AMATH 563: FINAL PROJECT REPORT

IKE GRISS SALAS, SARA ICHINAGA, JAMES HAZELDEN

Applied Mathematics Department, University of Washington, Seattle, WA
grisal@uw.edu, sarami7@uw.edu, jhazelde@uw.edu

Abstract. Discovery of operators underlying dynamics of PDEs is a fundamental problem in essentially all physical sciences. Long et al. [6] introduced an approach for PDE discovery through kernel smoothing and regression. Their framework may better facilitate noisy or coarse observations and has more mathematical guarantees than alternatives such as deep operator networks [7]. In this study, we reproduce the kernel PDE framework and validate it, as in the original work, on a pendulum ODE and the Darcy flow PDE. Further, we investigate the broader applicability by applying it to the chaotic Lorenz system and an empirical "walking water droplet" dataset with further extensions proposed.

1. Introduction and Overview

The dynamics of Partial Differential Equations (PDEs) form the cornerstone of many physical science phenomena, with the discovery of their underlying operators posing a significant challenge. Addressing this issue, Long et al. proposed a robust method employing kernel smoothing, differentiation, and regression to uncover these operators [6]. Their method was found to be particularly effective for handling problems characterized by coarse or noisy observations, and more mathematically verifiable than other techniques, such as DeepONets or Fourier neural operators [7, 5]. Furthermore, their method, although more mathematically involved, is relatively easy to implement and train in practice since it primarily involves direct kernel regression.

The kernel pipeline works in two stages: (1) kernel smoothing and differentiation, and (2) kernel regression. (1) entails smoothing observed data using a kernel of choice to "regress" onto the problem domain. This effectively factors out noise and has the primary advantage of facilitating the calculation of approximate derivatives of observed data. With the smoothed data and the approximate derivatives, step (2) computes an approximate PDE operator model for the data by a single kernel regression. In practice, the choice of kernel for both stages may have a large influence on the outcome.

In this work, we aim to reproduce and validate the kernel smoothing and regression approach. We replicate their work using two specific cases as testbeds: a one-dimensional pendulum with forcing and the Darcy flow PDE, following their original examples. However, our exploration does not stop at mere reproduction. As we seek to substantiate the efficacy of this method, we expand its application to two additional complex systems: the Lorenz system, a classic example of chaotic behavior, and an empirical "walking water droplet" dataset, a real-world example from data observed in a lab.

2. Theoretical Background

Much more of the original detail and references can be found in the original paper [6], but we highlight the key results used for implementation here. Our primary tool for accomplishing this

Date: June 10, 2023.

two step procedure is the use of Representer Theorems (lecture 4 of class notes) enabling us to solve both optimal recovery and regression problems.

Working on the simply connected set $\Theta \subseteq \mathbb{R}^D$, where $D \geq 1$, and a set of collocation points $X = \{\mathbf{x}_j\}_{j=1}^J \subset \Theta$, we let $\mathcal{K}$ be a positive definite symmetric (PDS) kernel on Θ with the associated Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}_\mathcal{K}$. Then we consider solving the following optimal recovery problem:

$$\underset{u\in\mathcal{H}_\mathcal{K}}{\text{minimize}} \|u\|_\mathcal{K} \quad \text{s.t.} \quad \boldsymbol{\phi}(u) = \mathbf{o}, \tag{1}$$

where $\boldsymbol{\phi}$ is a vector map comprised of the mappings[1]

$$\phi_j^q := \delta_j \circ L_q, \quad u \mapsto L_q(u)(\mathbf{x}_j), \quad \text{with} \quad q = 1, \dots, Q$$

with $L_q : C^k(\Theta) \to C^0(\Theta)$ being bounded linear operators from the Banach space of k-times continuously differentiable functions, $\delta_j : C^0(\Theta) \to \mathbb{R}$ being the point wise evaluation operator, and $\mathbf{o}$ being a fixed vector. Letting $\boldsymbol{\phi}$ be the vector of ϕ_j^q for some ordering[2], the interpolation problem (1) can be solved through the use of Representer Theorems [8], with the solution given by

$$\overline{u}(\cdot) = \mathcal{K}(\cdot, \boldsymbol{\phi})\mathcal{K}(\boldsymbol{\phi}, \boldsymbol{\phi})^{-1}\mathbf{o}. \tag{2}$$

Furthermore, we can solve the nonlinear regression problem of the form

$$\underset{u\in\mathcal{H}_\mathcal{K}}{\text{minimize}} \|u\|_\mathcal{K}^2 + \frac{1}{\lambda^2}\|\boldsymbol{\phi}(u) - \mathbf{o}\|_2^2, \tag{3}$$

with solution given by [2][3]

$$\overline{u}(\cdot) = \mathcal{K}(\cdot, \boldsymbol{\phi})(\mathcal{K}(\boldsymbol{\phi}, \boldsymbol{\phi}) + \lambda^2 I)^{-1}\mathbf{o}. \tag{4}$$

