INSTITUT FÜR MATHEMATIK

TUHH Technische Universität Hamburg

Data-Driven Approaches for the Maxey-Riley Equation

Masterarbeit

von
Niklas Dieckow
aus Hamburg
Matrikelnummer: 7480680
Studiengang: Technomathematik

November 26, 2023

Erstprüfer: Dr. Jens-Peter M. Zemke
Zweitprüfer: Prof. Dr. Daniel Ruprecht
Betreuer: Dr. Jens-Peter M. Zemke

Eidestattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Masterstudiengang Technomathematik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ort, Datum

Unterschrift

Contents

List of Figures

Chapter 1

Motivation

Many phenomena in the natural sciences can be described in the language of differential equations, from transmission of heat [19] in the realm of physics, over the predator-prey model [1] in biology, all the way to studying reaction rates in chemical kinetics [31]. What these examples all have in common is their simplicity and relative ease of discovery.

As an example, consider the predator-prey (or Lotka-Volterra) model [1, pp. 43–45]. We have two populations, say rabbits and wolves, with respective population amounts of $r(t)$ and $w(t)$ at time t. If there were no wolves, and assuming unlimited food supply, the rabbit population would grow exponentially, hence $\dot{r}(t) = cr(t)$ for some reproduction constant $c > 0$. If the number of wolves is non-zero, it is reasonable to assume a decline in growth proportional to the number of rabbits multiplied by the number of wolves, so the new and improved equation is $\dot{r}(t) = r(t)(-aw(t) + c)$, where $a > 0$. We can proceed similarly for the wolves. If there are no rabbits, the wolves die of starvation at a rate proportional to their population, so $\dot{w}(t) = -dw(t)$. If rabbits are available, there is an increase in growth, again proportional to $r(t) \cdot w(t)$, with proportionality constant $b > 0$. Hence, the final system becomes

$$\begin{aligned} \dot{r}(t) &= r(t)(-aw(t) + c), \\ \dot{w}(t) &= w(t)(br(t) - d). \end{aligned}$$

See Figure 1.1 below for an example of the Lotka-Volterra dynamics.

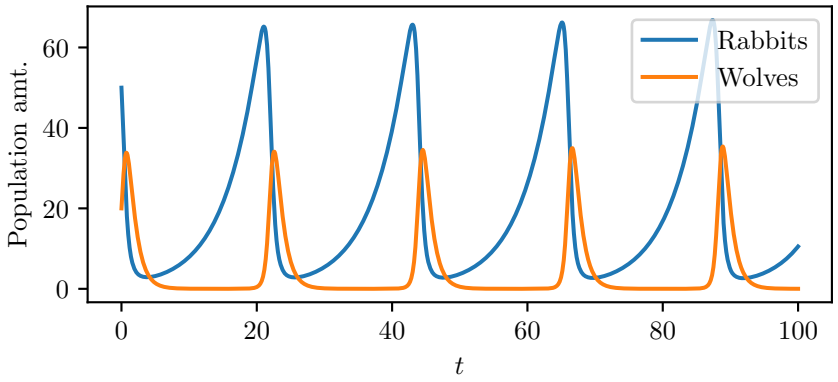


Figure 1.1: Solution of the Lotka-Volterra system over the time period $(0, 100)$ with parameters $a = 0.05, b = 0.05, c = 0.2$ and $d = 1.0$ as well as initial condition $(r(0), w(0)) = (50, 20)$.

However, the discovery of these *governing equations* is rarely such a straightforward endeavor once the complexity of the studied phenomena increases. More often, it takes clever minds and brilliant ideas along with countless experiments to come up with a mathematical model that accurately describes the phenomenon under study. Usually, this is a process spanning several years or decades, and starts with simple models that gradually become more and more refined. Even the Lotka-Volterra model is just an approximation to a complex biological process, and several environmental factors, such as seasonal effects, potential resource scarcity and other predatory species, have been ignored.

