Physica D 460 (2024) 134095

Contents lists available at ScienceDirect

Physica D

journal homepage: www.elsevier.com/locate/physd

A kernel framework for learning differential equations and their solution operators

Da Long [a], Nicole Mrvaljević [b], Shandian Zhe [a], Bamdad Hosseini [b,*]

[a] School of Computing, University of Utah, Salt Lake City, UT, 84112, United States of America
[b] Department of Applied Mathematics, University of Washington, Seattle, WA, 98195, United States of America

ARTICLE INFO

Keywords:
Equation discovery
Operator learning
Reproducing kernel Hilbert spaces
Physics informed machine learning

ABSTRACT

This article presents a three-step kernel framework for regression of the functional form of differential equations (DEs) and learning their solution operators. Given a training set consisting of pairs of noisy DE solutions and source/boundary terms on a mesh: (i) kernel smoothing is utilized to denoise the data and approximate derivatives of the solution; (ii) This information is then used in a kernel regression model to learn the functional form of the DE; (iii) The learned DE is then used within a numerical solver to approximate the solution of the DE with a new source term or initial data, thereby constituting an operator learning framework. Numerical experiments compare the method to state-of-the-art algorithms. In DE learning our framework matches the performance of Sparse Identification of nonlinear Dynamical Systems (SINDy) while in operator learning the method has superior performance compared to well-established neural network methods in low training data regimes.

1. Introduction

Differential equations (DEs) are ubiquitous in natural sciences such as physics [1], social sciences [2], biology [3] and engineering [4,5]. To some extent, DEs are the main subject of interest in the emergent field of Physics Informed Learning (PIL) [6–8] where machine learning (ML) is leveraged for the simulation or inference of physical processes and phenomenon. Traditionally, DEs are designed or discovered by experts based on mathematical and physical intuition, a process that relies on human expertise, data, and mathematical analysis. Once the DE is accepted as a model it is often solved using computer algorithms to simulate a real-world process of interest. Recent advances in ML along with the abundance of data have given rise to the idea of automating this workflow, thereby promising computer programs for learning a DE from limited and noisy data and solving the learned equations to predict the state of a physical system under previously unseen conditions. The goal of this paper is to present an example of such a workflow based on recent advances in the theory of kernel methods. Our proposed method is simple to implement, accurate, and robust to noise as demonstrated by a comprehensive list of numerical benchmarks.

Consider a generic nonlinear DE of the form

$$\mathcal{P}(\mathbf{x}, u) = f(\mathbf{x}), \qquad \mathbf{x} \in \Omega, \quad \text{and} \quad \mathcal{B}(\mathbf{x}, u) = g(\mathbf{x}), \qquad \mathbf{x} \in \partial\Omega,$$

defined over a compact domain $\Omega \subset \mathbb{R}^D$ for $D \geq 1$ with boundary $\partial\Omega$ with solution $u : \Omega \to \mathbb{R}$, forcing/source term $f : \Omega \to \mathbb{R}$, and boundary/initial data $g : \partial\Omega \to \mathbb{R}$. Assume that the boundary/initial data g is known and suppose we have access to limited (and possibly noisy) training data of the form $(u^i(X), f^i(X))_{i=1}^I$ where (u^i, f^i) are solution and source pairs that solve the DE and we used the shorthand notation $u(X) = (u(\mathbf{x}_1), \dots, u(\mathbf{x}_J))$ for a fixed set of observation points $X = \{\mathbf{x}_1, \dots, \mathbf{x}_J\} \subset \Omega$ (see Section 2 for our detailed setup). Given this limited and noisy data we consider two problems: (a) *Learn the DE*, that is, find an approximation to the map $\mathcal{P}$ that describes the nonlinear relationship between x and the pertinent partial derivatives of the solution u.[1]; (b) *Learn the solution operator of the DE*, that is, given a new forcing $\widetilde{f}$ provide an approximation to the corresponding solution $\mathcal{P}^{-1}\widetilde{f}$ using only the training data. Problem (a) is often referred to as *equation learning or discovery* and goes back, at least, to the seminal works [9,10] but it is broadly the subject of interest in the field of inverse problems [11] as well. More recently, it is often tackled by the Sparse Identification of nonlinear Dynamical Systems (SINDy) algorithm of [12] and its subvariants. Problem (b) is reminiscent of standard problems in numerical solution of DEs. However, the modern twist in our setting is that the true DE is assumed to be unknown and we only have access to limited information regarding samples of its solutions

* Corresponding author.
E-mail addresses: u1368737@utah.edu (D. Long), nesbihal@uw.edu (N. Mrvaljević), zhe@cs.utah.edu (S. Zhe), bamdadh@uw.edu (B. Hosseini).

[1] Equivalently, one can also cast the problem of learning the boundary operator $\mathcal{B}$ although we will not consider this setting in this article for brevity and will always assume that the boundary/initial data is known.

https://doi.org/10.1016/j.physd.2024.134095
Received 4 September 2023; Received in revised form 26 December 2023; Accepted 12 February 2024
Available online 15 February 2024

u^i corresponding to some source terms f^i. In this light, problem (b) is often referred to as *the operator learning problem* where one aims to learn/approximate the infinite-dimensional solution operator $\mathcal{P}^{-1}$. In PIL this problem is often cast as regression of an operator between two infinite-dimensional function spaces with the DeepONet algorithm of [13] and the Fourier Neural Operator (FNO) approach of [14] (and their variants) considered as state of the art; see also [15] for a recent competitive method using operator valued kernel regression.

To this end, our main contributions are three-fold:

(1) We present a three-step kernel method for learning (ordinary or partial) DEs and their solution operators from noisy and limited data: Step (i) kernel smoothing is utilized to denoise the training data and compute pertinent partial derivatives of the solution. This allows us to accommodate input data that are provided on unstructured grids; Step (ii) kernel regression is used to learn the functional form of the DE and provide the approximate map $\overline{\mathcal{P}}$; Step (iii) a kernel DE solver is used to numerically approximate $\overline{\mathcal{P}}^{-1} f'$ with the new source term of interest f' in order to approximate the true solution $\mathcal{P}^{-1} f'$. Our proposed framework is not only simple but it inherits the desirable robustness and stability properties of kernel methods and is amenable to kernel learning strategies such as cross validation (CV) or maximum likelihood estimation of hyper-parameters.

(2) Our three-step kernel approach for DE and operator learning is a special instance of an abstract, three-step approach that includes existing methods such as SINDy and PDE-FIND [16] but also extends their applicability in two directions: (a) our kernel approach for gradient estimation in Step (i) enables SINDy to deal with training data that are observed on irregular and inconsistent grids; (b) SINDy can be used in place of our kernel approach to Step (ii) in order to extend its applicability to operator learning tasks.

(3) We present a set of numerical experiments and benchmarks that demonstrate the superior performance of our three-step approach to operator learning in the low data regime where only a small data set of solution and forcing pairs are available. We intuit that this superior performance is due to the fact that our approach makes explicit use of the fact that the operator of interest is the solution map of a DE rather than a generic map between two function spaces as is often considered in the operator learning literature [13,14,17,18]

1.1. Review of the relevant literature

Below we present a review of the relevant literature to our work focusing on discovering/learning of DEs, operator learning, and variational DE solvers that are used in Step (iii) of our framework.

Discovering/learning DEs: Identifying the parameters of a DE is a well-known inverse problem; see the works of [19,20] on parameter identification of ordinary differential equations (ODEs) as well as the book of [11] and the article of [21] for examples involving PDEs. Such problems are also encountered in optimal control of PDEs [22]. However, these classic problems were considered under the assumption that the expression of the DE is known up to free parameters that need to be identified from experimental data.

Equation discovery/learning is a more recent problem attributed to [9,10] who used symbolic regression to discover underlying physical laws from experimental data. Compared with the aforementioned inverse problems, the goal here is to discover the functional form of the DE, that is the nonlinear relationship between the partial derivatives of the solution as well as possibly free parameters, from experimental data. DEs that describe real world physical systems involve only a few terms and often have simple expressions. Based on this philosophy, recent approaches to equation learning try to learn a DE from a dictionary of possible terms/features along with a sparsity assumption to ensure only a few terms will be active. Perhaps the best known example of such an approach is the SINDy algorithm of [12,16] and its many extensions and variants; see [23] and references therein. Other authors have also considered similar approaches [24,25] based on the idea of imposing sparsity structures on the terms involved in the learned DE. In this light, the main difference between the aforementioned methods is in the way they impose the requisite sparsity assumption and how they solve the resulting optimization problems.

Compared to the feature map perspective of SINDy-type methods, our approach employs a kernel perspective towards learning the DE. As a result, we give up the immediate interpretability of the learned equation in favor of richer and more flexible features that can be tuned using CV and a more convenient computational framework that is also able to deal with variable coefficient DEs; feature based methods often cannot deal with variable coefficients without strong prior knowledge injected in the their dictionaries. Our method can also be combined with the kernel mode decomposition approach of [26] to extract the dominant features of the learned DE, thereby making our approach more interpretable via a post-processing step although we do not pursue this direction here. Finally, due to its simple mathematical formulation, our method is amenable to mathematical analysis and opens the door for analyzing the accuracy and robustness of the estimator $\overline{\mathcal{P}}$ from the perspective of kernel methods and optimal recovery, providing a new perspective for the theoretical analysis of equation learning. Such theoretical questions have attracted attention very recently [27,28] although many open questions remain. Another closely related approach to our framework is the PDE-Net of [29,30] which, put simply, parameterizes $\overline{\mathcal{P}}$ via a convolutional neural network. In Section 4 we comment on how our abstract framework can be extended to include PDE-Net as well.

Finally we note that a similar kernel based approach to our method has been developed in the series of papers [31–36] aimed at the discovery, data assimilation, and extrapolation of dynamical systems. The main difference between those works and ours is that our method is aimed at wider families of DEs and in particular PDEs and distinguishes between the equation learning step and operator learning.

Operator Learning: Approximation or learning of the solution maps of DEs is a vast area of research in applied mathematics and engineering. In the setting of stochastic and parametric PDEs, the goal is often to approximate the solution of a PDE as a function of a random or uncertain parameter. The well-established approach to such problems is to choose or find appropriate bases for the input parameter and the solution of the PDE and then construct a parametric, high-dimensional map, that transforms the input basis coefficients to the output coefficients. Well-established methods such as polynomial chaos, stochastic finite element methods, reduced basis methods, and reduced order models [37–41] fall within this category. A vast literature exists on this subject and the theoretical analysis of these methods has been extensively researched; see for example [42–47]. More recent neural net based methods such as DeepONets [13], and FNO [14,17,48] also fall within the aforementioned category of methods where the main novelty appears to be the use of novel neural network architectures that are flexible and expressive, and allow the algorithm to learn and adapt the bases that are selected for the input and outputs of the solution map. See also the recent paper [15] for a comparison between these methods and a competitive kernel ridge regression approach based on the theory of operator valued kernels.

In contrast to the aforementioned methods, our three-step framework takes a different path towards operator learning. First, we use equation learning to approximate the functional form of the DE from the training data set, which is a much easier problem than direct approximation of the solution map. We then approximately evaluate the solution map by solving an variational problem that solves a "nearby" DE. To this end, our method is making explicit use of the knowledge that the operator of interest is the solution map of a DE. It is therefore

natural that our method is able to achieve better accuracy (since it is biased towards DE problems) but this higher accuracy comes at a higher computational cost since we still need to solve a DE every time we wish to evaluate the learned operator. Depending on the DE at hand and the desired accuracy, this may be an expensive calculation. In contrast, neural net operators are very efficient to evaluate although they may be more expensive and challenging to train.