3. Method Pipeline

For our pipeline, we consider PDEs of the form

$$\begin{aligned} &\mathcal{P}(\mathbf{x}, \partial^{\boldsymbol{\alpha}_1}u(\mathbf{x}), \dots, \partial^{\boldsymbol{\alpha}_P}u(\mathbf{x})) = f(\mathbf{x}), \ \mathbf{x} \in \Omega \\ &\mathcal{B}(\mathbf{x}, \partial^{\boldsymbol{\beta}_1}u(\mathbf{x}), \dots, \partial^{\boldsymbol{\beta}_B}u(\mathbf{x})) = g(\mathbf{x}), \ \mathbf{x} \in \partial\Omega, \end{aligned} \tag{5}$$

where $\mathcal{P}$, $\mathcal{B}$ are nonlinear functions on the domain interior Ω and boundary $\partial\Omega$ respectively, f and g are forcing functions, and integers $P, B \geq 0$ denote the number of partial derivatives. Note that $\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i \in \mathbb{Z}_{\geq 0}^D$ are simply shorthands for a multi-indexed set of non-negative integers denoting partial derivatives across the dimension of our true solution u. That is, our functions $\mathcal{P}$ and $\mathcal{B}$ are functions of the independent variable $\mathbf{x}$ and some number of partial derivatives of the true solution u.

For the pipeline, we focus on recovering an approximation $\overline{\mathcal{P}}$ of the true function $\mathcal{P}$, but the process can be extended similarly to approximate $\mathcal{B}$. Given a set collocations points $X = \{\mathbf{x}_j\}_{j=1}^J$ where $\{u(\mathbf{x}_j), f(\mathbf{x}_j)\}_{j=1}^J$ are a set of points that satisfy (5), we wish to recover an approximate $\overline{\mathcal{P}}$ provided we are only given noisy observations of our true solution,

$$\mathbf{u} = u(X) + \boldsymbol{\epsilon} \in \mathbb{R}^J \quad \text{and} \quad \mathbf{f} = f(X) \in \mathbb{R}^J,$$

where $\boldsymbol{\epsilon} \sim N(0, \lambda_\mathcal{U}^2 I)$. This will constitute our training data to which we apply our pipeline outlined below.

[1]Further generalization is done in [6].

[2]The selected ordering is innocuous.

[3]There is slight subtly to the existence of this solution as mentioned in [6].

3.1. **Kernel Smoothing and Differentiation.** First we de-noise our training data. We consider functions over the RKHS $\mathcal{H}$ that are continuously embedded in $C^{M_p}(\Omega)$, where $M_p := \max_{1\leq i\leq P} \|\boldsymbol{\alpha}_i\|_1$. Smoothing the data thus amounts to solving the regularized regression problem

$$\overline{u} = \operatorname*{arg\,min}_{v\in\mathcal{H}} \|v\|_{\mathcal{H}}^2 + \frac{1}{\lambda_{\mathcal{U}}^2}\|v(X) - \mathbf{u}\|_2^2.$$

By picking a PDS kernel $\mathcal{U} : \Omega\times\Omega \to \mathbb{R}$ so that its associated RKHS $\mathcal{H}_{\mathcal{U}}$ is continuously embedded in $C^{M_p}(\Omega)$, we can simply apply (4)—where $\boldsymbol{\phi} = (\delta_1, \dots, \delta_J)$ is the vector of pointwise evaluations—to solve the regression problem. The solution is given by

$$\overline{u}(\mathbf{x}) = \mathcal{U}(\mathbf{x}, X)(\mathcal{U}(X, X) + \lambda_{\mathcal{U}}^2 I)^{-1}\mathbf{u}. \tag{6}$$

We note for a sufficiently smooth choice of kernel $\mathcal{U}$, we can approximate any of the multi-index $\boldsymbol{\alpha}_j$ partial derivatives that appear in (5) by differentiating the kernel

$$\partial^{\boldsymbol{\alpha}_j}\overline{u}(\mathbf{x}) = \partial^{\boldsymbol{\alpha}_j}\mathcal{U}(\mathbf{x}, X)(\mathcal{U}(X, X) + \lambda_{\mathcal{U}}^2 I)^{-1}\mathbf{u}. \tag{7}$$

3.2. **Kernel Regression.** Next we learn the approximate $\overline{\mathcal{P}}$. For brevity we use the notation

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

where $J_P = D + P$. Since we do not have the true $\mathbf{s}_j$, we instead work with the approximate $\overline{\mathbf{s}}_j$ obtained from the previous step. Using a new RKHS, $\mathcal{H}'$, continuously embedded in $C^0(\mathbb{R}^{J_P})$, we can recover our approximate $\mathcal{P}$ by solving the following optimal recovery problem

$$\overline{\mathcal{P}} := \operatorname*{arg\,min}_{\mathcal{V}\in\mathcal{H}'} \|\mathcal{V}\|_{\mathcal{H}'} \quad \text{such that} \quad \mathcal{V}(\overline{\mathbf{s}}_j) = f(\mathbf{x}_j).$$

Similar to the kernel smoothing and differentiation step, by picking a PDS kernel $\mathcal{K} : \mathbb{R}^{J_P}\times\mathbb{R}^{J_P} \to \mathbb{R}$ where its associated RKHS $\mathcal{H}_{\mathcal{K}}$ is continuously embedded in $C^0(\mathbb{R}^{J_P})$, we can solve our optimization problem by applying (2). We have

$$\overline{\mathcal{P}}(\mathbf{s}) = \mathcal{K}(\mathbf{s}, S)(\mathcal{K}(S, S) + \lambda_{\mathcal{K}}^2 I)^{-1}\mathbf{f}, \tag{8}$$

where $S = [\mathbf{s}_1, \dots, \mathbf{s}_J]^T$ and $\lambda_{\mathcal{K}}^2$ is an optionally added "nugget" term to improve the conditioning of the matrix $\mathcal{K}(S, S)$.

