# Open-Source Dynamical Systems Research, with a Side of (Francis) Bacon

Jacob Stevens-Haas

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2025

Reading Committee:
Aleksandr Aravkin, Chair
Nathan Kutz
Bamdad Hosseini

Program Authorized to Offer Degree:
Applied Mathematics

University of Washington

**Abstract**

Open-Source Dynamical Systems Research, with a Side of (Francis) Bacon

Jacob Stevens-Haas

Chair of the Supervisory Committee:
Aleksandr Aravkin
Applied Mathematics

Sparse Identification of Nonlinear dynamics (SINDy) is a family of methods for explicitly identifying differential equations from data. The open-source Python package pysindy provides the engineering to support ongoing SINDy research.

I discuss original and community innovations in smoothing and sparse optimization, as well as a colocation approach to simultaneous estimation of states and sparse coefficients. The compatibility of these methods through the pysindy API has lessons for mathematics as an experimental field.

My contribution to the state of the art includes both original innovations and ongoing support for research contributions from the community. These innovations include smoothing methods such as kernel and Kalman, and sparse regression approaches involving Monte Carlo estimation, physics constraints, or mixed-integer optimization. The pysindy changes have also allowed a principled approach to simultaneous optimization of states and coefficients. Across these projects and more, the requirement for a consistent API has given rise to a common experimental language. This defense codifies that language in additional pacakges and suggests useful lessons for the open-source, numerical lab.

# TABLE OF CONTENTS

# LIST OF FIGURES

Figure Number Page

# ACKNOWLEDGMENTS

I would like to thank my advisors, Sasha Aravkin and Nathan Kutz, for all the guidance, support, and many engaging discussions over the course of my PhD. I am also grateful for the input of my committee members, who have helped shape this dissertation over the past few months. I would also like to thank all the friends, family, loved ones, and various support groups who have guided me through the past six years in everything non-academic. I would like to extend my thanks to the open source community, particularly those engaged with pysindy. This includes reveiewers like Alan Kaptanoglu and Zach Nicolau, predecessors such as Brian DeSilva, Markus Quade, contributors such as Wes Gurnee, Mikkel Bukke, and Ludger Paehler, preceding contributors like Jared Callaham, and obviously my collaborators, Ike, Alan, Mai, Alex H., Juan, Megan, Yash, Watcharin.

And the many users who raised github issues; as iron sharpens iron.

# Chapter 1

# INTRODUCTION

The rapid growth of measurement data from a wide variety of sensors has spawned interest in understanding more and more systems. Since the time of Newton, understanding a time-varying system has meant knowing and understanding a differential equation that governed the behavior of the system. To that end, scientists have developed various data-driven methods to estimate a differential operator. Methods include time series neural networks such as Physics-Informed Neural Networks and Neural Operators, the linear rank-reduction Dynamic Mode Decomposition (DMD), and direct regression on candidate functions, often encouraging sparsity. (as titled, Sparse Identification of Nonlinear Dynamics, or SINDy). Innovations of these categories multiply the number of variations, with each variation aimed at some class of subproblem or experiment. Of these categories, only direct regression on candidate functions provides what is conventionally recognizable as an ordinary or partial differential equation (ODE or PDE). Focusing, then, on SINDy, we are faced with two opportunities: Resolving the incompatilibilty between myriad innovations can open open up hitherto unresearched combinations. Secondarily, we can erase the disntinction between derivative estimation and sparse regression, allowing a more general approach. This dissertation approaches a solution to the latter, and along the way, builds the pysindy Python package into an open-source SINDy lab.

SINDy [15] is a family of emerging methods for discovering the underlying dynamics of a system governed by unknown differential equations. In pursuit of Occam's principle, it seeks to find a sparse expression for an autonomous differential equation

$$\dot{x} = \xi^T \theta(x),$$

where the coefficients $\xi \in \mathbb{R}^{n \times m}$, and $\theta = \theta_1, \ldots, \theta_m$. It can handle ODEs as well as PDEs [81] and has been used for chemical reaction networks [38], plasma physics [35], and more. Higher order differential equations can be trivially reorganized by letting $x = [y^(n-1), \ldots, y]^T$. The method generally proceeds in two steps:

(a) Estimate the time derivatives of the system.

(b) Choosing a sparse regression method, solve the regression for coefficients $\xi$.

The default method of finite-difference derivatives and Least Absolute Shrinkage and Selection Operator (LASSO) regression fails in cases with bad measurements, an incomplete function library, nonstationarity, or other more systemic issues. Innovations to SINDy attempt to address these and other problems by providing an alternate derivative or sparse regression method.

The two-step prescription was derived was a clear and practical approach where none previously existed. However, the most promising innovations, in this author's opinion, break the model's two-step prescription. Since derivative estimation (and equivalently, smoothing) and SINDy regression are each phrased as optimization objectives, it is the most natural thing to try to combine them.

Most invocations of SINDy occur through the `pysindy` Python package, but innovations such as Langevin Regression [16] or [80] exist as independent code. At the risk of simplification, the SINDy model object consists of a differentiation method, to estimate $\dot{x}$, a function library, specifying $\theta$, and a method for identifying sparse $\xi$. The associated repository was built by Brian DeSilva from an initial version of Markus Quade's. Since then, the code has been under stewardship of Zach Nicolau, Alan Kaptanoglu, and ultimately, myself. It served a dual purpose: to provide a working copy of people's innovations under a consistent Application Programming Interface (API), and to showcase experiments developed in related papers. As the package grew, dependencies sprung up between the previously clear abstractions. Improvements became blocked by onerous backwards compatibility requirements to support old research. The upshot of which was that SINDy was neither an effective support for applied or method research. I have been the primary maintainer for the last two years,

answering a few questions and bug reports per week from online users.

We take stock of emerging SINDy methodology and the existing pysindy package, improve the project, support related external contributions, add to the ecosystem, and introduce a new variant: Single-Step SINDy. The subsequent section of this chapter describes the SINDy method in more technical detail. It is followed by a chapter about my development of the pysindy package. The purpose is to clarify why the API matters and what engineering support is required for the collaborative efforts of the successive chapter. This chapter explains recent innovations of pysindy's regression that illustrate tradeoffs of different approaches to collaboration. My contribution in each progresses from reviewer to co-author. Moreover they demonstrate how an open source research lab can accomodate contributions from distant collaborators in various ways. The fourth chapter introduces the most significant innovation of SINDy during this dissertation: combining the derivative estimation step and sparse regression step. This begins by introducing Kalman smoothing as a differentiation method, generalizes the class of smoothers to kernel methods, then combines the smoothing and SINDy losses. From there, the focus returns to engineering. The fourth chapter explains the concept of the open source lab in greater detail, as well as the supporting contributions to pysindy, the style guide for research code, and the supporting packages in the pysindy ecosystem. Chief among them is the experiment management tool, mitosis. From all these chapters, the moral emerges that engineering is not mere implementation detail, but rather its own form of peer math communication and lab infrastructure.

### *1.1 SINDy background*

This section presents SINDy in more technical detail reviews the variants and use cases, and categorizes the failure modes it experiences in certain problems.

SINDy seeks to find a parsimonious expression for an autonomous differential equation.

$$\dot{x} = \xi^T \theta(x),$$

where the coefficients $\xi \in \mathbb{R}^{n\times m}$ and $\theta = \theta_1, \ldots, \theta_m$. This sparsity is an instance of Occam's principle: if one posits a large number of potential functions, only a few are present in the true dynamics.

When discretizing the timepoints $t_i$ $i \in \{1, \ldots, p\}$, the capital letter is used:

$$X = \begin{bmatrix} x_1(t_1) & x_1(t_2) & \ldots & x_1(t_p) \\ x_2(t_1) & x_2(t_2) & \ldots & x_2(t_p) \\ \vdots & \vdots & \ddots & \vdots \\ x_m(t_1) & x_m(t_2) & \ldots & x_m(t_p) \end{bmatrix} \in \mathbb{R}^{m\times p}.$$

Likewise, $\dot{X} \in \mathbb{R}^{m\times p}$ and $\Theta(X)$, which is $\Theta$ applied to each timepoint successively, is $\in \mathbb{R}^{p\times n}$.

Given some variable of interest $X$ and a library of functions $\Theta$ (including spatial derivatives, when relevant) SINDy seeks to find the coefficients $\Xi$ of the differential equation:

$$\dot{X} = \Xi\Theta(X), \tag{1.1}$$

where

$X \in \mathbb{R}^{n\times m} = \vec{x}(t_1)...\vec{x}(t_m)$: system of $n$ coordinates at $m$ timepoints.

$\Theta(X) \in \mathbb{R}^{p\times m}$: library of $p$ functions evaluated at $m$ timepoints

$\Xi \in \mathbb{R}^{n\times p}$: coefficients for $n$ equations of $p$ functions

The function library, written as a time-independent quantity, refers to the collection $\Theta = [\theta_1, \ldots \theta_p]^T$, where $\theta_i : \mathbb{R}^n \to \mathbb{R}$. Examples include the family of all degree-2 polynomials of $n$ inputs, mixed sines and cosines of certain frequencies, or any user-specified family. The method generally presumes the measurements ($Z$) faithfully reflect system state ($X$) and proceeds in two steps:

(a) Estimate the time derivatives of the system $\widehat{\dot{X}} = \mathrm{F}(Z)$ for some smoothing function F.

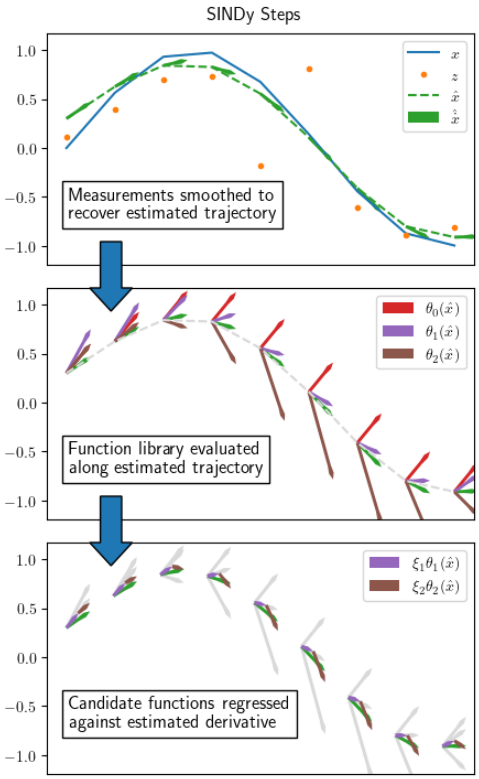



Figure 1.1: The SINDy method, for example, applied to fitting a sinusoid. It takes noisy data and smoothes it in the first panel, using the smoothed values to evaluate a library of functions at each point (second panel). Finally, it identifies the model $\dot{x} = \xi_1\theta_1(x) + \xi_2\theta_2(x)$, rejecting $\theta_0$.

(b) Choosing a sparse regression method, estimate the dynamics by solving the problem $\underset{\text{sparse } \Xi}{\arg\min} \left\| \dot{\widehat{X}} - \Xi\Theta(X) \right\|^2$.

This general process is sketched out in a representative scenario in Fig. 1.1. Each step described there has various innovations and options.

### *1.1.1 On what problems have people tried SINDy?*

The method can discover a variety of governing ODEs with [15] deomnstrating how it can identify the chaotic Lorenz attractor, low-dimensional, vortex-shedding modes of the Navier-stokes equation, and the Hopf normal form. Beyond these demonstrations, researchers use

SINDy to learn about the physical system but‘’ also as a useful engineering component. In many cases, problems have high system dimension. Such physical systems include learning chemical reaction networks such as yeast glycolosis and of Michaelis-Menten enzyme kinetics in [62], or gene regulatory networks such as mitogen-activated protein kinesis of [38]. In addition, low-dimensional representations of high-dimensional phenomena is its own problem class in dynamical system learning. This includes innately physical systems where the low dimensional system is a macroscopic observable such as hydrodynamic equations for ensembles of active matter, schools of fish, or autonomous agents in [76] or diatomic distances and dihedral angles as a low dimensional representation of atomic coordinates of large molecules as in [10]. Even more common is the reduced dimension problem of discovering a low dimensional ODE for the galerkin projection of a system evolving via a PDE. Cases include Navier stokes type problems of [51, 58, 69] atmospheric photochemical models in [90] and plasma convection of [24]. The latter showed how discovered Galerkin ODEs preserved the bifurcation and stability behavior of the original PDE. Reduced-order ODE/PDEs is a physics and engineering problem that extends beyond SINDy, and often seeks to find ways to resolve fine-scale features from coarse-grained models, such as eddies in global ocean circulation models. Finally, SINDy has use as a pure engineering component, such as how [93] uses it to accelerate a reinforcement learning agent's understanding of the environment.

### *1.1.2 A Taxonomy of Errors*

When SINDy has low measurement noise, is measured in the right coordinates, uses a complete feature library, and the ODE is, in fact, autonomous, the only thing that can cause SINDy to fail is bad numerics. But each of these idealized conditions can be violated in interesting ways, motivating more robust approaches along each violation.

Bad measurement degrades SINDy, particularly when measurements are taken to be accurate and derivatives estimated using finite difference. But "noise" encompasses a broad class of problems: at the simplest, measurement error makes it harder to estimate derivatives. Unsmoothed measurements are even more consequential inside the feature library, which,

because of nonlinearities, can have an arbitrarily bad effect on the regression. Another type of noise is the presence of unmodeled forcing. Finally, the last type of measurement malignancy is an insufficient sampling rate. The analog in linear systems is the Nyquist rate, but sampling location for nonlinear ODEs and PDEs has the added complexity of needing to deal with shocks boundary layers, where we need sufficient observations for terms that are elsewhere very small. All three issus are demonstrated in figure 1.2. Smoothing, ensembling, and WeakSINDy attempt to handle noise problems in SINDy. This doctoral work added several smoothing/differentiation approaches from the literature to pysindy. Handling measurement noise is in large part the role of chapter 4.

A bad feature library can have a substantial effect on the results of applying SINDy. The most obvious pathology is omitting certain features. And as we see in figure 1.3, fitting a third-degree polynomial with only second-degree SINDy terms fails to even recover the correct terms to second degree. But perhaps more subtly interesting is non-orthogonality. While orthogonal families of functions exist, measurement points rarely coincide with quadrature points. Orthogonality is what allows sparse regression to distinguish between coordinates, as described by the restricted isometry property in [17]. Consider the case of the pendulum in figure 1.3. Determining whether the system is linear or sinusoidal requires observing a large enough deflection. Even still, the problem of orthogonality makes it difficult to distinguish between $\sin\phi$ and its first two Taylor series terms.

In theory, choosing near-orthogonal basis features on the domain of sampling should be feasible, but doing so throws out the motivation that, in the appropriate basis, the dynamics are sparse. There is, however, no reason to believe that such is an orthogonal basis. Moreover, the question of function basis also applies to system coordinate basis. At its simplest, missing a coordinate may impair the ability to recover simulation dynamics, as stable attractors may not exist when a coordinate is lost. At worst, the regression picks up a variety of incorrect terms to compensate.

Several other issues can exist with coordinate system beyond simply missingness or not being the coordinate system in which the dynamics are sparse. Firstly, the size of the

feature library grows combinatorially with number of coordinates. The sparse regression part of SINDy then often fails to discover accurate information, even in the linear case Secondly, the coordinates may fail to capture enough derivatives of the system, e.g. fitting a third order ODE may fail if first and second derivatives are not estimated. Finally, and most challengingly, dynamics could occur on some manifold due to conservation of energy. Consider

$$E_0 = mgx_0 - 1/2\dot{x}_0^2$$

Here, there is a constant term, $E_0$, a linear term in $x$, and a quadratic term in $\dot{x}$. If one were to fit $\ddot{x}$ as a degree-two polynomial of $x$ and $\dot{x}$, features would be exactly collinear. Some apporaches to identify this *a priori* involve fitting an autoencoder as in [32] or checking the dimension of the hankel embedding. The time-delay coordinates of the hankel embedding have seen renewed interest in combination with pysindy, explored in [48, 77].

Finally, the last category of failure mode is when the modeled system does not obey the structure of the SINDy model. This category includes situations where the governing dynamics are inseparable in time and space, delay differential terms, stochastic uncertainty, or nonstationarity. Some atttempts have been made to codify a stochastic SINDy, such as [10, 16]. Other attemps tried to standardize the discovery of smoothly changing coefficients over time, e.g. [89].

### *1.1.3 "SINDy" spans a wide breadth of innovations*

In an attempt to improve the robustness of SINDy to the above problems, researchers have developed innovations to the methods of estimating derivatives, performing sparse regression, and reformulating as an integral equation.

Researchers have tried a few different methods for calculating the derivatives, broadly grouped into global methods (e.g. L-1 total variation minimization of [21]) and local methods (e.g. Savitzky-Golay smoothing). However, [12] suggests that the hyperparameters in each method share a common Pareto frontier.

The feature library is a less complicated piece of SINDy. For ambiguously controlled or forced systems, [47] describes how to fit SINDy on an additional feature library for those terms. The only challenge is when estimating spatial derivative features; [81] discusses SINDy with PDE features. While most differentiation methods for time might be used for spatial terms, some of the former only imply smoothness up to first order. An alternate approach, via the integral methods of [64, 65], is to try to fit the weak form of a PDE.

Different ways of applying sparsity has attracted more attention, including sequentially thresholding linear regression, nonconvex penalties such as L-0 with a relaxation-based minimization method [19, 92], an L-0 constraint [7], and Bayesian methods for a prior distribution such as spike-slab or regularized horseshoe priors [32, 37]. The latter two papers also demonstrate an interesting line of innovation, eschewing derivatives and using the integral of function library in the loss term. A related approach instead uses the weak form of the differential equation. Most of these methods can benefit from ensembling the data and library terms, as in [29], but others, such as [51, 69], for identifying Galerkin modes of globally stable fluid flows, require a specific form of function library.

The diversity of approaches to SINDy can be challenging to navigate. Many, however, exist as part of a python package, pysindy, which is discussed in the next chapter.

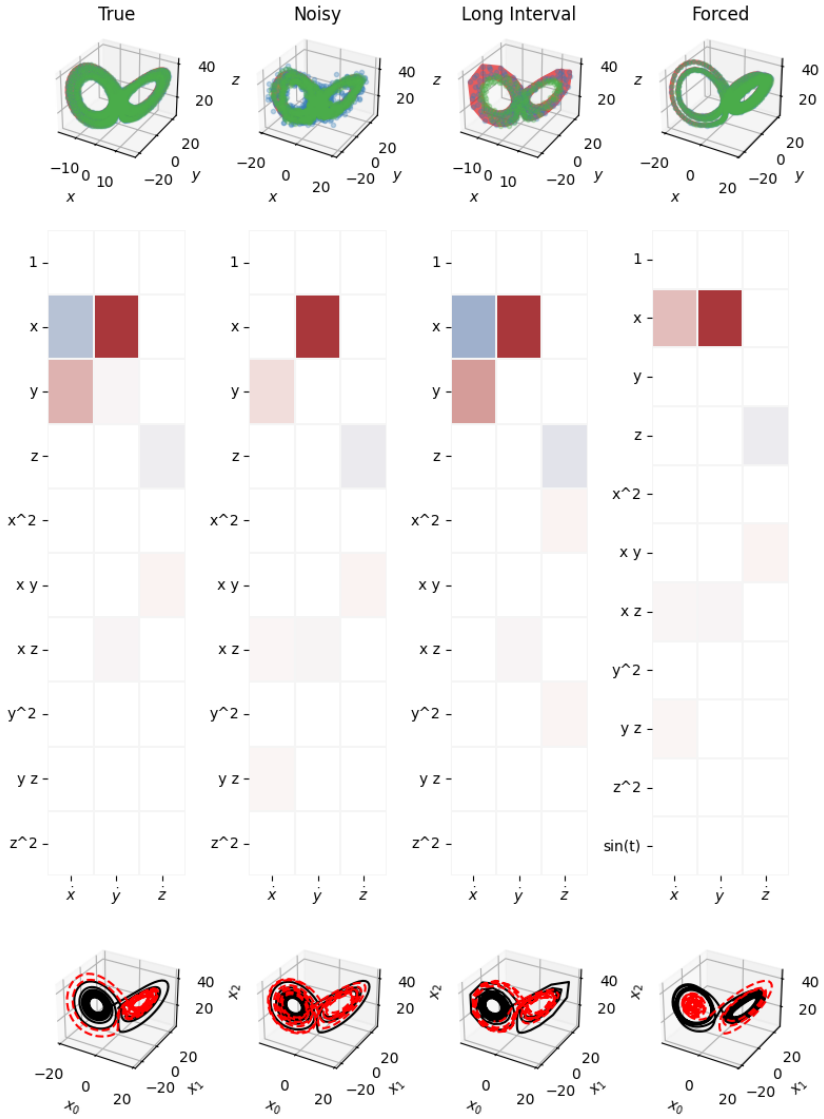



Figure 1.2: Measurement issues and noise degrades SINDy. On the left panel, a forced lorenz system is observed with sparse measurements and minimal measurement noise. The top plot includes the measurements, true value of the system, and smoothing. The middle plot shows that SINDy recovers the terms in the differential equation nearly exactly. The bottom plot demonstrates the fitted model in simulation; Due to the chaotic nature of the Lorenz system, small model errors mean that the model departs the true trajectory but remains on the stable attractor. Progressing rightward, the second column demonstrates the same problem but with noisier measurements The third column shows noiseless measurements but with a longer sampling interval. Finally, the last column shows noiseless, fast observations of a lorenz system experiencing sinusoidal forcing. While all panels after the first fail to recover even the linear coefficients correctly, each model discovers roughly the same stable attractor, even if the simulations vary widely in quality.

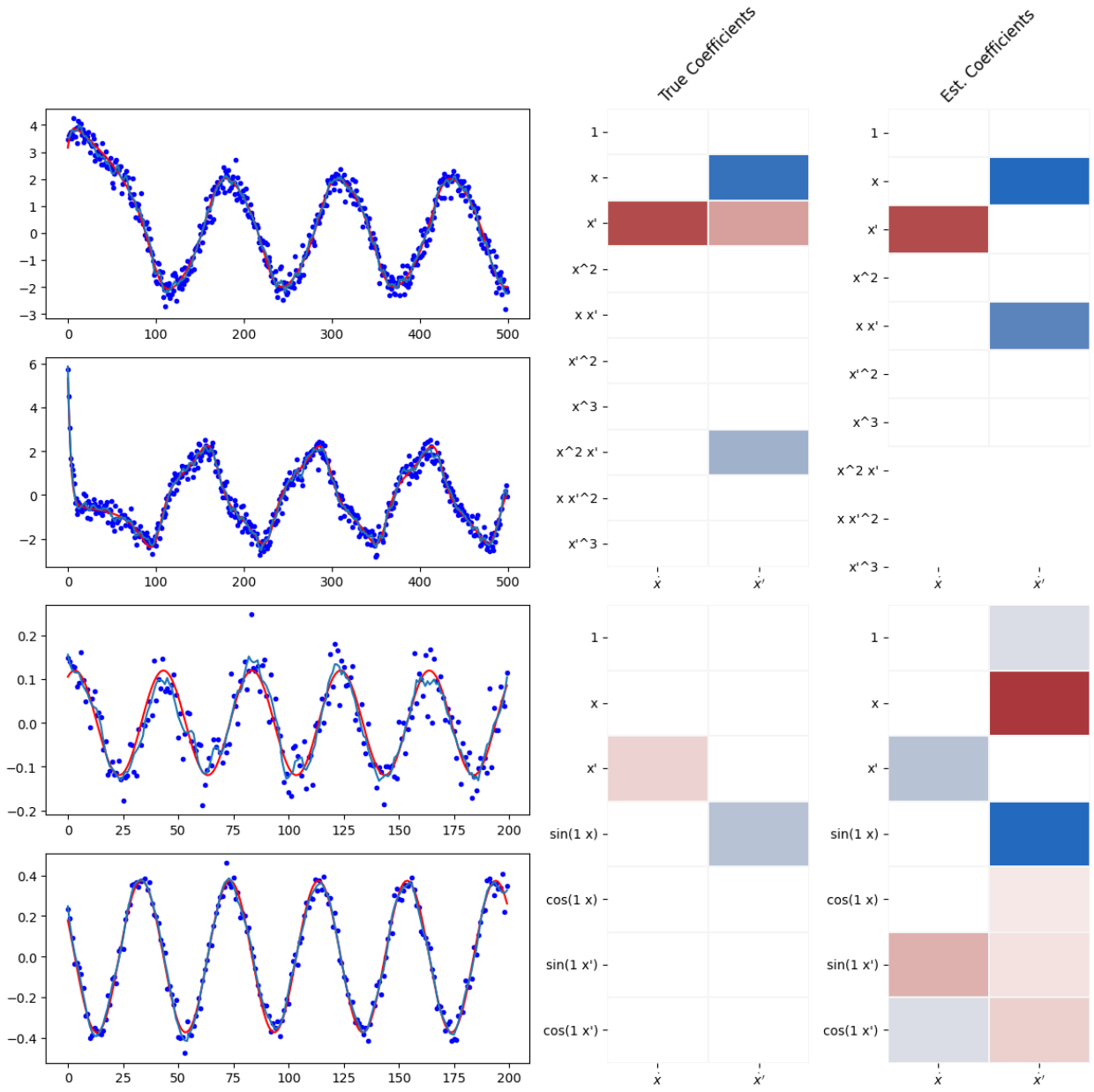



Figure 1.3: Even with good measurements, bad feature libraries can cause SINDy to struggle. In the top-left half of the figure, low-noise measurements are taken of a Van der Pol oscillator, then smoothed as in the first step of SINDy. Two nearly-identical SINDy models, one with a cubic polynomial library and one with a quadratic library are fit on the data. SINDy with a cubic library discovers all of the coefficients correctly. SINDy with a quadratic library obviously cannot discover the cubic coefficient, but this omission also causes it to incorrectly recover linear and quadratic features. In the bottom half, the system is a nonlinear oscillator perturbed slightly from the origin. The coefficients on the left show the true system, and the coefficients on the right show SINDy model recovery when given both Fourier and polynomial features. Because $\sin x$ and $x$ are nearly collinear near the origin, the sparse optimizer struggles to find a meaningful solution at all. Not shown: when the SINDy model is restricted to just polynomial terms, it discovers a good linear approximation of the nonlinear oscillator.

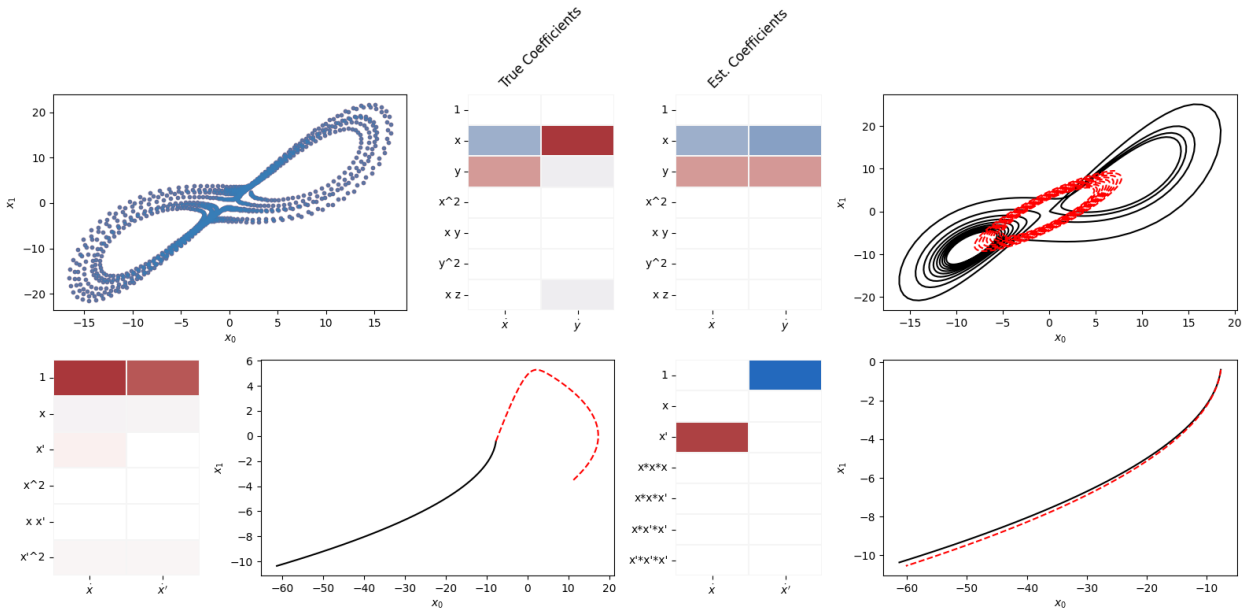



Figure 1.4: The measurement coordinates of a system can be a challenging aspect of SINDy. The top row shows a training, coefficient recovery, and simulation of a SINDy model trained on the Lorenz system with the $z$ coordinate missing. SINDy is unable to even discover the strongest linear term. A different problem can occur when SINDy is trained correctly, but the system obeys a conservation law and that conserved quantity can be formed by a linear combination of the features. This is the kinematic system of a particle in one dimension undergoing constant acceleration. The left shows a SINDy model with quadratic features, coefficients and simulation. The constant term, the linear position term, and the velocity squared term are rank-deficient columns because the trajectory obeys conservation of energy. To the right, when the quadratic features are replaced by cubic features, no linear combination of the features expresses conservation of energy. Not shown: This can also be achieved by removing the constant term from the quadratic library. When solutions like on a manifold, it is necessary to train on multiple trajectories with different energy levels.

# Chapter 2

# PYSINDY ENGINEERING

pysindy is a Python library, but it also represents a network of collaborators, contributions, and tools. It is widely used, but as academic software, it faces some common challenges. This chapter shows the complexity and breadth of the pysindy package, the software contributions of this dissertation to maintain it, and how those decisions support long-term growth of related research.

The pysindy project is a central part of my doctoral contribution. The pysindy code is the most identifiable part of the project, and consists at a minimum of answering bug reports and feature requests and incorporating contributions from the community. The community itself comprises current UW members, former UW members, and interested researchers at other academic institutions and corporations. There are more components to this lab, e.g. the derivative project ran by Andy Goldschmidt, the pysindy-experiments package of benchmarks, the documentation, including all the examples contributed by researchers. But such factors beyond the main codebase are left to the end of the paper.

As a part of the relevant scientific field, pysindy is widely used beyond collaborators. Several dozen copies per day are downloaded, excluding linux (according to PyPIStats). [1] In the past two and a half years, the community has posed 213 questions and issues, and unaffiliated researchers pull-requested two major pieces of functionality, described in chapter 3. In particular, the Kalman method in section 4.1 began appearing in pysindy questsions before it was submitted for publication. Nearly a dozen researchers have published their paper's experiments using pysindy. Clearly, the existence of the project, as opposed to

[1] Linux downloads are unusually high (85-90%), which may include CI of other projects and online notebook servers. Note that this number does not include conda downloads., an a total monthly download rate of 94,0685(as of Nov 2024).

independent reimplementation for each new paper, facilitates communication in the field. The project serves as a medium for disseminating research in a similar way as a journal.

Historically, pysindy has served two use cases: as a place to facilitate development of new pysindy methods and as a toolbox for users with their own scientific and engineering problems. Connecting these two encourages citations and uptake in the field. However, For the former, pysindy is a tool and benefits most from reliability and documentation. For the latter, pysindy is akin to a journal with its own review. Each group contributes and causes issues according to its motivations: the former group desries reliability and documentation, but reports research questions as bugs. the latter contriubtes methods, viewing package inclusion as a form of publication, but whose interest in rapidity mixes experimental code with library code, building technical debt. The maintainers' interest in reliability is a form of editorial review, and is encumbered in similar ways. Decisions to simplify the package may affect the API, limiting backwards compatibility with previously published experiments. Navigating the balance of needs is a challenge.

Thus, what is critical to introduce in this chapter are the code and the contributions accessory to this thesis's mathematical work, but critical to the larger contribution. This hopefully allows the reader to appreciate design decisions in subsequent sections. These design decisions are putatively in the purview of engineering, yet seriously affect the correctness or clarity of the math underlying use cases. Such considerations would not matter for an author writing the code for themselves. The rest of the lab management - in particular, experimentation - is left for a later chapter.

## *2.1 API*

This section provides a basic introduction to the pysindy API. It introduces the classes and functions that can be combined to make models to discover a wide variety of dynamical systems. Pysindy provides a single model object, SINDy, representing a particular approach to learn dynamics of an unknown system. It derives from scikit-learn's BaseEstimator, thus it has fit(), predict(), and score() methods. It as well has simulate() and print() methods.

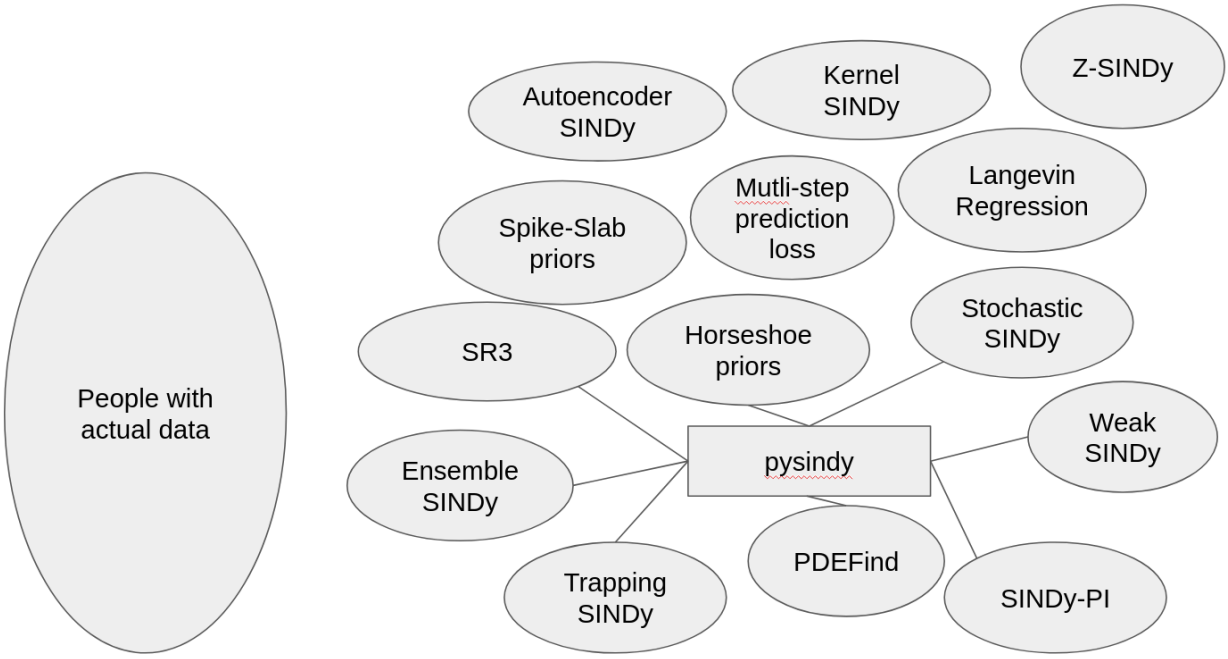



Figure 2.1: Methods research exists within and without pysindy. Development tends to be the responsiblity of authors on the right hand side. Such authors have deeper knowledge of the package than users on the left, allowing them to avoid the more difficult or buggy parts. Other authors contributions remain outside pysindy. Applications researchers, on the left, evaluate SINDy as a whole based upon the API's reliability, however.

As is common, all methods other than initialization and fit() require fit() to have been run, According to the scikit-learn API, it requires the data independent choices be made at initialization and the data-dependent ones be made at fitting. The pysindy model SINDy class comprises: (a) a differentiation/smoothing object that acts as $F$ in step 1, (b) a feature library that specifies the family of functions $\Theta$, and (c) an optimizer that specifies how to perform the sparse regression in step 2.

Differentiation methods inherit BaseDifferentiation, which requires innovations to specify the axis along which to differentiate at initialization and the units along the axis of differentiation. The default differentiation methods is 'FiniteDifference'.

The feature libraries inherit BaseFeatureLibrary. They are fit() to determine the to-

tal number of output functions for a particular number of input coordinates, or system dimension, then transform() applies the output functions to all the inputs. They also are responsible for the string representation of the functions, and can be concatenated or tensored with the builtin +/* operators.

While all previous objects inherit from sklearn.BaseEstimator, the BaseOptimizer inherits from sklearn.LinearRegression. The base class here handles some regression admin, such as normalizing the columns, unbiasing the regression, and most importantly, storing the saved coefficients after being fit(). The fit() method provides a hook for subclass customization in _reduce(). The default implementation is Sequentially-Thresholded Least-Squares (STLSQ).

This simple combination of components can be demonstrated in a few lines, as shown in listing 2.1.

```
import pysindy as ps

diff_spec = ps.SINDyDerivative(kind="kalman")
f_lib = ps.PDELibrary(order=2) + ps.PolynomialLibrary(degree=3)
opt = ps.STLSQ(alpha=0.0)
model = ps.SINDy(
    differntiation_method=diff_spec,
    function_library=f_lib,
    optimizer=opt
)

# consider u, t having been generated or supplied elsewhere
model.fit(u, t)
```

```
model.print()
```

Listing 2.1: A mock up of a pysindy model. It includes a differentiation method, a feature library, and an optimizer.