Solving PDEs with Gaussian Process/Kernel methods: Finally, we mention that the key to our operator learning framework is the existence of flexible, meshless, and general purpose nonlinear DE solvers such as the kernel method of [49] or the physics informed neural nets (PINNs) of [50] that allow us to "solve" DEs which, in general, may be ill-posed. This is crucial for us since the DE learning algorithms in Step (ii) do not impose any constraints that ensure the learned DE $\overline{\mathcal{P}}$ is in fact well-posed in the classical sense, i.e., these equations may not have solutions at all or may not be uniquely solvable. Additionally, the learned DEs may involve high order or stiff terms that cannot be tackled using classic numerical solvers such as finite differences and finite elements or may require expert intervention and specialized solvers. Methods such as the kernel solver of [49], allow us to overcome these difficulties since the solution of the equation is naturally regularized via a reproducing kernel Hilbert space (RKHS) norm penalty that provides numerical stability. We also note that the use of kernel methods and Gaussian Processes (GPs) for solving DEs has been an active area of research over the last decade; see for example [51–56]. Although the overwhelming majority of the research in this direction appears to be focused on the case of linear DEs. Some notable exceptions are [49,57,58].

1.2. Outline of the article

The article is organized as follows: Section 2 introduces our setup for nonlinear DEs and outlines our approach for equation and operator learning; Section 3 presents our numerical experiments; while Section 4 presents our concluding discussions. The appendix contains additional details of the setup of our experiments.

2. Methodology

Below we outline the details on our proposed approach for learning DEs and their solution operators. We start with the setup of the problem and our notation followed by abstract three-step framework for equation and operator learning that encompasses other families of methods SINDy and PDE-Net. Next we outline a simple kernel implementation which is used in our numerical experiments later. We only consider the case where the DE operator $\mathcal{P}$ is unknown while the boundary operator $\mathcal{B}$ and the data g are assumed to be given. However, the resulting approach can easily be extended to learn boundary conditions as well.

2.1. Setup for nonlinear DEs

Suppose $D \geq 1$ and let $\Omega \subset \mathbb{R}^D$ be a compact and simply-connected domain with boundary $\partial\Omega$. Consider the multi-index $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_D) \in \mathbb{N}^D$ (i.e., a d-dimensional vector of non-negative integers).[2] For a smooth function $u : \Omega \to \mathbb{R}$ we define the partial derivatives $\partial^{\boldsymbol{\alpha}} u := \partial_{x_1}^{\alpha_1} \partial_{x_2}^{\alpha_2} \dots \partial_{x_D}^{\alpha_D} u$ (see [59] for details regarding the multi-index notation in theory of DEs) and further consider two collections of multi-indices $\{\boldsymbol{\alpha}^1, \dots, \boldsymbol{\alpha}^P\} \subset \mathbb{N}^P$ and $\{\boldsymbol{\beta}^1, \dots, \boldsymbol{\beta}^B\} \subset \mathbb{N}^B$ for integers $P, B \geq 0$. Finally we define $M_P := \max_{1 \leq i \leq P} \|\boldsymbol{\alpha}^i\|_1$ and $M_B := \max_{1 \leq i \leq B} \|\boldsymbol{\beta}^i\|_1$. In the rest of the article we have in mind DEs of the form

$$\mathcal{P}\left(\mathbf{x}, \partial^{\boldsymbol{\alpha}_1} u(\mathbf{x}), \dots, \partial^{\boldsymbol{\alpha}_P} u(\mathbf{x})\right) = f(\mathbf{x}), \qquad \mathbf{x} \in \Omega, \tag{1a}$$

$$\mathcal{B}\left(\mathbf{x}, \partial^{\boldsymbol{\beta}_1} u(\mathbf{x}), \dots, \partial^{\boldsymbol{\beta}_B} u(\mathbf{x})\right) = g(\mathbf{x}), \qquad \mathbf{x} \in \partial\Omega, \tag{1b}$$

where, overloading our notation from Section 1, we defined $\mathcal{P} : \mathbb{R}^{J_P} \to \mathbb{R}$ and $\mathcal{B} : \mathbb{R}^{J_B} \to \mathbb{R}$, with $J_P = D + P$ and $J_B = D + B$, are nonlinear functions that define the functional relationships between $\mathbf{x} = (x_1, \dots, x_D)$ and values of u and its partial derivatives in the interior and boundary of Ω. The functions $f : \Omega \to \mathbb{R}$, often referred to as a forcing/source term, and $g : \partial\Omega \to \mathbb{R}$, the boundary condition, constitute the data of the PDE. In most practical problems, $M_B \leq M_P$ and $\max\{M_P, M_B\}$ denotes the *order* of the PDE.

For example consider the one-dimensional second order PDE

$$-\partial_x \left[a(x)\partial_x u(x)\right] + u^3(x) = f(x), \ \ x \in (0,1), \qquad u(0) = u(1) \quad = 0, \tag{2}$$

where $a \in C^1(\overline{\Omega})$ is a spatially varying coefficient, for example, drawn from a random field. We assume this coefficient along with its first derivative can be evaluated but in general it may have a complicated or unknown form. We can read that $\mathcal{B}(x, u(x)) \equiv u(x)$. Expanding the differential operator on the left hand side of the PDE we get $\mathcal{P}\left(x, u(x), \partial_x u(x), \partial_x^2 u(x)\right) = -\partial_x a(x) \partial_x u(x) - a(x)\partial_x^2 u(x) + u^3(x)$. Thus, defining the new variables $\mathbf{s} = (s_1, \dots, s_4) \equiv [x, u(x), u_x(x), u_{xx}(x)] \in \mathbb{R}^4$ and $\mathbf{t} = (t_1, t_2) \equiv [x, u(x)] \in \mathbb{R}^2$[3] we can write $\mathcal{P}(\mathbf{s}) = -a_x(s_1)s_3 - a(s_1)s_4 + s_2^3$ and $\mathcal{B}(\mathbf{t}) = t_2$. Throughout the rest of the article we will assume that whenever a DE is discussed, it is well-defined and has a unique strong solution in the classical sense, that is, defined pointwise.

2.2. An abstract framework for learning DEs and their solution operators

Suppose a set of mesh/observation points $X = \{\mathbf{x}_j\}_{j=1}^J \subset \Omega$ is fixed and let $\{u^{(i)}, f^{(i)}\}_{i=1}^I$ be pairs of solutions and forcing terms for the DE (1) with the same boundary conditions. Our training data consists of noisy observations of the pairs $(u^{(i)}, f^{(i)})$ at the points X, that is,

$$\mathbb{R}^J \ni \mathbf{u}^{(i)} = u^{(i)}(X) + \boldsymbol{\epsilon}^{(i)}, \quad \mathbb{R}^J \ni \mathbf{f}^{(i)} = f^{(i)}(X), \quad \boldsymbol{\epsilon}^{(i)} \sim N(0, \lambda_U^2 I).$$

At this point we may assume that the noise variance λ_U^2 is known but we will treat it as a hyper parameter later. We propose the following abstract three-step framework for learning $\mathcal{P}$ and the corresponding solution operator of the DE:

Step (i): Smoothing the Training Data and Estimating Derivatives. Consider a Banach space $\mathcal{H}$ that is continuously embedded in $C^{M_p}(\Omega)$ Then solve the regression problems

$$\overline{u}^{(i)} = \underset{v \in \mathcal{H}}{\arg\min} \|v\|_{\mathcal{H}}^r + \frac{1}{\lambda_U^2} \|v(X) - \mathbf{u}^{(i)}\|_2^2 \tag{3}$$

for $i = 1, \dots, I$ and $r > 0$. Proceed to compute the partial derivatives $\partial^{\boldsymbol{\alpha}_j} \overline{u}^{(i)}(X)$ for $j = 1, \dots, P$, i.e., the pertinent partial derivatives of the smoothed solutions involved in (1) evaluated at X; note that this is well-defined thanks to our assumption that $\mathcal{H} \subset C^{M_P}(\Omega)$.[4]

Step (ii): Learning the Functional Form of the DE. Define the set of vectors

$$\overline{\mathbf{s}}_j^{(i)} = \left(\mathbf{x}_j, \partial^{\boldsymbol{\alpha}_1} \overline{u}^{(i)}(\mathbf{x}_j), \dots, \partial^{\boldsymbol{\alpha}_P} \overline{u}^{(i)}(\mathbf{x}_j)\right) \in \mathbb{R}^{J_P}, \tag{4}$$

for $i = 1, \dots, I$. Now consider another Banach space $\mathcal{H}'$ that is continuously embedded in $C^0(\mathbb{R}^{J_P})$, so that pointwise evaluation is well-defined, and approximate the function $\mathcal{P}$ via the optimal recovery problem[5]

$$\overline{\mathcal{P}} = \underset{Q \in \mathcal{H}'}{\arg\min} \|Q\|_{\mathcal{H}'} \quad \text{s.t.} \quad Q(\overline{\mathbf{s}}_j^{(i)}) = f^{(i)}(\mathbf{x}_j), \qquad i = 1, \dots, I, j = 1, \dots, J. \tag{5}$$

[2] Henceforth we use bold letters to denote d-dimensional vectors of integers or reals for $d \geq 2$

[3] The entries s_i, t_i simply denote the values of x as well as u and its partial derivatives evaluated at x. This compact notation will be useful later on.

[4] If one wishes to learn the boundary operator $\mathcal{B}$ then the $\partial^{\boldsymbol{\beta}_j} \overline{u}^{(i)}$ should also be computed at a set of boundary collocation points.

[5] One can also formulate a regression problem analogous to Step (i) if the $f^{(i)}(X)$ are believed to be noisy.

Step (iii): Operator Learning by Solving the Learned DE. The goal of operator learning is to predict the solution of a DE given a source term from a training data set of solution and source pairs. To this end, Consider a new pair of solution and source term $(\widetilde{u}, \widetilde{f})$, that did not exist in the training data set $\{u^{(i)}, f^{(i)}\}_{i=1}^{I}$. Then our goal is to predict $\widetilde{u}$ given $\widetilde{f}$, but since $\mathcal{P}$ is unknown, we propose to formulate the following DE where, once again, we assumed the boundary conditions are known:

$$\begin{aligned} &\overline{\mathcal{P}}\left(\mathbf{x}, \partial^{\boldsymbol{\alpha}_1} u(\mathbf{x}), \ldots, \partial^{\boldsymbol{\alpha}_P} u(\mathbf{x})\right) = \widetilde{f}(\mathbf{x}), \ \mathbf{x} \in \Omega, \\ &\mathcal{B}\left(\mathbf{x}, \partial^{\boldsymbol{\beta}_1} u(\mathbf{x}), \ldots, \partial^{\boldsymbol{\beta}_B} u(\mathbf{x})\right) = g(\mathbf{x}), \ \mathbf{x} \in \partial\Omega. \end{aligned} \tag{6}$$

Note that $\overline{\mathcal{P}}$ is the function given by (5) and the resulting DE is not guaranteed to be well-posed. Henceforth we think of "solving" this DE simply as finding a function $\widehat{u}$ that (approximately) minimizes the residual of (6). To do so, take new sets of collocation points $\{\widetilde{\mathbf{x}}_1, \ldots, \widetilde{\mathbf{x}}_{\widetilde{J}_\Omega}\} \subset \Omega$ in the interior of Ω and $\{\widetilde{\mathbf{x}}_{\widetilde{J}_\Omega+1}, \ldots, \widetilde{\mathbf{x}}_{\widetilde{J}}\} \subset \partial\Omega$ on the boundary $\partial\Omega$. These new collocation points are independent of the observation points X and will only be used to solve (6). Choose parameters $r, \lambda_{\mathcal{P}}, \lambda_{\mathcal{B}} > 0$ and approximate $\widetilde{u}$ by solving the optimization problem[6]

$$\begin{aligned} \widehat{u} := \underset{u \in \mathcal{H}}{\arg\min} \ &\|u\|_{\mathcal{H}}^r + \frac{1}{\lambda_{\mathcal{P}}^2} \sum_{j=1}^{\widetilde{J}_\Omega} \left|\overline{\mathcal{P}}\left(\widetilde{\mathbf{x}}_j, \partial^{\boldsymbol{\alpha}_1} u(\widetilde{\mathbf{x}}_j), \ldots, \partial^{\boldsymbol{\alpha}_P} u(\widetilde{\mathbf{x}}_j)\right) - \widetilde{f}(\widetilde{\mathbf{x}}_j)\right|^2 \\ &+ \frac{1}{\lambda_{\mathcal{B}}^2} \sum_{j=\widetilde{J}_\Omega+1}^{\widetilde{J}} \left|\mathcal{B}\left(\widetilde{\mathbf{x}}_j, \partial^{\boldsymbol{\beta}_1} u(\widetilde{\mathbf{x}}_j), \ldots, \partial^{\boldsymbol{\beta}_B} u(\widetilde{\mathbf{x}}_j)\right) - g(\widetilde{\mathbf{x}}_j)\right|^2. \end{aligned} \tag{7}$$

The above formulation is at the heart of ML inspired DE solvers of [49,50] and can be viewed as finding a minimum norm solution $\widehat{u}$ by imposing the DE and boundary conditions using Lagrange multipliers rather than exact equality constraints (see [49] for a detailed discussion).