As another example, we may consider the process of a small spherical particle settling under gravity in a fluid at rest. Perhaps a grain of sand in a bathtub, or a particle of saliva in the air emitted by sneezing. Newton's second law states that the governing equation must be of the form $F = ma$, with m being the particle's mass, a its acceleration and F the sum of all forces acting on the particle. The big question mark is F — which forces act on the particle? Gravity, drag and buoyancy come to mind. One might proceed by incorporating these forces into the equation, but some experiments may indicate that this is not yet the full picture. Or perhaps they are correct but one wishes to expand this model, such that it also supports moving, non-constant flows. In either case, the procedure is not as straightforward as it was with the Lotka-Volterra equations. There is in fact an equation describing this phenomenon, called the *Maxey-Riley equation* (also MRE from now on). A detailed historic account of the development of this equation, which indeed took more than a century and several different scientific minds, is given in Section 3.1.2.

Based on this, one could ask: What if it were possible to automate the discovery of these equations? What if there was an algorithm that consumed data and spat out a differential equation that accurately models said data? With recent developments in the field of machine learning and the increasing amount of available data and computing power, this idea might not seem so far-fetched. Deep learning methods have been shown to outperform humans in medical imaging tasks [32], and Google DeepMind's AlphaZero – a product of reinforcement learning coupled with deep learning – is able to beat even the most skilled human players at complex games such as chess and Go [51]. However, it is crucial to note that discovering governing equations is a more intricate challenge compared to the tasks mentioned earlier. While machine learning can outperform humans in specific domains, the identification of governing equations demands a level of sophistication that might border on superhuman AI capabilities. Even if achievable, the question arises whether the automation of science is a desirable goal. The authors of one of the seminal papers on system identification propose an alternative perspective, which promotes human-AI cooperation instead of automation: "Might this process diminish the role of future scientists? Quite the contrary: Scientists may use processes such as this to help focus on interesting phenomena more rapidly and to interpret their meaning" [49].

Early approaches for the discovery of governing equations have employed symbolic regression, usually realized through genetic programming. In their 2009 seminal paper [49], Schmidt and Lipson develop an algorithm to discover physical invariants based on time series data and successfully apply it to a number of physical toy problems, including an air-track oscillator and a double pendulum swing. Their method initially generates symbolic functions at random. After every epoch, the fitness of each generated function f is evaluated. This is done by comparing its partial derivatives to the numerically calculated partial derivatives of the time series data, thus aiming for

$$\frac{\partial f}{\partial x} \Big/ \frac{\partial f}{\partial y} \approx \frac{\mathrm{d}x}{\mathrm{d}t} \Big/ \frac{\mathrm{d}y}{\mathrm{d}t}$$