The API is an abstraction between all the different research variants of SINDy on one hand, and use cases on the other. Innovations from nearly twenty research papers have been implemented as subclasses of the three main pysindy constituent objects, as described in table 2.1. Combining compatible API elements allows hundreds of variations of SINDy, many of them never evaluated in depth. Such combinations occur in uses cases across science and engineering. Scientifically, a compatible API allows evaluating different variations of SINDy in the same experiment, or choosing the method to investigate and the experimental data and evaluation separately. These same benefits accrue to engineering applications, when SINDy is a component of a larger controls system, as in [93]. Moreover, isolating research to the narrow differences it has with other innovations helps communicate what is new.

Although cleanly separated in this section, the API's simplicity is deceptive. Ideally, the API would make it easy to do anything simple, and illegal to do anything illegal. For math, this means that when components or methods that do not imply contradictory assumptions the code objects representing them should be compatible. When components or methods do imply contradictory assumptions, their use should be prohibited, ideally by the type system. Questions to the pysindy github repository suggest that users holds this belief very strongly, and are very confused when math abstractions are leaky.

**Guiding principle** The type system of a math library should enforce mathematically compatible assumptions.

In other words, Any combinations of SINDy arguments that does not result in an error, must be mathematically interesting, and not merely an incompatibility hid in confusion. The converse also represents an ideal: any variant of SINDy that combines methods available in pysindy should be possible to create in pysindy. Only the most minor deviations of this rule

| From paper to codebase | | |
|---|---|---|
| Innovation | API element | Reference |
| Spectral Derivative | *none* | *none* |
| Total Variation Smoothing/Derivative | derivative package TrendFiltered | [21] |
| Spline S/D | derivative package Spline | [21] |
| Savitzky-Golay S/D | derivative package SavitzkyGolay or pysindy's SmoothedFiniteDifference | [21] |
| Kalman S/D | derivative package Kalman | [88] |
| Radial Basis Function Kernel Smoothing | derivative package Kernel | [60] |
| PDEFind initialization and fit() require fit() to have been run, | feature library PDELibrary | [81] |
| Weak SINDy | feature library WeakPDELibrary | [64, 65] |
| SINDyPI | SINDyPILibrary (deprecated) SINDyPIOptimizer | [45] |
| SINDyCP | feature library ParametrizedLibrary | [67] |
| SINDy-E | optimizer EnsembleOptimizer | [29] |
| Forward Regression Orthogonal Least-Squares (FROLS) | optimizer FROLS | [8] |
| Stepwise Sparse Regression (SSR) | optimizer SSR | [10] |
| Sparse Relaxed Regularized Regression (SR3) | optimizers SR3, ConstrainedSR3 | [19, 92] |
| Mixed Integer Optimization, Sparse Regression (MIOSR) | optimizer MIOSR | [7] |
| Sparse Bayesian Regression (SBR) | optimizer SBR | [32, 37] |
| Trapping | optimizer TrappingSR3 | [51] |
| Stable Linear SR3 | optimizer StabilizedLinearSR3 | *none* |

Table 2.1: Research papers and efforts lead to innovations of base classes in pysindy. But implementing all the innovations in a consistent API is far from problem-free. Note that impelmentations in the derivative package are accessed through the SINDyDerivative wrapper class. SINDyPI requires additional settings.

are allowed. [2]

However, certain combinations of differentiation and feature library or feature library and optimizer are mathematically nonsensical and will execute without error. Similarly, some combinations of these objects and SINDy arguments break functionality, such as being able to simulate or print equations. On top of it all, more pedestrian bugs occasionally occur. Working to improve the code base: its maintainability, reliability, and conceptual integrity has occurred in parallel with development of new methods. The following section explains such contributions in greater detail.

### *2.2 Maintenance and Improvement*

This doctoral effort has dramatically improved the style and design of the pysindy codebase in order to support continued growth and maintenance of the lab's research. External users have a smaller, safer API with more capability. Maintainers and developers have cleaner abstractions and faster and safer tests. But it was not always the case; much work remains. This section describes the broad categories of obstacles to such a happy state, explains the consequences for the lab until the problems were fixed, and suggests general solutions to these problems. Implementation of additional functionality, added by pull request in chapters 3 and 4, must fit within the design principles discussed here, among others.

The most challenging bugs are the ones that occur when someone reuses an abstraction for something tha todes not quite fit, breaking the abstraction, but not, perhaps, any concretions that may occur in general use. Most commonly this occurs when a developer adds their method to pysindy by subclassing the smallest possible component that seems to make sense. These kinds of errors have additional problems in math code, as abstractions can quietly contain assumptions which, when ignored, can allow a developer to violate the above guiding principle. Table 2.2 describes the problem with several examples among types of

[2] Some deviations are unavoidable: for example, math code frequently requires hyperparameters to be "non-negative floats", which is not representable by the type system as anything other than a "float". The exception applies more to custom types.

differentiators, feature libraries, and optimizers. Errata of this kind can be sensed when seemingly separate modules or classes involve a substantial amount of coupling. They can also make new features more difficult to implement. As a case in point, the refactoring in #105 was required so as to fix another bug: When a differentiation method also smoothed the trajectory, the smooth trajectory was merely being discarded, and the noisy trajectory used in the feature library $\Theta(X)$. As is typical in these problems, the solution required introducing a new level of abstraction that unblocked a variety of improvements. Fixing that bug improved the performance of nearly every SINDy method on noisy data.

Other pysindy bugs were more pedestrian, yet had mathematical consequences: either a computation was done incorrectly, or some valid math raised an error, preventing people from using that method. Such bug fixes are generally local and individual, but systemic improvements in style and testing make them much rarer. The ultimate guard against bugs, however, is peer review. As [74] points out, all bugs are shallow with enough eyes. Better review of changes enforces a more understandable coding style, and a more understandable style makes intent clearer. As a case in point, the implementations of SR3 and SSR both introduced bugs that existed for several years. Conversely, the pull request for SBR included peer review and a deep comparison between the source code, documentation, and relevant papers, catching several bugs before the code went live. Some instructive bugfixes are listed in table 2.3

The final category of improvements this thesis has included are those which make the code more maintainable. This can involve making the style more comprehendable or improving the code infrastructure. Problems of comprehension, beyond bad abstractions, can often be solved by thinking of the code as a form of communication. Good style in coding makes wrong mistakes more obvious [40]. Consider this code which takes the derivative of an array (a custom numpy subclass AxesArray) of data:

```
>>> du_dt = derivative(u, axis=0)
```

Is it correct? Consider the same thing, written differently:

| Bad abstractions, high coupling | | |
|---|---|---|
| Problem | Description | Fixed in |
| Generalizing $\dot{u}$ to LHS | Weak SINDy was introduced as a feature library, WeakPDELibrary. However, Weak SINDy also changes the form of the left hand side of the ODE, and the SINDy class needed to ignore the differentiation method when combined with WeakPDELibrary Combining weak and nonweak feature libraries needed to be prohibited, and prediction using weak features was simply incorrect rather than raising an error. Although not ideal, SINDy now asks the feature library how to calculate the left hand side of the regression. | Partially fixed in #105 |
| SINDyPI as feature library, optimizer | SINDyPI implementation required both SINDyPILibrary, and SINDyPIOptimizer classes. This meant that two independent objects that do not directly interact still depended upon each other. Beyond checking for a legal combination, the SINDy object needed handle that SINDyPI assumes a single dynamics model, but involves many different regressions, leaving the SINDy object unable to print the model or simulate. | Remains to be solved. See future ideas in section 5.2. |
| Unbiasing handled external to optimizer | Unbiasing refers to a final unregularized regression after sparse optimizer has identified support for a model. This can violate constraints of optimizers that allow constraints. Previously, unbiasing was handled in SINDy.fit, which just assumed the user would manually choose whether to unbias. When people added optimizers, as in 3.1, they would not know to set unbiasing rules outside of the optimizer. This was a simple decision to move unbiasing inside BaseOptimizer and subclasses. | #380 |

Table 2.2: Design problems in pysindy resulting in high coupling. The common code smell indicating these problems was the presence of conditional checks and boolean flags for the type of an argument: WeakPDELibrary, SINDyPILibrary, or constrained optimizers. Innovations outside those libraries, such as new features, would need to know how to interact with those subclasses specifically. Changing how other objects behaved when interacting with certain subclasses is a clear sign that abstractions aren't working as abstractions.

```
>>> du_dt = derivative(u, axis=u.ax_spatial[0])
```

And it becomes clear that no, this code either differentiates the wrong axis or assigns the result to a wrongly-named variable. AxesArray works as expected for indexing, including all forms of basic and boolean/integer advanced indexing:

```
>>> u = AxesArray(arr, axes={``ax_spatial"": 0, ``ax_time'': 1, ``ax_coor
>>> u[0].ax_time
0
```

Infrastructural improvements generally involve testing and CI. Both are summarized in table 2.4.

All of these improvements yield an API that is more reliable and more useful in engineering and experimentation. Moreover, they provide a standard of acceptability for new features as described in the subsequent chapters.

| Math bugfixes | | |
|---|---|---|
| Example | Description & Consequence | Fixed in |
| SSR bug | Stepwise sparse regression is a model selection procedure that progressively eliminates or adds a feature until reaching an inflection point in the regression loss. The code implementation, however, was simply comparing an information criterion approach for models generated by dropping each feature in turn. Not only was this not the published method, but the default penalty for number of nonzero terms was zero. This poor default choice meant that without better choices of parameters, the SSR object was just performing least-squares regression. | #559 |
| SR3 bug | SR3 allows weighted and normal L1, L2, and L0 penalties. However, the shape of the weighting, as a 1D vector, was at times multiplied incorrectly, resulting in broadcasting the shape from a 1D vector to a 2D matrix. As a result, the norm and prox calculations were all incorrect. The consequence for users was that SR3 was primarily used for handling constraints, but without sparsification. | #544, #548 |
| Duplicate intercepts | Since BaseOptimizer inherits from sklearn.LinearRegression, it allowed fitting an intercept as a separate, specially-handled feature. This was somewhat more efficient, but it meant that by default, both the feature library and the optimizer would add an identical constant feature. SINDy now explicitly disables the optimizer's intercept, requiring constant features to arise in the function library. | #388 |
| PDE differentiation | PDE feature libraries need to calcualte numerical derivative estimates in spatial directions. pysindy allows many different derivative methods. One would be forgiven for thinking that PDEFind did not work with noisy data, as PDELibrary was ignoring any choice and simply doing finite difference. Alternative methods were made feasible, improving PDEFind by default. | Partially in #476, derivative #41,#43. |

Table 2.3: Some bugs are mathematically significant and are discovered with a careful review. If not discovered, they affect the results of using pysindy, impairing claims made by authors.

| | | |
|---|---|---|
| **Easier to Read and Maintain** | | |
| Fast, limited tests | In order to make changes, developers need to verify that proposed changes do not break anything. Tests that ran for around 300 seconds in 2022 made it difficult to improve the code. Now tests run in thirty seconds. | #334, #391, #393, #573 |
| Packaging | Installation of pysindy relied on setup.py, an explicit and more difficult to maintain file than pyproject.toml, accepted by the python community in 2020. | #332 |
| Benchmarking | In order to guard against performance regressions when moving finite difference functionality, I added automated runtime benchmarking capability. | derviative, #46 |
| Implicit axis conventions | Understanding which order axes need to be in is more than an external API challenge. Internally, knowing the axis layout of an array helps explain what mathematical operation some lines of code are responsible for. I introduced a numpy array subclass to automatically track axis meaning across all numpy operations. This then helped debug a variety of ineptly-named functions. | #185 |
| Confusing inputs_per_library | In order to dispatch some inputs to different libraries, users set the inputs_per_library argument. It previously required passing redundant information and was anectodally the second-most-commonly posted question. The change simplified and guarded it under test. | #362 |

Table 2.4: The most valuable pull requests are the ones that make future changes easier. While none of these explicitly fixed math or coding bugs, many make it much less likely for users or contributors to make such errors. This table includes changes that improve maintainability and legibility. Many other improvements are stylistic, but no less significant. Such changes make it easier to spot errors and learn the math of the code.

# Chapter 3

# OPTIMIZERS

SINDy has seen the most innovation in the last four years with the optimizers. Optimizers in this context means any approach to the meta-problem:

$$\underset{\text{sparse } \Xi}{\arg\min} \left\| \widehat{\dot{X}} - \Xi\Theta(X) \right\|^2. \tag{3.1}$$

As a code object, it combines the decision of what optimization problem to solve with the question of how to solve it. Nevertheless, at the risk of oversimplification, whereas the differentiation method contains prior information about the measurement, the optimizer represents our prior information about the system. [1]

Previous optimizers included Sparse Regularized Relaxed Regression (SR3), Stepwise Sparse Regression (SSR), Sequentially-Thresholded Least Squares (STLSQ), and Forward Regression Orthogonal Least-Squares (FROLS). The first of these is an optimization method for

$$\underset{C\Xi=D}{\arg\min} \left\| \widehat{\dot{X}} - \Xi\Theta(X) \right\|^2 + \lambda R(\Xi) \tag{3.2}$$

Where $R$ is an l0, l1, or l2 regularizer and $\lambda$ is a weighting (optionally, by observation). It is the primary way to apply linear constraints, when required. While the SR3 as generally introduced in [92] can accept norm constraints, the versions explored by [19] and pysindy don't attempt to use them. SSR, FROLS, and STLSQ are both heuristic model-selection algortithms, but with fewer convergence guarantees. STLSQ solves the ridge-regularized

[1] Strictly speaking, the differentiation/smoothing method contains prior information about the measurement and the system smoothness, whereas the optimizer contains prior information about the system coefficients

objective, drops terms that fall below a threshold, then runs the truncated problem, repeating until no terms are dropped in a round. FROLS iteratively adds terms to the regression based upon which feature is most correlated with remaining residual, then terminates after a predetermined number of iterations, choosing the model with the lowest L-0 penalized loss. SSR progressively ablates features based upon which term vitiates the residual the least (alternatively, similar to STLSQ, based upon coefficient absolute value) either until a desired level of sparsity is met or until an inflection point in the cross-validated loss. Regardless, each invocation of these methods leads to a single set of coefficients. They can be wrapped in the bagging of [29], and with the exception of constrained optimizers, they can be wrapped in library ensembling.

Three innovations have opened up promising optimizer approaches: Mixed Integer Optimization - Sparse Regression (MIOSR) provides an optimization approach for global minimization subject to an L0 constraint, in distinction to SR3's L0 regularizer.[2] Regularized Horseshoe Priors, with a Monte-Carlo solver, provides two nominally independent innovations: a truly sparse prior and an alternative to ensembling to discover estimator statistics. Trapping SINDy is a physically-informed method, applicable to a narrower class of problems, adds constraint or regularization to guarantee global or local stability of a Lyapunov function.

### *3.1 MIOSR*

One challenge optimizers face is the heavy use of heuristics and relaxations to apply sparsity. Enter MIOSR. MIOSR applies a sparsity constraint, rather than a penalty, and accepts the NP-complete cost of the nonconvex optimum. As [7] show, modern branch-and-bound methods for big M constraints and type-1 specially ordered sets (SOS-1) can handle most of the SINDy problems that have appeared in literature. Both constraint dialects introduce the binary variable $z_{ij} \in \{0, 1\}$, subject to a knapsack constraint by target or overall. $\sum_{i=1}^{D} z_{ij} \leq$

[2] SR3 has been extended to efficient solutions of L-0 constraints in [4, 61], but this functionality has not been used with SINDy, and at best identifies a local solution.

$k_j$ or $\sum_{i=1}^{D} z_{ij} \leq k$, depending on whether constraints are applied per coordinate or overall. With Big-M constraints, the problem is:

$$\underset{C\Xi=D}{\arg\min} \left\| \hat{\dot{X}} - \Xi\Theta(X) \right\|^2 + \lambda R(\Xi) \tag{3.3}$$

$$M_{ij}^L z_{ij} \leq \Xi_{ij} \leq M_{ij}^U z_{ij} \tag{3.4}$$

$$\sum_{i=1}^{D} z_{ij} \leq k \tag{3.5}$$

$$\tag{3.6}$$

for user-specified lower and upper bounds $M^L$ and $M^U$. Optionally, linear equality constraints on $\Xi$ may be added. The SOS-1 constraints, on the other hand, are:

$$(1 - z_{ij})\Xi_{ij} = 0 \tag{3.7}$$

The implementation in pysindy follows SOS-1, with [7] noting that so long as the number of features is less than several thousand, Existing commercial software such as Complex Linear Programming Expert (CPLEX) and Gurobi will be able to handle it. Indeed, they show that these NP-hard solvers, with optimal compilation, perform better than less optimal implementations of novel convex solvers. They also note advantages in the constrained approach for mutlicollinear problems such as SINDy.

The ideal use case for MIOSR is thus when trying to discover known systems. While the goal of SINDy is of course to discover unknown systems, innovation of SINDy requires evaluating against known systems. When evaluating other aspects of SINDy, an optimizer with an L0 constraint may best isolate effects to the aspects being tested. It becomes particularly useful when researching the effect of smoothing methods and process models. This makes it a frequent tool in the denoument of this thesis, chapter 4. Moreover, it provides the only other optimizer than SR3 to accept constraints, The API implications of this are discussed further in section 3.3. On the other hand, when the number of variables is large, such as in determining reaction networks [38] or neural connectivity, the NP-hard approach

may be prohibitive.

From a software perspective, MIOSR posed a new question to pysindy management: how to publish methods developed externally to the core group? MIOSR had been written with the core algorithm and API compatible with pysindy but as a separate repository. Generally, external functionality can be incorporated as either a

- a component of the core pysindy package,
- an optional component but part of the same repository,
- an externally managed plugin,
- or a separate package entirely.

The amount of user involvement required to specifically use MIOSR increases progressively through the list while the responsibility of the core maintainers decreases. The decision was made to include it as an optional component which allows easier updates and compatibility with the rest of the repository. This was validated as MIOSR benefitted from further updates to the package adding serializability (pickling).

### *3.2 Bayesian SINDy*

[37] introduced the regularized horseshoe prior to SINDy as a better sparsifier. While the L-1 regularizer promotes sparsity because of a useful geometry, it presumes a Laplace prior, which is actually sparse with probability zero. It also tends to bias data that is not sent to zero. The name "horseshoe prior" reflects the appearance of the distribution of posterior shrinkage weights under the horseshoe prior, as described in [18]: To paraphrase, weights that are not fully shrunk are likely not shrunk at all. [71] regularized the horseshoe prior when some shrinkage is desirable after all. Such a prior behaves very similarly to the spike-slab prior, which assigns a nonzero probability to zero. [32] introduced a third exotic prior, the Spike-Slab Gauss-Laplace, which is a convex combination of a Gaussian and a Laplace distribution These papers introduced other innovations to SINDy inference problems, such as autoencorders, a variant of weak solutions similar to [80], and Monte Carlo estimation.

The pull-request review process expanded the paper's organization and description of

the mathematics. Most of the paper's code was originally introduced *de novo* rather than modifying pysindy. But in late 2023, an interested researcher pull-requested some of this functionality as the SBR (Sparse Bayesian Regression) optimizer. This kicked off an involved review process lasting several months and added to the descriptions provided in [32, 37].

The Sparse Bayesian Regression is a maximum a posteriori estimator that follows equation 3.2. Here, the regularizer $R$ is the negative log likelihood of a regularized horseshoe prior on the coefficients $\Xi$:

$$\Xi_{ij}|\tau, \lambda, c^2 \sim \mathbf{N}\left(0, \tau \frac{c^2 \lambda_{ij}}{c^2 + \tau^2 \lambda_{ij}^2}\right) \tag{3.8}$$

$$\lambda_{ij} \sim \mathbf{C}^+(0, 1) \tag{3.9}$$

$$c^2 \sim \mathbf{Inv} - \mathbf{Gamma}(\alpha, \beta) \tag{3.10}$$

$$\tau \sim \mathbf{C}^+(0, \tau_0). \tag{3.11}$$

Here, $\tau$ controls the global shrinkage, and a larger value of $\tau_0$ will allow a less sparse solution. The heavy-tailed Cauchy distributions for $\lambda_{ij}$ allow individual features to escape shrinkage. When $c^2 \gg \tau^2 \lambda_{ij}$, we recover the unregularized horseshoe prior. Otherwise, we recover a normal prior with variance $c^2$. Adding an Inverse-Gamma hyperprior for $c^2$ with $\alpha = \nu/2$ and $\beta = s^2\nu/2$ allows a the variance for the less-shrunk coefficients to follow a student-T distribution of $\nu$ degrees of freedom and variance $s^2$.

While pysindy's SR3 optimizer uses the cvxpy package, to minimize equation 3.2, the SBR optimizer uses Monte Carlo estimation via numpyro, a python implementation of [9].

The most significant contribution was tracing back through the literature how to choose the different parameters $\alpha$, $\beta$,a nd $\tau_0$ from other literature. These parameters needed docstrings that could stand on their own as advice to a user who has not read [37]. Figure 3.1 shows a demonstration of this optimizer learning the lotka volterra equations. In this case, the distribution of coefficients shows that the predation coefficient is much more uncertain than the natural rate of reproduction/death.

Another problem with implementing the regularized horseshoe prior in pysindy, by cod-

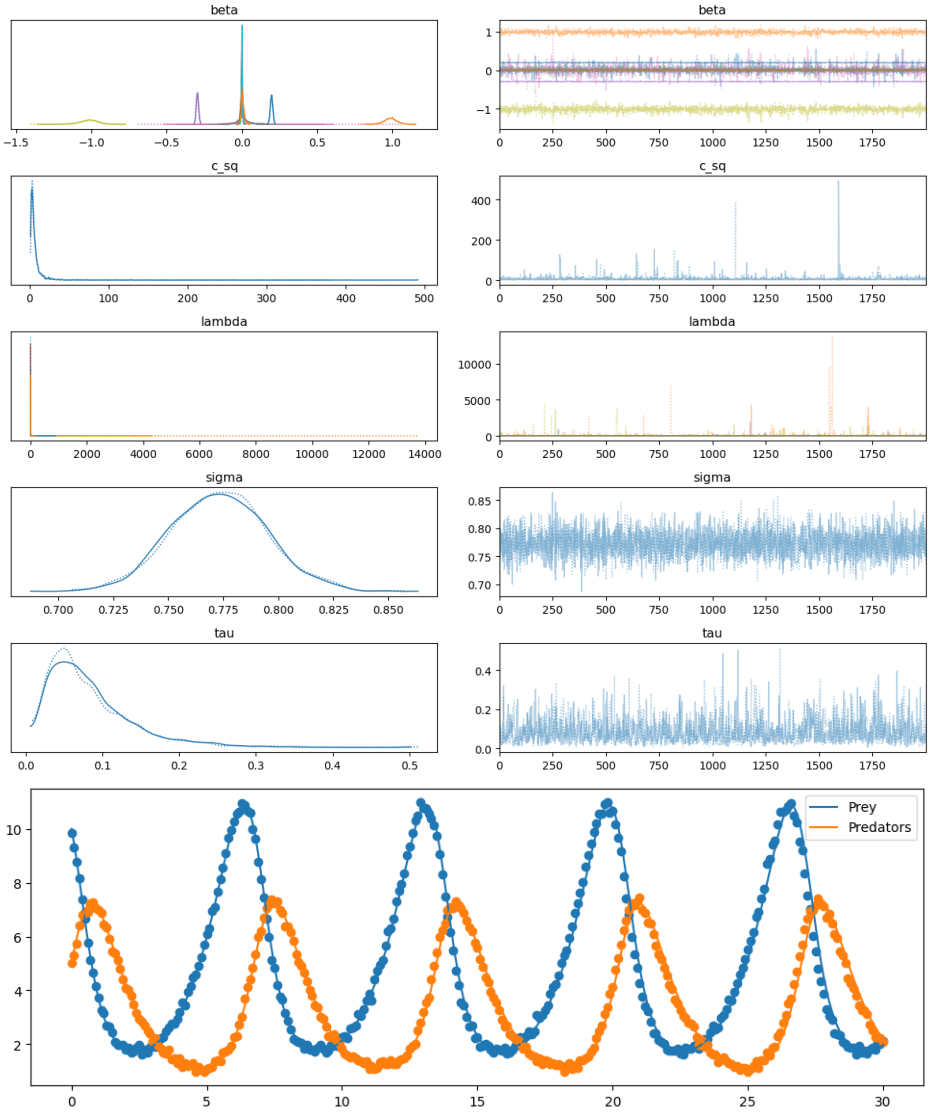



Figure 3.1: Learning the Lotka-Volterra equations for lynx/hare populations with the SBR optimizer. The first column shows the posterior distribution of each parameter calculated by the Monte Carlo optimizer: the SINDy coefficients, the variance of nonzero coefficients, the shrinkage prior for each coefficient, $\lambda_{ij}$, the predicted derivative, and the global shrinkage prior. The second column shows the values of the parameters for each sample The final plot is a simulation of the discovered model as compared to the training data.

ifying the method, people would expect it to be correct. This requirement meant testing, but what form the test would take became challenging. Previous optimizers are tested in that they recover an easy problem correctly, burt cause some shrinkage. The nature of the horseshoe prior is that shrinkage is nearly all-or-nothing, and it becomes difficult to engineer a problem with "just a bit" of shrinkage. This difficulty of choosing a parametrization to fail predictably could also be considered a strength. Nevertheless, some tests were necessary, which brought to light another problem: runtime. Monte carlo methods are noticeably slower than convex optimization when the problem admits the latter. The benefit is a distribution of coefficients, rather than their posterior mode, but the cost is something close to 100x slowdown for small problems. This meant even including fewer tests, and suggests that, if this isn't merely a numpyro framework problem. the SGLD optimizer of [32] may be a better Monte-carlo-like optimizer. It also bodes ill for monte carlo methods when we expand to single step SINDy, as described in chapter 4, which must simultaneously identify coefficients and smooth the trajectory.

The introduction of Sparse Bayesian Regression poses several problems to pysindy's typing system. Most notably, like the Ensemble Optimizer, monte carlo methods provide a distribution of coefficients. Thus, these two optimizers comprise a new subtype, one of probabilistic optimizers. This subtype can be use to distinguish between optimizers that can be ensembled and those that cannot. In the future, it would be desirable to standardize their statistical API. It may also be useful to build a small module to calculate relevant statistics from SINDy models, enabling further analysis. This is a similar design to the Ensemble DMD class in pydmd.

Finally, the most challenging part about these papers is that the most promising contributions are the most difficult to implement. Both attempts at a Bayesian SINDy learn the variable $x$ and $\Xi$ simultaneously, the one via an autoencoder formulation and the other implicitly, using a multi-step prediction error. As mentioned in section 2.1, SINDy solves an optimization for the data and smoothing loss first, then uses those values in discovering the coefficients. From an API standpoint, in order to eventually accomodate autoencoder or

multi-step prediction error, formulating the smoothing problem needs to be separated from solving it. Autoencoders would require a similar separation of concerns. So would extending the results of this paper to discrete SINDy, and likewise with the components in pysindy. We begin this separation of concerns in the subsequent chapter.

## *3.3 Trapping*

One of the major advantages for explainable operator learning is that the last two centuries of mathematics give us tools to understand differential equations. Where that matters most for SINDy is stability, particularly useful when discovering a reduced order model to simulate a PDE. The use of Galerkin Principal Orthogonal Decomposition is common: First, a PDE is simulated at full resolution. Then, the principal orthogonal decomposition of the simulated physics is taken, and SINDy fits an ODE of the principal modes. This model can often be simulated faster than the PDE. The limiting noise arises only from discretization noise and from mode truncation. Yet while SINDy may learn a model that is accurate in parameter space, the model may blow up in simulation. With Trapping SINDy, originally introduced in [51], and extended in [69] the physics understanding about bounded energy is brought forwards into the optimization objective and constraints. It is summarized here.

*Trapping Theorem*

We are concerned with systems of the form:

$$\dot{a}_i(t) = E_i + L_{ij}a_j + Q_{ijk}a_j a_k. \tag{3.12}$$

Typically, $a_i$ represent time variation of Galerkin POD modes, but more generally may represent any system that remains bounded.

To determine if a model is long-term bounded, [82] developed a "trapping theorem" with necessary and sufficient conditions for long-term model stability for systems that exhibit quadratic, energy-preserving nonlinearities. Trapping SINDy implements the conditions of

the Schlegel-Noack Trapping theorem as constraints and penalties on the SINDy objective. The theorem states the conditions under which solutions of a quadratic ODE remain bounded for all time.

The existence of a trapping region presumes some stable point at its center, $m$. The equation for the total energy $K = \frac{1}{2}\sum_i \|a_i(t)\|^2$ is better expressed shifted by $y_i = a_i - m$:

$$\dot{K} = Q_{ijk}y_i y_j y_k + A^s y_i y_j + d_i y_i \tag{3.13}$$

where

$$A^s = \quad L^S_{ij} + (Q_{ijk} + Q_{jik})m_k, \tag{3.14}$$

$$d_i = \quad E_i + L_{ij}m_j + Q_{ijk}m_j m_k, \tag{3.15}$$

and the superscript $^s$ refers to the symmetric component. It stands to reason that if $Q_{ijk}y_i y_j y_k$ is small enough, and if $A^S$ can be made negative definite, the system will be stable with respect to energy. The former is exactly zero if it is permutation-symmetric, that is:

$$\epsilon_Q = \max_{ijk} Q_{ijk} + Q_{jki} + Q_{kij} = 0. \tag{3.16}$$

$\epsilon_Q$ describes the size of the cubic energy term.

This allows us to restate the trapping theorem from [82]:

**Theorem 1** *For a quadratic system that obeys 3.16, and is effectively nonlinear ($Q$ is full rank), there exists a coordinate shift $m$ such that $A^S$ can be made negative definite, if and only if there exists a monotonically trapping region at least as small as the ball $B(m, R_m)$ The radius is then given by $R_m = \|d_m\|/|\lambda_1|$.*

The radius $R_m = \|d_m\|_2/\lambda_1$ depends upon how much the linear and quadratic terms can oppose each other. The left panel of figure 3.2 shows a graphical intuition of the proof, found in [51, 82].

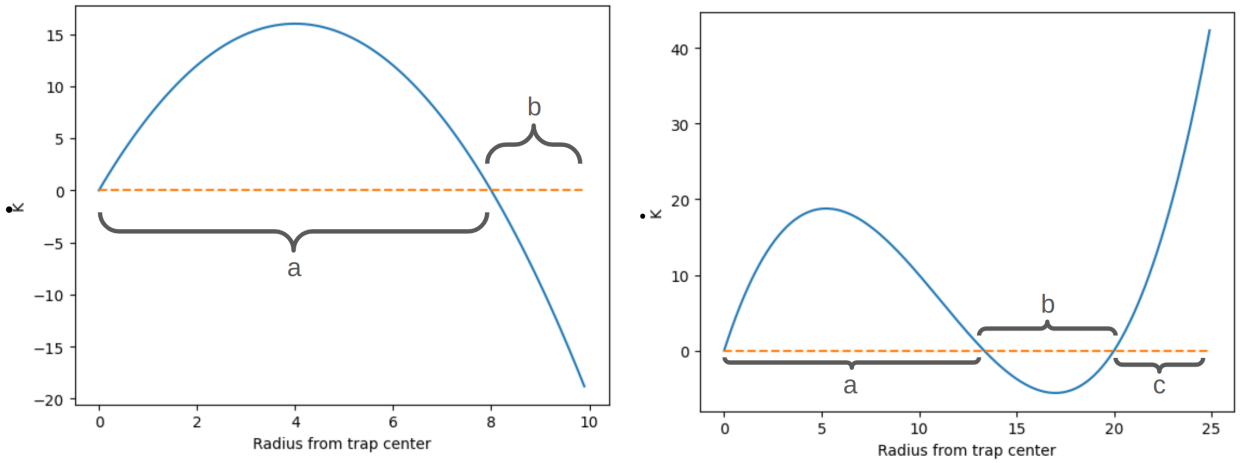



Figure 3.2: A one-dimensional graphical representation of trapping theorem, both global (left) and local (right). This phase-space of energy is chosen along the highest-growth rate direction. Within the trapping ball (a), trajectories may exchange energy across coordinates/system dimensions. Chaotic attractors exist in this region. Outside the trapping ball (b) trajectories fall monotonically into (a). This negative energy region extends to infinity when $\epsilon_Q = 0$. But finite-precsion arithmetic means that $\epsilon_Q$ will be slightly nonzero. In such cases, region (c) exists, where integration may diverge.

Writing the maximum eigenvalue as $\lambda_1(A^s)$, the SINDy objective:

$$\min_{\epsilon_Q=0} \|\dot{X} - \Xi^T \Theta(X)\|^2 + \lambda_1(A^s) \tag{3.17}$$

enforces or promotes all conditions of the trapping theorem. The challenge is in optimizing $A$, which is linear in $\Xi$ and $m$ but not jointly so. We provide more details of the optimization strategy in section 3.3.

*Local Stability Radii*

Since equality constraints can only be satisfied to machine precision, the previous, global formulation leaves us with a more restrictive class of models than is necessary. All we need is a guarantee that the discovered model will remain stable around the trap center, not for all possible starting points. This allows us to expand the class of allowed models slightly by

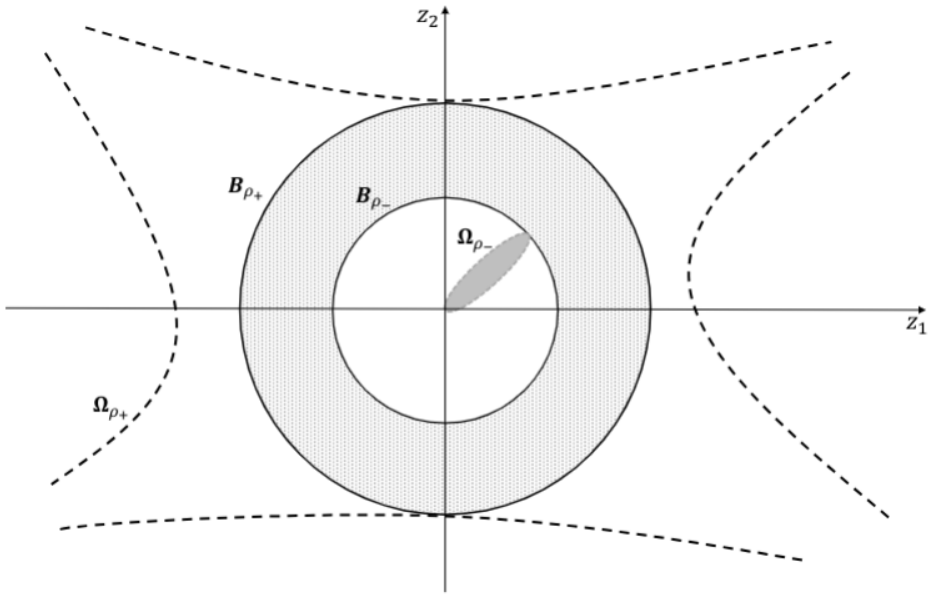



Figure 3.3: Local trapping in two dimensions visualized. Energy increases outside of $\Omega_{\rho^+}$ It decreases in the region between $\Omega_{\rho^+}$ and $\Omega_{\rho^-}$. Within $\Omega_{\rho^-}$, energy grows once again. The approximations of the complex boundaries $\Omega$ with bounding balls $\mathbf{B}$ are a result of simplifying the n-dimensional root finding problem with bounds based upon the one-dimensional directions of fastest growth.

writing the constraint as a penalty:

$$\min_{\Xi} \|\dot{X} - \Xi^T \Theta(X)\|^2 + \lambda_1(A) + \lambda_2 \epsilon_Q \tag{3.18}$$

This objective is supported by the local trapping theorem from [69]:

**Theorem 2** *If a system is effectively quadratically nonlinear, then a local monotonic trapping region exists with inner radius $\rho_-$ and outer radius $\rho_+$ if and only if $A^S$ is negative definite and $\lambda_1^2 - \frac{4r^{3/2}\epsilon_Q}{3}\|\tilde{d}_m\|_2 \geq 0$*

This theorem replaces the condition for no cubic energy growth with a limit on its size relative to the weakest direction of quadratic energy decay.

What effectively happens is that, so long as the cubic energy growth is weak enough, the negative quadratic part dominates long enough for the phase trajectory to cross the

horizontal-axis, as seen in the right panel of figure 3.2. The radii at which the local stability region appears, is naturally bounded by a function of the relative strength of the cubic term and the negative quadratic terms in the energy ODE, equation 3.13:

$$\rho_- = \frac{3|\lambda_1|}{2\epsilon_Q}\left(1 - \sqrt{1 - \frac{4\epsilon_Q}{3\lambda_1^2}\|\tilde{d}\|_2}\right) \tag{3.19}$$

$$\rho_+ = \frac{3|\lambda_1|}{2\epsilon_Q}\left(1 + \sqrt{1 - \frac{4\epsilon_Q}{3\lambda_1^2}\|\tilde{d}\|_2}\right) \tag{3.20}$$

The second panel of figure 3.2 shows a simplified graphical proof of the local trapping theorem. On the other hand, 3.3

*Applicability*

While introduced to SINDy for the purposes of Galerkin approximation, plenty of important ODEs meet or nearly meet the criteria for the trapping theorem. The most nuanced criterion is that of "effective" nonlinearity, which merely means that the trajectory never remains in a linear subspace where the nonlinear term vanishes. Such ineffective nonlinearity would exist if $E$ was an eigenvector of $L$ and also lived the null space of the $Q_i$. Many of the 135 chaotic systems provided by dysts the package developed in [33] to study differential equation learning are bounded and effectively nonlinear, e.g. Lorenz, Hadley, Finance, Lorenz-Stenflo, and Vallis El Nino. Others, such as the Burke-Shaw system and the Sprott A system have a linear subspace in which trajectories remain and grow unbounded. In such systems, which are technically disqualified from the trapping theorem, Trapping SINDy still manages to discover effective models. Figure 3.4 shows the result of Trapping SINDy on fifteen nonlinear systems, only five of which are "effectively nonlinear".