Remark 2.1. We also note that Eq. (7) is the only place in our three-step framework where the boundary operator $\mathcal{B}$ appears. Here we are imposing the boundary conditions using a variational/penalty technique which often requires careful tuning of the parameter $\lambda_{\mathcal{B}}$. However, depending on $\mathcal{B}$ one may be able to choose $\mathcal{H}$ in such a way to impose the boundary conditions more accurately as is often done with standard numerical PDE solvers such as imposing natural boundary conditions in finite element methods or Dirichlet boundary conditions in finite difference solvers.

2.3. Brief review of representer theorems for kernel regression

Before proceeding further we give a brief review of representer theorems in RKHSs that will be utilized to implement a kernel instance of the abstract framework of Section 2.2. For brevity we only discuss pertinent results from the literature and refer the reader to the Refs. [60–63] for in-depth treatment of the theory of RKHSs and kernel methods.

Consider a simply connected set $\Theta \subseteq \mathbb{R}^D$. We say that a function $\mathcal{V} : \Theta \times \Theta \to \mathbb{R}$ is a *Mercer kernel* if it is continuous in both arguments, symmetric, that is $\mathcal{V}(\mathbf{y}_1, \mathbf{y}_2) = \mathcal{V}(\mathbf{y}_2, \mathbf{y}_1)$, and positive definite, that is, for any collection of points $Y = \{\mathbf{y}_1, \ldots, \mathbf{y}_J\} \subset \Theta$ the matrix $[\mathcal{V}(Y, Y)]_{ij} = \mathcal{V}(\mathbf{y}_i, \mathbf{y}_j)$ is positive definite. We write $\mathcal{H}_{\mathcal{V}}$ to denote the RKHS associated to $\mathcal{V}$ with its norm denoted by $\|\cdot\|_{\mathcal{V}}$.

Now for $N \in \mathbb{N}$ let $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_N)$ be a vector of N-bounded linear functionals $\phi_j : \mathcal{H}_{\mathcal{V}} \to \mathbb{R}$ defining a vector valued bounded linear map $\boldsymbol{\phi} : \mathcal{H}_{\mathcal{V}} \to \mathbb{R}^N$ and consider regression problems of the form

$$\underset{v \in \mathcal{H}_{\mathcal{V}}}{\text{minimize}} \|v\|_{\mathcal{V}}^2 + \frac{1}{\lambda^2} \|\mathcal{F} \circ \boldsymbol{\phi}(v) - \mathbf{o}\|_2^2, \tag{8}$$

where $\mathcal{F} : \mathbb{R}^N \to \mathbb{R}^O$ is a nonlinear map, $\mathbf{o} \in \mathbb{R}^O$ is a fixed vector of *observations*, and $\lambda > 0$ is a regularization parameter. It follows from [49, Prop. 2.3] that every minimizer $\overline{v}$ of problem (8) is of the form

$$\overline{v}(\mathbf{y}) = \mathcal{V}(\mathbf{y}, \boldsymbol{\phi}) \mathcal{V}(\boldsymbol{\phi}, \boldsymbol{\phi})^{-1} \overline{\mathbf{z}}, \tag{9}$$

where $\mathcal{V}(\mathbf{y}, \boldsymbol{\phi}) : \Theta \to \mathcal{H}_{\mathcal{V}}^{\otimes N}$ is a row vector field on Θ with entries $[\mathcal{V}(\mathbf{y}, \boldsymbol{\phi})]_j = \phi_j(\mathcal{V}(\mathbf{y}, \cdot))$ for $j = 1, \ldots, N$, i.e., we fix $\mathbf{y}$ and apply ϕ_j to $\mathcal{V}(\mathbf{y}, \cdot)$ as a function of its second argument, and $\mathcal{V}(\boldsymbol{\phi}, \boldsymbol{\phi}) \in \mathbb{R}^{N \times N}$ is a symmetric matrix with entries $[\mathcal{V}(\boldsymbol{\phi}, \boldsymbol{\phi})]_{ij} = \phi_i([\mathcal{V}(\cdot, \boldsymbol{\phi})]_j)$. Finally $\overline{\mathbf{z}} \in \mathbb{R}^N$ is a vector that solves the optimization problem

$$\underset{\mathbf{z} \in \mathbb{R}^N}{\text{minimize}} \ \mathbf{z}^T \mathcal{V}(\boldsymbol{\phi}, \boldsymbol{\phi})^{-1} \mathbf{z} + \frac{1}{\lambda^2} \|\mathcal{F}(\mathbf{z}) - \mathbf{o}\|_2^2. \tag{10}$$

Eq. (9) is often referred to as a representer formula for (8) as it states that the minimizers of the latter equation are represented by the finite dimensional vector $\overline{\mathbf{z}}$.

In the particular case where $N = O$ and $\mathcal{F} = \mathrm{Id}$ one can solve for $\overline{\mathbf{z}}$ exactly and substitute in (9) to obtain the well-known representer formula for kernel regression

$$\overline{v}(\mathbf{y}) = \mathcal{V}(\mathbf{y}, \boldsymbol{\phi}) \left(\mathcal{V}(\boldsymbol{\phi}, \boldsymbol{\phi}) + \lambda^2 I\right)^{-1} \mathbf{o}. \tag{11}$$

Furthermore, letting $\lambda \to 0$ we obtain

$$\overline{v}(\mathbf{y}) = \mathcal{V}(\mathbf{y}, \boldsymbol{\phi}) \mathcal{V}(\boldsymbol{\phi}, \boldsymbol{\phi})^{-1} \mathbf{o}, \tag{12}$$

which is the representer formula for the minimizers of the kernel interpolation problem

$$\underset{g \in \mathcal{H}_{\mathcal{G}}}{\text{minimize}} \|g\|_{\mathcal{G}} \quad \text{s.t.} \quad \boldsymbol{\phi}(g) = \mathbf{o}. \tag{13}$$

2.4. Implementation of the three-step framework using kernels

We now present a simple, flexible, and efficient implementation of the framework of Section 2.2 by choosing $\mathcal{H}$ and $\mathcal{H}'$ to be RKHSs. The resulting algorithm relies heavily on the representer formulae discussed above.

Step (i): Let $\mathcal{U} : \Omega \times \Omega \to \mathbb{R}$ be a Mercer kernel with RKHS $\mathcal{H}_{\mathcal{U}}$ that is assumed to be continuously embedded in $C^{M_P}(\Omega)$. A simple choice would be the popular RBF kernel $\mathcal{U}(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(\frac{-1}{2\ell^2}\|\mathbf{x}_1 - \mathbf{x}_2\|_2^2\right)$ whose RKHS consists of infinitely smooth functions although one can choose any other family of Mercer kernels with sufficiently smooth RKHSs; see for example [64]. Consider the regression problem (3) with $\mathcal{H} \equiv \mathcal{H}_{\mathcal{U}}$ and $r = 2$. We can solve this problem by applying formula (11) with $\boldsymbol{\phi} = (\delta_1, \ldots, \delta_J)$ to obtain the minimizers

$$\overline{u}^{(i)}(\mathbf{x}) = \mathcal{U}(\mathbf{x}, X) \left(\mathcal{U}(X, X) + \lambda_{\mathcal{U}}^2 I\right)^{-1} \mathbf{u}^{(i)}, \tag{14}$$

where $\mathcal{U}(\mathbf{x}, X) = \left(\mathcal{U}(\mathbf{x}, \mathbf{x}_1), \ldots, \mathcal{U}(\mathbf{x}, \mathbf{x}_J)\right)$ is viewed as the row vector field with entries $\mathcal{U}(\cdot, \mathbf{x}_j)$ and $\mathcal{U}(X, X) \in \mathbb{R}^{J \times J}$ is a kernel matrix with entries $\mathcal{U}(X, X)_{ij} = \mathcal{U}(\mathbf{x}_i, \mathbf{x}_j)$ and $\lambda_{\mathcal{U}}^2 > 0$ is the regularization/nugget parameter. Since we assumed $\mathcal{H}_{\mathcal{U}}$ is continuously embedded in $C^{M_P}(\Omega)$, then for any multi-index $\boldsymbol{\alpha}_j$ we can directly differentiate this formula to get

$$\partial^{\boldsymbol{\alpha}_j} \overline{u}^{(i)}(\mathbf{x}) = \partial^{\boldsymbol{\alpha}_j} \mathcal{U}(\mathbf{x}, X) \left(\mathcal{U}(X, X) + \lambda_{\mathcal{U}}^2 I\right)^{-1} \mathbf{u}^{(i)}, \tag{15}$$

where $\partial^{\boldsymbol{\alpha}_j} \mathcal{U}(\mathbf{x}, X) = \left(\partial^{\boldsymbol{\alpha}_j} \mathcal{U}(\mathbf{x}, \mathbf{x}_1), \ldots, \partial^{\boldsymbol{\alpha}_j} \mathcal{U}(\mathbf{x}, \mathbf{x}_J)\right)$, the entries of which can be computed offline using analytic expressions or automatic differentiation as they do not depend on the data $\mathbf{u}^{(i)}$ and only depend on the kernel $\mathcal{U}$, the points $\mathbf{x}_j$, and the multi-indicies $\boldsymbol{\alpha}_j$.

Step (ii): With formula (15) at hand we compute the vectors $\overline{\mathbf{s}}_j^{(i)}$ following (4). We then choose another Mercer kernel $\mathcal{K} : \mathbb{R}^{J_P} \times \mathbb{R}^{J_P} \to \mathbb{R}$ with RKHS $\mathcal{H}_{\mathcal{K}}$; once again the RBF kernel would be a convenient choice although a polynomial kernel of the form $\mathcal{K}(\mathbf{s}, \mathbf{s}') = (\mathbf{s}^T \mathbf{s}' + 1)^b$ for some integer $b \in \mathbb{N}$ was found to be very effective in our

[6] One can take u in a different space than $\mathcal{H}$ if prior knowledge of its regularity is available but without such knowledge it is reasonable to assume that is belongs to the same function class as the solutions in the training set.

experiments in Section 3. We then formulate the optimal recovery problem (5) with $\mathcal{H}' \equiv \mathcal{H}_{\mathcal{K}}$. Let us write $\overline{S} := \{\bar{\mathbf{s}}_1, \dots, \bar{\mathbf{s}}_{IJ}\} = \left\{\bar{\mathbf{s}}_1^{(1)}, \dots, \bar{\mathbf{s}}_J^{(1)}, \bar{\mathbf{s}}_1^{(2)}, \dots, \bar{\mathbf{s}}_J^{(2)}, \dots, \bar{\mathbf{s}}_1^{(I)}, \dots, \bar{\mathbf{s}}_J^{(I)}\right\}$ and $\mathbf{f} := (\mathbf{f}^{(1)}, \dots, \mathbf{f}^{(I)})$ denoting the column vector obtained by stacking the $\mathbf{f}^{(i)}$'s. Then applying Eq. (11) once more we have,

$$\overline{\mathcal{P}}(\mathbf{s}) = \mathcal{K}(\mathbf{s}, \overline{S}) \left(\mathcal{K}(\overline{S}, \overline{S}) + \lambda_{\mathcal{K}}^2 I\right)^{-1} \mathbf{f}, \tag{16}$$

where, analogous to Step (i), we write $\mathcal{K}(\mathbf{s}, \overline{S}) = \left(\mathcal{K}(\mathbf{s}, \bar{\mathbf{s}}_1), \dots, \mathcal{K}(\mathbf{s}, \mathbf{s}_{IJ})\right)$ regarded as a row-vector field and $\mathcal{K}(\overline{S}, \overline{S}) \in \mathbb{R}^{IJ \times IJ}$ with entries $\mathcal{K}(\overline{S}, \overline{S})_{ij} = \mathcal{K}(\bar{\mathbf{s}}_i, \bar{\mathbf{s}}_j)$. Furthermore, we introduced the artificial regularization/nugget parameter $\lambda_{\mathcal{K}}^2 > 0$ that improves the conditioning of $\mathcal{K}(\overline{S}, \overline{S})$.