3.3. **Kernel Functions.** We include the two kernel functions used in our project:

$$\mathcal{K}_{\text{RBF}}(\mathbf{x}, \mathbf{y}) = \exp\big(- \gamma\|\mathbf{x} - \mathbf{y}\|^2\big), \quad \mathbf{x}, \mathbf{y} \in \mathbb{R}^D$$
$$\mathcal{K}_{\text{Poly}}(\mathbf{x}, \mathbf{y}) = \big(\mathbf{x}^\top\mathbf{y} + c\big)^d, \quad \mathbf{x}, \mathbf{y} \in \mathbb{R}^D.$$

For all experiments discussed in Section 5, we used the RBF kernel with hyper-parameter $\gamma > 0$ for the kernel smoothing and differentiation step. For the kernel regression step, we either used the RBF kernel or the polynomial kernel with hyper-parameters $c > 0$ and $d \in \mathbb{N}$.

4. Algorithm Implementation and Development

For algorithm development, we implemented the formulas mentioned in section 2 with all packages used cited in the repository [3]. We benchmarked each implementation as summarized below. To benchmark our kernel smoothing and differentiation steps, we examined the function

$$u(x) = \sin(kx)$$

for $k = 4$. In particular, we drew $J = 100$ uniformly-spaced points from the interval $x \in [0, 4\pi]$ and evaluated u at these points. We then computed the smoothed data $\overline{u}$ and the derivatives u_x and u_{xx} using equations (6) and (7) equipped with the RBF kernel for $\gamma = 1$. We then compared our smoothed data and approximate derivatives to the ground truth values of u and its derivatives. This

Function	No Gaussian Noise	10% Gaussian Noise
$u(x)$	6.0928×10^{-7}	6.1694×10^{-2}
$u_x(x)$	4.6498×10^{-4}	4.6946×10^{-2}
$u_{xx}(x)$	2.9346×10^{-4}	5.4162×10^{-2}
$\mathcal{P}(\mathbf{s})$	6.8475×10^{-13}	–

Table 1. L_2 relative error in our benchmarking function approximations.

process was performed first with noise-free u measurements and then with u measurements polluted with 10% additive Gaussian noise. We used $\lambda_{\mathcal{U}} = 10^{-5}$ in the noise-free case and $\lambda_{\mathcal{U}} = 10^{-2}$ in the noisy case, which we chose empirically. Based on our error results in Table 1, we conclude that our kernel smoothing and differentiation implementation is functional even for noisy data, as long as the hyper-parameters of the method are properly tuned.

We similarly benchmarked our kernel regression implementation against the differential equation

$$u_{xx}(x) + k^2 u(x) = 0$$

for $k = 10$, which we note has the ground truth solution

$$u(x) = A\cos(kx) + B\sin(kx)$$

for $A, B \in \mathbb{R}$. Using $A = 1$ and $B = 0$, we evaluated u, u_x and u_{xx} at $J = 100$ uniformly-spaced collocation points from $x \in [0,\ 4\pi]$. We then sought to ensure that applying equation (8) with $\mathbf{s} = (x, u, u_x, u_{xx})$ and $\mathbf{f} = -k^2 u$ yields an operator such that $\overline{\mathcal{P}}(\mathbf{s}) \approx u_{xx}$. The L_2 relative error in our approximation $\overline{\mathcal{P}}(\mathbf{s})$ is presented in Table 1, from which we conclude that our kernel regression implementation is functional. To obtain these results, we utilized the polynomial kernel with $d = 3$ and $c = 0.015$, along with the nugget term $\lambda_{\mathcal{K}} = 10^{-5}$.

5. Computational Results

All results and code can be found at our GitHub repository [3].

5.1. **Pendulum with Forcing.**

Just like in the original paper [6], we first considered the motion of a pendulum subject to external forcing, which is governed by the following system of ODEs.

$$\begin{cases} \dot{u}_1(t) = u_2(t) \\ \dot{u}_2(t) = -k\sin(u_1(t)) + f(t) \end{cases}$$

In particular, we considered this system for $u_1(0) = u_2(0) = 0$ and $k = 150$, where each instance $i \in \{1, \ldots, I\}$ of the system is influenced by the forcing $f^{(i)}$ drawn from an RBF Gaussian Process with lengthscale 0.2. The training data consists of the collocation points $\{u^{(i)}(t_j), f^{(i)}(t_j)\}_{j=1}^{J}$ collected from time $t = 0$ to time $t = 1$. We visualize the training data in Figure 1, where we considered $I = 3$ trajectories, each with $J = 30$ collocation points. We then applied our pipeline while assuming a system of ODEs of the form

$$\begin{cases} \dot{u}_1(t) = \overline{\mathcal{P}}_1(u_1(t), u_2(t)) \\ \dot{u}_2(t) - f(t) = \overline{\mathcal{P}}_2(u_1(t), u_2(t)). \end{cases}$$