In addition, bounds behave better for different Lyapunov functions than energy. While trapping SINDy was derived from the energy equation 3.13, an equivalent global or local trapping method can be derived for any Lyapunov function $K = y^T P y$ where $P$ is any Hurwitz matrix. This may be useful for systems where enstrophy is a more useful descriptor.

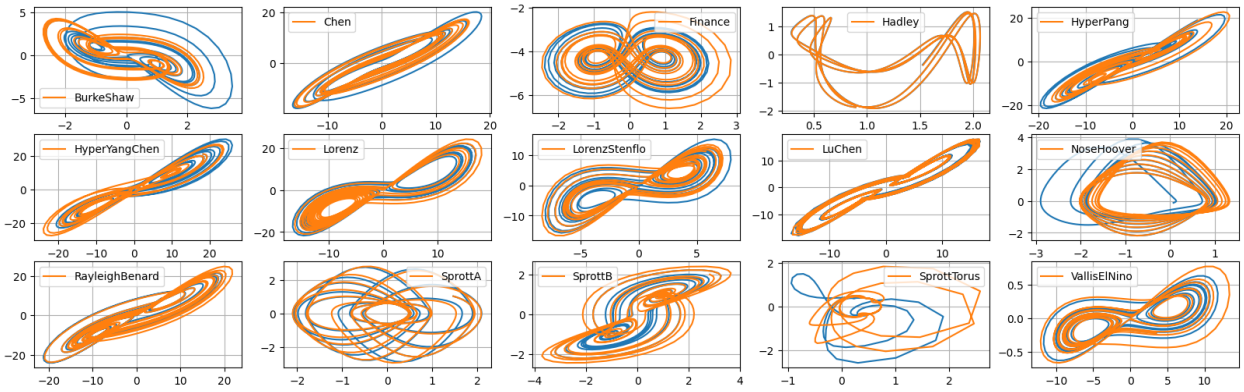



Figure 3.4: Trapping theorem and SINDy, applied to representative chaotic systems from [33]. Many of these systems have some linear subspace that allows them to grow unbounded. Nevertheless, Trapping SINDy is able to discover their dynamics and simulate them effectively. In these cases, simulated annealing was used to further improve the trap center.

As an example, figure 3.6 shows, with an experiment on Von Karman vortex shedding, how the local trapping theorem provides wider stability radius for a lyapunov function defined by enstrophy rather than energy. More work is needed, however, to determine when, an enstrophy matrix will improve the performance of Trapping SINDy.

*Optimization strategy*

The objective in equations 3.17 and 3.18 is nonlinear, because the two optimization variables, the trap center $m$ and the SINDy coefficeints $\Xi$, are mutiplied together in the formula for $A$. The consequence is that we alternate between optimzing $m$ and $\Xi$. We also use a relaxation scheme, introducing $A$ as a relaxation of the $A^s$ calculated firectly from $m$ and $\Xi$. Thus we

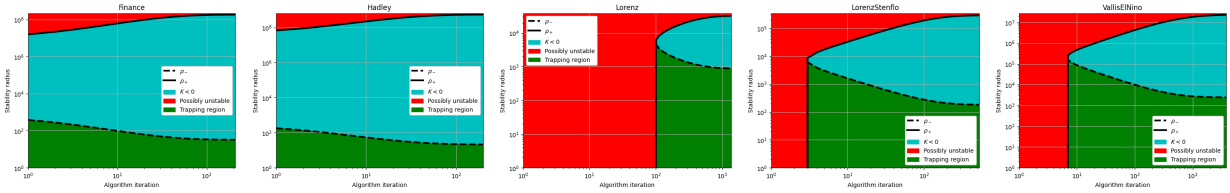



Figure 3.5: The Trapping SINDy algorithm progressively grows the region of stability, emerging from a saddle-node in the energy bounds. Here are the inner and outer stability radii for the effectively nonlinear systems from [33]

optimize three expressions in a cycle:

$$\Xi^+ = \arg\min_{\Xi} \|\dot{X} - \Xi^T \Theta(X)\|^2 + \gamma R(\Xi) + \beta \|\epsilon_Q\| + \frac{1}{\eta}\|Qm - A\|^2 \tag{3.21}$$

$$m^+ = \arg\min_{m} \frac{1}{\eta}\|Qm - A\| \tag{3.22}$$

$$A^+ = \arg\min_{\lambda_1(A) < -\gamma} + \frac{1}{\eta}\|P_m^A \Xi - A\|^2 \tag{3.23}$$

As mentioned previously, $\epsilon_Q$ is the maximum of linear combinations of $Q$, and $Q$ just represents the quadratic terms of $\Xi$.

In practice, the algorithm is not always able to enforce negative definiteness of $A$. Achieving negative definiteness requires some sense a priori where the center of the trapping region is. The best heuristic available is that the learning rate for $\Xi$ and $m$ should be near each other by a factor of around 1e-2. When the algorithm converges, it also provides bounds for the stability radii, as in figure 3.5. Nevertheless, the relationship between finding the trap center and enforcing negative definiteness of $A^S$ suggests where future research could improve; e.g. [34] optimizes over the root of $A^S$, enforcing negative definiteness by construction.

Sparsity is an additional challenge and has not been succesfully combined with Trapping. There is little reason a priori to believe that a galerkin POD model would obey a sparse ODE anyways. Nevertheless, as mentioned above, many sparse systems do exhibit effective nonlinearity and thus sparse Trapping deserves future research. Since Trapping research predated some of the fixes in section 2.2, the SR3 and ConstrainedSR3 superclasses implemented regu-

larization incorrectly, except in some L-0 cases. That sparse Trapping SINDy struggled could be due either to such bugs, to bugs introduced in implementing the projection matrices of [51] that were solved in the course of [69] to other bugs introduced by a divergent formulation using CVXPY, to more mathematical questions about the degrees of freedom that remain after constraints in equation 3.16. More work is needed to test out and improve the sparse capabilities of Trapping SINDy. In particular, it would be excellent if the trapping SINDy constraints could be adapted to MIOSR for solving the optimization problem. This desire reflects one instance of the conflation, discussed in section 2.1, of optimization objective and optimization method.

*Code Planning and Coordination*

The implementation of Trapping SINDy, through substantial merge conflict, shows how shared code fosters collaboration. Beyond the generalizations of local trapping and enstrophy, we refactored the trapping optimizer, removing redundancies with existing ConstrainedSR3, reducing the API for creating constraints, and substantial refactoring. However, these improvements occured in parallel, leading to hundreds of lines of merge conflicts. While the technical resolution of these conflicts is beyond the scope of this paper, it demonstrated two lessons. Firstly, when publishing research via open-source software, incremental improvements to the state of the art have an advantage over larger improvements. Secondly, the existence of merge conflicts helps facilitate a practical communication within an open-source lab.

Trapping SINDy challenges the pysindy API, in the sense discussed in section 2.1: To summarize the thesis of the relevant section, this work seeks a type system that prevents mathematical incoherence. Other variants of SINDy (Weak SINDy and discrete SINDy) eschew derivatives and thus ought to be promoted to subclasses of the SINDy object. They naturally can take any combination of feature library and optimizer - except WeakSINDy can use Trapping, discrete SINDy cannot. There is a latent concept of a problem where continuous physics matters, that is nonsensical when applied to discrete data or with the

wrong feature library. Trapping has traditionally been implemented as an optimizer, since its primary work involved an optimization penalty and constraint. Overloads can enforce that an optimizer and feature library are compatible types. However, trapping demands that its feature library be the same instance as in the containing class, about which it can currently know nothing. Potential near-term solutions involve runtime type checking or only exposing Trapping via a factory. Longer term, perhaps the latent concepts of physics understanding can be formalized, along with divorcing the specification of optimization problem from the method of solving the optimization problem.

*Experimental Results*

In practice, trapping SINDy has performed well. The ability to evaluate stability before integration is useful to catch problems before they occur when discovering and evaluating models automatically. Such facility is an example of the kind of problem analysis desired for the API in section 2.1. As an optimizer, it has the strongest physical intuition, and thus does well on systems with a chaotic attractor, even on unsmoothed trajectories as shown on the Lorenz system in figure 3.8. Another example, the lid-cavity experiment of [69] shows how impressively Trapping SINDy improves model discovery from a good fit that diverges rapidly to one closely matching system harmonics in figure 3.7. Systems that live on a manifold in which energy is conserved in general pose a problem for SINDy, as described in the kinematic example of section 1.1. At least where the coordinates are such that the system is identifiable, as in the inviscid magnetohydrodynamic (MHD) model of

$$\begin{bmatrix} \dot{V}_1 \\ \dot{V}_2 \\ \dot{V}_3 \\ \dot{B}_1 \\ \dot{B}_2 \\ \dot{B}_3 \end{bmatrix} = \begin{bmatrix} 4(V_2V_3 - B_2B_3) \\ -7(V_1V_3 - B_1B_3) \\ 3(V_1V_2 - B_1B_2) \\ 2(B_3V_2 - V_3B_2) \\ 5(V_3B_1 - B_3V_1) \\ 9(V_1B_2 - B_1V_2) \end{bmatrix}, \tag{3.24}$$

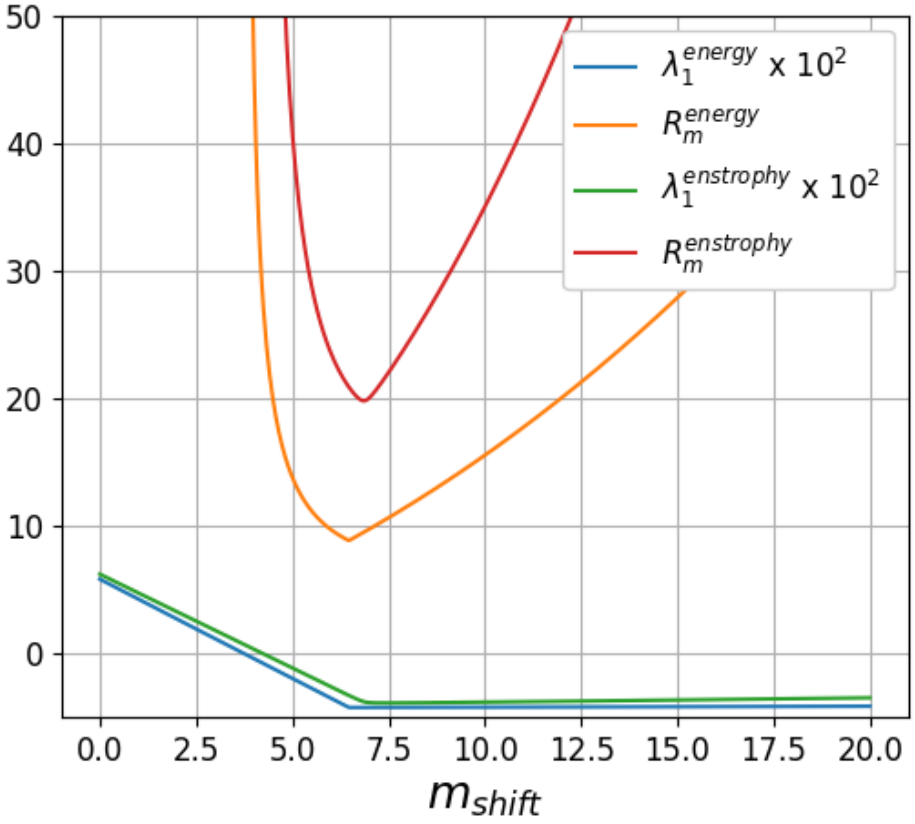



Figure 3.6: While stability in one lyapunov function implies stablity in another, certain lyapunov matrices act to even out the axes of ellipsoid stability giving rise to an easier optmization problem. These are the stability radii for the POD of Von Karman vortices behind a cylinder. Using the enstrophy matrix to define a lyapunov function gives a larger estimate for the radius of stability.

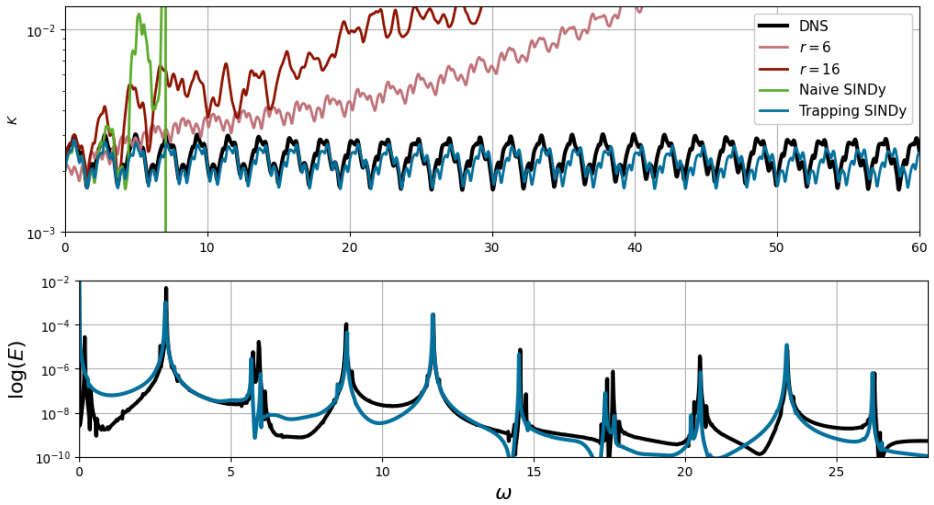



Figure 3.7: The lid-cavity POD flow is an example of how well Trapping SINDy can do even when it does not discovery a negative-definite $A^S$. In this case, the maximum eigenvalue was $4x10^3$. And yet encouraging stability made it was dramatically more stable than naive SINDy. This stability is not a naive coercion with a dominating negative polynomial. The trapping stability also causes the model to identify the correct power spectrum.

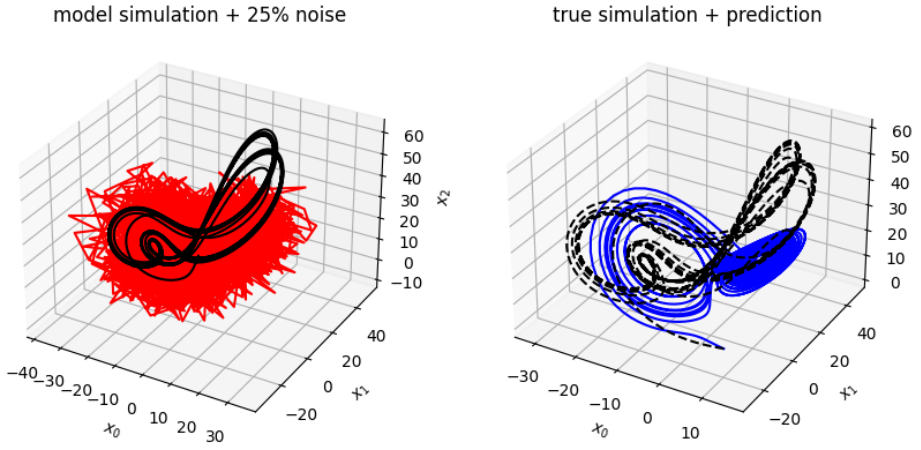



Figure 3.8: Sparsity is perhaps a weaker prior belief than boundedness. Here, the SINDy is able to recover the lorenz system despite substantial noise. The left panel compares the very noisy training data in red, to the discovered model's believable simulation. The right panel shows that the model does discover a stability region roughly corresponding to the attracting manifold.

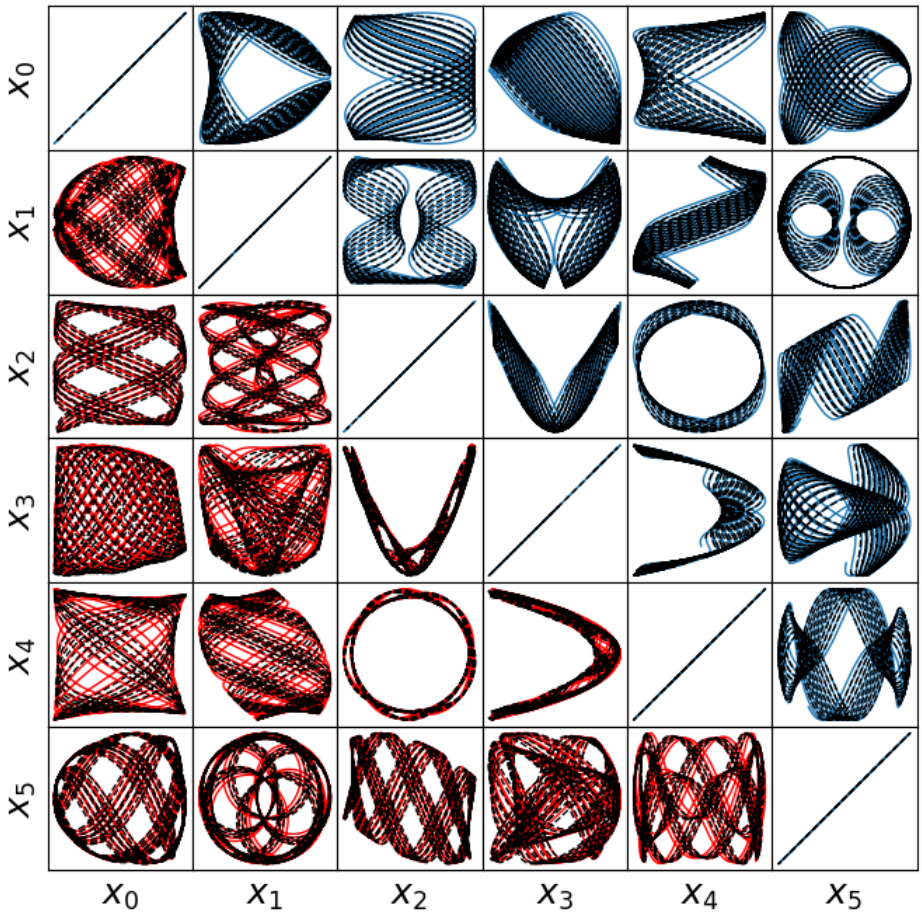



Figure 3.9: The 2-D projections of true and learned dynamics of the MHD coordinates. The system is stable, but does not posses a trapping region. Nevertheless, Trapping SINDy discovers an accurate model.

trapping SINDy can provide useful information. In such a case, the $A^s$ matrix cannot be made negative definite, but it *can* be made zero, and Trapping SINDy identifies this: In such a case, we know the solutions are oscillations on a manifold, as depicted by the Lissajou plots for an MHD system discoevered by Trapping SINDy, figure 3.9.

The alternative to trapping is to simply add a small, negative, high-order, odd-polynomial term to a discovered ODE. In the plume research listed in appendix C, we used both. The data came from a reduced-order model of video data. Although trapping's assumptions come

from fluid system research, and there was no guarantee that the data's dynamics obeyed those assumptions, we did know that the model was (a) nonlinear and (b) stable. By comparison to the alternative, Trapping SINDy required a PolynomialLibrary of degree two, whereas the negative polynomial stability can work for any library that is polynomially-dominated (including FourierLibrary). On the other hand, the stability of Trapping is learned from the data, whereas negative polynomial stability is ad hoc and specified a priori. Nevertheless, trapping performed reliably in stabilizing dynamics of plume ROMs.

*Future*

Trapping research can benefit most from a better trap center optimization. The diagnosis for not being able to discover a negative-definite quadratic energy growth matrix, $A^s$, is almost uniformly that the trap center has not converged. Implementing the alernate parametrization of [34] may solve this problem. In addition, more work to establish experimental criteria for discovered stability would go a long way to benchmarking the algorithmic improvements desired. Similarly, many of the experiments depend upon saved data which has been deeply investigated a priori. As a case in point, research into trapping the Von Karman POD relies not just on simulation but also on knowing how to calculate the essential time-shift mode. Finally, analysis of trapping should be brought from the examples into an experimental pacakge, later described in section 5.1.

The trapping approach of [51] has been extended into system identification methods several other times, e.g. [25, 53, 55]. The ability to promote global stability guarantees has broad potential for data-driven models, and has already been extended to neural-network-based system identification methods [68]. For general quadratic systems, [54] discusses the condition for such systems to be locally stable by means of a Lyapunov function and provides better estimates of the radii of the domain of attraction by a novel optimization-based approach. For systems whose dynamics can be described by more general polynomial functions, the sum-of-squares of polynomials technique provides a constructive method for generating Lyapunov functions to achieve global stability and control of dynamical systems and

ROMs [23, 39]. However, a sum-of-squares polynomial Lyapunov function for a given system is nontrivial to discover and not guaranteed to exist.

# Chapter 4

# SINGLE STEP SINDY

This chapter seeks to build a more robust SINDy to some of the degenerate cases of section 1.1.2. Specifically, it aims to continue to improve the ability to handle noise as well as a mis-specified library. It begins by introducing two differentiation/smoothing methods, each implemented in pysindy. Then, it combines the derivative estimation objective with the smoothing objective for a joint optimization.

In the basic formulation of SINDy, estimating the derivatives often involves an optimization problem, one that inherently smooths the zeroth deriative as well. Examples of such smoothers from pysindy include Savitzky-Golay smoothers and total variation smoothers. A growing corpus of work attempts to present SINDy without derivative estimation, or does so concurrently with estimating the system coefficients. Such work comprises "soft adherence to governing equations" as in [80], Weak SINDy of [64], and Bayesian papers whose implementation admits Monte Carlo estimation of the states as well. Yet only one, [28], have sought to just combine the derivative estimation step and the dynamics estimation step. That investigation made a variety of specific choices about smoothing/interpolation and presaged our current research and development effort.

We aim to combine smoothing and dynamics estimations of SINDy in a single step. While there are several different smoothing expressions that could be amenable to this treatment, we choose two founded in stochastic calculus that may adapt best beyond the current work: Kalman smoothing and Kernel smoothing. The first contribution was refactoring BaseDifferentiation in pysindy to mean differenation and smoothing, as discussed in section 2.2. The second innovation was introducing Kalman and Kernel smoothers to pysindy (or more accurately, to the derivative package) The penultimate improvement is to introduce

the class structure that supports a single-step SINDy. Finally, we develop and experiment on simultaneous, or single-step SINDy.

*Alternate methods*

Alternate methods to assimilate noisy data include Total Variation regularization, Savitzky-Golay smoothing ([21]), and Weak SINDy ([66]). Total variation regularization emerged from denoising overhead imagery in [79]. Slight variations in the optimization method exist, but the implementation used in derivative seeks to minimize:

$$\lambda\|\dot{x}\|_1 + \| \int \dot{x} - z\|_2^2$$

for some measurements $z$ and smoothing hyperparameter $\lambda$. Similar to Kalman smoothing, there is a regularity condition on the derivative and a hyperparameter that balances it with the measurement error term. However, in this TV implementation, the state $x$ is the deterministic integral of $\dot{x}$ (up to an approximation error), whereas Kalman allows for some covariance between state and its derivatives.

On the other hand, Savitzky-Golay smoothing seeks to find a local polynomial regression of some window. As a local method, the effects of outliers are ignored beyond the window width, and it can handle nonstationary problems better. However, unlike Total Variation and Kalman Smoothing, Savitzky-Golay's smoothing hyperparameter (window width) affects runtime: a wider window means more points to account for at each interpolation.

The most widely explored existing single-objective SINDy method is Weak SINDy, of [64, 65]. Weak SINDy eschews the need to estimate derivatives by building a regression from the weak form of the ODE, equation 1.1. For some suitable test function $\phi(t)$ that perishes at

the boundary,

$$\begin{aligned} \int_a^b \dot{X}\phi(t)dt &= \int_a^b \Xi\Theta(X)\phi(t)dt \\ X(b)\phi(b) - X(a)\phi(a) - \int_a^b X\phi_t(t)dt &= \int_a^b \Xi\Theta(X)\phi(t)dt \\ -\int_a^b X\phi_t(t)dt &= \int_a^b \Xi\Theta(X)\phi(t)dt. \end{aligned}$$

With a judicious choice of test function, one can remove the derivative from $X$ and put it on the test function using integration by parts. By removing the derivative estimation error, this method is more robust to noise, but still uses the noisy observations in the feature library. [1] In practice, the domain of observation is split into a set subdomains, each with an appropriately decaying test function. These new domains become the samples in the regression objective. Weak SINDy is a compelling approach. It does not need two separate steps and may be better conditioned than Kalman smoothing, but it cannot take advantaged of a smoothed state in $\Theta(X)$.

Other SINDy variants [37, 80] have also removed derivative estimation, but instead of using the weak form, they integrate the dynamics library for the loss in 3.1:

$$\|X - \int \Xi\Theta(X)\|.$$

Finally, [28] comes close to what we plan to do. However, their choice of spline smoothing and an STLSQ optimizer can be broadened substantially.

The rest of this chapter proceeds as follows. We begin with introducing the Kalman smoothing/differentiation step in the next section. [60] introduces Kernel smoothing for PDE discovery, and we reprise some basics of kernel smoothing from that paper next, as its

[1] It would be instructive to observe how much the reduction in sample variance is due to the averaging effect of integration versus removing ill-conditioned derivative estimation.

part of the foundation for the subsequent section. Both Kalman and Kernel smoothing are then used as a format for building single-step SINDy in the subsequent section. I explore the advantages of single-step SINDy in a series of experiments in the final section, though note that the results are preliminary.

## *4.1 Kalman SINDy*

The first improvements made to pysindy in the direction of single step SINDy is adding a Kalman smoothing/differentiation option. This section, borrowing heavily from my work in [88], introduces Kamlan smoothing and a useful hyperparameter-selection method. It then builds a series of experiments comparing Kalman smoothing to alternatives.

### *Background*

Kalman filtering and smoothing refers to a group of optimal estimation techniques to assimilate measurement noise to a random process. Filtering refers to incorporating new measurements in real-time, while smoothing refers to estimating the underlying state or signal using a complete trajectory of (batch) measurements. Kalman smoothing and filtering are widely used in engineering design for real-world control and prediction, such as tracking and navigation, in radar systems, econometric variables, and weather prediction ([3]).

The Kalman smoother can be considered as a best-fit Euler update, as the maximum likelihood estimator of integrated Brownian motion, or as the best linear fit of an unknown system. While SINDy is typically concerned with deterministic processes, in the first step of SINDy they are unknown, and so probabilistic language is appropriate. The best fit/-maximum likelihood view extends the classic Kalman updates to a rich family of efficient generalized Kalman smooth algorithms for signals corrupted by outliers, nonlinear models, constraints through side information, and a myriad of other applications, see [1, 2, 43]. Providing these innovations to the SINDy smoothing/differentiation step API is a worthy effort beyond the ambit of this thesis, however.

In the simplest invocation, the Kalman estimator is determined given only the ratio

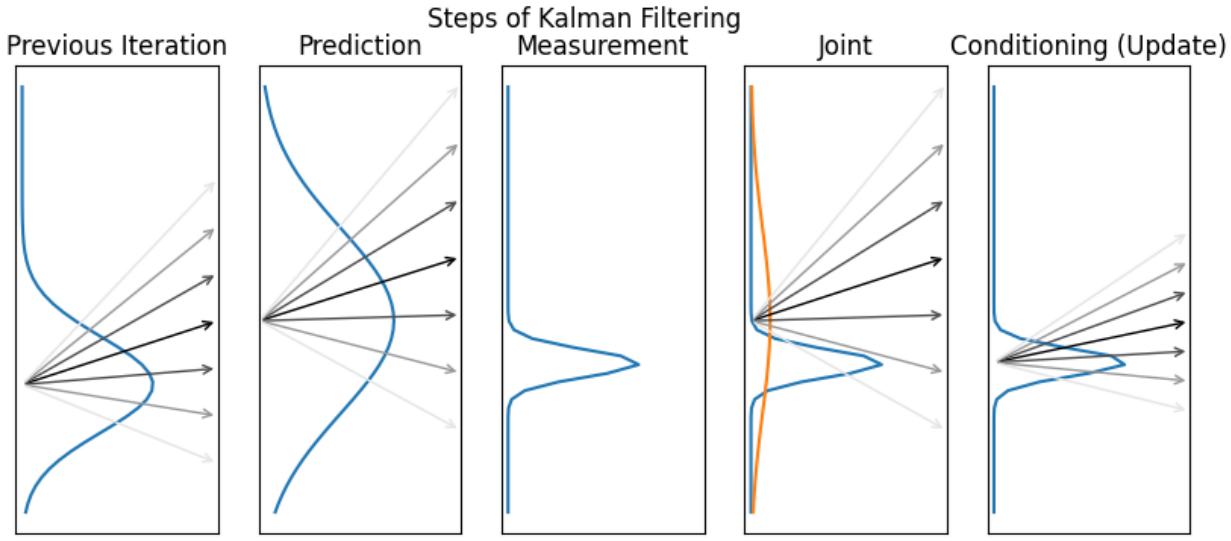



Figure 4.1: Explanatory depiction of Kalman filtering. A previous iteration gives a distribution $p(x_{i-1}, \dot{x}_{i-1})$. Multiplication by an update matrix produces the predictions $p(x_i, \dot{x}_i | x_{i-1}, \dot{x}_{i-1})$. Simultaneously, measurements $z_i$ are taken that, with known measurement noise, give $p(z_i|x_i)$. Multiplication gives the joint distribution $p(x_i, \dot{x}_i, z_i | x_{i-1}, \dot{x}_{i-1})$, from which the conditional distribution $p(x_i, \dot{x}_i | z_i, x_{i-1}, \dot{x}_{i-1})$ can be calculated, shown in [27]

of measurement noise to the process's underlying stochastic noise. Fixing both of these parameters allows Kalman methods to also identify the variance of the associated estimator. Furthermore, a line of research aims to identify parameters purely from data, including [5, 12, 42]. Many such methods include their own hyperparameters and are not guaranteed to find a solution, but are an improvement on the indeterminate nature of direct maximum likelihood or a prior choice for the variance hyperparameter.

*Equations and more technical part*

In adding Kalman smoothing to SINDy, we introduce a distinction between the measurement variables and the state variables of the dynamical system in equation 1.1. As such, the inputs to the problem become $m$ time points of measurements of $k$ variables ($Z \in \mathbb{R}k \times m$) and a linear transform from the state to the measurement variables $H \in \mathbb{R}n \times k$ describing how

the process is measured. This innovation will allow us to use the same syntax when defining additional collocation points in the next sections.

Measurement error is assumed to be normally distributed with $HX - Z \sim \sigma_z \mathcal{N}(0, R)$ where the covariance matrix $R \in \mathbb{R}k \times k$. Measurement regimes where noise is autocorrelated or varies over time can be accomodated by flattening $HX - Z$ and describing $R \in \mathbb{R}nk \times nk$. Two parameters are required: $\sigma_z$, the measurement noise standard deviation, and $\sigma_x$, the process velocity standard deviation per unit time. If only point estimates of the state are required, and posterior uncertainty is not, it suffices to use the ratio $\rho = (\sigma_z/\sigma_x)^2$.

Each process is assumed to have an independent, Brownian velocity. This leads to Kalman smoothing estimator:

$$\arg\min_{X,\dot{X}} \|HX - Z\|_{R^{-1}}{}^2 + \rho \|G[\dot{X}, X]\|_{Q^{-1}}{}^2 \tag{4.1}$$

Here, $G$ is a linear transform to separate $[\dot{X}, X]$ into independent, mean-zero increments, and $Q$ is the covariance of the Brownian process velocity $\dot{X}$ and its integral, $X$ over the increments. A graphic displaying Kalman filtering is shown in Figure 4.1. To illustrate the ideas, the figure presents step-by-step filtering updates; however, batch smoothing is used in pysindy.

*GCV parameter estimation*

We use the generalized cross validation of [5] to choose $\rho$. The method witholds some measurement points in order to find the values of $H, R, G$, and $Q$ that produce estimates $\hat{X}$ that fit witheld data most accurately. In our work, we presume to know the measurement parameters and most of the process parameters - after all, "position is the integral of velocity", implies certain constraints on $G$ and $Q$. The method accomodates these constraints through specifying a prox function, resulting in an ideal $\rho$. GCV can apply to many data science techniques, but comes with the burdens of nonconvexity. Although not guaranteed to find a minimum, particularly for our univariate optimization, heuristic experience has showed that

the longer the trajectory, the more likely the algorithm will succeed.

The engineering problem then became how to incorporate the GCV approach in the derivative package. The simplest solution we approached was to add an entrypoint that users could request instead of specifying $\rho$. Our experiment package then registered a plugin for that entrypoint, calling functions packaged and published alongside [5]. This allows us to refer to the kalman functionality as a single type in the code, as opposed to having a separate type for the hyperparameter-optimized version. It comes at a cost of repeating the Kalman smoothing, which is naturally a part of [5]'s package's funcitonality. Although redundant, this single linear solve at the end of an iterative optimization does not affect runtime substantially.

*Experiments*

We seek to evaluate Kalman smoothing as a step in SINDy in comparison to other noise-mitigation innovations. We simulate eight dynamical systems [2] with noisy measurements across a variety of initial conditions, discovering ODEs from SINDy with different smoothing methods. These systems are of moderate complexity: two chaotic, one linear, three quadratic, four cubic and all existing in `pysindy`'s repertoire. We aim to explore the behavior under varying noise regimes. An extension of the experiments to more systems in order to draw conclusions about the behavior across different dynamical characteristics represented in [33] would be a worthy extension, but beyond these first steps. The trials are run across a range of durations and relative noise levels, calculated as the noise-to-signal ratio of the measurement variance with the system's mean squared value. To compare the methods, we then integrate the discovered equations and observe how well they preserve the system's structure as well as directly comparing the coefficients.

We compare the results of SINDy with three different differentiation/smoothing methods, parametrized across a gridsearch: Kalman smoothing, Savitzky-Golay, and L-1 total

[2] Cubic Harmonic Oscillator, Duffing, Hopf, Lotka-Volterra, Rossler, Simple Harmonic Oscillator (SHO), Van Der Pol Oscillator, and Lorenz-63

variation minimization [3], as well as with Kalman smoothing with GCV hyperparameter optimization [5].

The gridsearch parameters sweep a range of reasonable values around the default provided to a user. We explore noise regimes ranging from minimal to severe, and data availability ranging from parsimonious to copious. It is worth noting that choosing the gridsearched optimum requires knowledge of the true system, in distinction to the hyperparameter optimization method used for Kalman smoothing. The GCV method itself requires hyperparameters only for the proportion of witheld data, stopping criteria, and any regularization.

Methods can be compared in several ways: by the coefficients of the equations they discover, by their accuracy in forecasting derivatives, and by how well the discovered system recreates observed dynamics in simulation. There are many metrics for scoring dynamical system discovery, and the merit of a metric depends upon both the use case and whether the trajectory considered is one of importance. For instance, in controls engineering, the local derivative and very short-term forecasting is the primary imperative. On the other hand, for reduced-order PDE models, recreating larger-scale phenomena in simulation may be more important. Finally, in high-dimensional network dynamics, the accuracy of identifying connectivity, as measured by coefficient F1 score, is most important.

The integration metrics require additional parametrization and are best suited for low-noise systems. Coefficient metrics are the most straightforwards, and we compare methods by F1 score and MAE (mean absolute error) as the duration of training data increases, and separately, as the measurement noise increases. These metrics, based upon L1 and L0 norms, reflect the goal of sparsity better than metrics based upon the L2 norm. We also visually evaluate how well the discovered ODEs, simulated from random initial conditions in a test set, track the true data and display relevant behavior.

Beyond the differentiation step, the SINDy models also specify a function library and optimizer. The feature library used for all experiments was cubic polynomials, including

[3] TV requires a coefficient for the L-1 regularizer and Savitzky-Golay requires a smoothing window width.