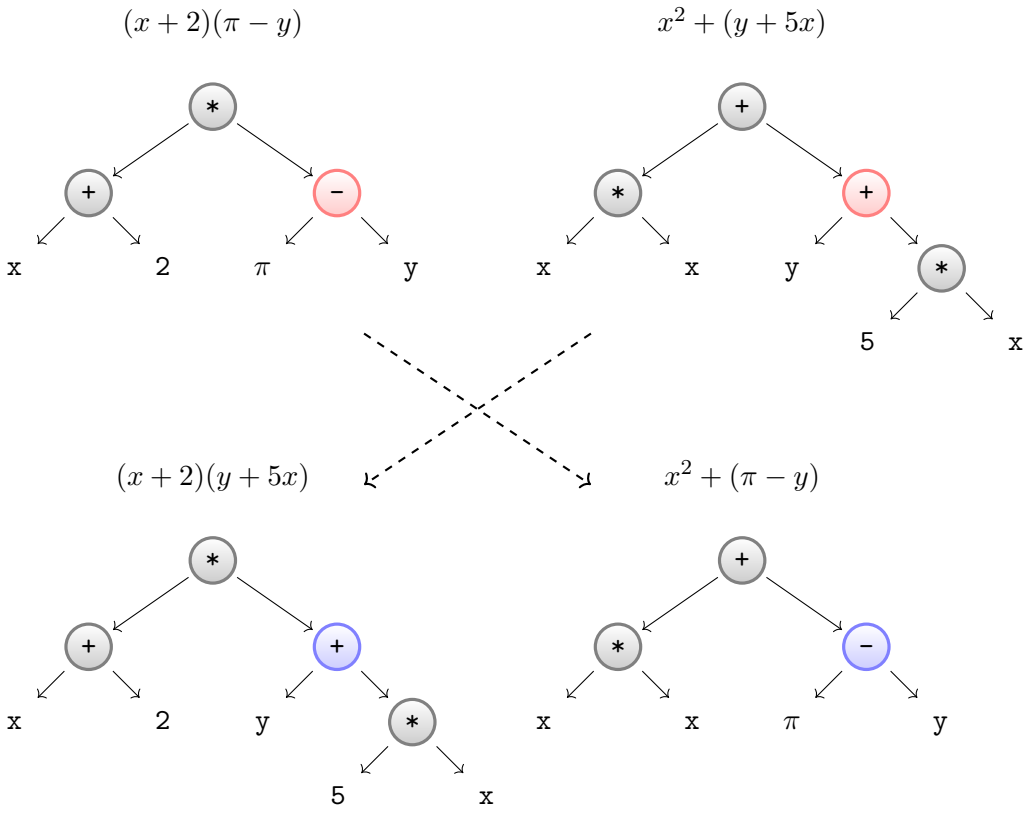


Figure 1.2: Example of a "genetic crossover" of two mathematical expressions. In both parents, a random subtree is chosen and swapped with the subtree of the other parent, producing two child expressions.

in the two-dimensional case. Only the best functions proceed to the next generation. The "surviving" candidate functions are augmented using mutations and crossovers as follows: By considering the parse tree of a computer program or a mathematical expression, two programs can be crossed, or "genetically bred," by choosing a random subtree in each of the programs and swapping them, as shown in Figure 1.2, creating two children in the process. As this is an operation on the level of parse trees, correctness of the resulting programs is ensured. A population is subsequently evaluated based on a fitness measure, causing the fittest programs to survive and produce new crossovers [42]. After sufficient fitness has been achieved, the most parsimonious function is selected as the final invariant.

The idea of applying the principle of natural selection to computer programs had already been mentioned by Alan Turing in his work "Computing Machinery and Intelligence" [34]. The theoretical groundwork was laid by John Holland in his 1975 book "Adaptation in Natural and Artificial Systems" [24] and the field was further developed and popularized by his PhD student John R. Koza [43].

A different, but related, approach is the *Approximate Vanishing Ideal* algorithm proposed by Heldt et al. in 2009 [25]. Given a number of data samples, the algorithm finds a set of polynomials that are close to 0 at each sampling point, thus uncovering polynomial relations in the data. The authors describe a possible application for the task of feature selection.

A more recent approach, known as the **S**parse **I**dentification of **N**onlinear **Dy**namics (SINDy) algorithm, relies on the assumption that the right-hand-side of the dynamics can be expressed

as a linear combination of candidate functions. If derivative data are available, one can express the problem as a system of linear equations, which can then be solved by any number of sparse regression methods, ideally yielding an interpretable dynamical system. We take a closer look at SINDy and such sparse regression methods in Section 3.2.

Comparing SINDy to symbolic regression, the latter exhibits the capability of building more complex equations from fewer building blocks. For example, employing just $+$, sin and cos, symbolic regression would be able to construct the function $\sin(1 + \cos(x))$. As SINDy uses candidate functions as a linear basis, it could only discover this function if it was already part of the candidate library. On the other hand, linear models are generally easier to interpret, provided that each of the basis functions is interpretable. Symbolic regression introduces a caveat, as it may produce highly complex expressions that fit the observed data well, but prove challenging, or even impossible, to interpret – a phenomenon commonly known as *overfitting*.

Chapter 2

Introduction

The aforementioned Maxey-Riley equation is a system of integro-differential equations, which can be written in the following form,

$$\frac{1}{R}\dot{\mathbf{v}} = \frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t} - \frac{1}{S}(\mathbf{v}-\mathbf{u}) + \sqrt{\frac{3}{\pi}\frac{1}{S}}\frac{\mathrm{d}}{\mathrm{d}t}\int_0^t \frac{\mathbf{v}-\mathbf{u}}{\sqrt{t-s}}\,\mathrm{d}s, \tag{2.0.1}$$

with dimensionless parameters $R, S \in \mathbb{R}$ we shall not concern ourselves with at the moment. It describes the motion of a small, spherical particle suspended in a fluid flow that is described by the vector field $\mathbf{u}\colon \mathbb{R}^n \times \mathbb{R}_{\geqslant 0} \to \mathbb{R}^n$. The particle's absolute velocity is denoted by $\mathbf{v}\colon \mathbb{R}_{\geqslant 0} \to \mathbb{R}^n$. In general, we would expect $\mathbf{v}$ to be close to, or at least strongly influenced by, the surrounding flow field $\mathbf{u}$. Still, even a small particle interacts with the fluid and is subject to drag, drift, buoyancy and other forces, causing the resulting behavior to be different from what is dictated by the flow field. This deviation $\mathbf{v}-\mathbf{u}$ is called the *slip velocity* of the particle, and whether it is zero or not distinguishes *tracers* – usually radioactive substances that follow the flow field precisely – from *inertial particles*.

