convolutional Neural Network. 25

COO Coordinate format. 36, 37

Coupled coupled PDEs. 5, 6, 16, 17, 29, 32, 33, 35, 37, 40, 43, 44, 49

CSC Compressed Sparse Column. 36, 37

CSR Compressed Sparse Row. 36, 37

ELU Exponential Linear Unit. 30

FDM Finite Difference Method. 3, 11–13, 21, 27, 33, 40–42, 49

FEM Finite Element Method. 11, 13, 27

FFT Fast Fourier Transform. 30

FNO Fourier Neural Operator. 3, 7, 12, 13, 24, 25, 27, 29, 30, 40–42, 45, 46, 49

GCN Graph Convolutional Network. 25, 26

GNN Graph Neural Network. 3, 7, 11–13, 25–27, 29, 30, 40–42, 45, 46, 49

ICLR International Conference on Learning Representations. 12, 28, 49

ICs Initial Conditions. 7, 16, 23, 29, 32, 36, 41, 44, 45

IMEX Implicit-Explicit. 9, 34, 45, 46

KM Kansa Method. 11, 13, 26, 27, 29, 36, 40–42, 45

MLP Multi-Layer Perceptron. 29

MP message-passing. 7, 25, 26, 30, 42, 45, 46

NNZ Number of Non-Zero entries. 36

NO Neural Operator. 7, 11, 27, 29

ODE Ordinary Differential Equation. 21

PDE Partial Differential Equation. 7, 11–16, 21–23, 26–29, 31, 33–35, 40, 41, 49, 50, 58–61

PINN Physics-Informed Neural Network. 3, 7, 11–13, 22–24, 27, 29, 40–42, 45, 46, 49, 60

PINO Physics-Informed Neural Operator. 24, 27

RBF Radial Basis Function. 26, 28, 31, 35, 41, 45, 49

ReLU Rectified Linear Unit. 35, 44

Sin. Sinusoidal. 26, 44

std. standard deviation. 42, 45, 46

UML Unified Modeling Language. 7, 29, 30, 50