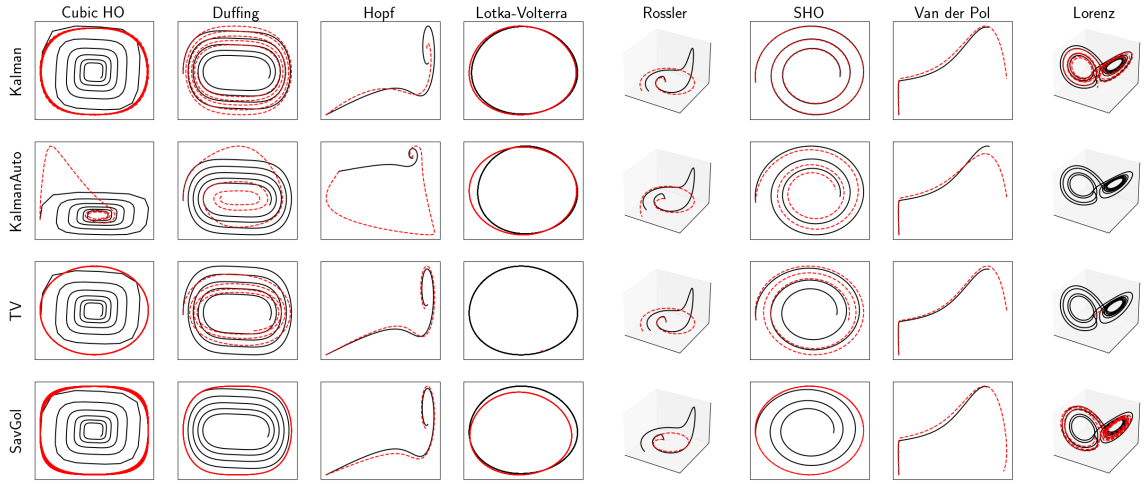



Figure 4.2: The simulation of discovered models compared to test data. Kalman appears better for half of eight ODEs. It represents the essential behavior of more ODEs than TV and Savitzky-Golay. Kalman with auto-hyperparameter selection performs similarly to total variation on a gridsearch. 10% relative noise, 8 seconds of data.

mixed terms and a constant term. The optimizer was MIOSR, configured with the correct number of nonzero terms a priori, and ensembled over 20 models each trained on 60% of the data. Presenting SINDy with the known number of nonzero coefficients is an attempt to present a best case, where we can ameliorate any interaction between the smoothing method and sparsification parameters. A full list of ODE, simulation, and experimental parameters are shown in the Appendix, tables A.1 and A.2.

We find that SINDy with Kalman smoothing recovers the problem structure in application as well or better than competing methods in seven of eight systems. Models discovered in this manner track the essential dynamics in most cases. GCV hyperparameter optimization is noticeably not as good as the ideal gridsearched optima, but still provides accurate smoothing in six of eight cases. Somewhat counterintuitively, however, the best smoothing parameter does not necessarily lead to the most accurate solution, either in coefficient accuracy or in simulation.

SINDy with Kalman hyperparameter optimization tends to perform worse than that with

Savitzky-Golay, but on par with Total Variation gridsearched optima, and is itself outperformed only slightly by the Kalman gridsearched optima. While hyperparameter optimization imposes some runtime cost, it does not require access to the true data, making those results all the more inspiring for field use cases. Simulations of discovered models across all ODEs and methods are shown in Fig. 4.2.

There are cases where the MAE scores of different methods do not indicate which method performs better in simulation, and where effective smoothing does not predict effective system recovery. As one case in point, Kalman GCV and Total Variation smoothing appears most accurate for the Hopf system in Fig. 4.3. However, the coefficient metrics show that Total Variation, Kalman gridsearch, and Savitzky-Golay recovered the system equations better that Kalman GCV, and simulation in Fig. 4.2 confirms Kalman GCV is the outlier in reconstructing the dynamics. Similarly, methods have a wide range of performance on MAE and F1 score on the Rossler system, despite all simulations missing the chaotic behavior. As a final case in point, the Kalman-smoothed training data itself does not seem as accurate as data smoothed by L-1 Total Variation in the the SHO trial. Yet a SINDy model based upon it performs better.

Just as surprisingly, despite performing well in simulating the Lotka-Volterra system, SINDy with Kalman hyperparameter optimization performs poorly in the coefficient metrics. Coefficient metrics by data duration is shown in Fig. 4.4, and by noise level shown in Fig. 4.5. If anying appears consistent about Kalman with GCV, it is that, with a long enough duration, it becomes more consistent. Across the range of noise levels sampled, either Savitzky-Golay or Kalman (gridsearched) perform the best, depending upon system. As expected, Kalman (gridsearched) always outperforms Kalman GCV, but it is interesting to note that at some durations and noise levels Kalman GCV occasionally outperforms Savitzky-Golay (e.g. Rossler, SHO).

Overall, Kalman SINDy performs well and is a worthwhile addition to pysindy. However, it does not obviously stand out from the next most competitive method, Savitzky-Golay smoothing. Looking to [12] for an explanation, the limit could be due to an inherent Pareto

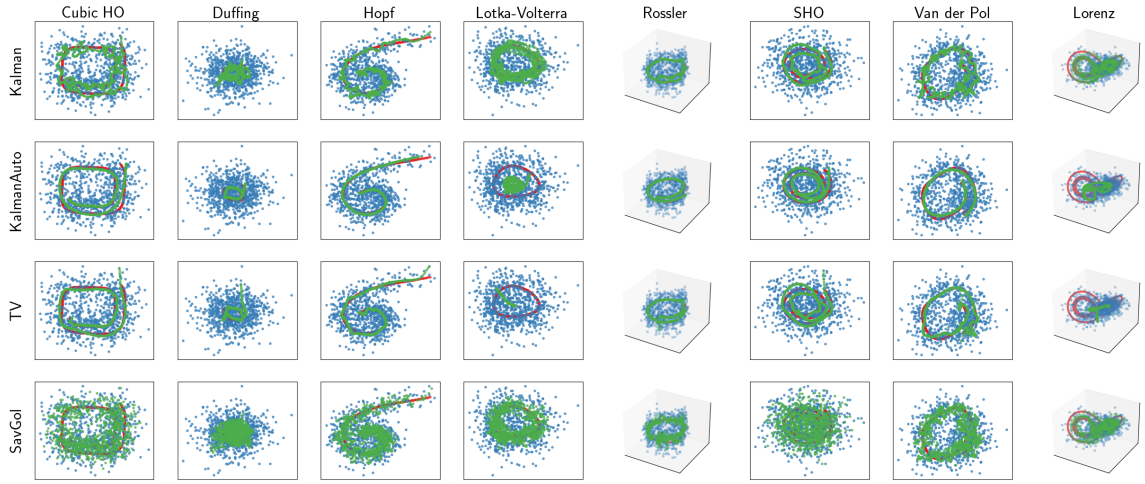



Figure 4.3: The smoothing of training data, performed by different differentiation methods prior to SINDy fit. It does not appear to be the case that a more visually accurate smoothing yields a model that behaves more correctly in simulation. Nevertheless, as Fig. 4.2 shows, Kalman-smoothed trajectories lead to better models in simulation. 10% relative noise, 8 seconds of data. Of interest, note the particular cases of SHO, cubic HO, and Hopf systems, in which the best smoother does not necessarily lead to the best simulation.

frontier between smoothness and accuracy: Nonlinear ODEs are not necessarily uniformly smooth. We may expect any smoothing method to over-smooth in some areas and under-smooth in others. Moreover, with regular measurement points, Kalman can also be described as cubic spline interpolation. This makes the result, that it is slightly better than the local cubic polnomials of Savitzky-Golay, unsurprising.

The assertion by [12], that all generic smoothers are subject to a Pareto frontier, suggests that single-step SINDy may be a promising approach. However, before proceeding with that innovation, we want to consider a broader class of smoothers of which Kalman is just one part.

## *4.2 Kernel Background*

In addition to the views of the Kalman smoother around Brownian motion, Euler updates, or cubic splines, another way to consider the Kalman smoother is as an indefinite kernel.

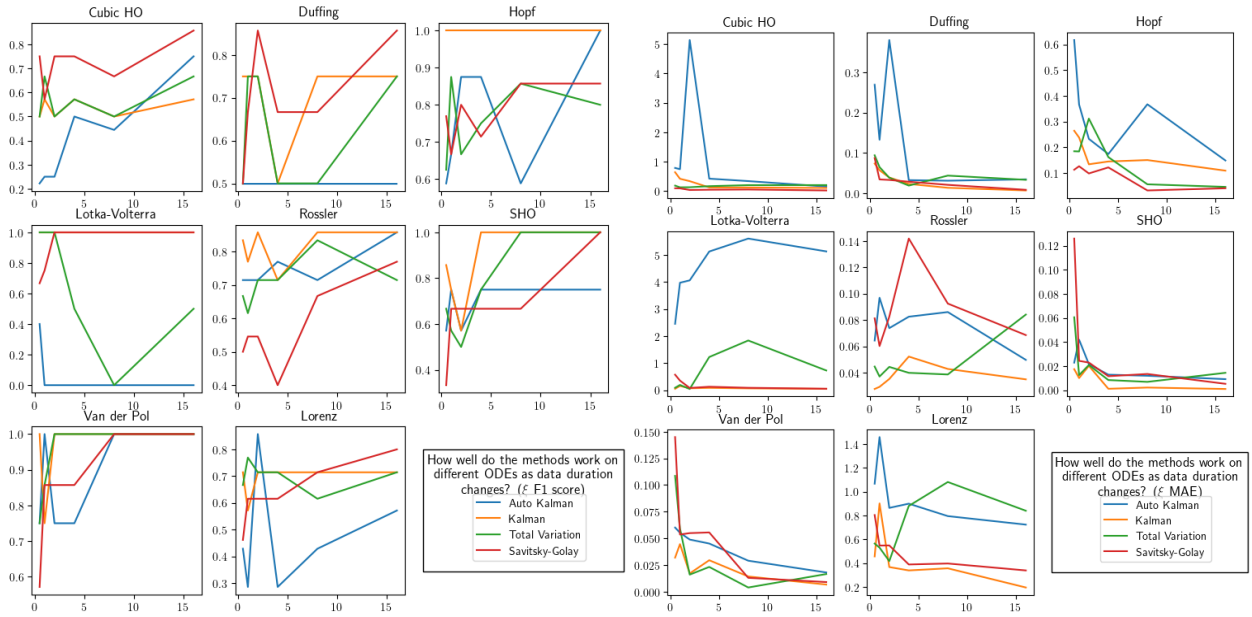



Figure 4.4: How well different smoothing methods in SINDy recover the ODE coefficients as data duration increases. 10% relative noise

Kernel smoothing complements Kalman smoothing as a related class of methods to smooth an estimate derivatives of noisy measurement data. We added functionality to derivative so that Kernel smoothing could be used as the derivative estimation step in SINDy, as in section 4.1. This chapter introduces relevant background and some brief experimental results, before we elevate both Kalman and Kernel smoothing to Simultaneous SINDy in the subsequent section.

Gaussian processes, via kernel methods, are used throughout machine learning [73]. The kernel of a process refers to the covariance function of a process at two different times. In the condext of differential equations, kernel methods have been used for PDE inverse problems as well as for learning solution operators to PDEs. [6, 22]. [60] used kernels as a step specifically to smooth measurements before solving the inverse problem, using either SINDy or a learned kernel operator for the second step.

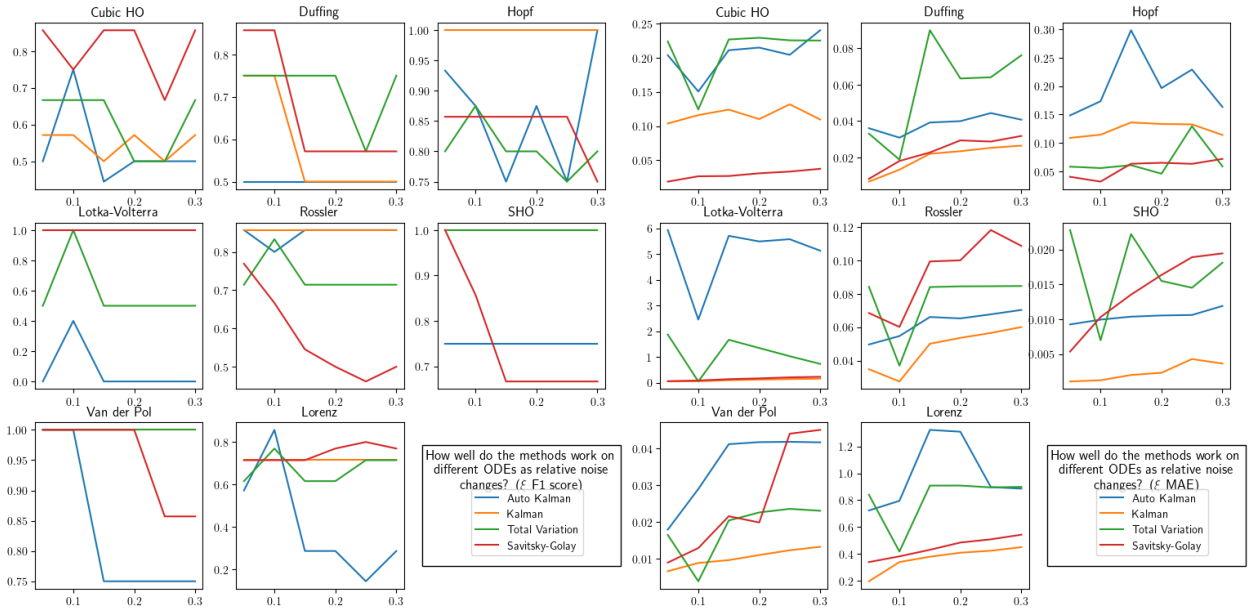



Figure 4.5: How well different smoothing methods in SINDy recover the ODE coefficients as noise increases. 8 seconds of data.

**Kernel Background** A Kernel is a function between to points of an input space to the real number line that reflects the similarity of some process $u(x)$ on that space. Here, the input space is $x \in \Omega$, and the kernel function $k : \Omega \times \Omega \to \mathbb{R}$. The similarity measure must obey certain criteria, most commonly that it is a covariance function, and by extension, gives rise to a positive-definite matrix $K \in \mathbb{R}^{m,m}$ when evaluated on $m$ points in $\Omega$. This matrix $K$ can be used to define an inner product and its induced, elliptic norm for a space of functions. That is to say, choosing a kernel function and some points of interest, [4] the kernel matrix gives us a way to compare continuous functions based on their evaluation at finite points. This connection becomes most useful when paired with the representer theorem, which gives

[4]Measurement points, Colocation points, etc

a shortcut to solve problems of the form:

$$\arg\min_{\mathcal{C}(u)} \|u\|_{H_K}, \tag{4.2}$$

where $\mathcal{C}$ represents some convex constraints.

**Theorem 3** *If $k$ is a positive-definite kernel, the solution to problem 4.2 is a linear combination of basis functions defined by the kernel and points of interest:*

$$u^*(x) = \sum_{i=1}^{m} \alpha_i k(x, x_i) = u_\alpha \tag{4.3}$$

This Invanov formulation, treating constraints as a prior belief, has corollary theorems for Morozov and Tikhonov formulations. Proof can be found in [83]. The theorem also provides an explicit relationship between the continuous $u$ and the discrete parameter $\alpha$: $\|u\|_{H_k} = \|\alpha\|_K$.

The representer theorem invites a wide variety of optimization problems. Most interesting to us are smoothing problems. The kernel function $K$ contains prior belief about the process that generated the smooth function. Descibing the measurements as $z$ and regression problem can be phrased as

$$\arg\min_{\alpha} \|u_\alpha - z\|^2 + \lambda \|\theta\|_K^2. \tag{4.4}$$

Similar to Kalman smoothing, the optimization seeks to minimize the sum of measurement likelihood and likelihood due to a prior belief of the process. The choice of kernel will affect the degree of smoothness and other properties of the interpolant function.

**Different Kernels** Different kernels $k$ represent different prior beliefs that a Gaussian process has covariance $k(x, y)$ between position $x$ and position $y$. All kernels we consider are stationary, where $k(x, y) = k(y - x)$, so that covariance between two points in the process is only a function of their distance. Kernels are often parameterized by the scale of the

independent variable, which affects the width of the basis functions $k(\cdot, x_i)$. Too short of a length scale will create a function that only represents a function very near to some observed points. Too long of a length scale will create a function that merely linearly interpolates. In addition to the length scale, kernels often have an additional absolute scale, and some also possess a smoothness parameter.

While [26] describes kernel options in more detail, we present a summary of three most pertinent kernels:

**Gaussian Radial Basis Function (RBF)** The Gaussian RBF Kernel as a solution to a variational problem penalizing nonsmoothness. Its basis functions are Guassians, and the scale parameter affects the width of the basis function.

**Matérn Kernel** The Matérn kernel arises as the solution to Laplace's equation on a disc, and thus its basis functions are Bessel functions. It is a generalization of the Gaussian RBF to finite-order smoothness. This allows it to decay more slowly than the Gaussian RBF.

**"Kalman" Kernel** Kalman smoothing, as described in the previous section, posesses a solution evocative of the representer theorem for kernels. Moreover, its derivation as the maximum likelihood estimator for Brownian motion means that it is a Gaussian process. Thus we talk about the Kalman smoothing matrix as a kernel of sorts. Like a kernel interpolant, Kalman smoothing creates a solution in terms of basis functions: cubic splines. Similar to Matérn, Kalman smoothers of different orders of smoothness can be derived (See appendix B for an example).

The essential element of Kalman smoothing, that the process is chopped into independent updates, means that the covariance of the process overall is singular. The reformulation into independent increments means that the model is flat, with a singular covariance matrix. Significantly, the gram matrices for most kernels are both nonsingular with dense covariance matrices. Whereas the function that minimizes Matérn and Gaussian RBF RKHS norms is the zero function, The Kalman smoothing matrix represents a seminorm, and any straight line has a seminorm of zero. Thus, in some sense, it represents a less-informative assumption about the underlying process. As distance goes up, similar behavior to Kalman's flatness


Figure 4.6: Simulation of systems learned using kernels. Uses same parameters as figure 4.2.

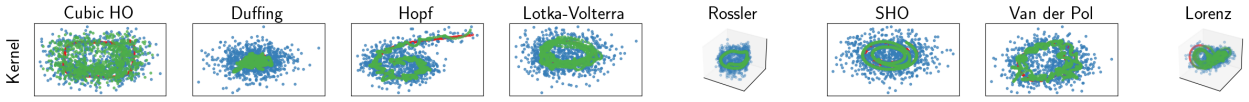



Figure 4.7: Smoothing of systems learned using kernels. Uses same parameters as figure 4.3.

appears: in the flat limit of kernels, basis functions act locally, and the condition number grows. [1]

**Engineering** Adding a second smoothing/differentiation method to pysindy was facilitated by the work done on the first. Because pysindy had been updated to use the smooth trajectory, Adding Kernel smoothing was as simple as adding a new class of differentiator to derivative. Only a Gaussian RBF was added, however, but since the hyperparameter optimization plugin added in section 4.1 is general enough, users can bring their own kernel hyperparameter optimization methods to pysindy.

### *4.2.1 Experiments*

Adding Kernel smoothing to traditional, two-step SINDy will serve as a comparator for single-step kernel smoothing in the coming sections. We have recreated some of the experiments from the previous section, replacing kalman smoothing with Gaussian RBF Kernel smoothing. See figures 4.7 and 4.6

### *4.2.2 Segue*

The early kernel SINDy work immediately suggested several avenues of improvement. Already, substantial kernel research was being done in jax. Jax was new and untested, so while

kernel smoothing was implemented in numpy, the addition of kernel smoothing identified a long-term need for multiarray compatability, spanning numpy, jax, and cvxpy. As the changes in 3.2 introduced Secondly, while all smoothing methods allow for some smooth interpolation, usually via a measurement matrix, the form of Kernel regression makes it more obvious that some value may be gained by handling the second step of SINDy across a tighter mesh of colocation points. These opportunites, as well as the general API challenges of section 2.1, prompted a reinvention of pysindy's core SINDy object.

### *4.3 Simultaneous Colocation SINDy*

We now combine the derivative estimation and sparse coefficient estimation steps of SINDy. In the abstract, describing the measurement loss as $\ell_m$, the random process loss (e.g. RKHS norm in kernel smoothing, or integration term in total variation) as $\ell_p$, and the SINDy dynamics loss as $\ell_S$ yields the meta-pro1blem:

$$\underset{\dot{X},X,\text{sparse } \Xi}{\arg\min} \lambda_m \ell_m(X) + \lambda_p \ell_p(\dot{X}, X) + \lambda_S \ell_S(\Xi, \dot{X}, X), \tag{4.5}$$

allowing weighting for each term, and one again leaving sparsification choices to an investigation described later in this section. Choosing Kalman or Kernel smoothing, along with normally distributed measurement error concretizes the expression:

$$\underset{\dot{X},X,\text{sparse } \Xi}{\arg\min} \lambda_m \|HX - Z\|^2 + \lambda_p \|(\dot{X}, X)\|_M^2 + \lambda_S \|\dot{X} - \Xi^T \Theta(X)\|^2. \tag{4.6}$$

Here $M$ is a matrix from whichever linear smoothing/interpolation method is used.

This joint optimization is more difficult than traditional SINDy, with traditional SINDy existing as a special case. The difficulty arises because $\Xi^T \Theta(x)$ in the final term is nonlinear. That nonlinearity ceases to exist in cases when measurements are exact, such as the original SINDy paper, providing one avenue for direct comparison with traditional SINDy methods. The three hyperparameters can be reduced to two without loss of generality: the ratios of

measurement to process noise ($\lambda_p/\lambda_m$) and the ratio of trajectory interpolation to ODE fit ($\lambda_S/\lambda_p$). By comparison, the directly applicable regular SINDy approaches have only the first hyperparameter: the relative weighting of measurement and process noise in smoothing. However, we can recover traditional bi-level SINDy optimization by choosing the ratio of trajectory interpolation to ODE fit low enough that the first two terms are optimized first, then $\Xi$ is fit as best as possible to minimize the third term.

**Gauss-Newton updates** Attempting to solve the truly nonlinear case will, however, be challenging. We aim to solve equation 4.6 via Levenberg-Marquardt. similar to [22], but with various sparsification options. This involves a Gauss-Newton approach of linearizing the residual $[\lambda_m \ell_m, \lambda_S \ell_S]^T = F(\Xi, \dot{X}, X)$ around $\Xi^k, \dot{X}^k, X^k$ to

$$\widetilde{F}^k(\Xi, \dot{X}, X) = F(\Xi^k, \dot{X}^k, X^k) + J[F](\Xi^k, \dot{X}^k, X^k)\Delta(\Xi, \dot{X}, X) \tag{4.7}$$

and iteratively minimizing the local linear approximation

$$\Xi^{k+1}, \dot{X}^{k+1}, X^{k+1} = \underset{\substack{(\Xi,\dot{X},X)\text{close to}(\Xi^k,\dot{X}^k,X^k) \\ \text{sparse } \Xi}}{\arg\min} \|\widetilde{F}^k(\Xi, \dot{X}, X)\|^2 \tag{4.8}$$

Iterates continue until termination. [5] This meta-problem requires several more decisions to be made concrete:

**Globalization strategies** Iterates need to remain close the previous value in order to maintain the acceptability of the linear residual interpolation. "Acceptibility" is enforced by how accurately the linearization approximates the new iterates:

$$\|\widetilde{F}^k(\Xi, \dot{X}, X) - \widetilde{F}^k(\Xi^+, \dot{X}^+, X^+)\| / \|F(\Xi, \dot{X}, X) - F(\Xi^+, \dot{X}^+, X^+)\| < c \tag{4.9}$$

[5] Timing note: In practice, using autograd methods to calculate the Jacobian can be expensive, so it is faster to include only the smooth nonlinear terms in $F$ and $F^k$. On paper, however, the mathematics of including $\lambda_p \|(\dot{X}, X)\|_M^2$ in $F^k$ are the same as adding it as a separate term in equation 4.8.

For some scalar threshold $c > 0$. [6] When the condition fails, $\alpha$ can be tightened and the iterate regenerated. While acceptability is defined by accuracy of the linear approximation, closeness can be enforced in different ways, all using a paramater $\alpha$:

- *Trust Regions*: Requiring $\|(\Xi, \dot{X}, X) - (\Xi^k, \dot{X}^k, X^k)\| < \alpha$
- *Backtracking Line Search*: $\alpha$ is a ratio of the proposed gradient step size.
- *Levenberg-Marquardt*: Adding a penalty $\alpha\|(\Xi, \dot{X}, X) - (\Xi^k, \dot{X}^k, X^k)\|^2$ to the regression.

We choose Levenberg-Marquardt. In addition, the value of the acceptability criteria at any iterate allows for modifying $\alpha$, how close or far the next iterate is allowed to move. This feedback gives rise to acceleration.

**Sparse Regression** We posit sparsity via several different approaches. Most obviously is via a regularizer:

$$\underset{(\Xi,\dot{X},X)\text{close to}(\Xi^k,\dot{X}^k,X^k)}{\arg\min} \|\widetilde{F}^k(\Xi, \dot{X}, X)\|^2 + \lambda_\Xi R(\Xi). \tag{4.10}$$

The linear, least-squares variables $\dot{X}, X$ can be projected out, such that

$$\tilde{g}^k(\Xi) = \min_{\dot{X},X} \widetilde{F}^k(\Xi, \dot{X}, X). \tag{4.11}$$

pysindy can achieve this minimization either implicitly, solving for $\dot{X}(\Xi), X(\Xi)$ and forming

$$\tilde{g}^k(\Xi) = \widetilde{F}^k(\Xi, \dot{X}(\Xi), X(\Xi)), \tag{4.12}$$

or approximately, establishing $\dot{X}^k, X^k = \arg\min_{\dot{X},X} \widetilde{F}^k(\Xi^{k-1}, \dot{X}, X)$ and forming

$$\tilde{g}^k(\Xi) = \widetilde{F}^k(\Xi, \dot{X}^k, X^k). \tag{4.13}$$

[6] This is similar to an Armijo-Goldstein condition, but the improvement threshold is relative to the magnitude of the function.

While the former is preferable due to its accuracy and convergence behavior, the latter is easier to implement, as described in the subsequent paragraph on implementation.

**Algorithm 1** Single-Step SINDy using Levenberg-Marquardt

$u_0, \alpha_0, \gamma, \beta, \delta$
$u^k \leftarrow u_0$
**while** $\|\nabla_u F\| < \delta$ **do**
  $J^k \leftarrow J_F(u^k)$
  $F^k \leftarrow F(u^k) + J^k(u - u^k)$
  $\Delta L = 0$
  $\alpha \leftarrow \alpha_0$
  **while** $\Delta L < .05$ **do**
    $u^+ = \text{inner}_\text{o}\text{pt}(F^k, u^k, \alpha)$
    $\alpha = \gamma\alpha$
    $\Delta L = \|F^k(u) - F^k(u^+)\|/\|F(u) - F(u^+)\|$
  **end while**
  $u^k \leftarrow u^+$
  **if** $\Delta L < .2$ **then**
    $\alpha_0 \leftarrow \alpha_0\beta$
  **else if** $\Delta L > .8$ **then**
    $\alpha_0 \leftarrow \alpha_0/\beta$
  **end if**
**end while**

We present two distinct approaches: When $R(\Xi)$ is an L-1 optimizer, we utilize the implicit variable projection of equation 4.12 and update $\Xi$ using proximal gradient descent, which involves a series of inner iterations $i$ of the form:

$$\Xi^{k,i+1} = \text{prox}_{l_1,\lambda_\Xi}(\Xi^{k,i} - \nabla\tilde{g}^k(\Xi^{k,i})) \tag{4.14}$$

Alternatively, if we have an explicit $\dot{X}^k, X^k$, then $\widetilde{F}^k(\Xi, \dot{X}^k, X^k)$ has a traditional form of a SINDy objective. This means we can use any existing, traditional SINDy optimizer to solve the sparse meta-problem, equation 3.1.

Our single-step SINDy algorithm is described in algorithm 1. In addition to the description so far, there are several smaller implementation decisions to describe. We terminate

the outer loop when the absolute value of the loss function changes less than a threshold, or when the gradient of the loss drops below a threshold in an L-2 sense. Both can be absolute or relative, although our implementation in pysindy uses absolute criteria. Initialziation also plays a role; we provide many in pysindy, but recommend starting the nonlinear optimization with values from a related linear problem. In our case, that means using a pure linear interpolant to get $\dot{X}, X$, followed by a least-squares regression to get a non-sparse $\Xi$. However, the method can become more complex with a schedule of optimizations (e.g. switching methods based upon termination criteria). All of these initialization strategies are provided as part of the single-step SINDy API.

**Engineering** Engineering colocation into pysindy provided an excellent opportunity to consider what all other single-step approaches might need. It also highlighted some of the problems of existing approaches to integrating variant methods such as WeakSINDy and discrete SINDy, Most notably, the SINDy object itself represented both a problem's admin (inputs, shapes, and how to print) and how to use a particular combination of constituent objects (differentiation, feature library, and optimizer). The former was extracted into a base class, from which a new SSSINDy object derived. It was the root of a new heirarchy that separated the combined loss expression, with its trajectory interpolant, measurement and process models, and its ODE feature library, from an optimizer that understood only the difference between process coefficients and ODE coefficients. This communication is mediated by a series of container classes. Optimization is currently handled solely through Levenberg-Marquardt, but the global regularization and sparsification can be customized via subclasses. As SSSINDy uses jax for numerical methods, SSS uses specialized feature libraries. Most feature libraries, however, require limited changes to enable their use with jax. Multi-array compatibility is a long-term goal of pysindy.

The new composite types are depicted in figure 4.8. New objects in Single-Step SINDy include:

- Trajectory interpolation: TrajectoryInterpolant object is used to generate the data residual

as well as convert from basis coefficients to trajectory values.

- Expressions: An expression is specified with an interpolant, a feature library, and the relative weight weight of data and dynamics. When given data, it creates a ResidualObjective.
- ResidualObjective: This object stores the optimization problem information: the components include callable data and dynamic loss functions, their Jacobians, and array offsets for different components of the residual and optimization variable.
- _BaseSSOptimizer: A single-step optimizer, in addition to the needs of _BaseOptimizer, also tracks process coefficients. LMSolver is the subclass that implements our optimization
- ObjEvaluation: Used by Levenberg-Marquardt, this contains the evaluation of the objective at a parameter value: the residuals, the Jacobian, the loss, the gradient, and Hessian approximation
- _LMRegularizer: Used by Levenberg-Marquardt, this ABC contains method eval, which adds to the loss, gradient, and Hessian approximation of an ObjEvaluation, and step, which descibes how, at a particular ObjEvaluation, the parameters should be updated. Subclasses include LML2Cholesky, LML1, and LMAlternatingSINDy.
- _take_prox_step Used by Levenberg-Marquardt, this can be swapped out for different approaches, e.g. taking a step in a trust region or taking a linesearch step.

### *4.4 Future work*

The method of Single-step SINDy has the potential to replace existing SINDy implementations, in part because it can incorporate many of the existing innovations. However, much work remains to evaluate different inner sparsifiers and sparsifying wrappers. Beyond such evaluation, we aim to improve the ability to choose hyperparameters: the smoothing scale, the relative weight of the dynamics term to the interpolant term, and the relative weight of data and smoothness in the interpolant. Along these lines, we hope to test out different interpolants: notably Matérn kernel and Kalman kernel/cubic spline interpolant. From an engineering perspective, there are a few more things to try: most notably, to expand the sense of derivative in order to allow wrapping methods from the derivative package in

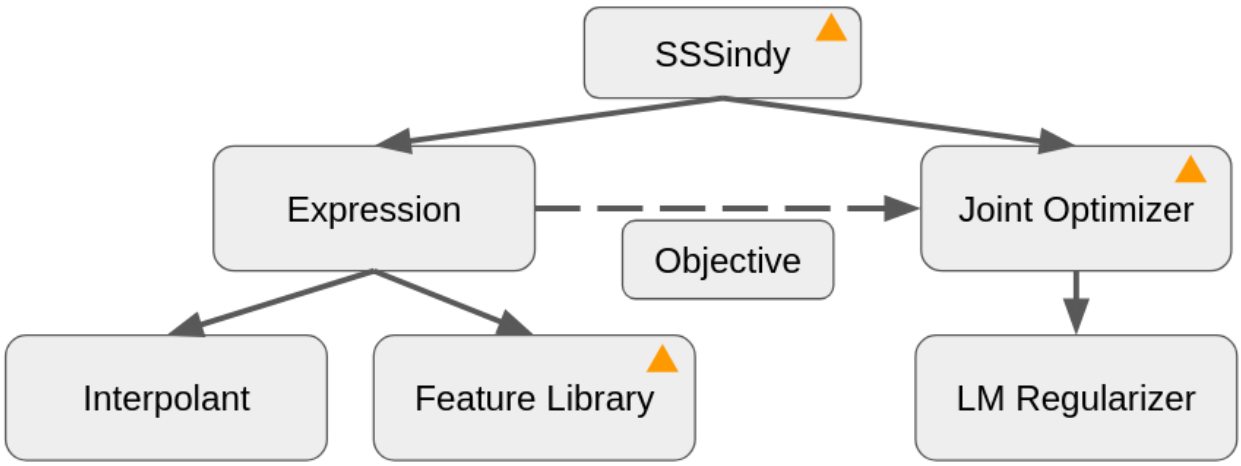



Figure 4.8: Single-Step SINDy abstractions/classes. As a provisional attempt at factoring code to implement algorithm 1, a single-step SINDy model is initialized with an expression, that balances interpolation and dynamics, and an optimizer, which can contain sparsification options. The overall model object, when fit with data, requests information about an objective function from the Expression, which it then passes to the optimizer to solve. Classes with a triangle inherit some functionality from the a base class shared with traditional SINDy. The overall SSSINDy object still has the same methods as a SINDy class and can be used in any experiment designed for the latter.

aTrajectoryInterpolant subclass.

There are plenty of directions to improve Single-step SINDy. The lowest-hanging fruit is to make more feature libraries compatible with jax. This would enable spatial derivatives, and thus PDEs. Similarly, TrajectoryInterpolant should fit to a spatiotemporal grid, allowing spatial and time smoothing in a consistent manner. Beyond these first approaches, the optimization algorithm could use more innovation including a monte carlo estimator, better acceleration, or hyperparameter optimization via marginal likelihood or GCV likelihood. Certain optimizers, like SBR or SR3, might benefit from reformulating with the complete optimization objective, rather than alternating as a Levenberg-Marquardt subproblem. And for speed, it may make sense for the ObjectiveResidual to separate out the linear terms from the nonlinear terms. Finally, we need to explore better heuristic approaches to initialization and termination across a wider range of problems.

On the interpolant side of the algorithm, we could add spline smoothing and make the equivalence with Kalman exact. We hope to explore the differences and similarities between interpolants, and in particular if there is a hyperparameter range where they behave similarly. Though the kernel smoothers are nonsingular, in the hyperparameter limit, they approach a singular matrix and may behave similar to Kalman smoothing.

The objective itself also deserves some attention. It is worth noting and disclaiming that combining the two steps of SINDy does not, in itself, yield a likelihood estimator consistent with a continuous-time model. Consider the following stochastic processes:

(a) $\dot{y} = \xi^T \theta(y)$ where $\xi$ is a random variable.

(b) $dy = \xi^T \theta(y) dt + \int_0^t B_s$, where $B_t$ is Brownian motion

(c) $\dot{y} = \theta(y) + \widetilde{\theta(y)}$ where $\widetilde{\theta}$ is a random process in $y$

The first is a nonstationary process, representing the situation where we do truly know the functional form of $\dot{y}$. Here, the only thing random is our prior belief in the coefficients $\xi$. If they obey i.i.d. normal distributions, $\theta$ is linear, and we can accurately observe $y$ and $\dot{y}$, then the loss may indeed be normally distributed according to $\dot{y}|y \sim \mathcal{N}(\xi^T \theta)$. However, even with perfect observations, as soon as $\theta$ is nonlinear, the distribution is no longer normal, leading to

an estimator that may differ significantly from SINDy. The second stochastic process reflects random forcing. If the time between observations is very short, the distribution of $x_t|x_{t-1}$ may be close to normal. However, if the dynamics are nonlinear (e.g. $\theta$ reflects a double-well potential), the distribution will be non-normal. The addition of randomness in $\xi$ makes a likelihood expression for this process even more difficult. Finally, the third expression reflects a random field. With perfect measurements, and modeling $\widetilde{\theta}$ as a gaussian process equipped with a kernel, it would give rise to an expression similar to neural operators. We hope to explore these generalizations of the problem in future work.

# Chapter 5

# PYSINDY PROJECT MANAGEMENT

Make haste slowly

---

Benjamin Franklin (as Richard Saunders), An Almanack For the Year of Christ 1744

The final chapter is about research engineering contributions around pysindy: One lesson in the pysindy chapter, 2, was that APIs matter because they allow durable systems to be built on top. In the first section, I discuss the experiment as a code object. Plenty of people use jupyter notebooks, and plenty more use hard-coded scripts. Many of same people understand the value of publishing methods as a library. I argue that another level of useful abstraction exists between the experiment and the end user: The end user interacts with experiment-running software, and the experiment-running software handles all of the experiment administrative tasks. I describe the value it brings in the next section, where I introduce the mitosis tool I've developed, and the pysindy-experiments package, a package for benchmarking of dynamical systems.

The second section describes plans for future development of the library. It takes the lessons learned in the API chapter, 2, and considers the ways to decrease the maintenance burden, determine the most valueable features to add in order to support continued research.

## *5.1 Mitosis and the Experiment API*

> The natural and spontaneous action of the mind is suspect... our only remaining hope and salvation is to begin... using mechanical aid.
>
> Francis Bacon, *Novum Organum*

.