Most unusual about equation (2.0.1) is the so-called *Boussinesq-Basset force*, corresponding to the last term of the equation, which involves the derivative of an integral with a singular kernel. This integral is taken over the entire history of the particle, because the past movement of the particle locally perturbs the flow field it is subject to. This naturally has an effect on the current movement of the particle. A consequence is that any exact solver for equation (2.0.1) needs to recompute the history force at each time step, requiring the entire particle history to be stored in memory and causing the computation time to grow quadratically in the number of time steps [16]. As the history force is integral to the equation, this increase in complexity cannot be avoided without making approximations. One mitigation is to consider higher-order methods, as they require fewer time steps. Numerical methods of arbitrarily high order are constructed in [16].

In certain applications, it may however be desirable to sacrifice accuracy for speed of computation. An example of this is optimization, as it requires repeated calls to a solver; the faster the solver, the quicker the optimization. A different example is given by closed-loop control systems, as they require the availability of solutions in real time. Approximate models used for these kinds of applications are often referred to as *surrogate models*. These, in turn, often come in the form of *grey-box* models, which are evidently a mixture of *white-* and *black-box* models. While white-box models are exact physical models that were constructed from theory and experiments, black-box models are fully data-driven models, such as neural networks. Thus,

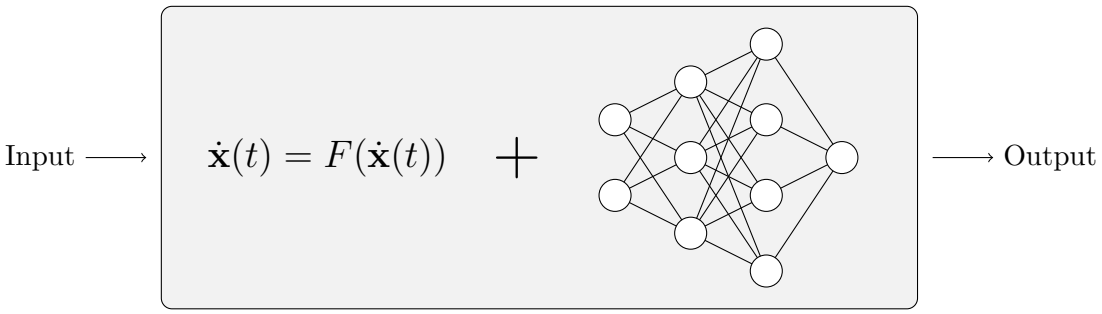


Figure 2.1: Symbolic representation of grey-box models. They combine physical information, such as known differential equations, with data-driven approaches, such as neural networks. Note that the "+" symbol does not explicitly refer to addition but rather some form of combination in general, although the former is a frequently used way of combining physical models with data-driven models.

grey-box models represent the middle ground: Partially exact models that are augmented with data-driven methods.

An example of a technique usable for black- and grey-box modeling are neural ordinary differential equations [13], in which the right-hand side of an ODE is replaced by a neural network,

$$\frac{d\mathbf{h}(t)}{\mathrm{d}t} = f(\mathbf{h}(t), t; \theta).$$

This is a special case of the much more general framework of universal differential equations (UDE) [44], which, for instance, also allows to add known terms of the dynamics to the right-hand side. Physics-informed neural networks [45] are another method for including prior physical knowledge or constraints into a neural network-based model. Contrary to the UDE approach, the goal is not to learn the right-hand side of the equation, but its solution from limited data. This is achieved by incorporating a known differential equation into the loss function, causing the network to naturally learn to adhere to the dynamics.

The goal of this thesis is to find a differential equation that approximates the Maxey-Riley equation. While several methods to do this are available, as outlined, we focus exclusively on the SINDy algorithm [7]. Compared to approaches like Neural ODEs mentioned above, which also learn the right-hand side of an ODE from data, SINDy has the advantage of producing interpretable models.