Step (iii): Finally we consider problem (7) and, following [49], we take $\mathcal{H} \equiv \mathcal{H}_{\mathcal{U}}$ and $r = 2$. Let $\widetilde{\delta}_j$ denote the pointwise evaluation operator at $\widetilde{\mathbf{x}}_j$ and define the maps $\widetilde{\boldsymbol{\phi}}_j^i = \widetilde{\delta}_j \circ \partial^{\boldsymbol{\alpha}_i}$, for $i = 1, \dots, p$ and $j = 1, \dots, \widetilde{J}_\Omega$ as well as $\widetilde{\boldsymbol{\psi}}_j^i = \widetilde{\delta}_j \circ \partial^{\boldsymbol{\beta}_i}$, for $i = 1, \dots, q$ and $j = \widetilde{J}_\Omega + 1, \dots, \widetilde{J}$; note that the $\widetilde{\boldsymbol{\phi}}_j^i$ and $\widetilde{\boldsymbol{\psi}}_j^i$ are well-defined bounded linear operators on $\mathcal{H}_{\mathcal{U}}$ due to our assumption that $\mathcal{H}_{\mathcal{U}}$ is continuously embedded in $C^{M_{\mathcal{P}}}(\Omega)$. Further define the vector valued maps $\widetilde{\boldsymbol{\phi}}_j := (\widetilde{\boldsymbol{\phi}}_j^1, \dots, \widetilde{\boldsymbol{\phi}}_j^p)$ and $\widetilde{\boldsymbol{\psi}}_j := (\widetilde{\boldsymbol{\psi}}_j^1, \dots, \widetilde{\boldsymbol{\psi}}_j^q)$. We can now rewrite (7) as

$$\begin{aligned}\underset{u \in \mathcal{H}_{\mathcal{U}}}{\text{minimize}} \; & \|u\|_{\mathcal{U}}^2 + \frac{1}{\lambda_{\mathcal{P}}^2} \sum_{j=1}^{\widetilde{J}_\Omega} \left| \overline{\mathcal{P}}\left(\widetilde{\mathbf{x}}_j, \widetilde{\boldsymbol{\phi}}_j(u)\right) - \widetilde{f}(\widetilde{\mathbf{x}}_j) \right|^2 \\ & + \frac{1}{\lambda_{\mathcal{B}}^2} \sum_{j=\widetilde{J}_\Omega+1}^{\widetilde{J}} \left| \mathcal{B}\left(\widetilde{\mathbf{x}}_j, \widetilde{\boldsymbol{\psi}}_j(u)\right) - g(\widetilde{\mathbf{x}}_j) \right|^2.\end{aligned}$$

Using the representer theorems recalled in Section 2.3, and in particular realizing that this equation is of the same form as (8), we evoke formula (9) to identify

$$\hat{u}(\mathbf{x}) = \mathcal{U}(\mathbf{x}, \widetilde{\boldsymbol{\phi}}) \mathcal{U}(\widetilde{\boldsymbol{\phi}}, \widetilde{\boldsymbol{\phi}})^{-1} \hat{\mathbf{z}},$$

where $\widetilde{\boldsymbol{\phi}} := (\widetilde{\boldsymbol{\phi}}_1, \dots, \widetilde{\boldsymbol{\phi}}_{\widetilde{J}_\Omega}, \widetilde{\boldsymbol{\psi}}_{\widetilde{J}_\Omega+1}, \dots, \widetilde{\boldsymbol{\psi}}_{\widetilde{J}})$ is a concatenated vector of bounded linear functionals and $\hat{\mathbf{z}}$ is a concatenated vector that solves

$$\begin{aligned}\underset{\mathbf{z}=(\mathbf{z}_1, \dots, \mathbf{z}_{\widetilde{J}})}{\text{minimize}} \; & \mathbf{z}^T \mathcal{U}(\widetilde{\boldsymbol{\phi}}, \widetilde{\boldsymbol{\phi}})^{-1} \mathbf{z}^T + \frac{1}{\lambda_{\mathcal{P}}^2} \sum_{j=1}^{\widetilde{J}_\Omega} \left| \overline{\mathcal{P}}\left(\mathbf{z}_j\right) - \widetilde{f}(\widetilde{\mathbf{x}}_j) \right|^2 \\ & + \frac{1}{\lambda_{\mathcal{B}}^2} \sum_{j=\widetilde{J}_\Omega+1}^{\widetilde{J}} \left| \mathcal{B}\left(\mathbf{z}_j\right) - g(\widetilde{\mathbf{x}}_j) \right|^2.\end{aligned} \tag{17}$$

In practice we solve this problem using a gradient descent algorithm, such as the Gauss–Newton algorithm proposed in [49] or L-BFGS.

Remark 2.2. Our proposed kernel method involves the tuning of a number of hyper-parameters such as the regularization parameters $\lambda_{\mathcal{U}}, \lambda_{\mathcal{K}}, \lambda_{\mathcal{P}}$ and $\lambda_{\mathcal{B}}$ as well as other parameters in the kernels $\mathcal{U}, \mathcal{K}$. The tuning of such parameters is a well-studied problem in kernel regression and, more broadly, in statistical theory with methods such as maximum likelihood estimation (MLE) [65], CV [66], and expectation maximization (EM) [67] regarded as standard in the literature; see also [68] and references within.

3. Experiments

Below we compare our computational framework to state-of-the-art algorithms for equation discovery and operator learning. Here we focus on presenting the results and give a brief summary of the setup. Further details of experiments such as the form of kernels or the choices of hyper-parameters are summarized in the Appendix.

Three benchmark DEs were considered: a pendulum model (18), a nonlinear diffusion PDE (19), and the Darcy flow PDE (20). For the DE learning task we compared our kernel method to SINDy [16] for the pendulum and diffusion PDEs. Both our method and SINDy were trained using the same training data with our kernel method used to denoise the training solutions $u^{(i)}$ and to compute the relevant partial derivatives in Step (i). All kernel parameters as well as the regularization parameters $\lambda_{\mathcal{U}}, \lambda_{\mathcal{K}}$ were chosen using CV (cross validation); see Remark 2.2. A test data set was then constructed by taking the same training source terms $f^{(i)}$ from the training set, perturbing them in a controlled manner, and solving the DEs using an independent solver. The Darcy flow PDE was excluded from these experiments since it is unclear how to choose a SINDy dictionary for PDEs with spatially variable coefficients without injecting explicit prior information about the form of the PDE and its dependence on the unknown coefficient. Furthermore, in all three benchmarks we used Dirichlet boundary conditions that are imposed using additional collocation points on the boundary as in (17). Following Remark 2.1 one could incorporate other standard boundary conditions using the same approach as outlined in [49].

For operator learning we used our kernel method and SINDy for Steps (i) and (ii) and used the resulting $\overline{\mathcal{P}}$'s coupled with the kernel solver of [49] for Step (iii); The hyper parameters $\lambda_{\mathcal{P}}, \lambda_{\mathcal{B}}$ were chosen using CV once more and following the same approach as [49]. Results were further compared with the DeepONet algorithm [69] (both the original version and the POD-DeepONet) and the Fourier Neural Operator (FNO) method of [14], trained using the same training data set, to learn the mapping from the source term f to the solution u. Throughout the experiments we also used a second POD-DeepONet, denoted as POD-DeepONet (L) in our tables, which is a large network that we tuned to maximize performance and achieve the closest results to our kernel method. This model serves to show the additional complexity of the neural net that is needed to match the performance of the much simpler three step approach. All operator learning methods were validated on a test set consisting of new pairs of solutions and source terms. Errors were computed via comparison to an independent high-resolution PDE solver that was taken as ground truth.

3.1. Pendulum

The following system of ODEs modeling the motion of a pendulum was considered

$$(u_1)_t(t) = u_2(t), \qquad (u_2)_t(t) \quad = -k \sin(u_1(t)) + f(t), \tag{18}$$

subject to $u_1(0) = u_2(0) = 0$. Note that here we used the parameter t as our input parameter rather than just x as is common notation in ODE and PDE literature. The training data for this experiment consists of the pairs of solutions and forcing functions $(u^{(i)}(t_j), f^{(i)}(t_j))$ for $i = 1, \dots, I$ (we took $I = 10$ or 20). Each forcing $f^{(i)}$ was drawn from a GP (Gaussian Process) and the points t_j were distributed uniformly over a fine mesh; see Fig. 1.

Equation Learning: The function $\overline{\mathcal{P}}$ was learned using our kernel approach for Step (ii) as well as SINDy.[7] We took $I = 20$ (size of the training set) and for testing, the forcing terms $f^{(i)}$ were perturbed using the formula $f_\beta^{(i)} = f^{(i)}(t) + \beta \sin(5\pi t)$, the parameter β controls size of the perturbation and hence, the departure of the test and training sets. The ODEs were then solved using an independent solver to obtain the perturbed solutions $u_\beta^{(i)}$. The kernel smoothing of Step (i) was then used to estimate the pertinent derivatives of the $u_\beta^{(i)}$ which were then used to define a new set of inputs over which the error between $\overline{\mathcal{P}}$ and $\mathcal{P}$ was computed for our kernel method and SINDy. The results are reported in Fig. 2 (left) where we observe that our approach with $\mathcal{K}$ taken to be the polynomial kernel almost perfectly matches SINDy (the points appear to overlap) and the learned equations are very robust to perturbations of the test set, a sign that $\overline{\mathcal{P}}$ is a good *global* approximation to $\mathcal{P}$. Taking $\mathcal{K}$ to be the ARD kernel (an anisotropic variant of the RBF family)

[7] see the Appendix for details such as the SINDy dictionary and definition of kernels in our method.

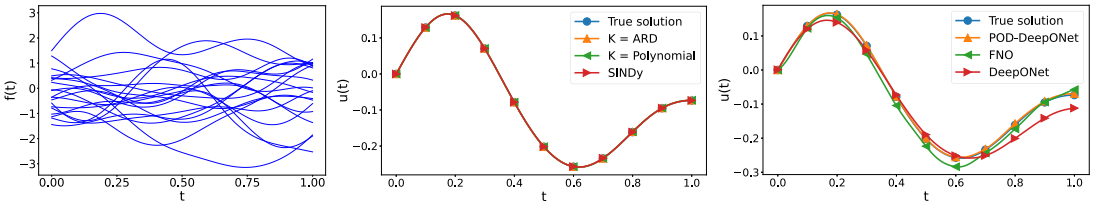
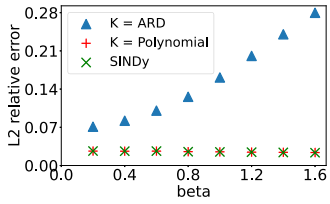


Fig. 1. (Left) the forcing terms used to generate the training data for the pendulum ODE. (Middle) comparing the ODE solutions obtained from our model and SINDy for one of the test forcing terms. (Right) comparing the ODE solutions obtained from POD-DeepONet, FNO, and DeepONet for one of the test forcing terms. Results were obtained with training set of size 20.