Beyond the method API, perhaps the next most critical piece of lab software infrastructure is the experiment API. The goal of experiments is to convince someone: the researcher running the experiment, collaborators, or the greater community. In this section, I present a tool developed during the projects of this defense designed to manage computational experiments and give examples of its use throughout the papers in this defense. This tool simplifies the process of running and documenting research, making tracking experiments require next to no mental overhead or effort Moreover, its development shed light on the qualities that make an experiment convincing. One widely-recognized quality is *reproducibility*, but the best experiments are much more than reproducible. The chapter then discusses additional desiderata and how mitosis provides them. It concludes with a brief description of pysindy—experiments, the benchmark experimental companion to pysindy

Mitosis [86] is a python package that handles all the admin of running an experiment. That admin includes tracking the version of an experiment and all its dependencies, along with the visual artifacts (e.g. plots), the processed or intermediate data, any metrics calculated, logs emitted, and the particular parameterization, as described in figure 5.1. Named parameterizations would ideally be serialized to a database in a way that equality on disk implied equality of the associated runtime values, and vice versa. [1] This property would restrict the types of python objects tracked in a database too much; mitosis chooses a bal-

[1] For an example of where the difficulty arises, consider unordered collections, or stateful objects that reflect the same system resource under different conditions. (like a connection that can be open or closed).

anced compromise: equal objects can result in different serializations; but different objects cannot result in the same serialization without intentional subversion. These choices only slightly limit which objects can be parameters (e.g. lambda functions, objects without a __dict__ attribute), while practically removing the chance of spuriously changed results. Tor instance, these checks mean that if the "low-noise" variant of an experiment specifies values for the standard deviation, data duration, and sampling interval, those remain the same for every trial of the experiment with "low-noise". Users also have the option to smartly opt-out of checking individual parameters, e.g. for plot preferences or CPU/GPU device choice.

mitosis is invoked via the command line. In any given project, different experiments or experiment steps are specified in the pyproject.toml along with the dictionary to look up named parameterizations. An experiment step is a callable with a certain signature: optionally, an argument to accept data from a previous step, and returning a particular typed dictionary with metrics and record data. Along the way, different plotting functions will be called and output captured. Different experiments can be composed so long as the return type of one is the correct input type to another. The experimental repository does not require any explicit dependence upon mitosis. This strict dependency inversion not only allows mitosis to be so lightweight, but also allows experiments to be ran with different experiment runners. In fact, neither the experiment nor the experiment runner nor the methods packages need any explicit dependence on each other. Instead, I advocate a separate package, focused on a particular problem set or investigation contain the dependencies on all three. This relationship is depicted in figure 5.1.

**Experimental Desiderata** The workflow supported by mitosis provides more than reproducibility. The most obvious advantage to having an experiment runner is that the iterations of rerunning experiments, regenerating figures, and recording metadata, as the final bugs and tweaks are fixed, are easier and verifiable. As [41] emphasizes, "If the process takes any more than one step, it is prone to errors." Another desiderata is *reusability*. That is to say, if I can borrow the experiment and test out a different intervention, I find

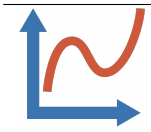
| | |
|---|---|
|  | Sqlite Database associating trials with their git commit, file location, and variants of parameters. |
|  | Metadata folder for each experiment containing all of the following: |
|  | Environment file for recreating dependency environment. |
|  | Source code to rerun the exact same experiment outside mitosis. |
|  | All of the printed and drawn output from the experiment, including plots. Saved as a static jupyter html notebook. The source code should be able to recreate these plots. |
|  | Pickled output data from the experiment. |
|   | All of the INFO-level (and optionally, DEBUG-level) logs from the experiment and its dependnencies. |

Table 5.1: Mitosis produces a variety of files to support reproducibility and other desiderata.

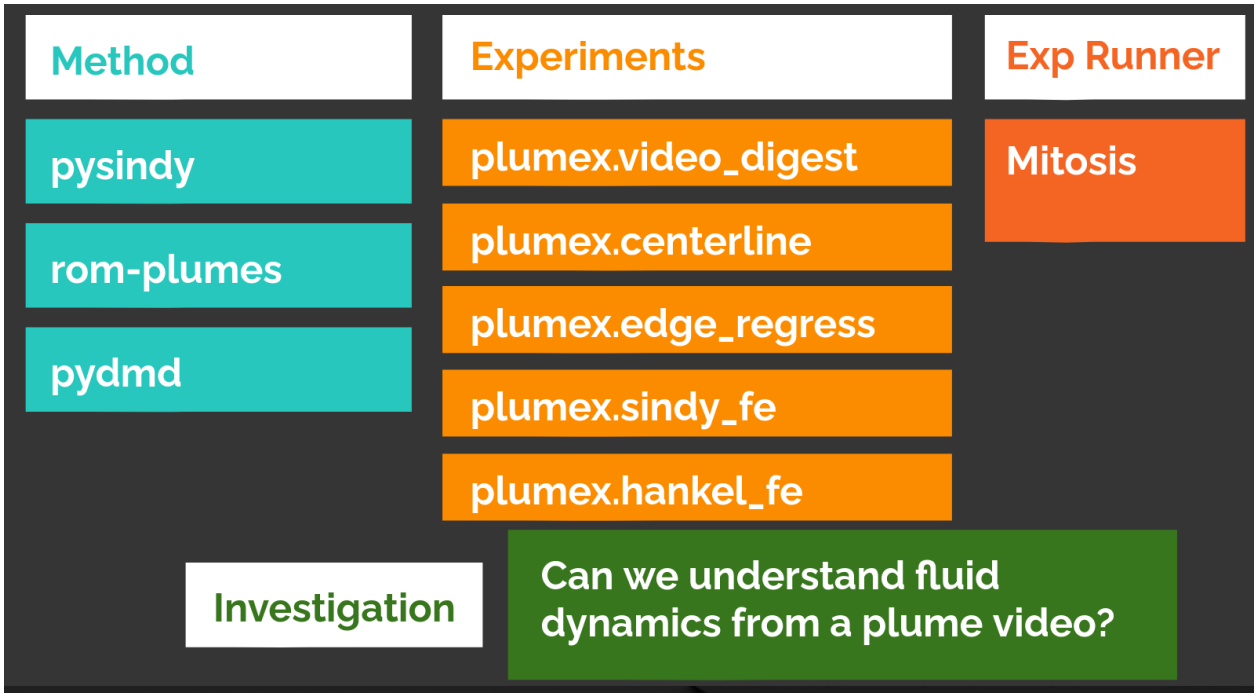



Figure 5.1: A depiction of the project organization for the plumes project. This consists of a package to process plume videos into various reduced-order models, a package to run experiments to evaluate different parameterizations of those reduced order models, a copy of mitosis, and a project that represents the investigations in the paper and presentation, specifying the particular choices of parameters to investigate. This layout was used for several hundred invocations of experiments as we improved and debugged the experiments and identified the best parameterizations.

the experiment more convincing. Here, reusability is provided merely by the experiment's parametrization, and shipping it as a standalone API. While many experiments are neither *reproducible* nor reusable, I argue that one can ask two more desiderata of experiments: that they are *describable*, and that they are *composable*. The former refers to a vernacular for its parameterization and a single source of truth for what that vernacular means. Mitosis contributes this by diligently checking all named variants against a database, with appropriate considerations for repeatable serialization. The dictionary definition of vernacular is a semi-declarative syntax, as shown in figure 5.2. This figure shows value of a declarative syntax: its simple to see what someone is experimenting on by reviewing the diff of named

```
diff --git a/images/gen_image/ksindy_figs/config.py b/images/gen_image/ksindy_figs/config.py
index e37938e..1c41f1d 100644
--- a/images/gen_image/ksindy_figs/config.py
+++ b/images/gen_image/ksindy_figs/config.py
@@ -95,6 +95,7 @@ sim_params = {
     "test-r1": ND({"n_trajectories": 2, "noise_rel": 0.01}),
     "test-r2": ND({"n_trajectories": 2, "noise_rel": 0.1}),
     "test-r3": ND({"n_trajectories": 2, "noise_rel": 0.3}),
+    "heavy-tail": ND({"n_trajectories": 10, "noise_shape": "student-t"}),
     "10x": ND({"n_trajectories": 10}),
     "10x-r1": ND({"n_trajectories": 10, "noise_rel": 0.01}),
     "10x-r2": ND({"n_trajectories": 10, "noise_rel": 0.05}),
```

Figure 5.2: This diff shows that a user is running experiments with a different noise shape. Moreover, we know that the name "heavy-tail" refers specifically to student's T distribution. When experiment parameters are stored declaratively, the changes made to support new experiments are obvious to all. Good refactoring isolates the places of interesting variation, such as the simulation noise. Anything that makes the code easier to read, in turn, improves the ability for collaborators to communicate via code.

parameterizations. Composability refers to the use of the experiment in some type of series. Methods like pysindy are often used not only for scientific discovery, but also in engineering systems like [93]. Therefore, providing the option of composing the experiment's processing with evaluation of a latter step helps make the experiment more useful to researchers. Similar to reusability, the definition of an experiment as a callable makes it composable as well. Since experiments in mitosis do not need to depend upon mitosis, the avoid any additional obstacles to reusability due to dependencies.

**Pysindy experiments package** In addition to the pysindy package, this dissertation also supplies the pysindy—experiment package [87]. This is a collection of experiments on single-step and traditional SINDy. The intent is to allow these to be used by other researchers writing papers to compare their methods with other SINDy methods. As it becomes more well-developed, the reusability of experiment API may become a force to standardize dynamical systems learning APIs. In other words, providing a new method that obeys the SINDy API grants access to all experiments used to evaluate SINDy problems. This, combined

with the easy declarative naming, provides the blueprint for a true benchmark in dynamical systems research. Future work aims to outsource data generation to the dysts package, [33] which includes excellent work to categorize different chaotic systems used to evaluate dynamical systems.

### *5.2 Development Plan*

The ecosystem of compatible dynamical systems learning code makes incremental improvement more powerful. Progress can unlock new combinations of methods for other researchers, or facilitate research well beyond an initial circle of collaborators.. Planning work is an important contribution to the pysindy lab management, A correct or well-thought out bug fix or new capability can improve dozens of related project at once, whearas a faulty patch can various previously published research and require effort of another researcher to fix. This section presents five major innovations planned for pysindy that will imporve most users' dynamical systems research. The first four improvements are lingering problems with the type system, where as discussed in section 2.2, the goal is to use the type system to enforce mathematical compatibility. The final probp pysindy aims to continue to reduce its maintenance burden: the cost of understanding and modifying code.

**Typing** The differentaition API is the most challenging and rewarding to fix. As pysindy imports a dedicated package for differentiating, it does not itself need to be in the business of differentiation. Every method that it implements is also implemented by derivative. Moreover, the static type of differentiation has several holes. The first is that smoothing in n-dimensions is different than smoothing in each dimension in turn. That is to say, calculating a Savitzky-Golay filter in space and then in time gives rise to derivatives estimated from different points. The second hole in differentiation typing is the ability to specify the nth order of derivative. Not all methods allow arbitrary order derivatives, and those methods may not be useful for spatial derivatives. Moreover, methods that could provide different orders of derivative do not necessarily provide a consistent signature that allows it. These

limitations could entirely the cause the apparent difficulty handling noise in PDEs. Finally, a type system that allowed differentiation methods to provide their loss function, rather than the smooth coordinates, would allow implementation in a TrajectoryInterpolant subclass and thus evaluation in a joint optimization problem. Clarifying a Liskov-safe type system that addresses all internal needs would help get pysindy out of the business of differentiation and smoothing. Ultimately, this reduced maintenance burden would allow people to use different implemenations, such as the pynumdiff library, based upon the hyperparameter theory of [12].

Another error in the type system is the SINDy object itself. The implementation is rife with conditional checks for Weak, SINDY-PI, and discrete cases. These conditional checks follows the "if-statement-dodge" antipattern in [75]. Weak SINDy cannot simulate, SINDy-PI cannot print equations, and SINDy internal logic differs when consitutent objects indicate one of those approaches. The solution is to promote the Weak and discrete cases from the SINDy object, and instead handle such problems as sibling classes with common functionality extracted into an appropriate BaseSINDy class. Doing so would allow simulating weak models with no additional engineering. Elevating SINDy-PI to its own class that contains SINDy models, or at least contains various optimizers, would allow it its own grammar for interacting with the list of rational functions, SINDy-PI also requires a specific choice of feature library and optimizer. Promoting SINDy-PI to a composite object would allow users access to all the existing sparse optimizers, and remove the code from PDE library that is not specific to spatial derivatives.

Similarly, splitting the BaseOptimizer class into and abstraction for regression setup and one for optimization approach may allow combining different approaches that were not previously amenable to SINDy. It might also simplify the space of decisions that a user needs to handle.

Finally, the last typing improvement is a challenge: handling arrays in a package-agnostic manner, especially in the feature libraries. IOn an ideal world, SINDy operations would preserve types for jax arrays, numpy arrays, and CVXPy atoms. This change would allow

users to build additional models on top of pysindy components, but retain access to all the more powerful tools of their preferred array implementation. A common case-in-point are the autoencoder SINDy methods of [20, 32]. This is easier than it may seem: Although the Python array API standard, [72], is limited, many packages publish a much larger compatible API.

**Additional Features** Along with improving existing type system, several additional features would help accelerate research. The first of these is solution analysis functionality. Adding facilities to pysindy-experiments to assess the performance and internal behavior of sindy models after fitting would allow more detailed experimental results. Some of those facilities could be borrowed from related methods, e.g. to extracting the eigenvalues of linear components of a SINDy model would help comparisons with DMDs. Pysindy-experimetns should also promote experiments from relevant literature, aiding reproducion of results and allowing easy comparison.

**Maintainability** Finally, in order to build the developer base, better documentation is needed. This involves clearer developer documentation: what are the internal types in pysindy and how to contribute. Along the way, better documentation means removing the legacy experiment notebooks and cleaning the git history to shrink the repository, as described in table 2.4. The goals of improved documentation, type safety, and experimental facilities will empower researchers to more deeply investigate innovations with less effort and more rapidity.

## *5.3 Conclusion*

The contributions in this chapter have broader lessons for research engineering generally. Much has been written about research engineering: commentary has aptly noted the fundamentals, such as project organization and good practice required, but misses many aspects of research project management. But the lessons from this chapter are not another “How

to code for researchers" but rather "software as part of a lab" Open-source research project management has relatively little written doctrine.

The first conclusion to draw is that Software engineering is an essential aspect of peer communication in research, not merely a technical requirement. As [36] point out, "One challenge to investing in improved software practices and processes for science is the perception by some that a focus on improving software skills falls under the category of engineering, not science". I argue that it is as much a part of science as peer communication and lab management is part of science. Open source provides rich faculties for that communication. If written well, nuances that are missing from a paper are detailed in the code. Code review can be as robust as journal review, as demonstrated in section 3.2. Stars, forks, comments, and PRs all catalogue the collaborative impact of a group of authors.

The corollary of "code as communication" is that research engineering supports a more incremental form of research. It is always easier to improve existing methods when there is some familiar code. An open-source API takes that to its logical conclusion. Individual improvements are harder to achieve, because of the requirements for compatibility. However, each improvement provides greater value based upon the compatible ecosystem: capabilities using a common API can synthesize with the improvement, and existing experiments that researchers can use to evaluate the improvement. In addition, the reuse of experiments across innovations promises to raise the standard of proof for claims.

The final conclusion to draw is that an open source research project blurs the traditional distinction of a lab. Consider a lab to be a group of occasional collaborators who communicate, share infrastructure, and study similar problems and methods. Code is infrastructure. Good code allows a lab to be productive, and records communication between collaborators. For labs that have little physical infrastructure, software is potentially *the most expensive* investment it makes. [2] The experience of software engineering is such that success depends

[2]Funding is often part of a lab as well. Sources of funding for research engineering include the regular research funding as well as nontraditional funding from organizations that sponsor open-source code. The latter often come with nontraditional governance requirements to guarantee the investment will not wither with a single author's interest. Funding management for open-source research is beyond the scope of this

not just upon tools, or even upon work flow, but upon organization and project management. And thusly research software engineering becomes lab management.

---

paper, however.

# BIBLIOGRAPHY

[1] A. Aravkin, J. V. Burke, and G. Pillonetto. "Robust and trend-following kalman smoothers using student's t." *IFAC Proceedings Volumes* **45**.16 (2012), pp. 1215–1220.

[2] A. Aravkin *et al.* "Generalized kalman smoothing: modeling and algorithms." *Automatica* **86** (2017), pp. 63–86.

[3] F. Auger *et al.* "Industrial applications of the kalman filter: a review." *IEEE Transactions on Industrial Electronics* **60** (12 2013), pp. 5458–5471. ISSN: 02780046.

[4] R. Baraldi, R. Kumar, and A. Aravkin. "Basis pursuit denoise with nonsmooth constraints." *IEEE Transactions on Signal Processing* **67**.22 (Nov. 2019). Conference Name: IEEE Transactions on Signal Processing, pp. 5811–5823. ISSN: 1941-0476. URL: https://ieeexplore.ieee.org/document/8861392/?arnumber=8861392 (visited on 03/10/2025).

[5] S. T. Barratt and S. P. Boyd. "Fitting a kalman smoother to data." *American Automatic Control Conference*. (July 2020).

[6] P. Batlle *et al.* "Kernel methods are competitive for operator learning." *Journal of Computational Physics* **496** (Jan. 1, 2024), p. 112549. ISSN: 0021-9991. URL: https://www.sciencedirect.com/science/article/pii/S0021999123006447 (visited on 10/01/2024).

[7] D. Bertsimas and W. Gurnee. "Learning sparse nonlinear dynamics via mixed-integer optimization." *Nonlinear Dynamics* **111**.7 (Apr. 1, 2023), pp. 6585–6604. ISSN: 1573-269X. URL: https://doi.org/10.1007/s11071-022-08178-9 (visited on 11/25/2024).

[8] S. A. Billings. In: *Nonlinear System Identification: NARMAX Methods in the Time, Frequency, and Spatio-Temporal Domains*. John Wiley & Sons, Ltd, 2013, pp. i–xvii. ISBN: 9781118535561. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118535561.fmatter. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118535561.fmatter.

[9] E. Bingham *et al.* "Pyro: deep universal probabilistic programming." *J. Mach. Learn. Res.* **20** (2019), 28:1–28:6. URL: http://jmlr.org/papers/v20/18-403.html.

[10] L. Boninsegna, F. Nüske, and C. Clementi. "Sparse learning of stochastic dynamical equations." *Journal of Chemical Physics* **148** (24 June 2018). ISSN: 00219606.

[11] G.-J. Both *et al.* "DeepMoD: deep learning for model discovery in noisy data." *Journal of Computational Physics* **428** (Mar. 1, 2021), p. 109985. ISSN: 0021-9991. URL: https://www.sciencedirect.com/science/article/pii/S0021999120307592 (visited on 02/19/2025).

[12] F. van Breugel, J. N. Kutz, and B. W. Brunton. "Numerical differentiation of noisy data: a unifying multi-objective optimization framework." *IEEE Access* **8** (2020), pp. 196865–196877. ISSN: 21693536.

[13] F. J. Brooks. *Mythical Man-Month, The: Essays on Software Engineering*. Anniversary Edition. Addison-Wesley Professional, Aug. 1995.

[14] S. L. Brunton and J. N. Kutz. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2022.

[15] S. L. Brunton, J. L. Proctor, and J Nathan Kutz. "Discovering governing equations from data by sparse identification of nonlinear dynamical systems." *Proceedings of the National Academy of Sciences* **113**.15 (2016).

[16] J. L. Callaham *et al.* "Nonlinear stochastic modelling with langevin regression." *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*

**477**.2250 (June 2, 2021). Publisher: Royal Society, p. 20210092. URL: https://royalsocietypublishing.org/doi/10.1098/rspa.2021.0092 (visited on 09/24/2024).

[17] E. Candes and T. Tao. "Decoding by linear programming." *IEEE Transactions on Information Theory* **51**.12 (Dec. 2005). Conference Name: IEEE Transactions on Information Theory, pp. 4203–4215. ISSN: 1557-9654. URL: https://ieeexplore.ieee.org/document/1542412 (visited on 02/19/2025).

[18] C. M. Carvalho, N. G. Polson, and J. G. Scott. "Handling sparsity via the horseshoe" (2009).

[19] K. Champion *et al.* "A unified sparse optimization framework to learn parsimonious physics-informed models from data." *IEEE Access* **8** (2020), pp. 169259–169271. ISSN: 21693536.

[20] K. Champion *et al.* "Data-driven discovery of coordinates and governing equations." *Proceedings of the National Academy of Sciences* **116**.45 (Nov. 5, 2019). Publisher: Proceedings of the National Academy of Sciences, pp. 22445–22451. URL: https://www.pnas.org/doi/10.1073/pnas.1906995116 (visited on 03/24/2025).

[21] R. Chartrand, L Marin, and D Xiao. "Numerical differentiation of noisy, nonsmooth data." *International Scholarly Research Network ISRN Applied Mathematics* **2011** (2011), p. 11.

[22] Y. Chen *et al.* "Solving and learning nonlinear PDEs with gaussian processes." *Journal of Computational Physics* **447** (Dec. 15, 2021), p. 110668. ISSN: 0021-9991. URL: https://www.sciencedirect.com/science/article/pii/S0021999121005635 (visited on 10/01/2024).

[23] S. I. Chernyshenko *et al.* "Polynomial sum of squares in fluid dynamics: a review with a look ahead." *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* **372**.2020 (2014), p. 20130350.

[24] M. Dam *et al.* "Sparse identification of a predator-prey system from simulation data of a convection model." *Physics of Plasmas* **24** (2 Feb. 2017), p. 022310. ISSN: 1070-664X. URL: https://aip.scitation.org/doi/abs/10.1063/1.4977057.

[25] I. P. Duff, P. Goyal, and P. Benner. "Stability-certified learning of control systems with quadratic nonlinearities." *arXiv preprint arXiv:2403.00646* (2024).

[26] D. K. Duvenaud. "Automatic model construction with gaussian processes" ().

[27] M. L. Eaton. *Multivariate Statistics: A Vector Space Approach*. Vol. 53. 905 W. Main Street, Suite 18B. Durham, NC: Institute of Mathematical Statistics, 2007, pp. 116,117. ISBN: 9780940600690.

[28] Fangzheng Sheng *et al.* "PiSL: physics-informed spline learning for data-driven identification of nonlinear dynamical systems." *Mechanical Systems and Signal Processing* **191** (May 15, 2023), p. 110165. ISSN: 0888-3270. URL: https://www.sciencedirect.com/science/article/abs/pii/S0888327023000729 (visited on 01/29/2025).

[29] U. Fasel *et al.* "Ensemble-SINDy: robust sparse model discovery in the low-data, high-noise limit, with active learning and control." *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* **478**.2260 (Apr. 13, 2022). Publisher: Royal Society, p. 20210904. URL: https://royalsocietypublishing.org/doi/full/10.1098/rspa.2021.0904 (visited on 11/12/2024).

[30] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Second. Boston: Addison-Wesley Professional, 2018.

[31] E. Gamma *et al. Design Patterns: Elements of Reusable Object-Oriented Software*. Boston: Addison-Wesley Professional, 1994.

[32] L. M. Gao and J. N. Kutz. "Bayesian autoencoders for data-driven discovery of coordinates, governing equations and fundamental constants" (Nov. 2022). URL: http://arxiv.org/abs/2211.10575.

[33] W. Gilpin. *Chaos as an interpretable benchmark for forecasting and data-driven modelling*. 2023. URL: https://github.com/williamgilpin/dysts.

[34] P. Goyal, I. P. Duff, and P. Benner. "Guaranteed stable quadratic models and their applications in SINDy and operator inference." arXiv:2308.13819 (Jan. 7, 2024). arXiv: 2308.13819[cs,math]. URL: http://arxiv.org/abs/2308.13819 (visited on 05/07/2024).

[35] Y. Guan, S. L. Brunton, and I. Novosselov. "Sparse nonlinear models of chaotic electroconvection." *Royal Society Open Science* **8** (8 Aug. 2021). ISSN: 20545703.

[36] M. A. Heroux. "Research software science: expanding the impact of research software engineering." *Computing in Science & Engineering* **24**.6 (Nov. 2022). Conference Name: Computing in Science & Engineering, pp. 22–27. ISSN: 1558-366X. URL: https://ieeexplore.ieee.org/document/10078171/?arnumber=10078171 (visited on 08/01/2024).

[37] S. M. Hirsh, D. A. Barajas-Solano, and J. N. Kutz. "Sparsifying priors for bayesian uncertainty quantification in model discovery." *Royal Society Open Science* **9** (2 2022). ISSN: 20545703. URL: https://doi.org/10.1098/rsos.211823.

[38] M. Hoffmann, C. Fröhner, and F. Noé. "Reactive sindy: discovering governing reactions from concentration data." *Journal of Chemical Physics* **150** (2 Jan. 2019). ISSN: 00219606.

[39] D. Huang *et al.* "Sum-of-squares of polynomials approach to nonlinear stability of fluid flows: an example of application." *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* **471**.2183 (2015), p. 20150622.

[40] Joel Spolsky. *Making Wrong Code Look Wrong*. Joel on Software. May 11, 2005. URL: https://www.joelonsoftware.com/2005/05/11/making-wrong-code-look-wrong/ (visited on 11/27/2024).

[41] Joel Spolsky. *The Joel Test: 12 Steps to Better Code*. Joel on Software. May 11, 2005. URL: https://www.joelonsoftware.com/2000/08/09/the-joel-test-12-steps-to-better-code/ (visited on 03/23/2025).

[42] J. Jonker, P. Zheng, and A. Y. Aravkin. "Efficient robust parameter identification in generalized kalman smoothing models." *IEEE Transactions on Automatic Control* **66**.10 (2020), pp. 4852–4857.

[43] J. Jonker *et al.* "Fast robust methods for singular state-space models." *Automatica* **105** (2019), pp. 399–405.

[44] Joseph Bakarji and Daniel M. Tartakovsky. "Data-driven discovery of coarse-grained equations - ScienceDirect." *Journal of Computational Physics* **434** (2021), p. 110219. ISSN: 0021-9991. URL: https://www.sciencedirect.com/science/article/pii/S0021999121001145 (visited on 02/18/2025).

[45] K. Kaheman, J. N. Kutz, and S. L. Brunton. "Sindy-pi: a robust algorithm for parallel implicit sparse identification of nonlinear dynamics." *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* **476**.2242 (2020), p. 20200279. eprint: https://royalsocietypublishing.org/doi/pdf/10.1098/rspa.2020.0279. URL: https://royalsocietypublishing.org/doi/abs/10.1098/rspa.2020.0279.

[46] K. Kaheman, J. N. Kutz, and S. L. Brunton. "SINDy-PI: a robust algorithm for parallel implicit sparse identification of nonlinear dynamics." *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* **476**.2242 (Oct. 7, 2020). Publisher: Royal Society, p. 20200279. URL: https://royalsocietypublishing.org/doi/10.1098/rspa.2020.0279 (visited on 02/21/2025).

[47] E. Kaiser, J. N. Kutz, and S. L. Brunton. "Sparse identification of nonlinear dynamics for model predictive control in the low-data limit." *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* **474** (2219 Nov. 2018). ISSN: 14712946. URL: http://dx.doi.org/10.1098/rspa.2018.0335.

[48] R. S. Kandezy and J. N. Jiang. *Mixed Algorithm of SINDy and HAVOK for Measure-Based Analysis of Power System with Inverter-based Resources*. Mar. 14, 2024. arXiv: 2403.09536[eess]. URL: http://arxiv.org/abs/2403.09536 (visited on 02/21/2025).

[49] A. Kaptanoglu *et al.* "Pysindy: a comprehensive python package for robust sparse system identification." *Journal of Open Source Software* **7** (69 Jan. 2022), p. 3994.

[50] A. A. Kaptanoglu *et al.* "Benchmarking sparse system identification with low-dimensional chaos." *Nonlinear Dynamics* **111** (14 Feb. 2023), pp. 13143–13164. ISSN: 1573269X. URL: https://arxiv.org/abs/2302.10787v1.

[51] A. A. Kaptanoglu *et al.* "Promoting global stability in data-driven models of quadratic nonlinear dynamics." *Physical Review Fluids* **6** (2021), p. 94401.

[52] A. A. Klishin *et al. Statistical Mechanics of Dynamical System Identification*. Feb. 4, 2025. arXiv: 2403.01723[cond-mat]. URL: http://arxiv.org/abs/2403.01723 (visited on 02/18/2025).

[53] T. Koike and E. Qian. "Energy-Preserving Reduced Operator Inference for Efficient Design and Control." In: *AIAA SCITECH 2024 Forum*. 2024, p. 1012.

[54] B. Kramer. "Stability domains for quadratic-bilinear reduced-order models." *SIAM Journal on Applied Dynamical Systems* **20**.2 (2021), pp. 981–996.

[55] S.-C. Liao *et al.* "A convex optimization approach to compute trapping regions for lossless quadratic systems." *arXiv e-prints* (2024), arXiv–2401.

[56] B. Liskov. "Keynote address — data abstraction and hierarchy." In: *OOPSLA '87: Addendum to the Proceedings on Object-oriented Programming Systems, Languages and Applications (Addendum)*. A keynote address in which Liskov first formulated the principle. 1987, pp. 17–34. ISBN: 0897912667.

[57] J.-C. Loiseau. "Data-driven modeling of the chaotic thermal convection in an annular thermosyphon." *Theoretical and Computational Fluid Dynamics* **34**.4 (Aug. 1, 2020), pp. 339–365. ISSN: 1432-2250. URL: https://doi.org/10.1007/s00162-020-00536-w (visited on 02/19/2025).

[58] J.-C. Loiseau and S. L. Brunton. "Constrained sparse galerkin regression." *Journal of Fluid Mechanics* **838** (Mar. 2018), pp. 42–67. ISSN: 0022-1120, 1469-7645. URL: https://www.cambridge.org/core/journals/journal-of-fluid-mechanics/article/abs/constrained-sparse-galerkin-regression/0E18A4A55FF5AC1401D236C0E4D1CAAE (visited on 02/18/2025).

[59] D. Long *et al. A Kernel Approach for PDE Discovery and Operator Learning*. 2023. arXiv: 2210.08140 [stat.ML]. URL: https://arxiv.org/abs/2210.08140.

[60] D. Long *et al.* "A kernel framework for learning differential equations and their solution operators." *Physica D: Nonlinear Phenomena* **460** (Apr. 2024), p. 134095. ISSN: 01672789. URL: https://linkinghub.elsevier.com/retrieve/pii/S0167278924000460 (visited on 06/26/2024).

[61] K. Maass, M. Kim, and A. Aravkin. "A nonconvex optimization approach to IMRT planning with dose–volume constraints." *INFORMS Journal on Computing* **34**.3 (May 2022), pp. 1366–1386. ISSN: 1091-9856, 1526-5528. URL: https://pubsonline.informs.org/doi/10.1287/ijoc.2021.1129 (visited on 03/10/2025).

[62] N. M. Mangan *et al.* "Inferring biological networks by sparse identification of nonlinear dynamics." *IEEE Transactions on Molecular, Biological, and Multi-Scale Communications* **2**.1 (June 2016). Conference Name: IEEE Transactions on Molecular, Biological, and Multi-Scale Communications, pp. 52–63. ISSN: 2332-7804. URL: https://ieeexplore.ieee.org/document/7809160/?arnumber=7809160 (visited on 02/18/2025).

[63] M. Mendoza *et al. PEP 646: Variadic Generics*. 2020. URL: https://peps.python.org/pep-0646/ (visited on 11/18/2024).

[64] D. A. Messenger and D. M. Bortz. "Weak sindy for partial differential equations." *Journal of Computational Physics* **443** (Oct. 2021). ISSN: 10902716.

[65] D. A. Messenger and D. M. Bortz. "Weak sindy: galerkin-based data-driven model selection." *Multiscale Modeling and Simulation* **19** (3 2021), pp. 1474–1497. ISSN: 15403467. URL: https://doi.org/10.1137/20M1343166.

[66] D. A. Messenger and D. M. Bortz. "Weak sindy: galerkin-based data-driven model selection." *Multiscale Modeling and Simulation* **19** (3 2021), pp. 1474–1497. ISSN: 15403467. URL: https://doi.org/10.1137/20M1343166.

[67] Z. G. Nicolaou *et al.* "Data-driven discovery and extrapolation of parameterized pattern-forming dynamics." *Phys. Rev. Res.* **5** (4 Nov. 2023), p. L042017. URL: https://link.aps.org/doi/10.1103/PhysRevResearch.5.L042017.

[68] S. Ouala *et al.* "Bounded nonlinear forecasts of partially observed geophysical systems with physics-constrained deep learning." *Physica D: Nonlinear Phenomena* **446** (2023), p. 133630.

[69] M. Peng *et al.* "Local stability guarantees for data-driven quadratically nonlinear models." *Journal of Fluid Mechanics* ().

[70] T. Peters. *PEP 20: The Zen of Python*. 2004. URL: https://peps.python.org/pep-0020/ (visited on 11/18/2024).

[71] J. Piironen and A. Vehtari. "Sparsity information and regularization in the horseshoe and other shrinkage priors." *Electronic Journal of Statistics* **11** (2 2017), pp. 5018–5051. ISSN: 19357524.

[72] *Python array API standard — Python array API standard 2024.12 documentation*. URL: https://data-apis.org/array-api/latest/ (visited on 03/24/2025).

[73] C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. 3. print. Adaptive computation and machine learning. Cambridge, Mass.: MIT Press, 2008. 248 pp. ISBN: 978-0-262-18253-9.

[74] E. S. Raymond. *The cathedral and the bazaar: musings on Linux and open source by an accidental revolutionary*. eng. 2., überarb. und erw. A. With a foreword by Bob Young. Beijing; Cambridge; Farnham; Köln; Paris; Sebastopol; Taip: O'Reilly Media, 2001, p. 241. ISBN: 0-596-00108-8. URL: http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/.

[75] B. Rhodes. *Python Design Patterns*. 2018. URL: https://python-patterns.guide/ (visited on 11/18/2024).

[76] Rohit Supekara,b *et al.* "Learning hydrodynamic equations for active matter from particle simulations and experiments." *Proceedings of the Natural Academy of Sciences* **120**.7 (Apr. 25, 2022). URL: https://www.pnas.org/doi/epub/10.1073/pnas.2206994120 (visited on 02/18/2025).

[77] L. Rosafalco *et al.* "EKF-SINDy: empowering the extended kalman filter with sparse identification of nonlinear dynamics." *Computer Methods in Applied Mechanics and Engineering* **431** (Nov. 2024), p. 117264. ISSN: 00457825. arXiv: 2404.07536[math]. URL: http://arxiv.org/abs/2404.07536 (visited on 02/21/2025).

[78] C. W. Rowley, T. Colonius, and R. M. Murray. "Model reduction for compressible flows using pod and galerkin projection." *Physica D* **189** (2004), pp. 115–129.

[79] L. I. Rudin, S. Osher, and E. Fatemi. "Nonlinear total variation based noise removal algorithms." *Physica D* **60** (1992), pp. 259–268.

[80] S. H. Rudy, S. L. Brunton, and J. N. Kutz. "Smoothing and parameter estimation by soft-adherence to governing equations." *Journal of Computational Physics* **398** (2019), p. 108860. URL: www.elsevier.com/locate/jcp.

[81] S. H. Rudy *et al.* "Data-driven discovery of partial differential equations." *Science Advances* **3**.4 (2017), e1602614. eprint: https://www.science.org/doi/pdf/10.1126/sciadv.1602614. URL: https://www.science.org/doi/abs/10.1126/sciadv.1602614.

[82] M. Schlegel and B. R. Noack. "On long-term boundedness of galerkin models." *Journal of Fluid Mechanics* **765** (Feb. 25, 2015), pp. 325–352. ISSN: 0022-1120, 1469-7645. URL: https://www.cambridge.org/core/product/identifier/S0022112014007368/type/journal_article (visited on 05/07/2024).

[83] B. Schölkopf, R. Herbrich, and A. J. Smola. "A Generalized Representer Theorem." In: *Computational Learning Theory*. Ed. by D. Helmbold and B. Williamson. Berlin, Heidelberg: Springer, 2001, pp. 416–426. ISBN: 978-3-540-44581-4.

[84] Scikit-Learn. *Developing Scikit-Learn Estimators*. Version 1.5.2. 2024. URL: https://scikit-learn.org/dev/developers/develop.html/ (visited on 11/18/2024).

[85] B. de Silva *et al.* "Pysindy: a python package for the sparse identification of nonlinear dynamical systems from data." *Journal of Open Source Software* **5** (49 May 2020), p. 2104.

[86] J. Stevens-Haas. *mitosis*. Version 0.5.1. Apr. 2024. URL: https://github.com/Jacob-Stevens-Haas/mitosis.

[87] J. Stevens-Haas and Y. Bhangale. *pysindy-experiments*. Version 0.1.1. Apr. 2024. URL: https://github.com/Jacob-Stevens-Haas/gen-experiments.

[88] J. M. Stevens-Haas *et al.* "Learning nonlinear dynamics using kalman smoothing." *IEEE Access* **12** (2024), pp. 138564–138574.

[89] D. Voina, S. Brunton, and J. N. Kutz. *Deep Generative Modeling for Identification of Noisy, Non-Stationary Dynamical Systems*. version: 1. Oct. 2, 2024. arXiv: 2410.02079[cs]. URL: http://arxiv.org/abs/2410.02079 (visited on 02/21/2025).

[90] X. Yang *et al.* "Atmospheric chemistry surrogate modeling with sparse identification of nonlinear dynamics." *Journal of Geophysical Research: Machine Learning and Computation* **1**.2 (2024). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1029/2024JH000132, e2024JH000132. ISSN: 2993-5210. URL: https://onlinelibrary.wiley.com/doi/abs/10.1029/2024JH000132 (visited on 02/18/2025).