We used the RBF kernel for the smoothing and differentiation step, whereas we used the polynomial kernel for the regression step. For each instance i of the system shown in Figure 1, we tuned the hyper-parameters of our pipeline, which we summarize in Table 2. Doing so yields an approximation $\overline{\mathcal{P}}_2(u_1, u_2)$ of the true governing system $\mathcal{P}_2(u_1) = -k\sin(u_1)$ for each instance i, which we then evaluate at at the training points $u^{(i)}(t_j)$ and visualize in Figure 1.

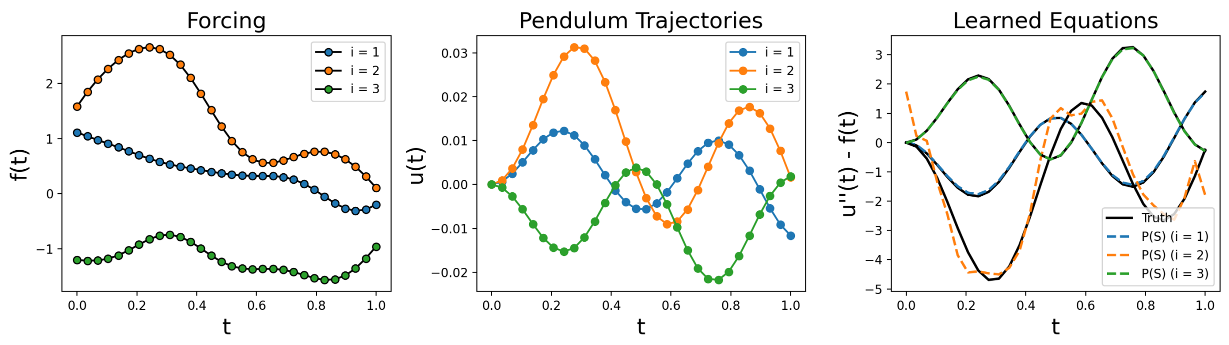


FIGURE 1. Forcing data points $f^{(i)}(t_j)$, pendulum data points $u^{(i)}(t_j)$, and learned equations $\overline{\mathcal{P}}_2$ evaluated at the training data points. The approximations $\overline{\mathcal{P}}_2(u_1, u_2)$ are compared against the true governing function $\mathcal{P}_2(u_1) = -k \sin(u_1)$.

Hyper-parameter	Values Tested	$i = 1$	$i = 2$	$i = 3$
γ	$[1,\ 10,\ 20,\ 50]$	10	1	10
$\lambda_{\mathcal{U}}$	$[10^{-8},\ 10^{-6},\ 10^{-4}]$	10^{-4}	10^{-4}	10^{-4}
d	$[3,\ 4,\ 5]$	5	5	5
c	$[10^{-2},\ 10^{-1},\ 10^{0}]$	10^{-1}	10^{-1}	10^{-1}
$\lambda_{\mathcal{K}}$	$[10^{-8}, 10^{-6}, 10^{-4}]$	10^{-4}	10^{-4}	10^{-4}

TABLE 2. Kernel method hyper-parameters examined and chosen to produce the pendulum results found in Figure 1. Hyper-parameters are defined in Section 3.

Notice that after tuning for the appropriate hyper-parameters, our pipeline is able to accurately recover the true governing equation for pendulum instances $i = 1$ and $i = 3$. We obtain a less accurate reconstruction of $\mathcal{P}_2$ in the $i = 2$ case, even after tuning our hyper-parameters. However, we note that our method is still able to recover the general structure of the true governing equation and that the observed inaccuracies are likely due to oversight in the hyper-parameter sweep.

5.2. **Chaotic Lorenz System.** For our second experiment, we considered the Lorenz system, which is generally defined by the following system of ordinary differential equations.

$$\begin{cases} \dot{x} = \sigma(y - x) \\ \dot{y} = x(\rho - z) - y \\ \dot{z} = xy - \beta z \end{cases}$$

We considered Lorenz parameterized with $(\sigma,\ \rho,\ \beta) = (10,\ 28,\ 8/3)$ and $(x_0,\ y_0,\ z_0) = (-8,\ 8,\ 27)$, which we note defines a chaotic attractor in three-dimensional space. Much like the pendulum example discussed previously, we applied our kernel approach by assuming a system of the form

$$\begin{cases} \dot{x} = \overline{\mathcal{P}}_1(x, y, z) \\ \dot{y} = \overline{\mathcal{P}}_2(x, y, z) \\ \dot{z} = \overline{\mathcal{P}}_3(x, y, z), \end{cases}$$

while using training data that consists of the collocation points $\{x(t_j), y(t_j), z(t_j)\}_{j=1}^{J}$. In particular, we utilized training data collected uniformly from time $t = 0$ to time $t = 5$ with $J = 100$ collocation points. We then applied our pipeline using an RBF kernel for smoothing and differentiation and a polynomial kernel for kernel regression. Select hyper-parameters were tuned during this process,