Method	$I = 10$	$I = 20$
$\mathcal{K} =$ ARD	$7.5e^{-3}(1.4e^{-3})$	$2.3e^{-3}(3.1e^{-4})$
$\mathcal{K} =$ Polynomial	$\mathbf{6.3e^{-3}(1.1e^{-3})}$	$\mathbf{2.1e^{-3}(1.7e^{-4})}$
SINDy	$7.1e^{-3}(8.5e^{-4})$	$4.5e^{-3}(4.5e^{-3})$
POD-DeepONet	$1.8e^{-1}(2.7e^{-2})$	$5.7e^{-2}(7.1e^{-3})$
POD-DeepONet (L)	$3.7e^{-2}(5.3e^{-3})$	$1.0e^{-2}(1.1e^{-3})$
FNO	$1.2e^{-1}(1.3e^{-2})$	$4.1e^{-2}(3.8e^{-3})$
DeepONet	$2.9e^{-1}(2.8e^{-2})$	$1.3e^{-1}(1.2e^{-2})$

Fig. 2. Experimental results for the pendulum ODE (18). (Left) Test error of the learned function $\overline{\mathcal{P}}$ with our method *vs.* SINDy for the pendulum ODE. The parameter β controls the departure of the test and training forcing terms. Results for the polynomial kernel overlapped with SINDy. (Right) Average L^2 relative errors for the operator learning task of pendulum system computed for 50 test forcing functions. Standard deviations are reported in brackets. (L) indicates the large network variant of POD-DeepONet. Bold text indicates the best errors.

results in drastically different behavior where the error is larger and grows with β, a sign that $\overline{\mathcal{P}}$ approximates $\mathcal{P}$ only *locally* in this setting.

Operator Learning: For operator learning we used our method and SINDy to learn $\overline{\mathcal{P}}$ as above with training data of size $I = 10$ and 20 and compared our three-step approach to DeepONets and FNO. The trained models were then validated on a test set of 50 solution-forcing pairs that were generated by the same procedure as the training set. Fig. 2 (right) compares the average L^2 errors for the operator learning of the pendulum model. We observed that our method with the polynomial kernel is able to achieve the best performance although the errors are close to the ARD kernel and SINDy. The POD-DeepONet (L) model is the next competitive model despite being an order of magnitude worse and using a much larger neural network, i.e., more expensive parameterization. A sample of the predicted solutions for all seven methods is presented in Fig. 1. In particular, it is visually clear that the predicted solutions using the learned DEs are more accurate than the neural net methods.

We also repeated our experiments by adding Gaussian noise to the training data (we used a noise to signal ratio of 0.1), meaning that the solution-forcing terms are no longer satisfying the underlying DE exactly. Results for this experiment are summarized in Table 1. As expected, this additional noise reduces the accuracy of all models but our method using the ARD kernel was still able to achieve the best performance. We note that the SINDy method also had very close performance. FNO achieved the next best result but it was still worse by a factor of 2. Overall the performance gap between the DE learning approach and the neural net methods was smaller in this case indicating that the loss of information due to the additional noise had likely diminished the advantage of our framework.

3.2. Nonlinear diffusion PDE

The following second order nonlinear PDE was considered for our second set of experiments

$$u_t(x,t) = 0.01u_{xx}(x,t)+0.01u^2(x) + f(x), \quad (x,t) \in (0,1)\times(0,1], \tag{19}$$

subject to boundary conditions $u(0,t) = u(1,t) = 0$ for $t \in (0,1]$ and initial conditions $u(x,0) = 0$, for $x \in (0,1)$. Similar to Section 3.1, the training data was generated by drawing random sources $f^{(i)}(x)$ from a GP with the RBF kernel; note that f is only a function of x here.

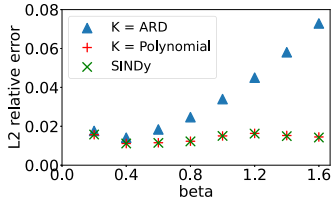
Table 1
Average L^2 relative errors for the operator learning task computed for 50 test forcing functions with 0.1 noise level in the training data. Standard deviations are reported in brackets. For our method, we report the best one from the ARD kernel and the polynomial kernel. (L) indicates the large network variant of POD-DeepONet. Bold text indicates the best errors.

Method	Pendulum	Diffusion	Darcy Flow
Our method	$\mathbf{3.9e^{-2}(2.3e^{-3})}$	$\mathbf{6.3e^{-2}(4.6e^{-3})}$	$7.7e^{-2}(5.0e^{-3})$
POD-DeepONet	$9.7e^{-2}(1.3e^{-2})$	$1.4e^{-1}(1.1e^{-2})$	$9.8e^{-2}(7.2e^{-3})$
POD-DeepONet (L)	$8.1e^{-2}(1.0e^{-2})$	$1.0e^{-1}(8.8e^{-3})$	$\mathbf{7.2e^{-2}(6.5e^{-3})}$
FNO	$8.0e^{-2}(6.8e^{-3})$	$7.7e^{-2}(5.0e^{-3})$	$8.8e^{-2}(9.0e^{-3})$
DeepONet	$1.5e^{-1}(1.9e^{-2})$	$2.3e^{-1}(1.8e^{-2})$	$1.5e^{-1}(1.6e^{-2})$
SINDy	$4.1e^{-2}(3.8e^{-3})$	$6.8e^{-2}(2.3e^{-3})$	N/A

As a benchmark PDE solver in this example we used the same finite-difference solver used by [13] with a higher resolution to serve as an independent proxy for exact solutions of the PDE.

Equation learning:. We followed the same recipe as the equation discovery experiments from Section 3.1 to compare our kernel approach: We used the RBF kernel in step (i), and took $\mathcal{K}$ to be the ARD and polynomial kernels in Step (ii) and compared with SINDy. we trained the models using a training data set of size $I = 20$ and tested the learned $\overline{\mathcal{P}}$ functions on a test set that was obtained via perturbation of the training set, parameterized by the β parameter controlling the deviation of the test set from the training set. The results of our experiments are presented in Fig. 3. Here we see a similar picture to the case of the pendulum ODE, i.e, the polynomial kernel matched the performance of SINDy, and yielded a global approximation while the ARD kernel resulted in a local approximation that for Small β appears and for test points close to the training set appears to match the performance of SINDy but the errors grow rapidly as we deviate from the training set.

Operator learning:. For operator learning experiments we followed the recipe of Section 3.1 once more. All models were trained on data sets of size $I = 10$ and 20 and validated on a test set of size 50, all generated using the same procedure but independently. Fig. 3 (right) summarizes our results with the exact training data. We observe similar trends as the pendulum example with the polynomial kernel achieving the best errors with SINDy achieving slightly worse performance. Interestingly, in this case the ARD kernel appears to perform significantly worse. Among the neural net methods the large POD-DeepONet was most



Method	$I = 10$	$I = 20$
$\mathcal{K} =$ ARD	$1.3e^{-2}(2.1e^{-3})$	$7.5e^{-3}(1.1e^{-3})$
$\mathcal{K} =$ Polynomial	$\mathbf{7.0e^{-3}(5.7e^{-4})}$	$\mathbf{4.1e^{-3}(2.4e^{-4})}$
POD-DeepONet	$1.7e^{-1}(1.5e^{-2})$	$7.8e^{-2}(6.1e^{-3})$
POD-DeepONet (L)	$4.4e^{-2}(3.6e^{-3})$	$1.4e^{-2}(1.3e^{-3})$
FNO	$5.8e^{-2}(4.1e^{-3})$	$1.6e^{-2}(1.3e^{-3})$
DeepONet	$3.4e^{-1}(1.9e^{-2})$	$1.8e^{-1}(1.5e^{-2})$
SINDy	$9.6e^{-3}(9.3e^{-4})$	$4.2e^{-3}(2.3e^{-4})$

Fig. 3. Experimental results for the nonlinear diffusion PDE (19). (Left) Test error of the learned function $\overline{P}$ with our method *vs.* SINDy for the pendulum ODE. The parameter β controls the departure of the test and training forcing terms. Results for the polynomial kernel overlapped with SINDy. (Right) Average L^2 relative errors for the operator learning task of nonlinear diffusion computed for 50 test forcing functions. Standard deviations are reported in brackets. (L) indicates the large network variant of POD-DeepONet. Bold text indicates the best errors.

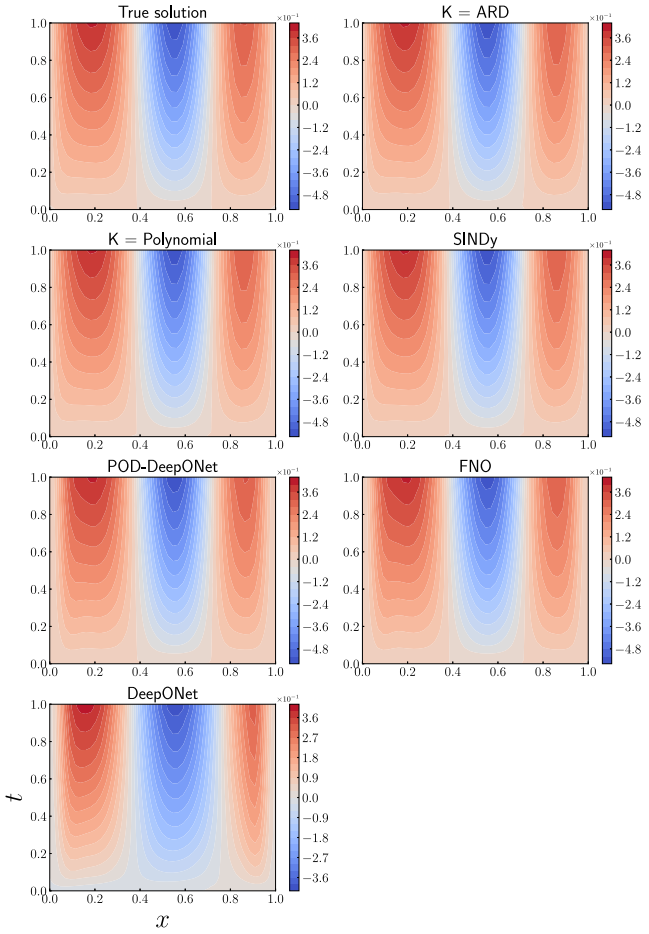


Fig. 4. A comparison of the estimated solutions to the diffusion PDE for one of the forcing terms in the test set with training set of size $I = 20$.

competitive. We also performed the experiments after adding artificial noise to the training data; the results are presented in Table 1. Once again we found that our method achieved the lowest error, followed closely by SINDy. The FNO was once again the best performing neural net based methods. A sample of the predicted solutions of all methods over the test set is presented in Fig. 4.

3.3. Darcy flow

For our third and final example we considered the Darcy flow PDE

$$-\operatorname{div}(a\nabla u)(x) = f(x),\ x \in (0,1)^2, \tag{20}$$

subject to homogeneous Dirichlet boundary conditions. The coefficient a is a spatially variable field given by $a(x) = \exp\left(\sin(\pi x_1) + \sin(\pi x_2)\right) + \exp\left(-\sin(\pi x_1) - \sin(\pi x_2)\right)$. In this experiment we excluded SINDy as the construction of an appropriate dictionary for PDEs with spatially variable coefficients is not possible without additional prior knowledge; see Section 4. Therefore, here we focus primarily on the operator learning problem and compare with the neural nets.

Table 2
Average L^2 relative errors for the operator learning task of Darcy Flow computed for 50 test forcing functions. Standard deviations are reported in brackets. (L) indicates the large network variant of POD-DeepONet. Bold text indicates the best errors.