[91] L. Zanna and T. Bolton. "Data-driven equation discovery of ocean mesoscale closures." *Geophysical Research Letters* **47**.17 (2020). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1029/e2020GL088376. ISSN: 1944-8007. URL: https://onlinelibrary.wiley.com/doi/abs/10.1029/2020GL088376 (visited on 02/19/2025).

[92] P. Zheng *et al.* "A unified framework for sparse relaxed regularized regression: sr3." *IEEE Access* **7** (2018), pp. 1404–1423.

[93] N. Zolman *et al. SINDy-RL: Interpretable and Efficient Model-Based Reinforcement Learning*. Mar. 14, 2024. arXiv: 2403.09110[cs]. URL: http://arxiv.org/abs/2403.09110 (visited on 02/18/2025).

# Appendix A

# EXTRA MATERIAL FOR KALMAN SECTION

| System | ODE | Experiment parameters | $x_0$ mean |
|---|---|---|---|
| Linear Damped Oscillator | $\dot{\vec{x}} = \begin{bmatrix} -\alpha & \beta \\ -\beta & -\alpha \end{bmatrix} \vec{x}$ | $\alpha = .1$, $\beta = 2$ | (0,0) |
| Lorenz | $\dot{\vec{x}} = \begin{bmatrix} \sigma(x_2 - x_1) \\ x_1(\rho - x_3) - x_2 \\ x_1 x_2 - \beta x_3 \end{bmatrix}$ | $\sigma = 10$, $\rho = 28$, $\beta = 8/3$ | (0, 0, 15) |
| Cubic Damped Oscillator | $\dot{\vec{x}} = \begin{bmatrix} -\alpha & \beta \\ -\beta & -\alpha \end{bmatrix} \begin{bmatrix} x_1^3 \\ x_2^3 \end{bmatrix}$ | $\alpha = .1$, $\beta = 2$ | (0,0) |
| Duffing | $\dot{\vec{x}} = \begin{bmatrix} x_2 \\ -\alpha x_2 - \beta x_1 - \gamma x_1^3 \end{bmatrix}$ | $\alpha = .2$, $\beta = .05$, $\gamma = 1$ | (0,0) |
| Hopf | $\dot{\vec{x}} = \begin{bmatrix} -\alpha x_1 - \beta x_2 - \gamma x_1(x_1^2 + x_2^2) \\ \beta x_1 - \alpha x_2 - \gamma x_2(x_1^2 + x_2^2) \end{bmatrix}$ | $\alpha = .05$, $\beta = 1$, $\gamma = 1$ | (0,0) |
| Lotka-Volterra | $\dot{\vec{x}} = \begin{bmatrix} \alpha x_1 - \beta x_1 x_2 \\ \beta x_1 x_2 - 2\alpha x_2 \end{bmatrix}$ | $\alpha = 5$, $\beta = 1$ | (5, 5) |
| Rossler | $\dot{\vec{x}} = \begin{bmatrix} -x_2 - x_3 \\ x_1 + \alpha x_2 \\ \beta + (x_1 - \gamma) x_3 \end{bmatrix}$ | $\alpha = .2$, $\beta = .2$, $\gamma = 5.7$ | (0,0,0) |
| Van der Pol Oscillator | $\dot{\vec{x}} = \begin{bmatrix} x_2 \\ \alpha(1 - x_1^2) x_2 - x_1 \end{bmatrix}$ | $\alpha = .5$ | (0,0) |

Table A.1: The parametrization of ODEs used in these experiments. Mostly from defaults in the pysindy package.

| Parameter | Value |
|---|---|
| *Simulated Data* | |
| Number of trajectories | 10 |
| Initial Condition ($x_0$) variance | 9 |
| Initial Condition ($x_0$) distribution | Normal* |
| Measurement error elative noise (default) | 10% |
| Trajectory duration (default) | 16 |
| Measurement interval | 0.01 |
| Random seed | 19 |
| *SINDy model* | |
| Feature Library ($\Theta$) | Polynomials to degree 3 |
| Optimizer | Mixed Integer Optimizer |
| L2 regularization (coefficitents) ($\alpha$) | 0.01 |
| Target sparsity | (true value from equation) |
| Unbiasing | Yes |
| Feature normalization | No |
| Ensembling | data bagging |
| Number of bags | 20 |
| *Experiment* | |
| Trajectory duration (grid) | 0.5, 1, 2, 4, 8, 16 |
| Relative noise (grid) | 0.05, 0.1, 0.15, 0.2, 0.25, 0.3 |
| Measurement:Process variance (Kalman grid) | 1e-4, 1e-3, 1e-2, 1e-1, 1 |
| L1 regularization (derivative) (TV grid) | 1e-4, 1e-3, 1e-2, 1e-1, 1 |
| Window length (Savitzky-Golay grid) | 5, 8, 12, 15 |
| GCV witheld rate | every fourth observation |
| GCV Tikhonov regularization | 1e-1 |
| All other GCV params default | |

Table A.2: Parametrization of data, SINDy models, and experiments conducted.
*Lotka-Volterra uses a gamma distribution, rather than normal, in order to enforce nonnegativity.

# Appendix B

# OCEAN GLIDER SMOOTHING

*This appendix documents my work on a tangentially-related subject: Seaglider navigation and measurement of the water column. It demonstrates an interesting engineering problem that called for better smoothing, and includes several variations of Kalman or Guass-Markov smoothing. The project also originated the code that later ended up as mitosis.*

# Theoretical Advances in Current Estimation and Navigation from a Glider-Based ADCP



Jacob Stevens-Haas [a], Aleksandr Aravkin [a], Sarah Webster [b]

[a] *Department of Applied Mathematics*
*University of Washington*
*Lewis Hall #201 Box 353925*
*Seattle, WA 98195-3925*

[b] *Applied Physics Laboratory*
*University of Washington*
*1013 NE 40th Street*
*Box 355640*
*Seattle, WA 98105-6698*

**Abstract**

We examine acoustic Doppler current profiler (ADCP) measurements from underwater gliders to determine glider position, glider velocity, and subsurface current. ADCPs do not directly observe the quantities of interest – they measure the relative motion of the vehicle and the water column. A range of past approaches assumes independence of current and state errors. Here, we leverage recent advances to form a joint probability model, extending Kalman smoothing to avoid the independence assumption. Detailed simulations affirm the efficacy of our approach for computing estimates and their uncertainty. The joint model developed here sets the stage for future work to incorporate constraints, range measurements, and robust statistical modeling.

*Email addresses:* `Corresponding author: jacob.stevens.haas@gmail.com. phone: (206) 543-5493, fax: (206) 685-1440` (Jacob Stevens-Haas), `saravin@uw.edu` (Aleksandr Aravkin), `swebster@apl.washington.edu` (Sarah Webster).

## 1 Introduction

In order to map subsurface currents far away from fixed and mobile infrastructure, underwater gliders embark small, 1 MHz acoustic Doppler current profilers (ADCP). These high-resolution profilers measure the relative velocity of a local slice of the water column and depend on both glider velocity and current. Additional measurements including GPS or acoustic beacon range can help with navigation. Using this information along with process models of current variability and vehicle motion makes it possible to separate the relative measurement into its ground-referenced components: the subsurface ocean current profile and vessel navigation velocity. The choice of process smoothing assumptions can have a significant impact on the quality of solutions, and the goal of this paper is to provide improved models for current and vehicle processes.



Most methods for inferring currents and navigational data from ADCP measurements derive from the inversion method of Visbeck (2002). The basic method centers on the linear equation:

$$\mathbf{b} = \mathbf{A}\mathbf{x}, \tag{1}$$

where the vector $\mathbf{b}$ contains the set of ADCP measurements. The state vector, $\mathbf{x}$, represents the desired information as a stacked vector: $\mathbf{x} := \begin{bmatrix} \mathbf{x}_v^T & \mathbf{x}_c^T \end{bmatrix}^T$, with $\mathbf{x}_v$ as the vehicle velocity and $\mathbf{x}_c$ as the ocean currents. The measurement matrix or observation operator, $\mathbf{A}$, selects the appropriate current and vehicle velocity for each ADCP measurement of relative velocity. If $\mathbf{x}_v$ has entries at each time of $n$ measurements and $\mathbf{x}_c$ has entries for each depth of the $n$ measurements, then $\mathbf{A} \in \mathbb{R}^{n \times 2n}$ is row-rank deficient by $n$.

The inversion method can remove $n-1$ degrees of freedom by smoothing $\mathbf{x}_c$ and $\mathbf{x}_v$. These appear as additional $n-1$ rows on $\mathbf{A}$ and $\mathbf{b}$. The inversion method can also bin observations to a time-depth grid with multiple observations per grid point, reducing the size of the state vector and the columns of $\mathbf{A}$. Nevertheless, even after binning and/or smoothing, one degree of freedom remains. [1] The least-squares model requires absolute measurement of either current or vehicle kinematics, in order for $\mathbf{A}$ to have sufficient rank and admit a solution using the pseudoinverse:

$$\mathbf{x} = \mathbf{A}^{\dagger}\mathbf{b} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}. \tag{2}$$

[1] Every row of $\mathbf{A}$ in ADCP measurement or smoothing subtracts two values, which means that any $\mathbf{x}$ that satisfies equation 1 would also be satisfied by $\mathbf{x} + \alpha\mathbf{1}$ for some scalar $\alpha$.

Medagoda et al. (2016b) presents a literature review of methods derived from Visbeck (2002). Variants differ according to how vehicle and current processes are modeled and smoothed, whether the model includes other sensors, and what measurement errors, such as sensor mount misalignment and ADCP bias, the method seeks to control. More complex sensors and vehicle process models, such as the physics-based model and alternative filters of Arnold et al. (2018) and Medagoda et al. (2016a) aim to support propeller-driven AUVs. Gliders, on the other hand, often travel straight, unaccelerated paths and do not have the battery power for additional sensors.

99

**Smoothing.** Two broad approaches exist to smoothing in the literature. Visbeck (2002) and Todd et al. (2017) regularize the second finite difference of current across bins, and optionally do the same for vehicle velocity. Alternatively, one may use a smoother that derives from a continuous stochastic process. The most simple is the Kalman Filter/Smoother, as in Medagoda et al. (2015), Medagoda et al. (2016a) and Jonker et al. (2019), with other variants in Medagoda et al. (2010) and Arnold et al. (2018) employing a physics-based model for vehicle kinematics. Such smoothers have an advantage in that they readily adapt to variable bin sizes steps. Few innovations to the glider inversion method address the current process, with the notable exception of Medagoda et al. (2015) allowing horizontal variation in the water column. However, all authors smoothing current and velocity do so in separate terms, implying that the vehicle's velocity and position) are independent of current. Only Jonker et al. (2019) remarks that vehicle smoothing ought to properly apply to vehicle through-the-water velocity, not over-the-ground velocity. However, their method still smooths over-the-ground velocity.

**Measurements.** Visbeck (2002) offers two mutually-exclusive measurement terms: One can either coerce the depth average of current (DAC) to be zero, calculate it from the difference between GPS displacement and the modeled vehicle displacement, and add it in as a post-processing step. Alternatively, one can use the GPS displacement as a least-squares measurement of the integral of vehicle velocities. Todd et al. (2017) improves upon the former by introducing a hydrodynamic model to estimate vehicle displacement, compares this to GPS displacement, and then includes the difference as a least-squares measurement of DAC. On the other hand, Jonker et al. (2019) adds the hydrodynamic model estimates directly as another least-squares measurement alongside GPS in the overall matrix inversion. Using GPS and hydrodynamic model terms in such a manner precludes the calculations of measurement biases in Todd et al. (2017). However, Medagoda et al. (2015) demonstrates how those measurement bias terms can be added into the state vector to try to achieve the same effect, retaining the single-step least-squares problem. In addition to direct GPS measurements, Medagoda et al. (2016b) also incorporates bottom-tracking Doppler Velocity Log (DVL) and bottom mapping measurements.

**Contributions and Roadmap.** We leverage recent advances to form a joint probability model, extending Kalman smoothing to avoid the independence assumption. To do so, we start with a stochastic model that allows us to derive the proper smoothing covariance between the current and the vehicle's over-the-ground velocity and position. To demonstrate the value of different current process models *ceteris paribus*, we flesh out a minimal model along the lines of Jonker et al. (2019) including ADCP data, hydrodynamic model data, and GPS location fixes (at the start and, optionally, at the end of the dive). We then form the joint optimization problem from the posterior mode of vehicle and current states which, unlike standard inversion, is flexible enough to accommodate nonlinear measurements. Finally, we compare results from simulation using different process models.

The paper proceeds as follows. In Section 2 we describe the general state-space model, derive the covariance between current and vehicle process models, and describe the measurements we include in our model. We also derive higher order process model variants and discuss uncertainty quantification. Section 3 describes simulations and numerical experiments as well as their results. We conclude with a discussion and outline future work.

## 2 Methods

The inversion method is equivalent to maximum likelihood estimation using ADCP measurement terms. This section emphasizes our contribution to the process model priors and forms the complete likelihood formulation for the probability of the state vector, $\Pr(\mathbf{x})$, including measurement errors, of the inversion method. In the likelihood view, the addition of vehicle or current smoothing regularizers represents Bayesian prior knowledge of system behavior.

The process model innovations are compatible with other authors' inversion-method models and their independent innovations. In the least-squares method, any such innovation independent of current is realized as additional rows on the matrix $\mathbf{A}$ or additional states in $\mathbf{x}$ that do not affect the current smoothing terms. Since this paper explores simulated data rather than real world data, the model here eschews important but independent states and measurements that exist elsewhere in the literature such as such as ADCP bias or yaw rate in Medagoda et al. (2015). As a final note, the assumption that preprocessing handles vehicle and geodetic orientation gives us all measurements in cardinal directions. As such, the northward and eastward components of our model comprise two non-interacting problems. When we specify a process or ADCP measurement term, we always mean two such terms: one for the northward component, and one for the eastward component.

We use the vector $\mathbf{x}$ to describe the combined state space of the vehicle and current. The state vector includes:

(1) $\mathbf{x}_v \in \mathbb{R}^{2n}$: vehicle velocity $\mathbf{x}_{\dot{q}}$ and position $\mathbf{x}_q$ on a local Cartesian grid, ordered as $[\dot{x}^1_{\dot{q}}, x^1_q, \ldots, \dot{x}^n_{\dot{q}}, x^n_q]^T$ 101
(2) $\mathbf{x}_c \in \mathbb{R}^m$: current components $[x^1_c, \ldots, x^m_c]^T$

We do not bin observations, so $\mathbf{x}$ has vehicle kinematic entries at all $n$ times an observation occurs and current entries at all $m$ depths an observation occurs. The above vectors represent just one horizontal dimension; double the vector lengths to include northward and eastward components.

The initial model for $\Pr(\mathbf{x})$ assumes current varies by depth and vehicle velocity varies by time as samples of independent Brownian motions, which gives rise to standard Kalman smoothing likelihood expressions. That choice of randomness does not represent a physics or controls process, but rather the structure of our uncertainty around these processes. The Kalman smoother for $\mathbf{x}_v$ specifies the update matrix $\mathbf{G}_v$ and covariance matrix between increments of velocity and position, $\mathbf{Q}_v$. It gives a negative log-likelihood term of:

$$-\ell(\mathbf{x}_v) = \frac{1}{2\sigma_v^2} \|\mathbf{G}_v \mathbf{x}_v\|^2_{\mathbf{Q}_v^{-1}}. \tag{3}$$

where $\sigma_v^2$ is a user-defined covariance scaling. As part of our evaluation, we will evaluate how sensitive the solution is to the correct choice of $\sigma_v^2$.

The matrix $\mathbf{G}_v$ is responsible for creating mean-zero, independent increments from velocity $x^j_{\dot{q}} = W_{t_j}$ and position $x^j_q = \int_0^{t_j} W_r dr$, where $j \in \{1, \ldots, n\}$ and $W$ is a Brownian motion. That is,

$$(\mathbf{G}_v \mathbf{x}_v)^j = \begin{bmatrix} x^j_{\dot{q}} - x^{j-1}_{\dot{q}} | x^{j-1}_{\dot{q}} \\ x^j_q - x^{j-1}_q - \Delta t \cdot x^{j-1}_{\dot{q}} | x^{j-1}_q, x^{j-1}_{\dot{q}} \end{bmatrix}$$

$\mathbf{Q}_v$ is block diagonal, reflecting the independence of $(\mathbf{G}_v \mathbf{x}_v)^j$, with blocks:

$$\text{var}[(\mathbf{G}_v \mathbf{x}_v)^j] = \mathbf{Q}_v^j = \begin{bmatrix} \Delta t_j & \Delta t_j^2/2 \\ \Delta t_j^2/2 & \Delta t_j^3/3 \end{bmatrix}, \tag{4}$$

Appendix A details the complete derivation of $\mathbf{G}_v$ and $\mathbf{Q}_v$.

The current process adds an analogous term, with a few notable differences. While current varies in all spatial directions and in time, all observations on

descent or ascent occur very close to a straight, unaccelerated path. If each direction and time determined an independent, additive Brownian motion for current variation, the line integral of such a path would be equivalent to a one-dimensional scaled Brownian motion. With that in mind, we index our position along that path by depth $s$ rather than time $t$. Vehicle behavior at apogee violates our 1-D path assumption, but for simplicity, we treat the ascent portion as merely continuing the dive. Phrased differently, if we model a dive down to 750 m and back to the surface, we treat it as a single dive down to 1500 m. As a result, our 1-D current model will allow more variance at apogee than a true spatiotemporal model of variance. Finally, the basic model only requires the water velocity and has no need for a current "position." The negative log likelihood contribution for the current process mirrors the one for $\mathbf{x}_v$:



$$-\ell(\mathbf{x}_c) = \frac{1}{2\sigma_c^2}\|\mathbf{G}_c\mathbf{x}_c\|^2_{\mathbf{Q}_c^{-1}}. \tag{5}$$

A covariance scaling, $\sigma_c^2$, completes the term. These process terms add to form:

$$\ell(\mathbf{x}) = \ell(\mathbf{x}_v) + \ell(\mathbf{x}_c) \tag{6}$$

### *2.2 Extensions of Process Model*

#### *2.2.1 Higher-order Smoothing*

Subsection 2.1 describes vehicle over-the-ground velocity and current as Brownian processes. The vehicle's over the ground position then becomes the integral of Brownian motion. Whereas previous authors have described the concomitant regularizer as a smoothing term, strictly speaking, the assumption of Brownian motion implies nonsmoothness. We can assume smoothness if we expand our state space model to include higher order terms: vehicle acceleration and the change in current with respect to depth. These terms then become Brownian. Their integrals, vehicle velocity and current velocity, become smooth.

Now that the current process has more than one order, $\mathbf{x}_{\dot{p}}$ refers specifically to the velocity components of the current process and $\mathbf{x}_{\ddot{p}}$ to the change in velocity with respect to depth. $\mathbf{x}_c$ then contains all the current process terms in the same way that $\mathbf{x}_v$ contains $\mathbf{x}_q$, $\mathbf{x}_{\dot{q}}$, and now $\mathbf{x}_{\ddot{q}}$, vehicle acceleration. The process update matrices, $\mathbf{G}_v$ and $\mathbf{G}_c$, and the covariance matrices, $\mathbf{Q}_v$ and $\mathbf{Q}_c$

also change. The update matrix $\mathbf{G}_v$ is block-banded with blocks $\mathbf{G}_v^j$:

$$\mathbf{G}_v^j = \left[\begin{array}{ccc|c} -1 & 0 & 0 & \\ -\Delta t_1 & -1 & 0 & \mathbf{I}_3 \\ -\frac{{\Delta t_1}^2}{2} & -\Delta t_1 & -1 & \end{array}\right], \mathbf{Q}_v^j = \begin{bmatrix} \Delta t & \frac{\Delta t^2}{2} & \frac{\Delta t^3}{6} \\ \frac{\Delta t^2}{2} & \frac{\Delta t^3}{3} & \frac{\Delta t^4}{8} \\ \frac{\Delta t^3}{6} & \frac{\Delta t^4}{8} & \frac{\Delta t^5}{20} \end{bmatrix}, \tag{7}$$

103

where $\mathbf{I}_3$ is a $3\times3$ identity matrix. The derivation for the higher order smoother is similar to the basic Kalman smoother and is provided in Appendix B.

### *2.2.2 Independence.*

Likelihood expression 6 implies an incorrect assumption: Adding together $\ell(\mathbf{x}_v)$ and $\ell(\mathbf{x}_c)$ implies that the joint probability $\Pr(\mathbf{x}_v, \mathbf{x}_c) = \Pr(\mathbf{x}_v)\Pr(\mathbf{x}_c)$, i.e. that the vehicle's over-the-ground velocity is independent from current. As far as we can tell, all previous authors who modeled over-the-ground vehicle kinematics and smoothed both current and vehicle kinematics made the same assumption *de facto*. Instead, the glider's *through-the-water* velocity is independent of current; its over-the-ground velocity obviously depends upon current, and this dependence becomes more significant in larger currents:

$$\mathbf{x}_{\dot{q}} = \mathbf{x}_{\dot{r}} + \mathbf{x}_c, \tag{8}$$

where $\mathbf{x}_{\dot{r}}$ is the relative velocity of the vehicle through the water.

**Covariance between $\Pr(\mathbf{x}_v)$ and $\Pr(\mathbf{x}_c)$.** The correct likelihood expression relies on the conditional formula,

$$\Pr(\mathbf{x}_q, \mathbf{x}_{\dot{q}}, \mathbf{x}_c) = \Pr(\mathbf{x}_q, \mathbf{x}_{\dot{q}}|\mathbf{x}_c)\Pr(\mathbf{x}_c) \tag{9}$$

$$\ell(\mathbf{x}_q, \mathbf{x}_{\dot{q}}, \mathbf{x}_c) = \ell(\mathbf{x}_q, \mathbf{x}_{\dot{q}}|\mathbf{x}_c) + \ell(\mathbf{x}_c). \tag{10}$$

**Thorem 1** *If vehicle through-the-water velocity is to be smoothed across $n$ time points as a Brownian process in time with variance $\sigma_v^2$, current is to be smoothed as a Brownian process across $m$ depth points with variance $\sigma_c^2$, and depth rate is considered constant between vehicle states, the change in the vehicle's over-the-ground position and velocity, conditioned upon current $(\mathbf{x}_{\dot{q}}, \mathbf{x}_q|\mathbf{x}_c)$ is distributed as:*

$$\mathbf{G}\mathbf{x} \sim \mathcal{N}(0, \sigma_v^2\mathbf{Q}_v)$$

*where* $\mathbf{G}$ *has* $n-1$ *row-blocks* $\mathbf{G}^j \in \mathbb{R}^{2\times(2n+m)}$*:*

$$\mathbf{G}^j = \begin{bmatrix} \dots & -1 & 0 & 1 & 0 \dots \\ \dots & -\Delta t_i & -1 & 0 & 1 \dots \end{bmatrix} \left| \begin{matrix} \\ \mathbf{M} \\ \\ \end{matrix} \right| \begin{bmatrix} \dots \\ \dots \end{bmatrix}, \quad \begin{matrix} & & & \\ x_{\dot{q}}^{j-1} & x_q^{j-1} & x_{\dot{q}}^{j} & x_q^{j} \end{matrix}$$

104

*where* $i(j)$ *indicates the vehicle is at depth* $s_i$ *at time* $t_j$*. Submatrix* $\mathbf{M}$ *incorporates how the vehicle drifts in the current at every depth between* $s_{i(j-1)}$ *to* $s_{i(j)}$*:*

$$\mathbf{M} = \begin{bmatrix} 1 & \dots & -1 \\ \frac{\Delta s_{i(j-1)+1}}{2\dot{s}_j} & \frac{-\Delta s_{k-1}+\Delta s_k}{2\dot{s}_j} & -\frac{\Delta s_{i(j)}}{2\dot{s}_j} \end{bmatrix}, \quad \begin{matrix} x_c^{i(j-1)} & & x_c^{i(j)} \end{matrix}$$

*with* $k \in [i(j-1)+2, i(j)-1]$*.*

$\mathbf{Q}_v$ *is block diagonal with* $n-1$ *blocks:*

$$\mathbf{Q}_v^j = \begin{bmatrix} \Delta t_j & \frac{\Delta t_j^2}{2} \\ \frac{\Delta t_j^2}{2} & \frac{\Delta t_j^3}{3} + \Upsilon_j \end{bmatrix}.$$

*where,*

$$\Upsilon_j = \frac{\sigma_c^2}{\sigma_v^2} \sum_{k=i(j-1)+1}^{i(j)} \frac{\Delta s_k^3}{12\dot{s}_j^2}$$

Thus, when using the proper covariance relationship, the likelihood of the vehicle process is

$$-\ell(\mathbf{x}_v|\mathbf{x}_c) = \frac{1}{2\sigma_v^2}\|\mathbf{G}\mathbf{x}\|^2_{\mathbf{Q}_v^{-1}}. \tag{11}$$

Appendix C provides a step-by-step derivation, including the matrices when $\mathbf{x}_v$ and $\mathbf{x}_c$ smoothed an additional order as in section 2.2.1.

### *2.3 Measurement Terms*

While the process contribution is applicable to all inversion methods, our measurement model is motivated by the Spray glider with a Norktek Signature 1000 ADCP in Jonker et al. (2019). It uses ADCP measurements, a hydrodynamic model estimate of vehicle through-the-water velocity akin to the one in

Todd et al. (2017), and a pair of GPS fixes at either the beginning of the dive or spanning it measure the change in vehicle position.

We present the measurement error expressions below. Each term has a variance parameter, $\sigma^2$, representing the measurement error variance; one can set these parameters based on published sensor accuracy. None of these error terms includes a modeled bias or covariance across measurements, though as mentioned they can be included without affecting the process smoothers above.

(1) $\mathbf{z}_{adcp}$: ADCP data measured from the glider with error variance $\sigma^2_{adcp}$ (In our case, Nortek (2022) describes the standard deviation of the Nortek Signature 1000 ADCP as 3.16E-2 m/s):

$$-\ell(\mathbf{z}_{adcp}|\mathbf{x}) = \frac{1}{2\sigma^2_{adcp}}\|\mathbf{z}_{adcp} - \mathbf{B}_{adcp}\mathbf{x}\|^2. \tag{12}$$

Such ADCPs are mounted in either an upwards or downwards facing direction and get results out to O(10m) with accuracy degrading with distance as is common in sonar equipment. $\mathbf{B}$ matrices select the vehicle kinematics of the state vector at the appropriate measurement times. That is, for $\mathbf{B}_{\text{adcp}}$, if row $k$ represents measurement $k$ including vehicle velocity at time $t_j(k)$ and current at depth $s_i(k)$,

$$\mathbf{b}_k = [0 \ldots \underset{\mathbf{x}_q^{j(k)}}{-1} \ldots 0 \ldots \underset{\mathbf{x}_c^{i(k)}}{1} \ldots 0]$$

(2) $\mathbf{z}_{ttw}$: Through-the-water estimated velocity of the glider with error variance $\sigma^2_{ttw}$. While this quantity nominally relies on calculations from pitch and depth rate measurements using a vehicle's hydrodynamic model, we treat it as direct measurement. Eriksen et al. (2001) provide calculations for the velocity and estimate a vertical velocity standard deviation of 12.6 cm/s. Horizontal velocity variance is a function of angle of attack, but for a lift/drag ratio of 3:1, the modeled horizontal velocity error would be around 37.7 cm/s.

$$-\ell(\mathbf{z}_{ttw}|\mathbf{x}) = \frac{1}{2\sigma^2_{ttw}}\|\mathbf{z}_{ttw} - \mathbf{B}_{ttw}\mathbf{x}\|^2. \tag{13}$$

(3) $\mathbf{z}_{gps}$: Two GPS position measurements at the surface before and (optionally) after a dive. As a representative value, Garmin (2022) lists the GPS15xH to have 3-5 m accuracy 95% of the time, which works out to a

$\sigma_{gps}$ of 1.58-2.55m.

$$-\ell(\mathbf{z}_{gps}|\mathbf{x}) = \frac{1}{2\sigma_{gps}^2}\|\mathbf{z}_{gps} - (\mathbf{B}_{gps}\mathbf{x})\|^2. \tag{14}$$

These terms compose the complete likelihood specification for the model:

$$-\ell(\mathbf{x}|\mathbf{z}) = -\ell(\mathbf{x}_v) - \ell(\mathbf{x}_c) - \ell(\mathbf{z}_p|\mathbf{x}) - \ell(\mathbf{z}_{ttw}|\mathbf{x}) - \ell(\mathbf{z}_{gps}|\mathbf{x}), \tag{15}$$

106

with $-\ell(\mathbf{x}_v)$ replaced by $-\ell(\mathbf{x}_v|\mathbf{x}_c)$ when using the conditional smoothing.

### *2.4 Optimization and Uncertainty Quantification*

The likelihood approach of expression (15) results in a combined posterior estimator that is a Gaussian random variable. One can represent the posterior estimator of $\mathbf{x}$ as:

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} -\ell(\mathbf{x}) \tag{16}$$

Since every term in (15) is linear in $\mathbf{x}$, all of the terms can be consolidated to:

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} \|\mathbf{A}x - \mathbf{b}\|^2, \tag{17}$$

where $\mathbf{A}$ and $\mathbf{b}$ combine the various $\mathbf{Q}$, $\mathbf{B}$, $\mathbf{G}$, and $\mathbf{z}$. The solution of equation (16) is given in equation (2), i.e. the inversion method, and has variance:

$$\text{var}(\hat{\mathbf{x}}) = \mathbb{E}[(\hat{\mathbf{x}} - \mathbf{x})(\hat{\mathbf{x}} - \mathbf{x})^T] = (\mathbf{A}^T\mathbf{A})^{-1}. \tag{18}$$

### *2.5 Comparison: Dead Reckoning, Corrected for Depth-Averaged Current*

Dead Reckoning (DR) is the navigation method where, in the absence of external positioning information and current information, a vessel advances its position purely based upon inaccurate velocity information. Glider's have routinely used the difference between the end-of-dive GPS and dead reckoning as an estimate of the true depth-averaged current, accurate to $\sim 1$ cm/s in order to successfully aid in navigation. Notwithstanding measurement biases, this is effectively the inversion method approach of Visbeck (2002) and Todd et al. (2017). This method is used to compare our results to those generated by a competing inversion method. We do not compare with more complex methods that correct for issues outside of the current scope.

## 3 Experiment and Simulation

We conducted simulations to compare four methods based on different process models: (1) the basic one described in section 2.1, (2) the higher-order one described in section 2.2.1, (3) the process with proper vehicle-current covariance described in section 2.2.2, and (4) a combination of methods 3 and 4: a proper covariance method that smooths to higher order. In comparing methods, we use as criteria: (a) the horizontal position accuracy during the dive at optimal process parameters, (b) the current accuracy during the dive at optimal process parameters, (c) the breadth of process parameters that achieve satisfactory results, and (d) performance on simulations including only two closely-placed GPS fixes before the dive and no GPS fix at the end. While such cases could always be improved by including a GPS point at the end, they illustrate how well the process model controls uncertainty growth while submerged. It also behaves similarly to trials of Todd et al. (2017) that use initial surface drift velocity, as measured by GPS, as a term in their least-squares solution. Significantly, position accuracy upon resurfacing could also be used to choose optimal process parameters in a grid search. In plots, methods are compared to DR corrected for DAC as discussed in 2.5. While more complex methods exist in the art, they mostly add terms to correct for errors we will not simulate and which can be incorporated compatibly with our state-space model. The purpose of the simulation is to demonstrate the effectiveness *ceteris paribus*, of our process model, especially the effects of the conditional probability model.

107

### *3.1 Parameters and Grid-Search*

The dive simulation parameters, listed in table 1, reflect a randomized current profile and vehicle navigation path. Current and vehicle speed as well as sampling interval is similar to the Canada Acoustic Basin Glider Experiment (CABAGE) trials explored in Jonker et al. (2019). The choice of velocity magnitudes aims to demonstrate the larger impact of the conditional smoother with greater currents and observation intervals. We made compromises for ease of simulation, such as separate, uncorrelated sinusoids for current and vehicle propulsion on ascent and descent; this manifests as an apparent vehicle turn at apogee.

Published information about physical devices can inform setting the order of magnitude for measurement variance parameters. On the other hand, the process variance parameters for smoothing represent prior knowledge of vehicle and current dynamics, and the problem of simultaneous state and variance estimation for Kalman smoothing diverges. Knowing this, we wish to under-

| Parameter | Value |
|---|---|
| Duration | 3 hrs |
| Depth | 750 m |
| Hydrodynamic model measurements | 500 |
| ADCP measurements | 450 |
| ADCP bins | 4 |
| ADCP range | 12m, upwards-facing |
| Current | piecewise sinusoid with amplitude $\sim N(0, .3)$ m/s |
| Vehicle TTW velocity | piecewise sinusoid with amplitude $\sim N(0, .4)$ m/s |
| Hydrodynamic model noise | 1e-1 m/s |
| ADCP measurement noise | 1e-1 m/s |
| GPS measurement noise | 1 m |

Table 1
Simulation Parameters, reflecting likely dive of a Spray glider with Nortek ADCP

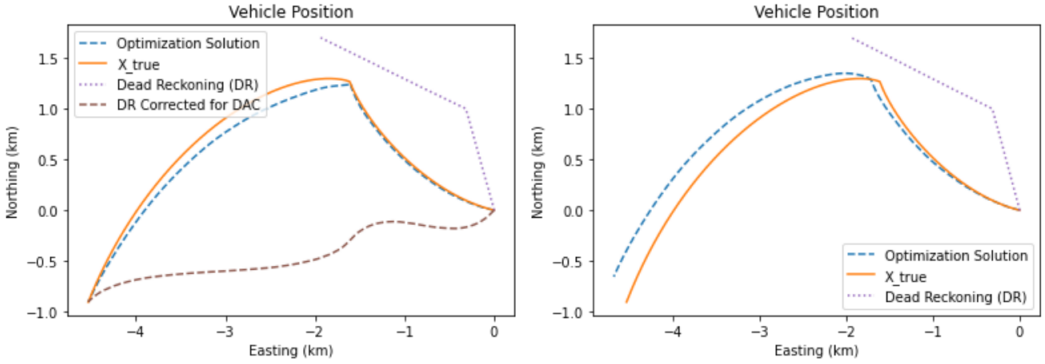
stand how much the model's success depends upon how exactly process model variances are chosen. Thus, rather than finding a particular best variance parameter, we are interested in the size of $\sigma_v^2$, $\sigma_c^2$ parameter space that gives a reasonable solution. In each simulation, we run a parameter search on the variance coefficients for the process terms, $\sigma_v^2$ and $\sigma_c^2$, and evaluate the effect on accuracy metrics. The measurement error terms in modeling, $\sigma_{adcp}^2$, $\sigma_{ttw}^2$, and $\sigma_{gps}^2$, are given the same value used in simulation.

### *3.2 Results*

Root mean squared error (RMSE) values averaged across twenty trials are displayed in Table 2. All solution methods tracked the vehicle to within three hundred yards after three hours before taking a GPS fix at the surface and reconstructed the current profile with less than ten centimeters per second error. Method 2 performed the best: including the GPS end point brought the error to under 100m and 4 cm/s. Most methods achieve around 6 cm/s error on average, with the strongest currents experiencing the highest error. Given that our simulation includes several thousand depth points, this equates to a DAC error on the order of $10^{-5}$ m/s. Fig.1 demonstrates the navigation accuracy of the method and Fig. 2 demonstrates the current accuracy. Looking in more detail at how the choice of vehicle and current process pa-

| Method | Variant | Nav Error | Curr Error | Nav err (no final GPS) | Curr err (no final GPS) |
|---|---|---|---|---|---|
| 1 | Basic | 151 | 5.30E-2 | 318 | 7.85E-2 |
| 2 | Higher-order | **97** | **3.67E-2** | **216** | **5.46E-2** |
| 3 | Vehicle-Current cov. | 134 | 4.92E-2 | 308 | 7.40E-2 |
| 4 | Cov. & Higher-order | 120 | 6.56E-2 | 264 | 6.72E-2 |

Table 2
Results, with the optimal method for each column boldfaced. Navigation RMSE uses units of meters, current RMSE uses units of meters per second. All methods perform competitively, with a different method taking top spot for each metric.




(a) (b)

Fig. 1. Navigation error (a) method 1, with start and end GPS points, and (b) method 4, including only starting GPS points. All of this paper's methods perform well, outperforming DR corrected by DAC, and the best track the vehicle to 55m/hr before receiving a resurfacing GPS fix.

rameters affect navigation error in Fig.3, the higher order methods (2 and 4) and correct covariance methods (3 and 4) admit a much wider choice of process variance parameters. This makes them better candidates for a production model. Among the higher-order methods, using the correct covariance provides a slightly smoother plot and larger acceptable parameter space, though averaged slightly worse performance.