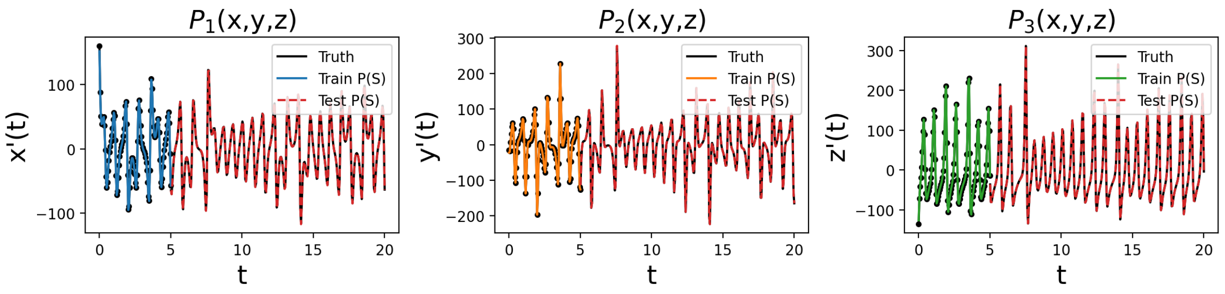


FIGURE 2. Lorenz system x, y, and z time derivatives predicted using the kernel approach of [6]. Model training was performed using 100 collocation points collected from the x, y, z Lorenz system trajectories at time $t = 0$ until time $t = 5$.

which we summarize in Table 3. We then evaluated our learned equations $\overline{\mathcal{P}}_i$ at both the training x, y, z data points as well as several data points beyond the training set, which we then compared against the true Lorenz equations. We see from our findings in Figure 2 that the kernel approach is able to recover the true Lorenz equations after hyper-parameter tuning when given sufficiently-sampled noise-free data.

We then sought to compare the performance of the kernel approach of Long et al. [6] to the performance of the Sparse Identification of Nonlinear Dynamics (SINDy) approach of Brunton et al. [1], as it has previously been shown in the original SINDy paper that SINDy is capable of recovering Lorenz system dynamics from data. In particular, we investigated the performance of the two methods as one varies either the number of training collocation points, or the magnitude of measurement noise on the training data. We visualize our findings in Figure 3, where we use relative error in approximating the true Lorenz equations evaluated at the training points as a proxy for accurate model prediction.

For each data point in Figure 3, we tune the hyper-parameters of the kernel approach by sweeping the values found in Table 3, with the exception of the data points in Figure 3B with noise magnitude $\sigma \leq 10^{-3}$. For these data points, we used the already-tuned hyper-parameters given in Table 3 since we empirically found Lorenz data with such noise magnitudes to be functionally similar to that of the noise-free case. To apply SINDy, we used sequentially thresholded least-squares for sparse regression, polynomials in x, y and z up to degree two for the feature library, and second-order centered finite-difference for derivative computations. We additionally trained our SINDy models using data obtained from kernel smoothing to better compare the two methods.

We see from Figure 3 that the two methods perform comparably well throughout all cases that we tested, with the kernel approach displaying slightly greater robustness to low sampling frequencies. Notice that when given few collocation points, neither method is able to accurately capture the chaotic dynamics of the Lorenz system. However, this issue resolves at a similar rate for both methods as the number of collocation points increases. Similarly, both methods degrade at comparable rates as the magnitude of measurement noise increases.

5.3. **Darcy Flow.**

As a third example, we aimed to reproduce the methods used in the original paper [6] to apply the kernel smoothing and discovery pipeline to the Darcy flow PDE. This is a two-dimensional PDE model of fluid flow through a porous material given by

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

with homogenous boundary conditions. As in [6], we let

$$a(x) = \exp(\sin(\pi x_1) + \sin(\pi x_2)) \exp(-\sin(\pi x_1) - \sin(\pi x_2)).$$

Hyper-parameter	Values Tested	Value Chosen
γ	$[1,\ 5,\ 10,\ 20,\ 50]$	50
$\lambda_{\mathcal{U}}$	$[10^{-5},\ 10^{-4},\ 10^{-3}]$	10^{-4}
$\lambda_{\mathcal{K}}$	$[10^{-5},\ 10^{-4},\ 10^{-3}]$	10^{-4}

TABLE 3. Kernel method hyper-parameters examined and chosen to produce the Lorenz system results found in Figure 2. Note that due to apriori knowledge of the second-degree polynomial structure of the Lorenz equations, we set $d = 2$ and $c = 1$.