Method	10 sources	20 sources
$\mathcal{K}$ =ARD	$\mathbf{1.4e^{-2}(1.5e^{-3})}$	$\mathbf{7.1e^{-3}(1.0e^{-3})}$
POD-DeepONet	$1.1e^{-1}(1.2e^{-2})$	$3.6e^{-2}(3.2e^{-3})$
POD-DeepONet (L)	$1.7e^{-2}(1.6e^{-3})$	$1.1e^{-2}(1.1e^{-3})$
FNO	$2.3e^{-1}(2.3e^{-2})$	$4.3e^{-2}(3.6e^{-3})$
DeepONet	$3.7e^{-1}(4.2e^{-2})$	$1.2e^{-1}(1.4e^{-2})$

Our experiments follow a similar setup to the previous problems. Once again the models were trained using data sets of size $I = 10$ or 20 and tested on a set of size 50 with forcing terms drawn from a GP. We also excluded the polynomial kernel as it was not competitive in this example. The results of our experiments with exact training data are summarized in Table 2 where our method with the ARD kernel achieved the lowest error followed closely by the large POD-DeepONet. Experimental results with the noisy training set are presented in Table 1. Interestingly, in this setting large POD-DeepONet achieved the best errors followed very closely by our method (the difference is well withing the standard deviation of the errors). In fact, the difference between our method, POD-DeepONet and FNO was quite small in this experiment compared to the previous two examples. Fig. 5 shows a sample of the predicted solutions from the test set for all methods.

3.4. Main takeaways from experiments

Our experiments focused on the two distinct tasks of equation learning and operator learning. We make three primary observations regarding equation learning: (a) kernel smoothing is a good pre-processing step for denoising and estimation of gradient information before learning DEs for both SINDy and the kernel approach. In fact, our kernel method for Step (i) extends the applicability of SINDy to training data that is provided on unstructured meshes; (b) the performance of the kernel method for step (ii) is closely tied to the choice of the kernel $\mathcal{K}$: With the polynomial kernel we matched the performance of SINDy in the pendulum and diffusion examples while the ARD kernel resulted in a local approximation to $\mathcal{P}$[8]; (c) Our kernel approach is more widely applicable than SINDy as demonstrated with the Darcy flow PDE where it is unclear how one could construct a dictionary for SINDy to begin with due to the unknown spatially variable coefficient. Here the ARD kernel appeared to yield good results while the polynomial kernel was far from being competitive. This is precisely due to the fact that the

[8] This is not surprising considering the fact that both methods are solving optimal recovery problems over the monomial basis. However, SINDy would impose a sparsity bias in that basis while the kernel approach does not.

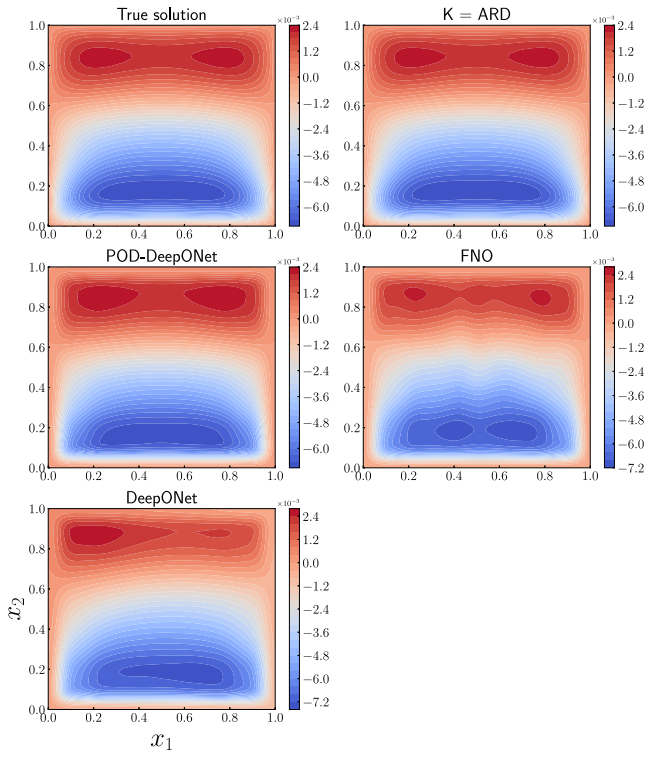


Fig. 5. A comparison of the estimated solutions to the Darcy flow PDE for one of the forcing terms in the test set with training set of size $I = 20$.

Polynomial kernel (and by extension dictionaries involving polynomial terms) are insufficient for capturing a variable coefficient PDE.

Our results concerning operator learning led to two primary observations: (a) operator learning via equation learning consistently outperformed neural net methods when exact training data was available (often by an order of magnitude); (b) the performance gap was smaller when noisy training data was involved but even then our method was (barely) beaten by the POD-DeepONet algorithm for the Darcy Flow example only. It is noteworthy that the POD-DeepONet was using a significantly larger set of parameters than our (much simpler) kernel method and it took significant tuning and architecture adjustment to achieve this level of performance.

4. Discussions and conclusions

Below we collection some discussions and concluding remarks that place our abstract framework as well as our kernel implementation of equation and operator learning within the context of existing methods in the literature for both tasks.

4.1. Our abstract framework

We highlight that our abstract three-step framework from Section 2 unifies many existing equation learning/discovery methods under the umbrella of optimal recovery and extends them to perform operator learning. For example, choosing $\mathcal{H}$ in Step (i) to be the appropriate RKHS associated to splines, we obtain the spline method implemented in the PySINDy package [23] for estimating gradients. One can also take $\mathcal{H}$ to be a Barron space [70] to obtain a neural net approximation for the derivatives as is done in PINNs. Choosing a sparsity promoting norm such as a 0-norm or a 1-norm (with $r = 1$) in Step (ii) yields methods such as SINDy while a Barron norm will yield a neural net approach such as PDE-Net [29,30]. The same is also true for Step (iii), one can choose $\mathcal{H}$ to be a neural net space to obtain solvers such as PINNs [50], or even a finite-dimensional subset of a Sobolev space towards obtaining a finite element solver.

We emphasize that the choice of the spaces $\mathcal{H}, \mathcal{H}'$ in Steps (i–iii) and more broadly, the models employed at each step, are largely dependent on the available data and information for the problem at hand, as well as downstream tasks in engineering or scientific pipelines as we will discuss in the paragraphs below.

4.2. Discovering DEs with kernels vs sparse recovery

The primary focus of the equation learning literature (see for example [9,10,12,24]) has been the extraction of explicit and interpretable equations that describe natural laws that govern physical processes. In our framework, this amounts to finding a human interpretable and simple expression for $\overline{\mathcal{P}}$. In this perspective, it is therefore natural to formulate Step (ii) over an appropriate set of features for $\overline{\mathcal{P}}$ and impose a sparsity assumption on those features as is customary in the SINDy method. Our kernel approach on the other hand, does not aim to find a human interpretable expression for $\overline{\mathcal{P}}$ but rather approximates $\mathcal{P}$ with a large number of features (possibly infinite) with the hope of achieving the most robust and accurate approximation to $\mathcal{P}$. The difference between these two perspectives has major implications in terms of their performance and applicability:

4.2.1. Downstream tasks

Whether or not one chooses to employ a sparse recovery approach to learning DEs or our kernel method should be decided by downstream tasks and pipelines and how the learned equation $\overline{\mathcal{P}}$ will be utilized. For example, in a scientific discovery application, where the goal is to discover new physical laws governing a phenomenon of interest, it is natural to employ sparse recovery to achieve a solution $\overline{\mathcal{P}}$ that is interpretable by a human as was done in the original works [9,10].

On the other hand, in operator learning or data-driven simulation scenarios, it is more important to obtain an accurate and robust $\overline{\mathcal{P}}$ over a large or redundant set of features in which case our kernel method is more desirable. We also emphasize that in such scenarios one should still take advantage of a good dictionary of features if additional a priori information is available. This can be easily achieved for the kernel method by taking the original kernel $\mathcal{K}$ and augmenting it with the kernel defined by the feature maps from the dictionary that is

$$\mathcal{K}_{\text{aug}}(\mathbf{s}, \mathbf{s}') := \mathcal{K}(\mathbf{s}, \mathbf{s}') + \sum_{\ell=1}^{L} F_\ell(\mathbf{s}) F_\ell(\mathbf{s}),$$

where we used $\{F_\ell\}_{\ell=1}^{L}$ for $F_\ell : \mathbb{R}^{J_P} \to \mathbb{R}$ to denote a set of features from a given dictionary. Then the set of feature maps of the resulting kernel $\mathcal{K}_{\text{aug}}$ is precisely the union of the F_ℓ with the feature maps of $\mathcal{K}$, and is therefore a more expressive kernel.

4.2.2. Choosing features and the role of a priori knowledge

It is well-known that the performance of sparsity promoting methods such as SINDy is closely tied to the construction of a good dictionary, in fact, in all of our experiments we used the dictionaries that were suggested by previous authors and were known to give competitive results. Due to its bias towards sparse solutions in the prescribed dictionary, it is often easy to setup SINDy to fail buy simply choosing a bad dictionary. We did not present such examples in this paper as we believe this to be an unfair comparison that is not related to how SINDy is often used in practice. To this end, if we have a good dictionary and the training data is sufficient, then we expect sparsity promoting methods to perform well. This fact has motivated various approaches, such as the Ensemble-SINDy [71], that aim to automate and improve the construction of dictionaries. However, there are various situations where the explicit construction of a dictionary is impossible. Consider our Darcy flow PDE (2) with a coefficient $a(x)$ that is unknown. In this case one cannot construct a simple dictionary

of functions (such as polynomials) for $\mathcal{P}$. This is of course possible if we knew the regularity of $a(x)$ and the manner in which $\mathcal{P}$ depends on a but then we are injecting strong prior information into the problem but this may be unrealistic.

Broadly speaking, the kernel approach, thanks to its large/infinite number of feature maps, is more suitable in situations where very little information about the form of $\mathcal{P}$ is available and variable coefficients exist. Another major advantage of the kernel approach is that it naturally accommodates the tuning/learning of kernel parameters which amounts to tailoring the feature maps (the dictionary terms) to the problem at hand. This extra flexibility is what allowed us to obtain superior results in our operator learning experiments using CV. To our knowledge, when sparsity promoting techniques are employed for equation learning, the dictionary itself is rarely tuned.

4.3. Operator learning via DE learning vs function space regression

At the moment the dominant approach to operator learning in the literature can be broadly categorized as regression of maps between function spaces. Many existing algorithms such as DeepONets [13,69], FNOs [14,48], the multipole graph neural operator [72], and the PCA-Net [17], fall within this category. Our approach to operator learning is fundamentally different from these methods as it relies on first learning the functional form of the PDE (that is $\mathcal{P}$), and then solving the learned PDE with a new forcing or boundary data. To our knowledge, our approach is the first of its kind and our experiments suggest that operator learning via PDE learning is significantly more data efficient and gives superior performance in small data regimes if our goal is to obtain the most accurate approximation to the operator. We conjecture this is due to the fact that our method uses explicit knowledge of the fact that the operator of interest is the solution map of a PDE. On the other hand, if our ultimate goal for operator learning is to obtain a cheap/fast approximation to the solution map, for example as a model emulator in an engineering workflow, then function space regression techniques may be more appropriate.

CRediT authorship contribution statement

Da Long: Investigation, Software, Validation, Visualization. **Nicole Mrvaljević:** Conceptualization, Formal analysis, Investigation, Validation, Visualization. **Shandian Zhe:** Conceptualization, Funding acquisition, Investigation, Resources, Supervision, Writing – original draft. **Bamdad Hosseini:** Conceptualization, Formal analysis, Investigation, Methodology, Supervision, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Bamdad Hosseini reports financial support was provided by National Science Foundation. Shandian Zhe reports financial support was provided by National Science Foundation.

Data availability

Data will be made available on request.

Acknowledgments

The authors would like to dedicated this article to Prof. Nilima Nigam in celebration of her 50th birthday. They are also grateful to the reviewers for their insightful comments and suggestions. SZ has been supported by NSF, United States of America grants CAREER IIS-2046295, and OAC-2311685. BH is supported by the NSF, United States of America grant DMS-2208535.

Table 3
The hyper-parameters we used for the pendulum experiments for exact training sets of size $I = 10, 20$ as well as the noisy training set of size $I = 20$.