The estimator variance plot in Fig.4 shows that while the Brownian assumptions fit the sinusoidal simulation. However, the off-diagonal blocks show that vehicle velocity estimates correlate highly with current estimates. Such structure indicates a near rank-deficiency in our model, a lack of identifiability in whether an individual velocity reflects vehicle motion or currents. It suggests that direct observation of a just a few true vehicle velocities or currents during the dive would dramatically improve the matrix conditioning and estimates. This perhaps provides an explanation for how Medagoda et al. (2016b) finds that a dramatic correction in navigation estimates occurs as soon as the vehicle

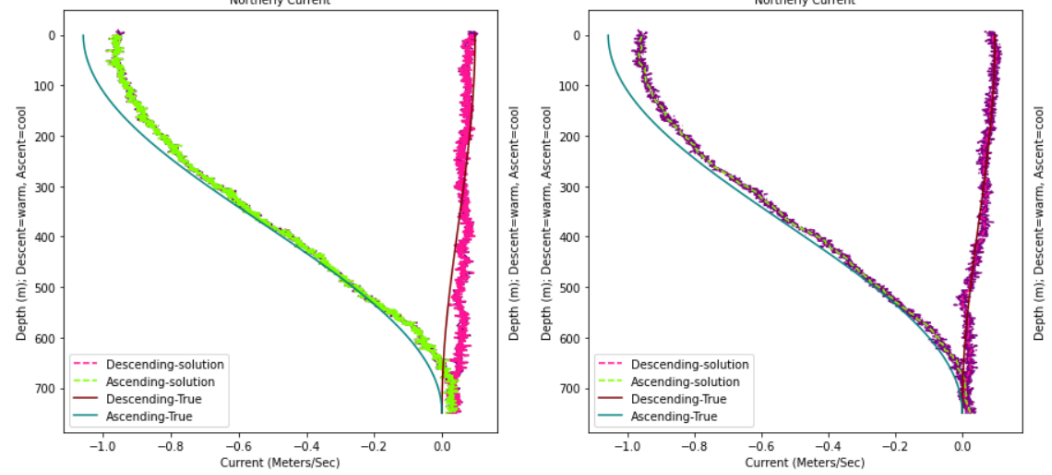



Fig. 2. Current error (a) in method 3 and (b) method 4. The purple series are the inferred ADCP measurement traces. Both solutions shown include only starting GPS points. Both the results on prepared metrics and the much smoother profile of the higher order methods recommend methods 2 and 4.

gets a DVL lock on the seafloor.

## 4 Conclusion and Further Research

Noticing previous ADCP current estimation models assume independence of vehicle and current velocities, we leverage recent advances to form a joint probability model, extending Kalman smoothing to avoid the independence assumption. We generated data and showed that our basic model, essentially a method of Jonker et al. (2019) without binning, recovered an accurate current profile and navigation track. Models were able to navigate within a few hundred meters without the final GPS point and track current to within 7 cm/s when the final GPS point was added. Extending the model to higher order to enforce smoothness of velocities improved the model substantially and results in a much wider region to choose acceptable variance parameters. Our other innovation, smoothing vehicle velocity and current velocity with proper covariance, widened the region of acceptable smoothing parameters slightly but was not more accurate in the best case. This performance and its additional rigor merit further investigation of the method, however. The conditional smoothing terms are most significant with large measurement gaps, and may help reduce the amount of data needed in glider missions or when observations are only taken on ascent or descent.

Various ancillary corrections may help improve the evidence for our major

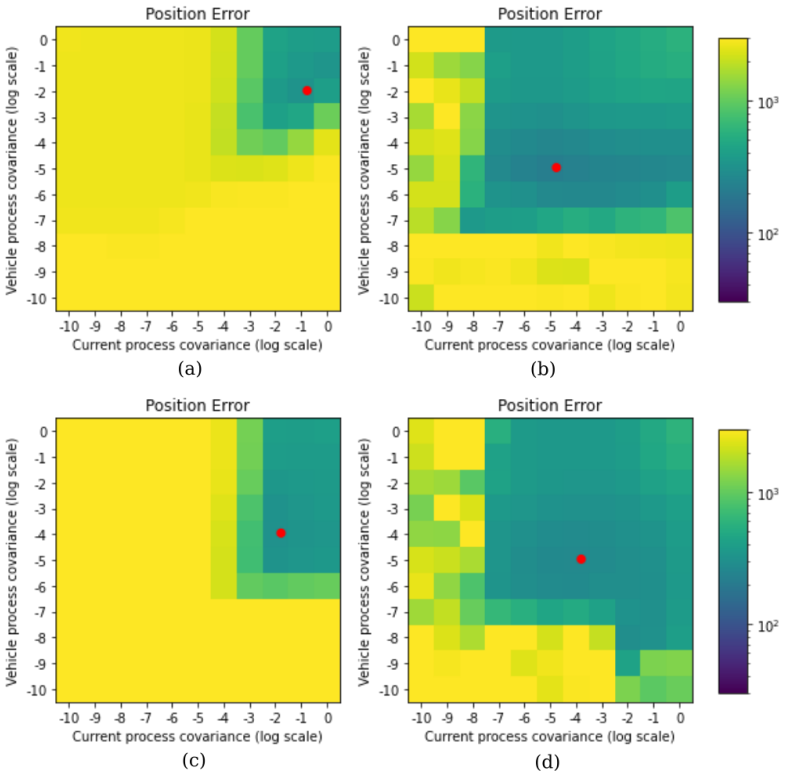



Fig. 3. Navigation error across process variance parameter search (a) in method 1, (b) method 2, (c) method 3, and (d) method 4. All cases shown include only starting GPS points, and the higher-order and correct-covariance smoothing methods have a much larger acceptable region of parameter space.

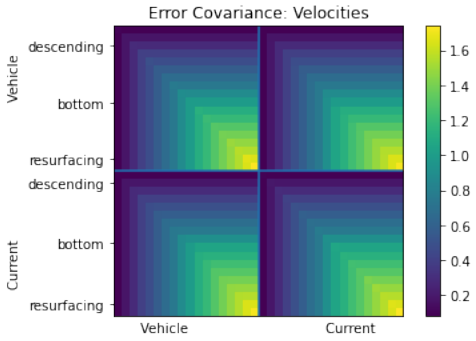



Fig. 4. Velocity rows/columns of the estimator error covariance matrix, organized to show vehicle and current states throughout the dive path. Method 1, without a final GPS point. The result demonstrates the same structure as the covariance of Brownian motion. Additionally, the repeated off-diagonal terms indicate a lack of identifiability. All methods including only starting GPS points demonstrate these effects.

contributions. Most significantly, we ran trials without binning, estimating current at $< 1$ m resolution. As a result, matrix conditioning caused issues with most models. Additionally, processes such as an Ornstein-Uhlenbeck/Gauss-Markov process could help by introducing the assumption that current centers around zero and the vehicle velocity centers around its design drift velocity. An O-U smoother can also incorporate the conditional structure described in section 2.2.2.

112

As a key avenue for improvement, one could replace the direct matrix inversion with a generic convex solver. This would allow for robust vehicle and current processes, as demonstrated in Aravkin et al. (2017). Robust process models would better handle the idiosyncracies we found in real data, such as very strong ADCP outliers at apogee. The generic convex solver would also permit more robust and nonlinear measurement forms in the likelihood expression, such as range measurements from as from fixed beacons or even range-azimuth-elevation of the USBL method in Jakuba et al. (2021).

## Acknowledgements

This research was supported by the NOAA Office of Exploration and Research award NA20OAR0110429 and by the Veterans Administration under the GI Bill.

One can generate the figures and data for this paper by downloading our Python package at https://github.com/UW-AMO/seaglider-navigation and running "publish_figures.py". The commit used to generate the exact figures in the paper is e63028d.

## References

Aravkin, Aleksandr, James V Burke, Lennart Ljung, Aurelie Lozano & Gianluigi Pillonetto (2017). "Generalized Kalman smoothing: Modeling and algorithms". In: *Automatica* 86, pp. 63–86.

Arnold, Sascha & Lashika Medagoda (2018). "Robust model-aided inertial localization for autonomous underwater vehicles". In: *IEEE conference on Robotics and Automation*, pp. 4889–4896. ISSN: 23318422.

Eriksen, Charles C. et al. (2001). "Seaglider: A long-range autonomous underwater vehicle for oceanographic research". In: *IEEE Journal of Oceanic Engineering* 26.4, pp. 424–436. ISSN: 03649059. DOI: 10.1109/48.972073.

Garmin (Apr. 2022). *GPS 15xH/15xL Technical Specifications*. Garmin International. URL: http://static.garmin.com/pumac/GPS15xH_15xL_TechnicalSpecifications.pdf.

Jakuba, Michael V., James W Partan, Sarah E Webster, Dennis Glaya & Christina Ramirez (2021). "Performance of a Low-Power One-Way Travel-Time Inverted Ultra-Short Baseline Navigation System". In: *OCEANS 2021: San Diego – Porto*.

Jonker, Jonathan, Andrey Shcherbina, Richard Krishfield, Lora Van Uffelen, Aleksandr Aravkin & Sarah E. Webster (2019). "Preliminary Results in Current Profile Estimation and Doppler-aided Navigation for Autonomous Underwater Gliders". In: pp. 1–8. DOI: 10.1109/oceanse.2019.8867108. arXiv: 1907.02897.

Medagoda, Lashika, Michael V. Jakuba, Oscar Pizarro & Stefan B. Williams (2010). "Water column current profile aided localisation for autonomous underwater vehicles". In: *OCEANS'10 IEEE Sydney, OCEANSSYD 2010*, pp. 1–10. DOI: 10.1109/OCEANSSYD.2010.5604016.

Medagoda, Lashika & James C Kinsey (2016a). "Water-Current and IMU Aided AUV Localization in Deep Mid-Water". In: *IEEE International Conference on Robotics and Automation* Dvl, pp. 2–4.

Medagoda, Lashika, James C. Kinsey & Martin Eilders (2015). "Autonomous Underwater Vehicle localization in a spatiotemporally varying water current field". In: *Proceedings - IEEE International Conference on Robotics and Automation* 2015-June.June, pp. 565–572. ISSN: 10504729. DOI: 10.1109/ICRA.2015.7139235.

Medagoda, Lashika, Stefan B. Williams, Oscar Pizarro, James C. Kinsey & Michael V. Jakuba (2016b). "Mid-water current aided localization for autonomous underwater vehicles". In: *Autonomous Robots* 40.7, pp. 1207–1227. ISSN: 15737527. DOI: 10.1007/s10514-016-9547-3.

Nortek (Feb. 2022). *Signature1000 Current Profiler*. Nortek Group. URL: https://www.nortekgroup.com/products/signature-1000/pdf.

Todd, Robert E., Daniel L. Rudnick, Jeffrey T. Sherman, W. Brechner Owens & Lawrence George (2017). "Absolute velocity estimates from autonomous underwater gliders equipped with doppler current profilers". In: *Journal of Atmospheric and Oceanic Technology* 34.2, pp. 309–333. ISSN: 15200426. DOI: 10.1175/JTECH-D-16-0156.1.

Visbeck, Martin (2002). "Deep velocity profiling using lowered acoustic Doppler current profilers: Bottom track and inverse solutions". In: *Journal of Atmospheric and Oceanic Technology* 19.5, pp. 794–807. ISSN: 07390572. DOI: 10.1175/1520-0426(2002)019<0794:DVPULA>2.0.CO;2.



## A Kalman Matrices

The Kalman smoother arrives in our likelihood expression (15) in the form of a prior. Let $W$ refer to a Brownian motion. Referring to time index $j$ for a coordinate with velocity $x_{\dot{q}}^j = W_{t_j}$ and position $x_q^j = \int_0^{t_j} W_r dr$, expression (3)

is exactly the likelihood of Brownian motion and its integral. That is:

$$\begin{aligned}
x_{\dot{q}}^j - x_{\dot{q}}^{j-1} &= W_{t_j} - W_{t_{j-1}} \\
&\sim W_{\Delta t_j} \\
x_q^j - x_q^{j-1} &= \int_{t_{j-1}}^{t_j} x_{\dot{q}}(t)dt \\
&= \int_{t_{j-1}}^{t_j} x_{\dot{q}}^{j-1} + x_{\dot{q}}(t) - x_{\dot{q}}^{j-1} dt \\
&= \Delta t_j x_{\dot{q}}^{j-1} + \int_{t_{j-1}}^{t_j} W_t - W_{t_{j-1}} dt \\
&\sim \Delta t_j x_{\dot{q}}^{j-1} + \int_0^{\Delta t_j} W_r dr.
\end{aligned}$$



We have used the independence of successive increments and stationarity to change from $[t_{j-1}, t_j]$ to simpler variables $[0, \Delta t_j]$. Thus,

$$\begin{bmatrix} x_{\dot{q}}^j - x_{\dot{q}}^{j-1} \\ x_q^j - x_q^{j-1} - \Delta t_j x_{\dot{q}}^{j-1} \end{bmatrix} \sim \begin{bmatrix} W_{\Delta t_j} \\ \int_0^{\Delta t_j} W_r dr \end{bmatrix} \tag{A.1}$$

Taken in blocks, matrix $\mathbf{G}_\upsilon \mathbf{x}_\upsilon$ is the left side of expression (A.1).

$$\mathbf{G}_\upsilon = \begin{bmatrix} -1 & 0 & 1 & 0 & & \\ -\Delta t_1 & -1 & 0 & 1 & & \\ & & -1 & 0 & 1 & 0 \\ & & -\Delta t_2 & -1 & 0 & 1 \\ & & & & & \ddots \end{bmatrix} \quad \text{and} \quad \mathbf{x}_\upsilon = \begin{bmatrix} x_{\dot{q}}^1 \\ x_q^1 \\ x_{\dot{q}}^2 \\ x_q^2 \\ \vdots \end{bmatrix}. \tag{A.2}$$

Because the right hand side are integrals of Brownian motion, we know:

$$\mathbb{E}[\mathbf{G}_\upsilon \mathbf{x}_\upsilon] = 0,$$

With the mean determined, all that remains is to determine the covariance for each timestep: $\mathbf{Q}_\upsilon$ in the expression $\mathbf{x}_\upsilon^j | \mathbf{x}_\upsilon^{j-1} \sim \mathcal{N}((\mathbf{G}_\upsilon \mathbf{x})^j, \mathbf{Q}_\upsilon^j)$. Because increments are independent, the matrix $\mathbf{Q}_\upsilon$ has block diagonals $\mathbf{Q}_\upsilon^j$:

$$\mathbb{E}\begin{bmatrix} W_{\Delta t_j}^2 & W_{\Delta t_j} \cdot \int_0^{\Delta t_j} W_r dr \\ W_{\Delta t_j} \cdot \int_0^{\Delta t_j} W_r dr & \left(\int_0^{\Delta t_j} W_r dr\right)^2 \end{bmatrix} = \begin{bmatrix} \Delta t_j & \Delta t_j^2/2 \\ \Delta t_j^2/2 & \Delta t_j^3/3 \end{bmatrix}. \tag{A.3}$$

We have omitted the calculation details involving Itô calculus. In $\mathbf{x}_c$, where the state vector only includes a velocity, one can simply drop the rows/columns of $\mathbf{G}$ and $\mathbf{Q}$ corresponding to position. Finally, since current varies by depth $s$, we replace $\Delta t_j$ with $\Delta s_i$.

## B Higher Order Smoothing

In cases when we include an acceleration term, we can simply assign $x^j_{\ddot{q}} = W_{t_j}$ and $x^j_{\dot{q}} = \int_0^{t_j} W_r dr$, giving the results from the previous appendix to the $x_{\ddot{q}}$ and $x_{\dot{q}}$ terms. We need to integrate the Brownian motion one more time to get position:



$$\begin{aligned}
x^j_q - x^{j-1}_q &= \Delta t_j x^{j-1}_{\dot{q}} + \int_{t_{j-1}}^{t_j} x_{\dot{q}}(t) - x^{j-1}_{\dot{q}} dt \\
&= \Delta t_j x^{j-1}_{\dot{q}} + \int_{t_{j-1}}^{t_j} \int_{t_{j-1}}^{t} x_{\ddot{q}}(r) dr dt \\
&= \Delta t_j x^{j-1}_{\dot{q}} + \int_{t_{j-1}}^{t_j} \int_{t_{j-1}}^{t} x^{j-1}_{\ddot{q}} + x_{\ddot{q}}(r) - x^{j-1}_{\ddot{q}} dr dt \\
&= \Delta t_j x^{j-1}_{\dot{q}} + \frac{\Delta t_j^2}{2} x^{j-1}_{\ddot{q}} + \int_{t_{j-1}}^{t_j} \int_{t_{j-1}}^{t} W_r - W_{t_{j-1}} dr dt \\
&\sim \Delta t_j x^{j-1}_{\dot{q}} + \frac{\Delta t_j^2}{2} x^{j-1}_{\ddot{q}} + \int_0^{\Delta t_j} \int_0^{t} W_r dr dt \\
&\sim \Delta t_j x^{j-1}_{\dot{q}} + \frac{\Delta t_j^2}{2} x^{j-1}_{\ddot{q}} + \int_0^{\Delta t_j} (\Delta t_j - t) W_r dt,
\end{aligned}$$

where the last step is Cauchy's formula for iterated integrals. Collecting terms as in the previous appendix and calculating covariances, these relationships give rise to the formulae for $\mathbf{G}_\upsilon$ and $\mathbf{Q}_\upsilon$:

$$\mathbf{G}_\upsilon = \begin{bmatrix}
-1 & 0 & 0 & 1 & 0 & 0 & & & & \\
-\Delta t_1 & -1 & 0 & 0 & 1 & 0 & & & & \\
-\frac{{\Delta t_1}^2}{2} & -\Delta t_1 & -1 & 0 & 0 & 1 & & & & \\
 & & & -1 & 0 & 0 & 1 & 0 & 0 & \\
 & & & -\Delta t_2 & -1 & 0 & 0 & 1 & 0 & \\
 & & & -\frac{{\Delta t_2}^2}{2} & -\Delta t_2 & -1 & 0 & 0 & 1 & \\
 & & & & & & & & & \ddots
\end{bmatrix}, \qquad \mathbf{Q}^j_\upsilon = \begin{bmatrix}
\Delta t_j & \frac{\Delta t_j^2}{2} & \frac{\Delta t_j^3}{6} \\
\frac{\Delta t_j^2}{2} & \frac{\Delta t_j^3}{3} & \frac{\Delta t_j^4}{8} \\
\frac{\Delta t_j^3}{6} & \frac{\Delta t_j^4}{8} & \frac{\Delta t_j^5}{20}
\end{bmatrix}. \tag{B.1}$$

The current process now has two total orders, and so has similar terms to vehicle smoothing in Appendix A, but again replacing $t$ with $s$.

## C Proper Covariance for Current and Vehicle True Velocity

We seek a specification for $\Pr(\mathbf{x}_{\dot{q}},\ \mathbf{x}_q|\ \mathbf{x}_c)$ in equation (9). We first form the correct covariance matrix for $\Pr(\mathbf{x}_{\dot{q}},\ \mathbf{x}_q,\ \mathbf{x}_c)$ and then derive the conditional covariance matrix and expected value. Since $\mathbf{G}_v$ uses the expected value to recenter the process, we need only concern ourselves with a single timestep $\Delta t$.



Although we do not utilize an $\mathbf{x}_r$ term, we still return to equation (8), integrating to get

$$x_q(t_j) - x_q(t_{j-1}) = \int_{t_{j-1}}^{t_j} x_{\dot{q}}(t)dt = \int_{t_{j-1}}^{t_j} x_{\dot{r}}(t) + x_c(s(t))dt.$$

Writing out all the terms as random variables driven by independent Brownian motions $x_c(s) = W_s^1$ driving current and $x_{\dot{r}}(t) = W_t^2$ driving the vehicle's speed through the water, we calculate:

$$\begin{aligned}
x_c^j - x_c^{j-1} &= \int_{s(t_{j-1})}^{s(t_j)} \sigma_c^2 dW_s^1 \sim \int_0^{\Delta s} \sigma_c^2 dW_s^1 \\
x_{\dot{q}}^j - x_{\dot{q}}^{j-1} &= \int_{t_{j-1}}^{t_j} \sigma_v^2 dW_t^2 + \int_{s(t_{j-1})}^{s(t_j)} \sigma_c^2 dW_s^1 \sim \int_0^{\Delta t_j} \sigma_c^2 dW_t^2 + \int_0^{\Delta s_j} \sigma_v^2 dW_s^1 \\
x_q^j - x_q^{j-1} &= \int_{t_{j-1}}^{t_j} dt \cdot x_{\dot{q}}(t) \\
&= \Delta t \cdot x_{\dot{q}}^{j-1} + \int_{t_{j-1}}^{t_j} dt \cdot \left(x_{\dot{q}}(t) - x_{\dot{q}}^{j-1}\right) \\
\int_{t_{j-1}}^{t_j} dt \cdot \left(x_{\dot{q}}(t) - x_{\dot{q}}^{j-1}\right) &\sim \int_{t_{j-1}}^{t_j} dt \int_{t_{j-1}}^{t} \sigma_v^2 dW_r^2 + \int_{s(t_{j-1})}^{s(t)} \sigma_c^2 dW_r^1 \\
&= \int_0^{\Delta t_j} dt \int_0^t \sigma_v^2 dW_r^2 + \int_0^{\Delta t_j} dt \int_0^{s(t)} \sigma_c^2 dW_r^1 \\
&= \int_0^{\Delta t_j} dt \int_0^t \sigma_v^2 dW_r^2 + \frac{1}{\dot{s}} \int_0^{\Delta s_j} ds \int_0^s \sigma_c^2 dW_r^1 \quad (\text{since } dt/ds = \dot{s}^{-1}).
\end{aligned}$$

For further simplification, we now let $\Delta x_c^j = x_c^j - x_c^{j-1}$, $\Delta x_{\dot{q}}^j = x_{\dot{q}}^j - x_{\dot{q}}^{j-1}$, and $\Delta x_q^j = x_q^j - x_q^{j-1} - \Delta t_j \cdot x_{\dot{q}}^{j-1}$. These mean-zero Gaussian random variables have the covariance matrix:

$$\mathbf{Q}^j = \begin{bmatrix}
\sigma_c^2 \Delta s_j & \sigma_c^2 \Delta s_j & \sigma_c^2 \frac{\Delta s_j^2}{2\dot{s}_j} \\
\sigma_c^2 \Delta s_j & \sigma_v^2 \Delta t_j + \sigma_c^2 \Delta s_j & \sigma_v^2 \frac{\Delta t_j^2}{2} + \sigma_c^2 \frac{\Delta s_j^2}{2\dot{s}_j} \\
\sigma_c^2 \frac{\Delta s_j^2}{2\dot{s}_j} & \sigma_v^2 \frac{\Delta t_j^2}{2} + \sigma_c^2 \frac{\Delta s_j^2}{2\dot{s}_j} & \sigma_v^2 \frac{\Delta t_j^3}{3} + \sigma_c^2 \frac{\Delta s_j^3}{3\dot{s}_j^2}
\end{bmatrix} \begin{matrix} \Delta x_c^j \\ \Delta x_{\dot{q}}^j \\ \Delta x_q^j \end{matrix}$$

$$\begin{matrix} \Delta x_c^j & \Delta x_{\dot{q}}^j & \Delta x_q^j \end{matrix}$$

Calculating the conditional distribution gives:

$$\Delta x_c^j \sim \mathcal{N}(0, \Delta s_j) \tag{C.1}$$

$$\begin{bmatrix} \Delta x_{\dot{q}}^j \\ \Delta x_q^j \end{bmatrix} \Bigg| \Delta x_c^j \sim \mathcal{N}\left( \begin{bmatrix} \Delta x_c^j \\ \frac{\Delta s_j}{2\dot{s}_j} \Delta x_c^j \end{bmatrix}, \sigma_v^2 \begin{bmatrix} \Delta t_j & \frac{\Delta t_j^2}{2} \\ \frac{\Delta t_j^2}{2} & \frac{\Delta t_j^3}{3} + \frac{\sigma_c^2}{\sigma_v^2} \frac{\Delta s_j^3}{12\dot{s}_j^2} \end{bmatrix} \right). \tag{C.2}$$



The above assumes an equivalent number of time and depth points. When there are intermediate depths in the state vector between $s(t_{j-1})$ and $s(t_j)$, the terms with $\Delta s$ need to be summed across all $k \in \{i(j-1)+1, \ldots, i(j)\}$, e.g. $\frac{\Delta s}{2\dot{s}} \Delta x_c \to \sum_{k=i(j-1)+1}^{i(j)} \frac{\Delta s_k}{2\dot{s}} \Delta x_c^k$ and $\frac{\Delta s^3}{12\dot{s}^2} \to \sum_{k=i(j-1)+1}^{i(j)} \frac{\Delta s_k^3}{12\dot{s}^2}$. Depth rate $\dot{s}$ is assumed constant between timepoints.

When smoothing the vehicle process to higher order and modeling acceleration conditionally, we also choose to smooth the current process to the same order. Thus, to smooth current to higher order, we re-notate $\mathbf{x}_c$ as $\mathbf{x}_{\dot{p}}$, the current velocity, and $\mathbf{x}_{\ddot{p}}$, the change in current with depth. Since $\mathbf{x}_{\ddot{p}}$ is a change in velocity with respect to depth, we must multiply by $\dot{s}$ in order to get the change in velocity with respect to time. That is,

$$x_{\ddot{q}}^j - x_{\ddot{q}}^{j-1} = \int_{t_{j-1}}^{t_j} \sigma_v^2 dW_t^2 + \int_{s(t_{j-1})}^{s(t_j)} \dot{s}\sigma_c^2 dW_s^1.$$

The $\Delta$ variables (e.g., $\Delta x_q^j$, $\Delta x_{\dot{q}}^j$) then also must adjust to maintain a mean-zero process in the joint distribution. The conditional distribution becomes:

$$\begin{bmatrix} \Delta x_{\ddot{q}}^j \\ \Delta x_{\dot{q}}^j \\ \Delta x_q^j \end{bmatrix} \Bigg| \begin{bmatrix} \Delta x_{\ddot{p}}^j \\ \Delta x_{\dot{p}}^j \end{bmatrix} \sim \mathcal{N}\left( \begin{bmatrix} \dot{s}\Delta x_{\ddot{p}}^j \\ \Delta x_{\dot{p}}^j \\ \frac{\Delta s_j}{2\dot{s}} \Delta x_{\dot{p}}^j - \frac{\Delta s_j^2 \Delta x_{\ddot{p}}^j}{12\dot{s}} \end{bmatrix}, \sigma_v^2 \begin{bmatrix} \Delta t_j & \frac{\Delta t_j^2}{2} & \frac{\Delta t_j^3}{6} \\ \frac{\Delta t_j^2}{2} & \frac{\Delta t_j^3}{3} & \frac{\Delta t_j^4}{8} \\ \frac{\Delta t_j^3}{6} & \frac{\Delta t_j^4}{8} & \frac{\Delta t_j^5}{20} + \frac{\sigma_c^2}{\sigma_v^2} \frac{\Delta s_j^5}{720\dot{s}^2} \end{bmatrix} \right). \tag{C.3}$$

In both expressions C.1 and C.3, we create $\mathbf{G}$ to subtract the conditional mean in order to recover a mean-zero random variable.

# Appendix C

# PLUME REDUCED-ORDER MODELING

*This appendix documents my work on a tangentially-related subject: smoke plume reduced-order modeling. It put into practice the engineering concepts of this dissertation de novo. In contrast, pysindy already had existed for five years before I began my work. As a colloboration, it also demonstrated that the benefit of separation of concerns: people could work on different parts of the packages without conflict.*

# Coarse Graining and Reduced Order Models for Plume Ejection Dynamics



Ike Griss Salas*, Megan R. Ebers*, Jake Stevens-Haas*, and J. Nathan Kutz*,‡

* Department of Applied Mathematics, University of Washington, Seattle, WA 98195 USA

‡ Department of Electrical and Computer Engineering, University of Washington, Seattle, WA 98195

March 20, 2025

**Abstract**

Monitoring the atmospheric dispersion of pollutants is increasingly critical for environmental impact assessments. High-fidelity computational models are often employed to simulate plume dynamics, guiding decision-making and prioritizing resource deployment. However, such models can be prohibitively expensive to simulate, as they require resolving turbulent flows at fine spatial and temporal resolutions. Moreover, there are at least two distinct dynamical regimes of interest in the plume: (i) the initial ejection of the plume where turbulent mixing is generated by the shear-driven Kelvin-Helmholtz instability, and (ii) the ensuing turbulent diffusion and advection which is often modeled by the Gaussian plume model. We address the challenge of modeling the initial plume generation. Specifically, we propose a data-driven framework that identifies a reduced-order analytical model for plume dynamics – directly from video data. We extract a time series of plume center and edge points from video snapshots and evaluate different regressions based to their extrapolation performance to generate a time series of coefficients that characterize the plume's overall direction and spread. We regress to a sinusoidal model inspired by the Kelvin-Helmholtz instability for the edge points in order to identify the plume's dispersion and vorticity. Overall, this reduced-order modeling framework provides a data-driven and lightweight approach to capture the dominant features of the initial nonlinear point-source plume dynamics, agnostic to plume type and starting only from video. The resulting model is a pre-cursor to standard models such as the Gaussian plume model and has the potential to enable rapid assessment and evaluation of critical environmental hazards, such as methane leaks, chemical spills, and pollutant dispersal from smokestacks.

## 1 Introduction

Wildfires, volcanic eruptions, chemical spills, and industrial emissions disperse plumes of pollutants into the atmosphere. For example, approximately 1.2 to 2.6 million tons of methane leakage occurs from natural gas pipelines per year [1]. Locating and monitoring air pollutant dispersion like methane leaks is an important action towards reducing harmful climate pollution [2, 3, 4], it is also important issue in national security [5]. By modeling plumes, scientists, public health officials, and emergency responders can understand the concentration and spread of pollutants carried by the plume to evaluate health, safety, and environmental risks. This paper seeks to develop a data-driven reduced order model for the initial plume formation which is driven by the Kelvin-Helmholtz instability. This is critical part of the overall plume dynamics and is a potential pre-cursor to more standard models for the ensuing diffusion and advection dynamics of the plume, thus scientists and emergency responders can use such reduced models to understand the overall drift and dispersion of pollutants. While other computational models to model plumes exist, the aim for our model is to construct models directly from data which characterizes the initial plume formation; the only required data collection being video such as from a smartphone.

Mathematical modeling of plume dynamics has allowed the prediction, management, and understanding of air pollutant dispersion [6]. Mathematical models can be used inversely for management, such as source localization or emission rate estimation; forward models can be used for prediction, like emergency response planning or designing emission control strategies. A standard mathematical model used in the community is

known as the Gaussian plume model [7, 8] which assumes the plume to be a Gaussian spreading through a coarse-grained description of the turbulent diffusion and advection of the atmosphere. The Gaussian plume model focuses on the dynamics once plume has already formed. A spectrum of additional dispersion models exist, with the most powerful being computational fluid dynamics (CFD) models, which offer high-fidelity simulations of complex flow scenarios [9, 10, 11, 12, 13]. However, CFD simulations of plume dynamics can be impractical for rapid assessment and evaluation in time-critical scenarios.

120

While high-fidelity simulations represent the state of the art in accurately tracking plume dispersion and localizing sources over time, their computational demands are often prohibitively high. Early research into air pollution dispersion dates back nearly 100 years, with Bosanquet and Pearson evaluating the spread of chimney smoke [14]. In the following decade, mathematical models evolved to incorporate additional complexities in air pollution dispersion equations, such as vertical dispersion, crosswind dispersion, and ground effects [15]. In the 1960s, environmental control regulations in the United States triggered a surge in computational approaches to modeling pollutant plumes [16]. The foundation for these computational approaches was the Gaussian dispersion model of continuous and buoyant air pollution plumes [17]. Additionally, the Briggs equations enabled trajectory modeling of more complex plume behaviors [18]. Advancements in computational technology have since enabled high-fidelity CFD simulations to more accurately model particulate dispersion in complex environments and flow regimes [19, 20, 21, 22]. However, such simulations can easily become computationally intractable for large spatial domains and fine spatial resolutions [13]. This is further compounded by the complexity of such simulations and their sensitivity to model parameters [23, 12]. Accurately modeling plume dynamics is highly dependent on the specific type of plume, such as methane or hydrothermal. This dependency limits the generalization of a model without significant upfront effort to fine-tune it using first principles. To address these challenges, reduced order models (ROMs) and data-driven modeling techniques have emerged as practical solutions.

Reduced order models (ROMs) can significantly reduce computational requirements while preserving accurate approximations of complex phenomena. Classical approaches like proper orthogonal decomposition and dynamic mode decomposition [24, 25, 26, 27, 28] are widely used to find a low-dimensional and data-driven basis that capture dominant flow structure. These methods have been used to study urban airflow and pollutant dispersion [29, 30, 31]. However, two major drawbacks of these methods are: (i) they have limited spatial extrapolation ability, and (ii) they learn optimal linear subspaces, while fluid dynamics such as plume behavior are strongly nonlinear. Indeed, nonlinear dimensionality reduction techniques have demonstrated improved system characterization of fluid flow [32, 33, 34]. However, the black-box nature of deep learning paradigms like autoencoder architectures can limit physical interpretability.

In this work, we introduce a data-driven framework that identifies a reduced-order analytical model of point-source plume dynamics, agnostic to plume type, directly from video data. Our framework operates in a two-step procedure: (i) time series extraction from video data (ii) and reduced-order model (ROM) discovery from the extracted time series data, described in detail in section 2. For the time series extraction, we obtain a best fit of the plume's mean path in each video frame by performing various regressions along the path of highest pixel density. Next, for each video frame, we fit a linearly-growing sinusoid to the plume edge. This process extracts the polynomial and sinusoidal cofficients over time that characterize the plume's average trajectory and spread. An overview the pipeline can be seen in Figure 1. In section 3, we interrogate our framework' s ability to automatically extract dominant plume features from video data by extensively testing plume (i) point reduction, (ii) center-line (mean) curve fitting, and (iii) edge-line (variance) curve fitting. Section 4 describes important future work, including time-dependent modeling of the plume center, time-dependent dimension reduction, and experiments needed to establish the utility of the model in situ.

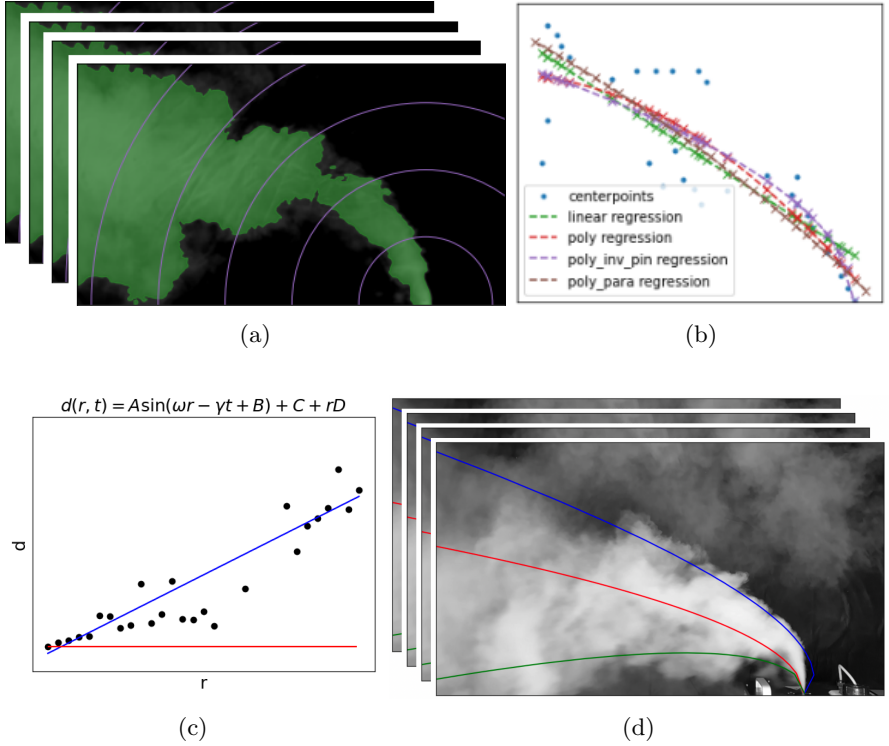



Figure 1: A sketch of each step in this paper and the composite product. (a) Cleaning the video and calculating the edge/center points. (b) Testing different regressions to find the curve that best extrapolates center points for each frame. (c) Modeling the spread of the edge points from the center path, and (d) the composite discovered centerline and edge dispersion, drawn on top of the original plume video

# 2 Methods

## 2.1 Video Pre-processing

An unprocessed video of a plume that is recorded with ideal background conditions.[1] All frames are converted to gray scale, that is $n \times d$ arrays taking on values between 0 and 1 (or 0 to 255). Two main steps are used in pre-processing prior to applying our model: background subtraction and Gaussian blurring.

**Background subtraction.** A fixed background subtraction method is applied to isolate the plume, where the first $k_{\text{fixed\_subtraction}}$ frames of video (where no plumes are have formed) are used to create an average background image. The average image is subtracted against the remaining frames to create the isolated plume frames. Once the plume dynamics have been isolated from background across selected frames, Gaussian blurring is applied.

**Gaussian Blurring.** Two separately tuned Gaussian filters are applied—a temporal and spatial filter, respecitvely. First, a Gaussian filter is applied across the time series of frames to add a time blur, reducing the high resolution of the plume dynamics. Second, a Gaussian filter is applied to each frame independently to coarse grain the image, adding spatial blur.

## 2.2 Coarse-graining for Centerline and Edge modeling

We denote the output of data pre-processing as $\boldsymbol{Z} = [\boldsymbol{z}(t_0), \ldots, \boldsymbol{z}(t_K)]$, where background subtraction and Gaussian filters have been applied. We extract a time series of second-order polynomial coefficients that

[1] A solid black background is used when filming plume dynamics.