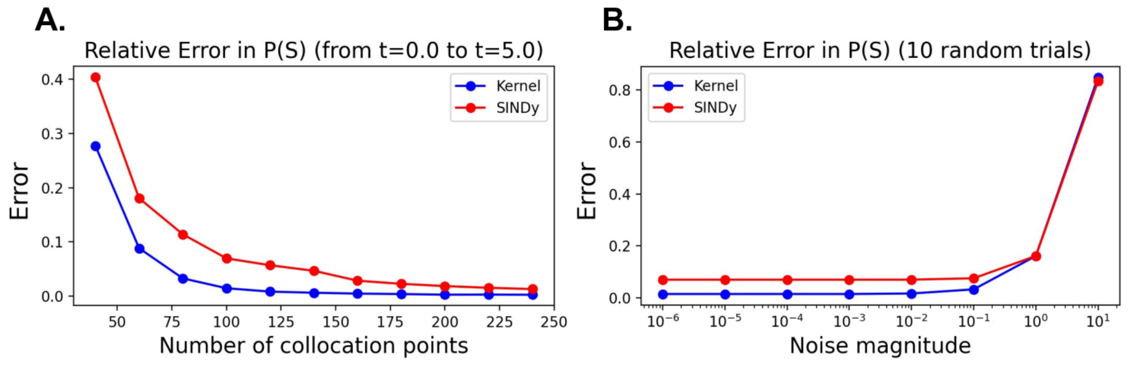


FIGURE 3. Kernel approach (blue) versus SINDy approach (red) as one varies either (A) the number of training collocation points, or (B) the magnitude of measurement noise polluting the training data. We use training data from $t = 0$ to $t = 5$ for both plots. Clean data is used to produce plot (A). In plot (B), we use and $J = 100$ collocation points and plot the average error over 10 instantiations of random noise.

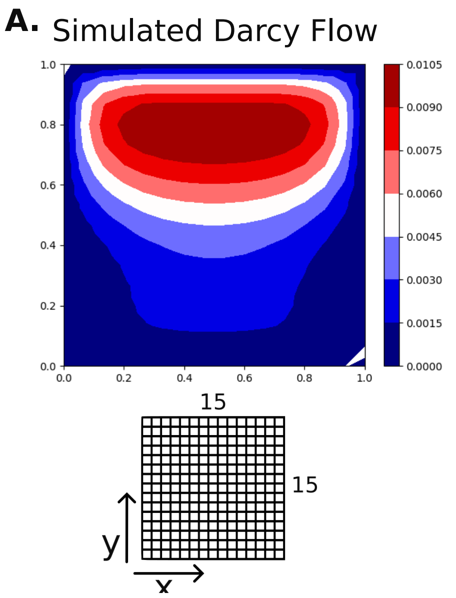
Hyper-parameter	Values Tested	Value Chosen
γ	`linspace`$(1,\ 10,\ 50)$	10
$\lambda_{\mathcal{K}}$	$10\hat{}($`linspace`$(-6,\ -3,\ 50))$	10^{-6}

TABLE 4. Darcy flow kernel regression hyper-parameters.

Furthermore, the forcing terms were constructed using a Gaussian process with an RBF kernel in the x direction and then repeated in the y direction, as in the original paper. The flow data was generated by numerically solving the exact PDE using the Fenics library (see Figure 4.A).

The kernel pipeline was applied to the input data on a coarse 15 by 15 grid, producing an estimated PDE operator $\overline{\mathcal{P}}$. One method for validation of this predicted operator is to then solve the resulting PDE defined by $\overline{\mathcal{P}}$. However, a simpler approach is to directly compute the forcing prediction (right hand side of the PDE) produced by the operator applied to the original data. These predictions using the RBF and Polynomial kernels are shown in Figure 4.B.

Cross validation was performed to determine the optimal choice of kernel parameters. Our results and choice of parameters are summarized in the Table 4. Note that just like in [6], we found that the polynomial kernel works poorly for the regression stage. In the original paper, an ARD kernel was used, which is a product of RBF kernels, but in our case we found a single RBF kernel resulted in good forcing reproduction.



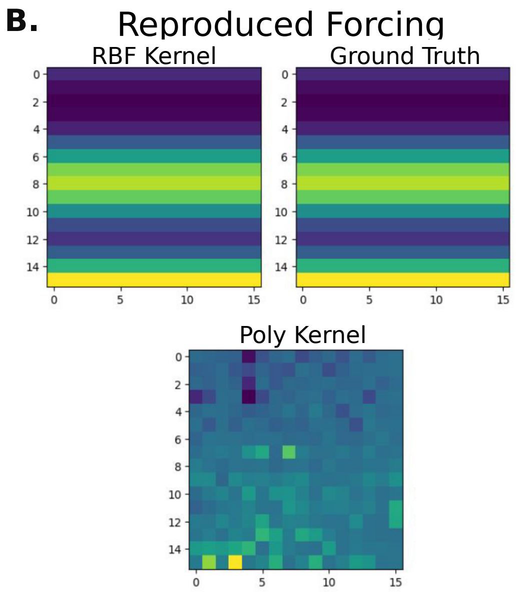


FIGURE 4. **A.** Illustration of simulated Darcy flow produced by using the Fenics library. A 15×15 grid was used for evaluation and collocation. **B.** Ground truth forcing (top right) and forcing predictions using kernel regression with an RBF kernel and polynomial kernel (after cross validation). The L^2 error for the RBF and polynomial kernel predictions shown were 1.016×10^{-12} and 0.887 respectively.