Hyper-parameters		$I = 10$	$I = 20$	$I = 20$ (Noisy)
(i):	$\lambda_{\mathcal{U}}$	$1.0e^{-8}$	$1.0e^{-8}$	$1.0e^{-8}$
u_1	σ	$(0.15, 0.45)$		$(0.15, 0.65)$
u_2	σ	$(0.1, 0.4)$		$(0.1, 0.8)$
(ii):	$\lambda_{\mathcal{K}}$	$1.0e^{-5}$	$1.0e^{-5}$	$1.0e^{-1}$
$\overline{\mathcal{P}}_1$	$\ell_1 = \ell_2$	0.52	1.0	1.0
	d	5	3	1
	c	3.5	0.015	0.01
$\overline{\mathcal{P}}_1$	$\ell_1 = \ell_2$	3.0	2.4	1.9
	d	5	3	1
	c	2.8	0.01	0.01

Table 4
The hyper-parameters we used for the diffusion experiment for exact training sets of size $I = 10, 20$ as well as the noisy training set of size $I = 20$. Reported "Failed" values indicate high errors that were not competitive.

Hyper-parameters		$I = 10$	$I = 20$	$I = 20$ (Noisy)
(i):	$\lambda_{\mathcal{U}}$	$1.0e^{-3}$	$1.0e^{-3}$	$1.0e^{-3}$
	σ	$(0.15, 0.7)$		$(0.4, 1.0)$
(ii):	$\lambda_{\mathcal{K}}$	$1.0e^{-3}$	$1.0e^{-3}$	$1.0e^{-3}$
	$(\ell_1, \ldots, \ell_3)$	$(0.50, 1.3, 0.13)$	$(0.50, 1.3, 0.13)$	$(0.50, 2.0, 0.25)$
	d	2	2	*Failed*
	c	0.23	0.0	*Failed*

Appendix. Details of experiments

Below we present additional details regarding our experiments in Section 3.

A.1. Common setup

For the kernel PDE solver in Step (iii) we used the implementation of [49] (https://github.com/yifanc96/NonLinPDEs-GPsolver). For estimation of derivatives in our method and the training of DeepONets we used Jax. For FNO we used the code base provided by the authors in [14]. The POD-DeepONet was implemented using Pytorch. We used Python to implement SINDy, with iterative thresholding, with NumPy for the least squares step.

For all three DEs we conducted the experiments with $I = 10$ and 20 pairs of solutions-sources (the $u^{(i)}, f^{(i)}$ pairs in Section 2) in the training set. In the $I = 20$ case we also conducted experiments with a noisy training set where a Gaussian noise of noise-to-signal ratio 0.1 was added to both the training solutions and the training sources. In all of these experiments we validated the models on the same test set of 50 solution–source pairs.

For solving the optimization problem (17) we used the Gaussian-Newton algorithm of [49] for the pendulum ODE and the Diffusion PDE with 50 iterations. In the case of the Darcy flow PDE we ran 4000 steps of L-BFGS with step sizes of 0.2 and 0.5. For all of the kernel matrices involved in our implementation we used diagonal nugget terms of the form ηI, where $\eta > 0$ is a constant and I is an identity matrix of the same size as the requisite kernel matrix; also see Section 2.4. The value of η was tuned for each experiment separately; see Tables 3–5 for a summary of the chosen nuggets.

For the POD-DeepONet we set the number of bases to maximum and varied the number of hidden layers from 2 to 3, and the width over 256, 512, and 1024. We trained for 100000 epochs to ensure convergence. We also trained a large variant of the POD-DeepONet (denoted as POD-DeepONet (L)) in all examples, where we set the width of the network to 8192. We implemented FNO using the standard four layer architecture for the integral operators, and varied the width over 64, 128, and 256. We trained the model for 4000 epochs to make sure it had converged. Finally we implemented the standard DeepONet with 2

Table 5
The hyperparameters we used for the Darcy Flow experiments for exact training sets of size $I = 10, 20$ as well as noisy training set of size $I = 20$.

Hyper-parameters		$I = 10$	$I = 20$	$I = 20$ (Noisy)
(i):	$\lambda_{\mathcal{V}}$	$1.0e^{-8}$	$1.0e^{-8}$	$1.0e^{-2}$
	σ	$(0.15, 0.35)$		$(0.05, 0.5)$
(ii):	$\lambda_{\mathcal{K}}$	$1.0e^{-3}$	$1.0e^{-3}$	$1.0e^{-1}$
	$(\ell_1, \ldots, \ell_6)$	$(1.2, 1.2, 8.0, 8.0, 10.0, 10.0)$	$(0.4, 0.4, 3.2, 3.2, 5.0, 5.0)$	$(0.64, 0.64, 2.0, 2.0, 3.0, 3.0)$

and 3 hidden layers and varied the width from 256, 512, and 1024 and trained for 100000 epochs. All of the above neural nets were trained using the Adam optimizer. We also used different activation functions (GELU, Tanh, and ReLU) and varied the learning rates from $1e^{-3}$, $1e^{-4}$, and $1e^{-5}$. Our reported results for each neural net method were the best test errors that were obtained by searching over the aforementioned set of architectures and hyperparameters.

A.2. The kernels

Throughout our experiments we used three kernels in Steps (i)–(iii) of our framework. The RBF kernel (also known as the squared exponential or Gaussian kernel)

$$\mathcal{V}_{\text{RBF}}(\mathbf{y}, \mathbf{y}') = \exp\left(-\frac{\|\mathbf{y} - \mathbf{y}'\|^2}{2\sigma^2}\right) \quad \mathbf{y}, \mathbf{y}' \in \mathbb{R}^D,$$

with hyper-parameter $\sigma > 0$. We primarily used this kernel in Step (i) of all of our experiments for smoothing the training data and estimating the requisite partial derivatives. The same kernel was also used in Step (iii) and during the implementation of the kernel solver of [49].

We also considered a tensorized anisotropic version of this kernel, which we referred to as the (automatic relevance determination) ARD kernel in our experiments:

$$\mathcal{V}_{\text{ARD}}(\mathbf{y}, \mathbf{y}') = \prod_{j=1}^{D} \exp\left(-\frac{|y_j - y_j'|^2}{2\ell_j^2}\right) \quad \mathbf{y}, \mathbf{y}' \in \mathbb{R}^D,$$

with hyper-parameters $\ell_j > 0$. The ARD kernel is simply a tensorization of 1D Gaussian kernels which uses a different length scale along each input coordinate. Finally, we also used the polynomial kernel

$$\mathcal{V}_{\text{Poly}}(\mathbf{y}, \mathbf{y}') = (\mathbf{y}^T \mathbf{y}' + c)^d, \quad \mathbf{y}, \mathbf{y}' \in \mathbb{R}^D,$$

with hyper-parameter $c \in \mathbb{R}$ and $d \in \mathbb{N}$. We only considered $d = 2, 3, 4$, and 5. For all experiments and CV to choose the hyperparameters. The ARD and polynomial kernels were used in Step (ii) of our framework.

A.3. Details for the pendulum benchmark

The training data was generated by the following recipe: the source terms $f^{(i)}$ for $i = 1, \ldots, I$ were drawn independently from a GP with the RBF kernel and lengthscale 0.2, For each source term the ODE was solved using the SciPy solve_ivp function on a fine grid and sub-sampled over a uniform grid of the t_j's for $j = 1, \ldots, 30$. The test data was generated using the same recipe except that 50 independent source terms were drawn. For operator learning the L^2 errors between the predicted solutions and the test solutions were computed over the t_j grid and then averaged over the test set. When implementing SINDy, we implemented the first equation exactly and only learned the second equation using the dictionary $\{(u_2)_t(t), u_1(t), u_1(t)^2, u_1(t)^3, \sin(u_1(t)), \cos(u_1(t)), 1\}$.

When implementing our method we learned each equation in the system separately assuming that the right hand side for each coordinate is a function of both u_1 and u_2, i.e., we considered the system of ODEs

$$(u_1)_t(t) = \overline{\mathcal{P}}_1\left(u_1(t), u_2(t)\right)$$
$$(u_2)_t(t) = \overline{\mathcal{P}}_2\left(u_1(t), u_2(t)\right).$$

All hyperparameters involved in the training of our kernel method for this example are summarized in Table 3. We used the Gaussian kernel for Step (i) but length scales were tuned for each instance of the data separately, therefore we report only the range of σ for each coordinate of the solution. The Gaussian kernel was also used for Step (iii) with a lenghthscale that was chosen in the same range that was tuned for Step (i). We also used different lengthscales for each of $\overline{\mathcal{P}}_1$ and $\overline{\mathcal{P}}_2$ as indicated in the table.

A.4. Details for the diffusion PDE benchmark

The test data set was generated by drawing the source terms from the same GP as in the pendulum example of Appendix A.3. The solution $u^{(i)}(x,t)$ for each force $f^{(i)}(x)$ was computed on a fine grid using an independent finite difference solver before they were subsampled to a space–time grid of size 15×15, constituting the training set, so for each tuple $(u^{(i)}, f^{(i)})$ we collected a total of 225 values for a total training set size of $I = 10$ and 20. The test data set was produced in the same manner for 50 pairs of solutions and sources. The errors were once again computed by averaging the L^2 errors over the test set. When implementing SINDy we used the dictionary of functions $\{u_t, u_{xx}, u, u^2, u^3, u \cdot u_{xx}, u^2 \cdot u_{xx}, u^3 \cdot u_{xx}, u \cdot u_t, u^2 \cdot u_t, u^3 \cdot u_t, 1\}$.

We parameterized the PDE as

$$\overline{\mathcal{P}}(u(x,t), u_t, u_{xx}(x,t)) = f(x).$$

All hyperparameters involved in the training of our kernel method for this example are summarized in Table 4. Once again we used the Gaussian kernel for Step (i) while the ARD and polynomial kernels were used for Step (ii). Step (iii) also used the Gaussian kernel with a lengthscale that was chosen in the same range that was found in Step (i). We also present an example of the predicted solutions of the PDE from the test set in Fig. 4.

A.5. Details for the Darcy flow benchmark

The training and test sources for the Darcy flow PDE were generated by taking $f(x_1, x_2) \equiv f(x_2)$ and drawing this function from a 1D GP with the RBF kernel and length scale 0.2. The PDE was then solved using a finite difference solver, on a fine mesh and the solutions were subsampled to a uniform grid of size 15×15, following a similar scheme to the diffusion PDE. The test set was generated in the same manner.

We parameterized the PDE as

$$\overline{\mathcal{P}}(x_1, x_2, u, u_{x_1}, u_{x_2}, \Delta u) = f(\mathbf{x}).$$

All hyperparameters involved in the training of our kernel method for this example as summarized in Table 5. The Gaussian kernel was used for Step (i) while the ARD kernel was used for Step (ii). Our experiments using the polynomial kernel for this step lead to bad results. Step (iii) also used the Gaussian kernel with a lengthscale that was chosen in the range that was tuned in Step (i). Example solutions from the test set are presented in Fig. 5.

References

[1] K.F. Riley, M.P. Hobson, S.J. Bence, Mathematical Methods for Physics and Engineering, Cambridge University Press, 1999.
[2] F. Black, M. Scholes, The pricing of options and corporate liabilities, J. Political Econ. 81 (3) (1973) 637–654.
[3] L. Edelstein-Keshet, Mathematical Models in Biology, SIAM, 2005.
[4] J.E. Marsden, T.J. Hughes, Mathematical Foundations of Elasticity, Dover Books, 1994.

[5] R. Temam, Navier-Stokes Equations: Theory and Numerical Analysis, vol. 343, American Mathematical Society, 2001.

[6] G.E. Karniadakis, I.G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, Nat. Rev. Phys. 3 (6) (2021) 422–440.

[7] G. Carleo, I. Cirac, K. Cranmer, L. Daudet, M. Schuld, N. Tishby, L. Vogt-Maranto, L. Zdeborová, Machine learning and the physical sciences, Rev. Modern Phys. 91 (4) (2019) 045002.