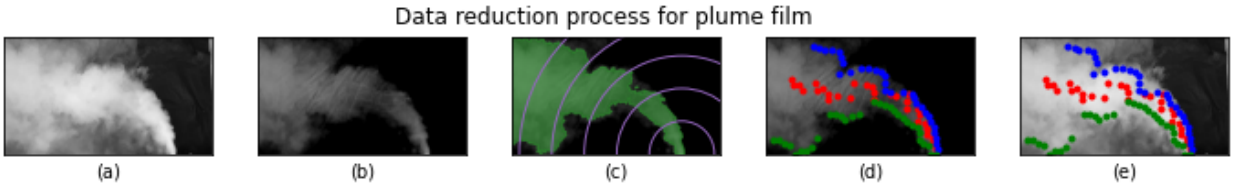





Figure 2: The plume point reduction step converts a frame of video into a scatter of edge and center points: (a) the raw frame image, (b) background subtraction, (c) contour selection, and finally (d) Along various ranges from origin, identify the max intensity (center) and intersection (edge) points with contours. (e) displays the final points on the original frame.

model the center path of the plume for each frame. Additionally, we learn the parameters of a growing sinusoidal function that best characterizes the spread, or edge-model, of the plume. We theorize there exists a connection between the Kevin Helmholtz shear velocity and the sinusoidal frequency.

Each frame $\boldsymbol{z}(t_i)$ is converted to a reduced order model describing the center and edge paths in a three step procedure: (i) contour detection, (ii) a concentric circle search, and (iii) regression. For each array, $\boldsymbol{z}(t_i)$, image recognition techniques are used to identify the plume contour and subsequently search along concentric circles, centered at the leak source, to identify the path of highest density, where we denote raw pixel value as a proxy for plume density, and the edge paths, as seen in Fig. 2.

(i) **Contour Detection.** We apply a binary threshold to identify the contours outlining the plume for each array $\boldsymbol{z}(t_i)$. Hyperpameter selection for the thresholding is done by using `opencv`'s Otsu's binarization. Optimal global threshold selection is performed by inspecting the image histogram for each array $\boldsymbol{z}(t_i)$. The $n$ largest contours, by area, are then selected for remainder of the pipeline. We denote the identified plume at time $t_k$ as $\boldsymbol{\psi}_k$.

(ii) **Concentric Circle Search.** A search is performed along a set of $\ell$ concentric circles, centered at the plume leak source, with incrementally growing radii of $r_i = r \cdot i$ for $i = 1, \ldots, \ell$ where $r$ is some fixed positive value. Along each concentric circle, three values are attained: the largest pixel value that lies within the identified contour, $\boldsymbol{\psi}_k$, and the two intersection points of the concentric circle $r_i$ with the identified plume contour, $\boldsymbol{\psi}_k$. These denote the points for the center and edge paths respectively.

(iii) **Regression.** Two regression techniques are implemented for the final steps for learning the centerline and edge plume paths. *Center Path.* Multiple second order polynomial regression are applied to the de-centered Cartesian coordinates of the centerline points identified in each frame from step (ii). Which produces a timeseries of collected polynomial coefficients. *Edge Paths.* At each time, $t_k$, along each concentric circle with radii $r$ the Euclidean distances between the edge points and center point is attained. Giving set of distances for each radial value. We denote this process as *flattening* the edge points. The flattened data is then bootstrapped and a series of growing sinusoid regression are applied via an Levenberg Marquardt optimization scheme. The mean set of values of the bootstrap bags are then selected for the final edge model.

# 3 Experiments

We investigate each step of our plume data-reduction method in order to identify the best parameter choices. To provide data for experiments, we filmed a smoke machine, where plumes were formed by vaporizing a water based triethylene glycol liquid, under three different intensities of crosswind, resulting in fifteen different videos. Each video lasts for between twenty and one hundred seconds. Ten of the videos are designated training videos, and five are reserved to test conclusions drawn on the training videos, balanced between crosswind levels. We conducted different experiments to learn the ideal parameters for fitting the model described in section 2: First, how best to reduce the greyscale frames to points along the center and edges of the plume; Secondly, how to fit a curve through the centerpoints that could extrapolate the direction of

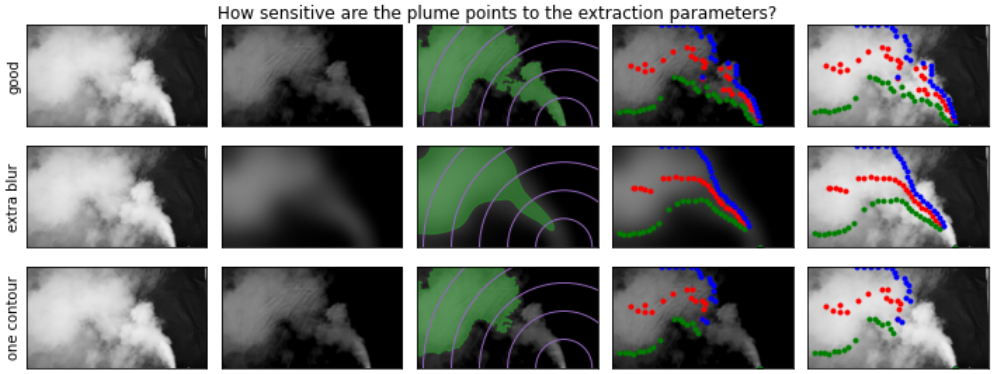





Figure 3: A visual comparison of the ideal parameters for our data set, along with selected less-than-ideal parameters.

the plume, potentially off-frame. Thirdly, how to identify spatiotemporal frequencies of the plume edge in hopes that these frequencies, associated with the Kelvin-Helmholtz instability, would reflect some physical parameters of the fluids. The appendix contains additional information to reproduce the experiments. A pseduo code of the complete pipeline is shown in algorithm 1.

## 3.1 Plume point reduction

At a macro scale, the first step is to transform video frames into scattered points representing the center and edges of the plumes. We investigate parameters for space and time Gaussian smoothing, as well as more exotic OpenCV parameters for detecting contours and determining the intersections of the contours and concentric circles. Because this step has no true value by which to evaluate goodness of fit, we manually review the same fifteen frames from each video's period of steady emission. While it is possible to add additional smoothing via a stronger Gaussian blur or minimizing the number of contour points or concentric circles, we aimed to reserve smoothing explicitly for the subsequent, curve-fitting steps. The goal of this step is merely to provide a good representation of the video.

We *a priori* identify the coordinates of the plume emission point in each video. We compare the results of different parameterizations by visually evaluating the how well the plume points match the raw video. Table 1 shows the different parameters we varied, and Figure 3 shows an example of a frame, comparing each step of plume point reduction against less-effective parameterizations.

The review of all the training frames suggested, and comparison of test frames confirmed, that a minor amount of spatial blurring and no time blurring were the best parameter selection. In addition, since contour thresholding often rejected part of the plume, finding up to three contours improved the accuracy of plume point reduction. An increased amount of spatial blurring, when tested, provided little to no benefit in reconstruction. We conclude that results were not terribly sensitive to blurring.

| Spatial Blur (px) | Time Blur (1 s/29.97) | Number of contours |
|---|---|---|
| 0 | **0** | 1 |
| **15** | 3 | 2 |
| 45 | 9 | **3** |
| 301 | 27 | 4 |

Table 1: The plume point reduction parameters compared across all trials. We additionally tested out blurring more strongly along the direction of the plume, different versions of OpenCV parameters on contour thresholding (settling on "OTSU") and concentric circle parameters (resulting in the saved defaults)

### 3.2 Center-line curve fitting

The next step is to fit a curve to the center points of the plume. With this in mind, we describe our coordinate origin as the plume emission point and compare four different curve-fitting methods: three that are a regression of $y = f(x)$ (linear, quadratic, and square-root), and one parametric quadratic fit that tries to identify $x = f(r), y = g(r)$. It is worth noting that while the expression of the square root curve is the inverse of the quadratic curve, they are both fit on $y$-error, making the former nonconvex and bounded by a nonconvex domain. To attempt to resolve these problems, we re-parameterize the square-root curve based upon the coordinates of the origin and the steepness of the curve and split the domain into four convex subdomains, fitting a constrained regression in each. Additionally, while quadratic and square-root curves can be described as instances of a parametric quadratic curve, in implementation, the regression does not require the fitted curve to enforce $r^2 = x^2 + y^2$ (nor can it, while retaining quadratic terms).



We seek, in the situation where video data is close to a point emission source, to describe where the plume is drifting off-camera. With this in mind, we split each frame into points *near* the origin, used to fit a curve, and points *away* from the origin, used to evaluate the curve, effectively creating a test regime for extrapolation. We compare the $l_2$ error of different curve fitting methods on the test points for each frame in order to determine the best method, as demonstrated in Figure 4.

From initial review of the training data, we hypothesized the square-root curve would be the best curve-fitting method based on visual inspection. We evaluated the results on the test dataset that comprises 6000 individual frames. With information on the best fitting method for each frame, we concluded that the square-root curve was the best fitting method overall at the 99% confidence level, with p-values near machine precision (see Table 2). However, the magnitude of the advantage over a linear regression was often small; relative error of extrapolation for each method is demonstrated in Figure 5. The advantage was most significant in low-wind datasets, where the plume arcs throughout the frame, and mostly disappeared in high-crosswind datasets, where the force from high-crosswind would rapidly dominate the initial point-source ejection force of the plume.

The results showed that the square-root curve was effective at extrapolating the direction of the fitted points with less than 20% relative error in most frames. This is even more substantial in context of how the fitted points were created in the previous step: the assumption that points of highest pixel intensity represent the greatest concentration of the plume, and that the points of greatest concentration best described the path of the plume. If, on the contrary, the plume's path at a particular radius is better described by its center of mass, or by the midpoint of the identified contours, the data fed into this experiment would naturally be smoother.

Nevertheless, the method and the experiments could be improved in a few ways. The choice of evaluating methods based upon extrapolation may result in worse models when trying to understand the physics before the dissipative regime. Changing the metric from extrapolation to interpolation would also unlock fitting with a spline curve, which allows parametric paths. Additionally, one could replace the parametric curve with a function $\theta(r)$ in polar coordinates. This would also make it easier to enforce the constraint that the fit curve crosses the origin, i.e. the plume emission point. Currently, only the square-root curve regression incorporates knowledge about the origin or rather, about the extremal points and the direction of the plume—a better comparison would provide similar information to the alternate regression methods.

### 3.3 Edge-line curve fitting

The goal is to understand plume spread with accurate edge modeling and capture the frequencies connected to Kelvin-Helmholtz instability. After the initial laminar emission, the plume appears to exhibit instability, with the edge experiencing rapid irregularities, before ultimately moving to a dissipative regime off-camera. This could be characterized by Kelvin-Helmholtz instability between the velocity different between two fluids: namely the plume vapor and surrounding air. The frequency of oscillations would relate to

| | Root curve | Linear | Parametric-Quadratic | Quadratic |
|---|---|---|---|---|
| Root curve | N/A | 0.0 | 0.0 | 0.0 |
| Linear | 1.0 | N/A | 0.0 | 0.0 |
| Parametric-Quadratic | 1.0 | 1.0 | N/A | 0.0 |
| Quadratic | 1.0 | 1.0 | 1.0 | N/A |



Table 2: P-Values for one-direction T-Test that method on left is more often a better curve-fitting method for extrapolation than method across top

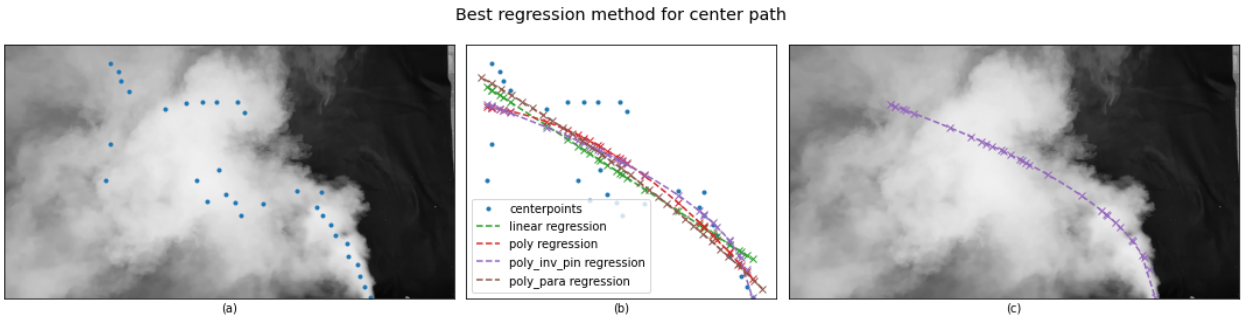



Figure 4: The experiment to find the best regression method for the plume path on a given frame: (a) The points of highest pixel intensity discovered by the previous step, overlayed on original frame. (b) A regression of each method against the centerpoints, up until the split between training and validation points. (c) The regression with the lowest validation score, overlaid on the original frame.

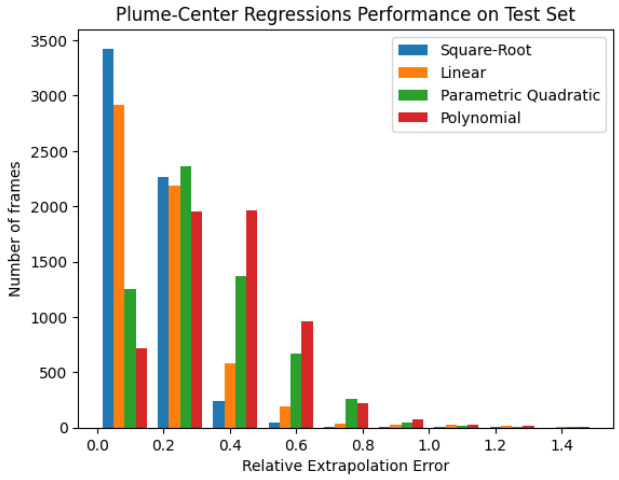



Figure 5: A histogram of the extrapolation errors for different regression methods across $\sim$ 6000 frames of test data. The square root regression performs the best, followed by linear, parametric quadratic, and then quadratic.

the mass flow rate of the plume emission to which we hope to identify the frequency directly form video data.

We opt to model the edge spread as a distance from the mean path of the plume trajectory. That is, before any regression, the edge points identified along the contour of the plume, $\psi_i$, from the step of concentric circles, are converted from Cartesian coordinates $(x, y)$ to the ordered pair $(t, r, d)$–where $d$ is the distance from the edge point to the corresponding center point identified along the concentric circle with radii $r$, at time $t$. For any given video this is done across all edge points from all frames. This "*flattening*" process transforms the data into a form more easily interpretable to apply regression.

126

Due to the oscillatory nature observed in the raw video data, we fit a growing sinusoid function of the form

$$d(r, t) \triangleq A \sin(\omega r - \gamma t + B) + Cr + D \tag{1}$$

to the flattened data, where $(A, \omega, \gamma, B, C, D)$ are to be discovered. For all videos a bootstrap method was used to reduce data variability. Approximately 80% of the data was randomly selected for training. Upon which, sampling with replacement was done to create 2000 to bags to fit equation (1) too. Fitting is done by solving the unconstrained optimization problem via the Levenberg-Marquardt algorithm, to which if no optimal parameters are found within a given number of iterations, the trail is rejected and disregarded. The mean parameters from the non rejected trials are selected for the final model output. The learned function is mapped back onto the original frames by a processing of *unflattenning*. Where points calculated from (1) are shifted by their polar angles against either the true center points or the learned center path regression. An example of points mapped back onto the original frames, where shifts were applied against the true center points can be seen in figure 7.

A distribution of learned parameters can been seen in figure 6. Where we note the observations (occurring frequently across all videos) namely the tight distribution for the "bottom" set of parameters and the bimodal behavior of the amplitudes, $A$. In general both the train and validation accuracies, shown in table 3, tended to also be higher on the "bottom" fit of frames. This is likely a computational artifact of the plume formation progressing off screen at the bottom, causing edge points to be identified along the boundaries of the video. Leading to a simple function to be fit.

Despite the narrow range of frequencies, the amplitude appears bimodal: positive and negative modes offset each other, resulting in a mean near zero. Such bimodal behavior, exhibited in figure 6, could be an artifact of the nonconvex optimization problem, but could just as well indicate a nonlinear phase shift. In order to investigate further, an alternative approach to frequency identification would be a 2D Fourier transform. Because our data is ragged, such a Fourier transform would need to be formulated as a matrix completion problem, but would be convex and could provide additional insights to the instability based upon a range of frequencies.

| **Video ID** | **Train Accuracy** | | **Validation Accuracy** | |
|---|---|---|---|---|
| | **Top** | **Bot** | **Top** | **Bot** |
| low 862 | 0.6166 | 0.3847 | 0.6097 | 0.3910 |
| low 865 | 0.4681 | 0.5089 | 0.4689 | 0.5106 |
| low 867 | 0.4005 | 0.4957 | 0.3982 | 0.5073 |
| low 869 | 0.4063 | 0.4985 | 0.4083 | 0.5026 |
| low 913 | 0.5581 | 0.6544 | 0.5563 | 0.6501 |
| med 871 | 0.4835 | 0.4937 | 0.4868 | 0.4936 |
| med 914 | 0.4610 | 0.6150 | 0.4562 | 0.6151 |
| med 916 | 0.4279 | 0.4821 | 0.4280 | 0.4801 |
| hi 919 | 0.4180 | 0.4992 | 0.4172 | 0.5059 |
| hi 920 | 0.5006 | 0.5364 | 0.4939 | 0.5411 |

Table 3: Train and Validation Accuracy (Top/Bottom)

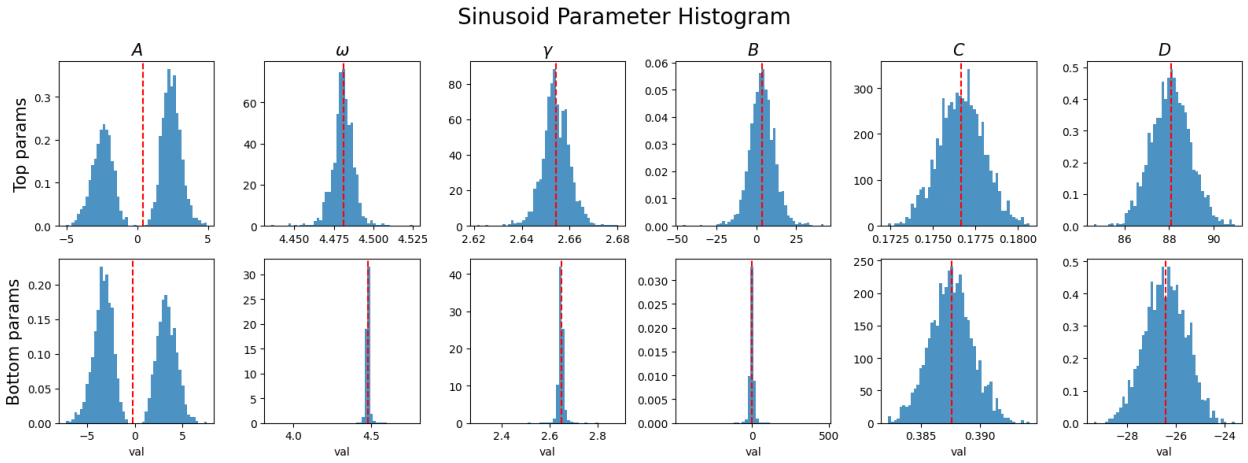



Figure 6: Histogram of amplitude and frequencies for video 920 across 2000 bootstrap bags. Top row denotes learned parameters for top sinusoid fit and bottom row is for the bottom edge of plume. The redline denotes the mean selected parameters used for the final model.

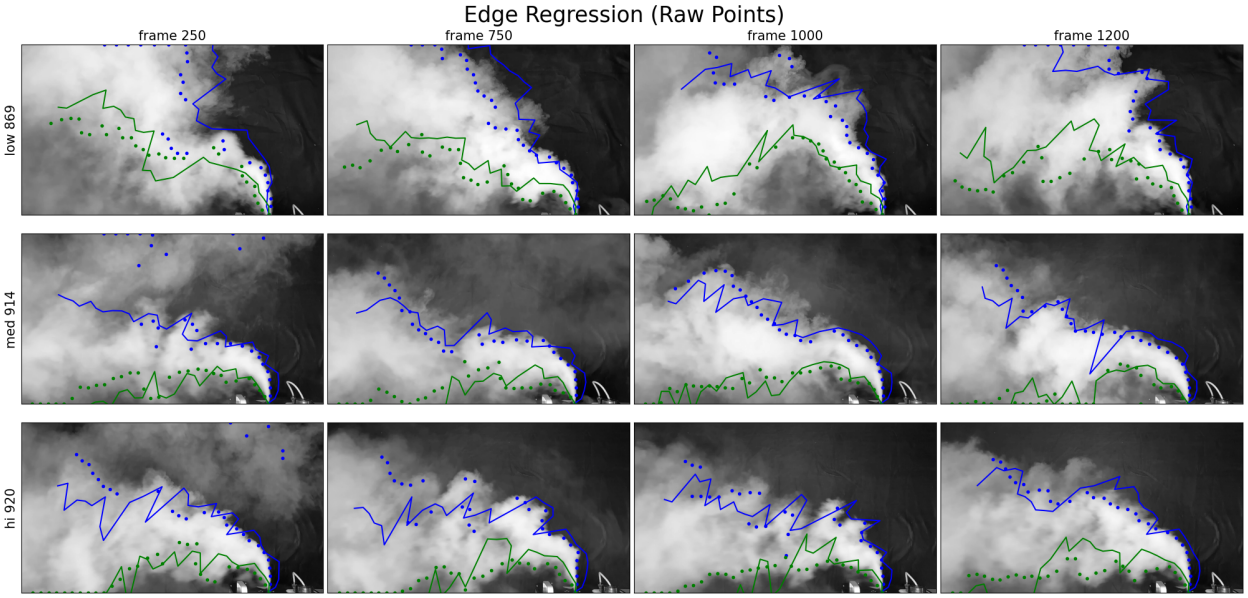
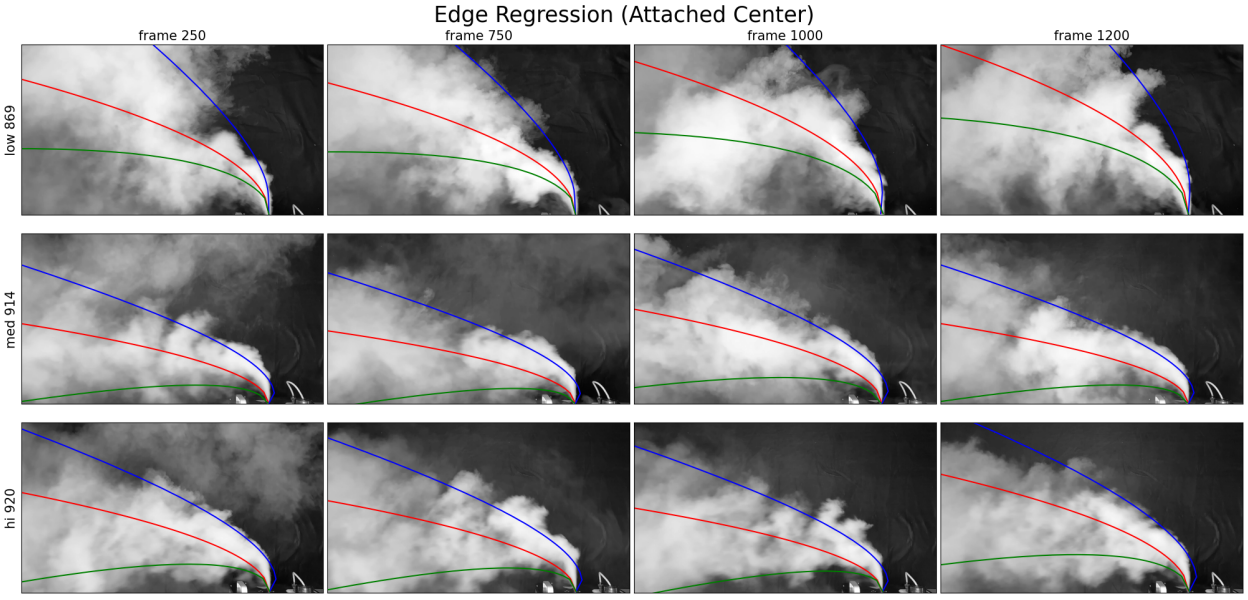



Figure 7: Edge regression path shifted by true center points detected.




128

Figure 8: The complete model applied to a video. Edge model is shown attached to the regressed center, rather than the raw center points.

# 4 Discussion

The result of the complete pipeline is demonstrated in Figure 8. Experiments have borne out the decision to treat the plume as following a square-root curve, which maintains low relative error in extrapolation. Alternate curve-fitting performed poorer in extrapolation, though promising variations remain unexplored. For modeling the edge of the plume, we found that a linear relationship, constant in time, does reasonably well to describe the edge of the plume as it drifts away from the source. While regression found substantial sinusoidal behavior, more work needs to be done to identify frequencies associated with the Kelvin-Helmholtz instability, and from there determine plume emission rate.

**Algorithm 1** Video to Reduced Order Model

**Clean video**
    1a. Background subtract
    1b. Apply Gaussian blur
**Calculate Plume Points**
    2a. Compute contours using Otsu's method
**for** each range ring from plume origin to edge of image **do**
    2bi. Of all contour pixels crossed by the Concentric Circle (CC), select the pixel with max intensity as center
    2bii. Find intersections of contour and CC
    2biii. Perform barycentric interpolation of intersections onto the range ring
    2biv. Choose the furthest clockwise (CW) and counterclockwise (CCW) interpolants as edge points
**end for**
**for** each frame **do**
    Regress the height of the plume center as a square-root function of horizontal distance
**end for**
Regress edge points for each frame

This effort fits within the ambit of "Go Pro physics", which seeks to conduct investigations on raw video of

dynamical phenomena. In the real world, however, conditions will be less than ideal. Issues such as camera jitter or low plume contrast would degrade the video, and its unclear how the difficulty would scale with these introduced factors. In particular, we aimed to remove as much Gaussian blur as possible, as additional smoothing in the first step would buoy regression metrics, potentially obscuring problems in our method. When using noisier data from the field, more blurring may be needed.

A more robust dataset would include not just more videos, but from more angles and with a wider aspect in order to capture higher Reynolds number behavior. The experiments could also be improved, as mentioned in their respective sections. But more ambitiously, once we can get useful frequencies from the we could simulate a plume using an accurate CFD engine such as OpenFoam[2] This would allow us, knowing the physics parameters a priori, to evaluate how well the discovered frequencies reflect the Kelvin-Helmholtz frequencies and thus the mass flow rate. In doing so, it may help to cluster the plume points in space and time in order to separate the laminar flow, unstable flow, and turbulent dissipation, e.g. using Spatio-temporal K-means [35]. More ambitiously, it may be possible to get a more robust model from using time-series data reduction models such as Dynamic Mode Decomposition [27] or Sparse Identification of Nonlinear Dynamics [36] directly on the raw video. These methods may also be able to directly recover the oscillatory frequency or fluid parameters of interest.

129

# 5 Conclusion

We have produced a reduced-order model for a point-source plume that, with enough crosswind, can describe the behavior of a plume on video. The model captures the dominant features of nonlinear plume dynamics using a low dimensional representation that, in contrast to Galerkin models, extrapolates beyond the spatial bounds of the frame. Against a handful of lab-condition but varied datasets, we have refined the parametrization and calibrated its effectiveness. We have opened the door to questions about using this reduced-order model to understand important physical properties of the plume: how much is being emitted in the first place, and how will it dissipate. These questions could, in the long run assist the management of industrial accidents, monitoring of pollution, and understanding natural phenomena such as volcanic and geothermal emissions.

We propose a natural extension to parameter initialization for Gaussian plume modeling, addressing a key challenge in this well-established technique—determining the initialization that corresponds to a desired plume formulation. Our approach integrates a data-driven paradigm to bridge video data with the commonly used first-principles model for plume dynamics, capturing the transition from source-dominated forces to turbulent diffusion and advection. Additionally, we have developed the open-source Python packages `rom-plumes` and `plumex` to facilitate experiment creation and replication.

# Acknowledgements

We would like to thank Laurel Doyle, Applied Mathematics at University of Washington for her early experiments and investigative work into plume modeling in contribution to this paper.

# Appendix

Code for the experiments are split between two Python packages: rom-plumes [37] and rom-plumex [38]. The former includes all functionality a user needs to apply the methods in this paper to their own data. The latter includes the the experiments in this paper, experimental configuration, as well as the code to regenerate the figures. Each experiment is a callable, run through the command line program mitosis [39]. Cloning the rom-plumex repository and editably installing it will install all the required dependencies for the experiments. To exactly reproduce the results of the experiments, download [40] and unzip as a folder

[2] We attempted with Blender, but it only simulates in the purely dissipative regime.

plume_videos in the repository. We have uploaded the mitosis artifacts of our experiment trials to [41], which includes an environment file (requirements.txt) and a source file (source.py), among other plot artifacts, logs, configuration, and data. Once the plume_videos folder is in the repository, installing a particular environment file and running the source file should produce the visual results in the experiment.html file and the python objects in results_0.dill, results_1.dill, etc. where each file refers to a different step of the experiment. To re-run the experiments with different parameters, see the documentation for mitosis.
While different experiments were run on different commits of rom-plumex, the final commit as of publication is tagged 0.2

# References


[1] Renee McVay. Methane emissions from u.s. gas pipeline leaksrenee mcvay, 2023.

[2] Ilissa B Ocko, Tianyi Sun, Drew Shindell, Michael Oppenheimer, Alexander N Hristov, Stephen W Pacala, Denise L Mauzerall, Yangyang Xu, and Steven P Hamburg. Acting rapidly to deploy readily available methane mitigation measures by sector can immediately slow global warming. *Environmental Research Letters*, 16(5):054042, 2021.

[3] Gunnar Myhre, Drew Shindell, F-M Bréon, William Collins, Jan Fuglestvedt, Jianping Huang, Dorothy Koch, J-F Lamarque, David Lee, Blanca Mendoza, et al. Anthropogenic and natural radiative forcing. *Climate Change 2013-The Physical Science Basis*, pages 659–740, 2014.

[4] Ilissa B Ocko, Vaishali Naik, and David Paynter. Rapid and reliable assessment of methane impacts on climate. *Atmospheric Chemistry and Physics*, 18(21):15555–15568, 2018.

[5] Gary S Settles. Fluid mechanics and homeland security. *Annu. Rev. Fluid Mech.*, 38(1):87–110, 2006.

[6] Gary A Briggs. Plume rise predictions. In *Lectures on air pollution and environmental impact analyses*, pages 59–111. Springer, 1975.

[7] S Pal Arya et al. *Air pollution meteorology and dispersion*, volume 310. Oxford University Press New York, 1999.

[8] John M Stockie. The mathematics of atmospheric dispersion modeling. *Siam Review*, 53(2):349–372, 2011.

[9] Yoshihide Tominaga and Ted Stathopoulos. Cfd simulations of near-field pollutant dispersion with different plume buoyancies. *Building and Environment*, 131:128–139, 2018.

[10] Julia E Flaherty, David Stock, and Brian Lamb. Computational fluid dynamic simulations of plume dispersion in urban oklahoma city. *Journal of Applied Meteorology and Climatology*, 46(12):2110–2126, 2007.

[11] Huixin Ma, Xuanyi Zhou, Yoshihide Tominaga, and Ming Gu. Cfd simulation of flow fields and pollutant dispersion around a cubic building considering the effect of plume buoyancies. *Building and Environment*, 208:108640, 2022.

[12] Schalk Cloete, Jan Erik Olsen, and Paal Skjetne. Cfd modeling of plume and free surface behavior resulting from a sub-sea gas release. *Applied Ocean Research*, 31(3):220–225, 2009.

[13] Steven R Hanna, Michael J Brown, Fernando E Camelli, Stevens T Chan, William J Coirier, Olav R Hansen, Alan H Huber, Sura Kim, and R Michael Reynolds. Detailed simulations of atmospheric flow and dispersion in downtown manhattan: An application of five computational fluid dynamics models. *Bulletin of the American Meteorological Society*, 87(12):1713–1726, 2006.

[14] CH Bosanquet and James L Pearson. The spread of smoke and gases from chimneys. *Transactions of the Faraday Society*, 32:1249–1263, 1936.

[15] OG Sutton. The problem of diffusion in the lower atmosphere. *Quarterly Journal of the Royal Meteorological Society*, 73(317-318):257–281, 1947.

[16] John S Irwin. A historical look at the development of regulatory air quality models for the united states environmental protection agency. 2002.

[17] Milton R Beychok. *Fundamentals of stack gas dispersion*. MR Beychok, 2005.

[18] Gary A Briggs. A plume rise model compared with observations. *Journal of the Air Pollution Control Association*, 15(9):433–438, 1965.

[19] GMD Joseph, DM Hargreaves, and IS Lowndes. Reconciling gaussian plume and computational fluid dynamics models of particulate dispersion. *Atmospheric Environment: X*, 5:100064, 2020.

[20] Silvana Di Sabatino, Riccardo Buccolieri, Beatrice Pulvirenti, and Rex Britter. Simulations of pollutant dispersion within idealised urban-type geometries with cfd and integral models. *Atmospheric environment*, 41(37):8316–8329, 2007.

[21] SA Silvester, IS Lowndes, and DM Hargreaves. A computational study of particulate emissions from an open pit quarry under neutral atmospheric conditions. *Atmospheric Environment*, 43(40):6415–6424, 2009.

[22] IS Lowndes, SA Silvester, SW Kingman, and DM Hargreaves. The application of an improved multi-scale computational modelling techniques to predict fugitive dust dispersion and deposition within and from surface mining operations. In *Proceedings of 12th US/North American Mine Ventilation Symposium, Wallace (ed)*, pages 359–366. Citeseer, 2008.

[23] Qingqing Pan, Stein T Johansen, Schalk Cloete, Mark Reed, and Lars Sætran. An enhanced k-epsilon model for bubble plumes. In *Proceedings of the Eighth International Conference on Multiphase Flow*, 2013.

[24] John Leask Lumley. The structure of inhomogeneous turbulent flows. *Atmospheric turbulence and radio wave propagation*, pages 166–178, 1967.

[25] Philip Holmes. *Turbulence, coherent structures, dynamical systems and symmetry*. Cambridge university press, 2012.

[26] Clarence W Rowley. Model reduction for fluids, using balanced proper orthogonal decomposition. *International Journal of Bifurcation and Chaos*, 15(03):997–1013, 2005.

[27] J Nathan Kutz, Steven L Brunton, Bingni W Brunton, and Joshua L Proctor. *Dynamic mode decomposition: data-driven modeling of complex systems*. SIAM, 2016.

[28] Jonathan H Tu. *Dynamic mode decomposition: Theory and applications*. PhD thesis, Princeton University, 2013.

[29] Shahin Masoumi-Verki, Fariborz Haghighat, and Ursula Eicker. A review of advances towards efficient reduced-order models (rom) for predicting urban airflow and pollutant dispersion. *Building and Environment*, 216:108966, 2022.

[30] D Xiao, F Fang, J Zheng, CC Pain, and IM Navon. Machine learning-based rapid response tools for regional air pollution modelling. *Atmospheric environment*, 199:463–473, 2019.

[31] Bastien X Nony, MC Rochoux, Thomas Jaravel, and Didier Lucor. Reduced-order modeling for parameterized large-eddy simulations of atmospheric pollutant dispersion. *Stochastic Environmental Research and Risk Assessment*, 37(6):2117–2144, 2023.

[32] Rui Fu, Dunhui Xiao, Ionel Michael Navon, Fangxin Fang, Liang Yang, Chengyuan Wang, and Sibo Cheng. A non-linear non-intrusive reduced order model of fluid flow by auto-encoder and self-attention deep learning methods. *International Journal for Numerical Methods in Engineering*, 124(13):3087–3111, 2023.

[33] Stefania Fresca, Luca Dede', and Andrea Manzoni. A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized pdes. *Journal of Scientific Computing*, 87:1–36, 2021.

[34] Songlin Xiang, Xiangwen Fu, Jingcheng Zhou, Yuqing Wang, Yizhou Zhang, Xiurong Hu, Jiayu Xu, Huazhen Liu, Junfeng Liu, Jianmin Ma, et al. Non-intrusive reduced order model of urban airflow with dynamic boundary conditions. *Building and Environment*, 187:107397, 2021.

[35] Olga Dorabiala, Devavrat Vivek Dabke, Jennifer Webster, Nathan Kutz, and Aleksandr Y. Aravkin. Spatiotemporal k-means. *ArXiv*, abs/2211.05337, 2022.

[36] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113, 2016.

[37] Ike Griss Salas and Jacob Stevens-Haas. rom-plumes, October 2024.

[38] Ike Griss Salas and Jacob Stevens-Haas. rom-plumex, October 2024.

[39] Jacob Stevens-Haas. mitosis, April 2024.

[40] Ike Griss Salas. Data in support of Plume Dynamics Reduced- Order Models, October 2024.

[41] Ike Griss Salas and Jacob Stevens-Haas. Experiments in support of Plume Dynamics Reduced- Order Models, October 2024.