5.4. **Walking Droplet.** For our fourth example, we consider the walking droplet experiment which consists of a droplet bouncing on a vertically vibrating bath of fluid. In our case, the fluid is silicon. The bath is in shallow coral (that may vary in its concavity) which vibrates at some specified acceleration creating Faraday waves in the bath. As the amplitude of acceleration exceeds the empirical values known as the *bouncing threshold* and the *walking threshold*[4], horizontal force is induced by small perturbations caused by the local waves generated by the droplet. This causes a "walking" phenomena, where in certain parameter regimes, walking can occur for an indefinite amount of time. For more information regarding the setup and theoretical investigation of the walking droplet, readers are referred to [9].

In the experiment we consider, there is approximately 4 minutes worth of video data where the position of the droplet is captured every $\Delta t = 1/30$ seconds using the `TrackingWalkers-YOLOv8` drop tracker [4]. Due the chaotic behavior of the system, we aimed for the modest task of learning and recreating the first $t = 5$ seconds of positional data using the kernel pipeline, where the path of the droplet can be seen in figure 5A. This amounts to $J = 150$ evenly-spaced collocation points, $\{x(t_j), y(t_j)\}_{j=1}^J$, to train our model. We additionally assume that we can write the governing system of the droplet in the following form:

$$\begin{cases} \dot{u}_1 = \mathcal{P}_1(u_1, u_2, u_3, u_4) \\ \dot{u}_2 = \mathcal{P}_2(u_1, u_2, u_3, u_4) \\ \dot{u}_3 = \mathcal{P}_3(u_1, u_2, u_3, u_4) \\ \dot{u}_4 = \mathcal{P}_4(u_1, u_2, u_3, u_4) \end{cases} \quad \text{where} \quad \begin{matrix} u_1 := x, \; u_2 := y \\ u_3 := \dot{x}, \; u_4 := \dot{y} \end{matrix} \tag{9}$$

We note for this particular example we do not know the true structure of (9), as we do in the previous examples. Thus in addition to the kernel pipeline, we solve the learned ODE system to measure the success of the kernel model. For the pipeline, we use an RBF kernel to apply kernel

[4]These value are determined by fluid viscosity, depth of coral, and surface tension.

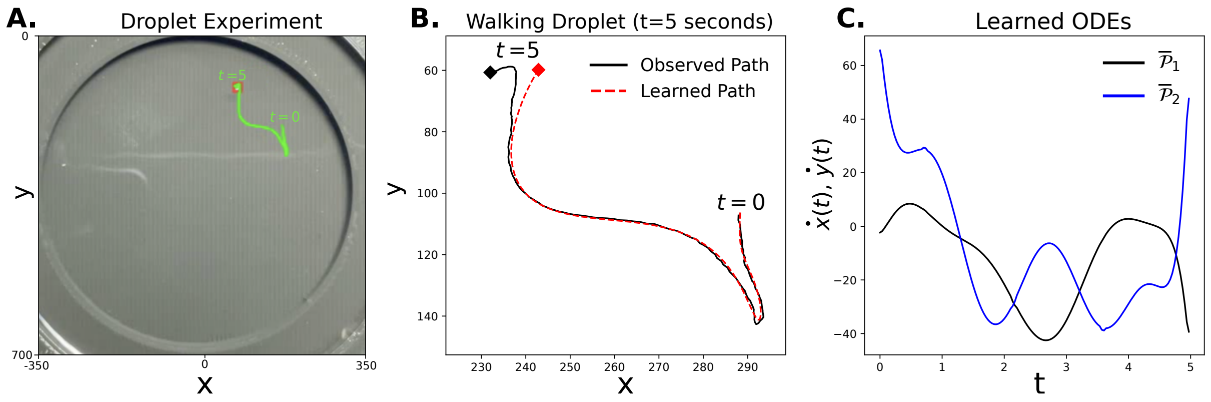


FIGURE 5. **A.** Experimental results of tracking droplet for $t = 5$ seconds where green line denotes path of droplet. **B.** Plotted observed path (black) using `TrackingWalkers-YOLOv8` and the learned path (dashed red) using kernel regression. A animated gif of droplet paths are included here. **C.** The learned ODE for $\dot{x}_1$ (black) and $\dot{x}_2$ (blue) using kernel regression.

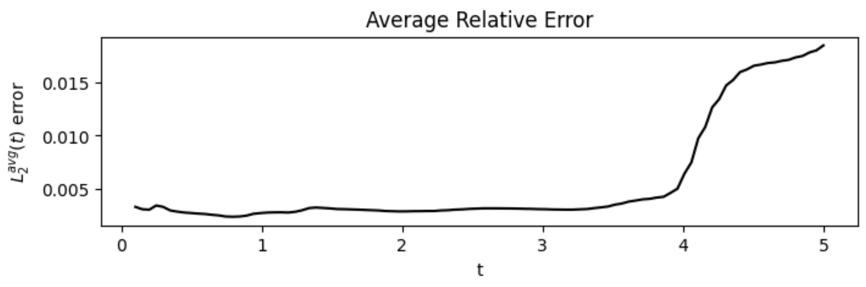


FIGURE 6. The average relative L_2 error of droplet path, up to time t, of both x and y. We define $L_2^{\text{avg}}(t) := \frac{1}{2}(L_2(x^{\text{true}}[:t] - x^{\text{approx}}[:t]) + L_2(y^{\text{true}}[:t] - y^{\text{approx}}[:t]))$.