[8] J. Willard, X. Jia, S. Xu, M. Steinbach, V. Kumar, Integrating physics-based modeling with machine learning: A survey, 2020, arXiv preprint arXiv:2003.04919.

[9] J. Bongard, H. Lipson, Automated reverse engineering of nonlinear dynamical systems, Proc. Natl. Acad. Sci. 104 (24) (2007) 9943–9948.

[10] M. Schmidt, H. Lipson, Distilling free-form natural laws from experimental data, Science 324 (5923) (2009) 81–85.

[11] J. Kaipio, E. Somersalo, Statistical and Computational Inverse Problems, vol. 160, Springer Science & Business Media, 2006.

[12] S.L. Brunton, J.L. Proctor, J.N. Kutz, Discovering governing equations from data by sparse identification of nonlinear dynamical systems, Proc. Natl. Acad. Sci. 113 (15) (2016) 3932–3937.

[13] L. Lu, P. Jin, G. Pang, Z. Zhang, G.E. Karniadakis, Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, Nat. Mach. Intell. 3 (3) (2021) 218–229.

[14] Z. Li, N.B. Kovachki, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, A. Anandkumar, et al., Fourier neural operator for parametric partial differential equations, in: International Conference on Learning Representations, 2020.

[15] P. Batlle, M. Darcy, B. Hosseini, H. Owhadi, Kernel methods are competitive for operator learning, 2023, arXiv preprint:2304.13202.

[16] S.H. Rudy, S.L. Brunton, J.L. Proctor, J.N. Kutz, Data-driven discovery of partial differential equations, Sci. Adv. 3 (4) (2017) e1602614.

[17] K. Bhattacharya, B. Hosseini, N.B. Kovachki, A.M. Stuart, Model reduction and neural networks for parametric PDEs, SMAI J. Comput. Math. 7 (2021) 121–157.

[18] Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Anandkumar, Neural operator: Learning maps between function spaces with applications to PDEs, J. Mach. Learn. Res. 24 (89) (2023) 1–97.

[19] H.G. Bock, Recent advances in parameter identification techniques for ode, in: Numerical Treatment of Inverse Problems in Differential and Integral Equations, Springer, 1983, pp. 95–121.

[20] H.G. Bock, Numerical treatment of inverse problems in chemical reaction kinetics, in: Modelling of Chemical Reaction Systems, Springer, 1981, pp. 102–125.

[21] A. Stuart, Inverse problems: a Bayesian perspective, Acta Numer. 19 (2010) 451–559.

[22] F. Tröltzsch, Optimal Control of Partial Differential Equations: Theory, Methods, and Applications, vol. 112, American Mathematical Soc., 2010.

[23] B.M. de Silva, K. Champion, M. Quade, J.-C. Loiseau, J.N. Kutz, S.L. Brunton, Pysindy: a python package for the sparse identification of nonlinear dynamics from data, 2020, arXiv preprint arXiv:2004.08424.

[24] H. Schaeffer, Learning partial differential equations via data discovery and sparse optimization, Proc. R. Soc. A 473 (2197) (2017) 20160446.

[25] S.H. Kang, W. Liao, Y. Liu, Ident: Identifying differential equations with numerical time evolution, J. Sci. Comput. 87 (1) (2021) 1–27.

[26] H. Owhadi, C. Scovel, G.R. Yoo, Kernel Mode Decomposition and the Programming of Kernels, Springer, 2021.

[27] Y. He, N. Suh, X. Huo, S.H. Kang, Y. Mei, Asymptotic theory of-regularized PDE identification from a single noisy trajectory, SIAM/ASA J. Uncertain. Quant. 10 (3) (2022) 1012–1036.

[28] Y. He, H. Zhao, Y. Zhong, How much can one learn a partial differential equation from its solution? 2022, arXiv preprint arXiv:2204.04602.

[29] Z. Long, Y. Lu, X. Ma, B. Dong, PDE-Net: Learning pdes from data, in: International Conference on Machine Learning, PMLR, 2018, pp. 3208–3216.

[30] Z. Long, Y. Lu, B. Dong, PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network, J. Comput. Phys. 399 (2019) 108925.

[31] B. Hamzi, H. Owhadi, Learning dynamical systems from data: a simple cross-validation perspective, part I: parametric kernel flows, Physica D 421 (2021) 132817.

[32] M. Darcy, B. Hamzi, J. Susiluoto, A. Braverman, H. Owhadi, Learning dynamical systems from data: a simple cross-validation perspective, part II: nonparametric kernel flows, 2021.

[33] J. Lee, E. De Brouwer, B. Hamzi, H. Owhadi, Learning dynamical systems from data: A simple cross-validation perspective, Part III: Irregularly-sampled time series, Physica D 443 (2023) 133546.

[34] B. Hamzi, H. Owhadi, Y. Kevrekidis, Learning dynamical systems from data: A simple cross-validation perspective, part iv: case with partial observations, Physica D (2023) 133853.

[35] L. Yang, X. Sun, B. Hamzi, H. Owhadi, N. Xie, Learning dynamical systems from data: A simple cross-validation perspective, Part V: Sparse kernel flows for 132 chaotic dynamical systems, 2023, arXiv preprint arXiv:2301.10321.

[36] L. Yang, B. Hamzi, Y. Kevrekidis, H. Owhadi, X. Sun, N. Xie, Learning Dynamical Systems from Data: A Simple Cross-Validation Perspective, Part VI: Hausdorff metric based training of kernels to learn attractors with application to 133 chaotic dynamical systems, 2023.

[37] R.G. Ghanem, P.D. Spanos, Stochastic Finite Elements: A Spectral Approach, Dover Publications, 2003.

[38] D. Xiu, Numerical Methods for Stochastic Computations: A Spectral Method Approach, Princeton University Press, 2010.

[39] A. Cohen, R. DeVore, Approximation of high-dimensional parametric PDEs, Acta Numer. 24 (2015) 1–159.

[40] J.S. Hesthaven, G. Rozza, B. Stamm, et al., Certified Reduced Basis Methods for Parametrized Partial Differential Equations, vol. 590, Springer, 2016.

[41] D.J. Lucia, P.S. Beran, W.A. Silva, Reduced-order modeling: new approaches for computational physics, Prog. Aerosp. Sci. 40 (1–2) (2004) 51–117.

[42] J. Beck, R. Tempone, F. Nobile, L. Tamellini, On the optimal polynomial approximation of stochastic PDEs by Galerkin and collocation methods, Math. Models Methods Appl. Sci. 22 (09) (2012) 1250023.

[43] A. Chkifa, A. Cohen, R. DeVore, C. Schwab, Sparse adaptive taylor approximation algorithms for parametric and stochastic elliptic PDEs, ESAIM Math. Model. Numer. Anal. 47 (1) (2012) 253–280.

[44] A. Chkifa, A. Cohen, C. Schwab, High-dimensional adaptive sparse polynomial interpolation and applications to parametric PDEs, Found. Comput. Math. 14 (4) (2014) 601–633.

[45] F. Nobile, R. Tempone, C.G. Webster, A sparse grid stochastic collocation method for partial differential equations with random input data, SIAM J. Numer. Anal. 46 (5) (2008) 2309–2345.

[46] F. Nobile, R. Tempone, C.G. Webster, An anisotropic sparse grid stochastic collocation method for partial differential equations with random input data, SIAM J. Numer. Anal. 46 (5) (2008) 2411–2442.

[47] M.D. Gunzburger, C.G. Webster, G. Zhang, Stochastic finite element methods for partial differential equations with random input data, Acta Numer. 23 (2014) 521–650.

[48] A. Anandkumar, K. Azizzadenesheli, K. Bhattacharya, N. Kovachki, Z. Li, B. Liu, A. Stuart, Neural operator: Graph kernel network for partial differential equations, in: ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations, 2020.

[49] Y. Chen, B. Hosseini, H. Owhadi, A.M. Stuart, Solving and learning nonlinear PDEs with Gaussian processes, J. Comput. Phys. 447 (2021) 110668.

[50] M. Raissi, P. Perdikaris, Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.

[51] C. Jidling, N. Wahlström, A. Wills, T.B. Schön, Linearly constrained Gaussian processes, Adv. Neural Inf. Process. Syst. 30 (2017).

[52] J. Schmidt, N. Krämer, P. Hennig, A probabilistic state space model for joint inference from differential equations and data, Adv. Neural Inf. Process. Syst. 34 (2021) 12374–12385.

[53] M. Gulian, A. Frankel, L. Swiler, Gaussian process regression constrained by boundary value problems, Comput. Methods Appl. Mech. Engrg. 388 (2022) 114117.

[54] S. Zhang, X. Yang, S. Tindel, G. Lin, Augmented Gaussian random field: Theory and computation, Discrete Contin. Dyn. Syst. S 15 (4) (2022) 931.

[55] N. Krämer, N. Bosch, J. Schmidt, P. Hennig, Probabilistic ODE solutions in millions of dimensions, in: International Conference on Machine Learning, PMLR, 2022, pp. 11634–11649.

[56] A. Besginow, M. Lange-Hegermann, Constraining Gaussian processes to systems of linear ordinary differential equations, in: Advances in Neural Information Processing Systems.

[57] M. Raissi, P. Perdikaris, G.E. Karniadakis, Machine learning of linear differential equations using Gaussian processes, J. Comput. Phys. 348 (2017) 683–693.

[58] C. Mou, X. Yang, C. Zhou, Numerical methods for mean field games based on Gaussian processes and Fourier features, J. Comput. Phys. 460 (2022) 111188.

[59] L.C. Evans, Partial Differential Equations, AMS, 2010.

[60] A. Berlinet, C. Thomas-Agnan, Reproducing Kernel Hilbert Spaces in Probability and Statistics, Springer Science & Business Media, 2011.

[61] B. Schölkopf, A.J. Smola, Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond, MIT Press, 2002.

[62] H. Owhadi, C. Scovel, Operator-Adapted Wavelets, Fast Solvers, and Numerical Homogenization: from a Game Theoretic Approach To Numerical Approximation and Algorithm Design, vol. 35, Cambridge University Press, 2019.

[63] K. Muandet, K. Fukumizu, B. Sriperumbudur, B. Schölkopf, et al., Kernel mean embedding of distributions: A review and beyond, Found. Trends Mach. Learn. 10 (1–2) (2017) 1–141.

[64] M.G. Genton, Classes of kernels for machine learning: a statistics perspective, J. Mach. Learn. Res. 2 (Dec) (2001) 299–312.

[65] C. Williams, C. Rasmussen, Gaussian processes for regression, Adv. Neural Inf. Process. Syst. 8 (1995).

[66] S. Sundararajan, S. Keerthi, Predictive app roaches for choosing hyperparameters in Gaussian processes, Adv. Neural Inf. Process. Syst. 12 (1999).

[67] M.E. Tipping, C.M. Bishop, Probabilistic principal component analysis, J. R. Stat. Soc. Ser. B Stat. Methodol. 61 (3) (1999) 611–622.

[68] S. Ameli, S.C. Shadden, Noise estimation in Gaussian process regression, 2022, arXiv preprint arXiv:2206.09976.

[69] L. Lu, X. Meng, S. Cai, Z. Mao, S. Goswami, Z. Zhang, G.E. Karniadakis, A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data, Comput. Methods Appl. Mech. Engrg. 393 (2022) 114778.

[70] C. Ma, L. Wu, et al., The Barron space and the flow-induced function spaces for neural network models, Constr. Approx. 55 (1) (2022) 369–406.

[71] U. Fasel, J.N. Kutz, B.W. Brunton, S.L. Brunton, Ensemble-SINDy: Robust sparse model discovery in the low-data, high-noise limit, with active learning and control, Proc. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci. 478 (2260) (2022) 20210904.

[72] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, A. Stuart, K. Bhattacharya, A. Anandkumar, Multipole graph neural operator for parametric partial differential equations, Adv. Neural Inf. Process. Syst. 33 (2020) 6755–6766.