Chapter 3 is dedicated to a discussion of the preliminaries. In 3.1, after informally introducing some basic terminology from fluid dynamics, the history and development of the Maxey-Riley equation is outlined, and the different components of the force F are explained. In Chapter 3.2, the SINDy algorithm is introduced and studied in closer detail.

In Chapter 4, the different approaches to and results of our numerical experiments are documented. The experiments are performed for two velocity fields of different difficulty.

Finally, in Chapter 5, the results are summarized and an outlook on possible future work is given.

Chapter 3

Preliminaries

In this chapter, we introduce the reader to the Maxey-Riley equations and SINDy, assuming no prior experience in either fluid dynamics or data-driven discovery methods of governing equations.

3.1 The Maxey-Riley Equations

Before the Maxey-Riley equation can be understood, we first introduce, on an informal level, several concepts from fluid dynamics that are required to understand the Maxey-Riley equations. Afterwards, a historical account of the development of the equation is given, while simultaneously explaining each of the terms. In a third section, more attention is placed on the history force already mentioned in the introduction.

3.1.1 Terminology from Fluid Dynamics

Fluid. A *fluid* is a liquid or gas that undergoes a continuous deformation when subject to shearing forces. The definition given in [52], a standard textbook on fluid mechanics, is the following: "the shearing forces necessary to deform a fluid body go to zero as the velocity of deformation tends to zero". A fluid can be characterized by several parameters, including density ρ, which is the ratio between mass and volume, and viscosity μ, which is explained next.

Viscosity. *Viscosity* is a measure of the resistance of a fluid against shear forces. An example of a fluid with high viscosity is honey, while water has low viscosity. To quantify viscosity, consider a thin layer of fluid between two solid plates with surface area A that are a distance d apart from each other, see Figure 3.1. The bottom plate is stationary while the upper plate moves with constant velocity v in one direction. Due to molecular phenomena, the velocity of a fluid at the surface of a solid is zero. This is known as the *no-slip condition*. Therefore, the lowest layer of the fluid layer is stationary, while the highest layer moves with velocity v. Between the lowest and highest layer, the velocity increases linearly. One can experimentally observe that the force F required to keep the upper plate moving at speed v is proportional to $A\frac{v}{d}$, that is,

$$F = \mu A \frac{v}{d}.$$

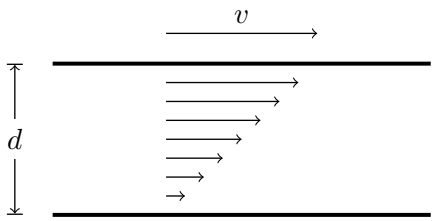


Figure 3.1: Illustration of the conceptual setup used to define viscosity.

The proportionality constant μ is then called *dynamic viscosity* [21]. Dividing by the fluid's density, one obtains the *kinematic viscosity* $\nu \coloneqq \frac{\mu}{\rho}$.

Types of fluid flow. Let the *velocity field* (or *flow field*) of a fluid flow be denoted by $\mathbf{u}\colon \mathbb{R}^m \times \mathbb{R}_+ \to \mathbb{R}^m$. If $\mathbf{u} \equiv 0$, the fluid is said to be at rest. If the flow is constant in time, that is, $\frac{\partial \mathbf{u}}{\partial t} \equiv 0$, it is said to be *steady* and otherwise *unsteady*. If it is constant in space, that is, $J\mathbf{u} \equiv 0$ (where $J\mathbf{u}$ denotes the Jacobian of $\mathbf{u}$ with respect to $\mathbf{x}$), the flow is called *uniform* and otherwise *nonuniform*.

Boundary Layer. Consider now an object (such as a particle) submerged in a viscous fluid. Due to the no-slip condition, $\mathbf{u}(x) = \mathbf{v}(x)$ holds at any point x on the boundary of the particle. As one moves further away from the boundary, the velocity monotonically increases until reaching the original fluid velocity. This region between the object's boundary and the point at which the original fluid velocity is reached is called the *boundary layer* of the object.