smoothing to our training data. We then obtain the approximate first and second derivatives for x and y by using a centered difference approximation. We opted not to use our kernel differentiation function to alleviate the task of parameter tuning, and due to the fine mesh of our collocation points, centered difference yields a good second-order approximation to our derivatives. Using an RBF kernel regression we obtain the approximations $\overline{\mathcal{P}}_1$, $\overline{\mathcal{P}}_2$, $\overline{\mathcal{P}}_3$, and $\overline{\mathcal{P}}_4$, and numerically solve the learned system of odes using `scipy.integrate.solve_ivp`, where all kernel hyper-parameters are summarized in Table 5.

We show the learned ODEs for $\overline{\mathcal{P}}_1$ and $\overline{\mathcal{P}}_2$ in Figure 5C and the learned droplet path compared against the observed path in Figure 5B. The learned path matches the observed path very closely for the first 4 seconds, yielding a average relative L_2 error less than 1%, but begins to diverge for $t > 4$, as can be seen in Figure 6. The results suggests the pipeline was successful in capturing the ODE dynamics of the droplet. However, this should be seen as an "unfinished" example with avenues to further explore. As an extension, the authors wish to test the predictive capabilities of the model on the walking droplet and have an exhaustive comparison against SINDy's performance, as is done in the Lorenz example.

6. SUMMARY AND CONCLUSIONS

In summary, we reproduced the work of Long et al. [6] in which a framework for discovery of PDEs from observed data was introduced using kernel smoothing & regression. We validated

	RBF Kernel Smoothing		**RBF Kernel Regression**	
Hyper-parameter	Values Tested	Values Chosen	Values Test	Values Chosen
γ	$[10^{-1}, 10^{-2}]$	10^{-1}	$[10^{-3}, 10^{-4}]$	10^{-4}
$\lambda_{\mathcal{K}}$	$\left[10^{-8}, 10^{-5},\ 10^{-3}\right]$	10^{-5}	$\left[10^{-3}, 10^{-2}, 10^{-1}\right]$	10^{-2}

Table 5. Walking Droplet hyper-parameters selected. Note $\lambda_{\mathcal{U}}$ is not included since a centered difference scheme was used instead of kernel differentiation.

this approach on two examples given in the original work: a 1D pendulum and the Darcy flow PDE and got comparable results to the original work after cross validation of kernel choice and hyper-parameters. Furthermore, we extended the original work by applying to the Lorenz system which can be complex to solve due to the chaotic dynamics it exhibits. We observed comparable performance to the SINDy approach [1]. Next, we applied the pipeline to "walking water droplet" data describing the positional behavior of a droplet over time, measured empirically in a lab. These examples highlighted that despite a rather simple implementation presented in [6] in application, each problem requires a substantial amount of work and familiarity in order to achieve presentable results. Much difficulty was faced with parameter tuning, replicating results, and other choices such as kernel selection and how one must formulate the PDE or ODE system to learn. Overall, we found that this kernel approach for discovering PDEs is effective and relatively simple to implement, making it a promising direction for further exploration, as well a rewarding experience.

Acknowledgements

The authors express their appreciation to Dr. Amin Rahman for providing both the data and insightful discussions regarding the Walking Droplet experiment. Thanks is extended to Dr. Bamdad Hosseini for their support and guidance in understanidng the implementation of the outlined pipeline. Additionally, the authors would like to acknowledge the support of their fellow colleagues in Amath, Alex Hsu and Juan Felipe Osorio Ramírez. Alex Hsu's assistance in solving the Darcy Flow PDE example and recommending of the utilization of `fenics` and Juan Felipe Osorio Ramírez's contribution to the pipeline implementation and assistance in theoretical understanding are greatly appreciated.

References

[1] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, Mar. 2016.
[2] Y. Chen, B. Hosseini, H. Owhadi, and A. M. Stuart. Solving and learning nonlinear pdes with gaussian processes, 2021.
[3] I. Griss Salas, S. Ichinaga, and J. Hazelden, 2023. `https://github.com/sichinaga/amath563_project`.
[4] E. Kara. Real-time droplet tracking with YOLOv8.
[5] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations, 2021.
[6] D. Long, N. Mrvaljevic, S. Zhe, and B. Hosseini. A kernel approach for pde discovery and operator learning. Mar. 2023.
[7] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, mar 2021.
[8] H. Owhadi and C. Scovel. *Operator-Adapted Wavelets, Fast Solvers, and Numerical Homogenization: From a Game Theoretic Approach to Numerical Approximation and Algorithm Design.* Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2019.
[9] A. U. Oza, R. R. Rosales, and J. W. M. Bush. A trajectory equation for walking droplets: hydrodynamic pilot-wave theory. *Journal of Fluid Mechanics*, 737:552–570, 2013.