Kolmogorov microscales and Reynolds number. Let ε denote the average rate of dissipation of turbulence kinetic energy per unit mass. The Kolomogorov microscales are defined by

- characteristic length $L = \left(\frac{\nu^3}{\varepsilon}\right)^{1/4}$
- characteristic time $T = \sqrt{\frac{\nu}{\varepsilon}}$
- characteristic velocity $U = (\nu\varepsilon)^{1/4}$,

where ε denotes the mean energy dissipation [29]. Informally, these are the smallest scales at which turbulence phenomena (such as eddies) occur. They satisfy the relationship $UT = L$. The *Reynolds number* of a flow is a dimensionless quantity related to the flow's turbulence. A higher Reynolds number indicates increased turbulence. It is defined as $\mathrm{Re} \coloneqq \frac{UL}{\nu}$.

Material Derivative. In fluid dynamics, there is a distinction between the *Eulerian specification* and the *Lagrangian specification* of the flow field. So far, we have considered the former, where the velocity field is given by a function $\mathbf{u}(\mathbf{x}, t)$. In the latter specification, each so-called *fluid parcel* is considered individually. Labeled $\mathbf{x}_0$ at some initial time, its position at time t is given by $\mathbf{X}(\mathbf{x}_0, t)$. Clearly, the two concepts are linked via

$$\mathbf{u}(\mathbf{X}(\mathbf{x}_0, t), t) = \frac{\partial \mathbf{X}}{\partial t}(\mathbf{x}_0, t). \tag{3.1.1}$$

If we are now given a vector field $\mathbf{F}(\mathbf{x}, t)$ in Eulerian terms, we can easily obtain its Eulerian time derivative, namely as $\frac{\partial \mathbf{F}}{\partial t}$. But we might also be interested in the rate of change with respect to a single flow parcel $\mathbf{x}_0$ – its time-derivative in Lagrangian terms, so to speak. The chain rule gives us

$$\frac{\mathrm{d}\mathbf{F}(\mathbf{X}(\mathbf{x}_0,t),t)}{\mathrm{d}t} = \frac{\partial \mathbf{F}}{\partial t}(\mathbf{X}(\mathbf{x}_0,t),t) + J\mathbf{F}(\mathbf{X}(\mathbf{x}_0,t),t)\cdot\frac{\partial \mathbf{X}}{\partial t}(\mathbf{x}_0,t) \tag{3.1.2}$$

$$\overset{(3.1.1)}{=} \frac{\partial \mathbf{F}}{\partial t}(\mathbf{X}(\mathbf{x}_0,t),t) + J\mathbf{F}(\mathbf{X}(\mathbf{x}_0,t),t)\cdot\mathbf{u}(\mathbf{X}(\mathbf{x}_0,t),t). \tag{3.1.3}$$

Hence, we define the *material derivative* of $\mathbf{F}$ as

$$\frac{\mathrm{D}\mathbf{F}}{\mathrm{D}t} \coloneqq \frac{\partial \mathbf{F}}{\partial t} + J\mathbf{F}\cdot\mathbf{u}.$$

Using the relation $J\mathbf{F}\cdot\mathbf{u} = (\mathbf{u}\cdot\nabla)\mathbf{F}$, we may also define the material derivative as a differential operator

$$\frac{\mathrm{D}}{\mathrm{D}t} \coloneqq \frac{\partial}{\partial t} + \mathbf{u}\cdot\nabla.$$

3.1.2 History of the Maxey-Riley equations

The Maxey-Riley model is a system of integral-differential equations describing the motion of a small, spherical, rigid particle in a nonuniform, unsteady flow at low Reynolds numbers. In its current form, it was first proposed by Maxey and Riley [36].

The Maxey-Riley equations have been used in applications in fields such as oceanography for e.g. the modelling of the drift of *sargassum* rafts in the ocean [5], which is a type of alga. During the recent COVID-19 pandemic, the airborne transmission of the virus has been studied by making use of the Maxey-Riley equations to model the aerosols, the particles with lower density than the surrounding fluid, emitted during talking or sneezing [47]. However, the inverse problem is also of interest: Given a collection of dispersed particles in a fluid, is it possible to find the source or sources? This is studied in [54] and the authors utilize the technique of projecting the particles onto a slow manifold in order to numerically find the source of a simulated anthrax outbreak.

Let $\mathcal{D} \subseteq \mathbb{R}^n$ be open and consider the fluid flow $\mathbf{u}\colon \mathcal{D}\times\mathbb{R}_+ \to \mathbb{R}^n$ with a kinematic viscosity of ν. Let $\mathbf{y}$ be the position of the particle and define $\mathbf{v} = \dot{\mathbf{y}}$ as the velocity of the particle. Any equation describing particle motion is nothing more than Newton's second law $\mathbf{F} = m\dot{\mathbf{v}}$ with an elaborate force term. In 1856, Stokes [53] first derived an expression of the drag force,

$$\mathbf{F}_d = 6\pi\nu\rho_f a(\mathbf{v}-\mathbf{u})$$

for a sphere with radius $a > 0$ with small Reynolds number in a uniform and steady flow. Notably, he assumed the slip velocity $\mathbf{v}-\mathbf{u}$ to be constant. Around 30 years later, Basset [3] and Boussinesq [6] independently extended this model by also incorporating added mass, gravitational force and memory effects. This led to the equation

$$\frac{4\pi a^3}{3}\rho_p\dot{\mathbf{v}} = -\underbrace{\frac{2\pi a^3}{3}\rho_f\dot{\mathbf{v}}}_{\text{Added mass}} \underbrace{- 6\pi\nu\rho_f a\left\{\mathbf{v} + \frac{a}{\sqrt{\pi\nu}}\int_{t_0}^{t}\frac{\dot{\mathbf{v}}(s)}{\sqrt{t-s}}\,\mathrm{d}s\right\}}_{\text{Stokes drag + Basset force}} \underbrace{- \frac{4\pi a^3}{3}(\rho_p-\rho_f)\mathbf{g}}_{\text{Gravity}} \tag{3.1.4}$$

for a fluid at rest, i.e., $\mathbf{u} \equiv 0$. *Added mass*, or *virtual mass*, refers to the additional force required to move the fluid around the particle, as both of them cannot occupy the same space. By rearranging equation (3.1.4) in such a way that the added mass term is on the left-hand side, one sees that this is equivalent to considering a particle with higher mass, thus explaining the name. The *Basset force*, or *history force*, accounts for the viscous effects caused by the particle's acceleration, namely the lagging development of the boundary layer [15]. Put differently, the particle locally modifies the velocity field through its movement, so that it is subject to a slightly different flow than $\mathbf{u}$. This change depends on the entire movement history of the particle, which is why this term is an integral over the time up to t.

To obtain a solution in the setting of a moving fluid that is uniform but unsteady, Tchen [55], in his 1947 PhD thesis, considered a sphere moving with velocity $\mathbf{v} - \mathbf{u}$ in a fluid at rest, allowing him to reuse the solution derived by Basset, Boussinesq and Oseen. This was followed by endowing the entire system with the fluid velocity $\mathbf{u}$, resulting in an additional pressure gradient term,

$$\frac{4\pi a^3}{3}\nabla \mathbf{p},$$

where $\mathbf{p}$ denotes the fluid's pressure field. He further generalized the equation to a flow that was both unsteady and nonuniform; however, Corrsin and Lumley [14] pointed out and corrected flaws in his approach, leading to the form

$$\frac{4\pi a^3}{3}\rho_p \dot{\mathbf{v}} = \frac{4\pi a^3}{3}\rho_f \left(\frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t} - \nu\nabla^2\mathbf{u}\right) - \frac{2\pi a^3}{3}\rho_f \left(\dot{\mathbf{v}} - \frac{\partial \mathbf{u}}{\partial t} - (\mathbf{v}\cdot\nabla)\mathbf{u}\right) \tag{3.1.5}$$

$$- 6\pi\nu\rho_f a(\mathbf{v} - \mathbf{u}) - 6\pi\nu\rho_f a \frac{a}{\sqrt{\pi\nu}} \int_{-\infty}^{t} \frac{\dot{\mathbf{v}}(s) - \frac{\partial \mathbf{u}}{\partial t} - (\mathbf{v}\cdot\nabla)\mathbf{u}}{\sqrt{t-s}}\,\mathrm{d}s \tag{3.1.6}$$

$$- \frac{4\pi a^3}{3}(\rho_p - \rho_f)\mathbf{g}. \tag{3.1.7}$$

In 1983, Maxey and Riley [36] revisited the problem and re-examined the effects of nonuniform flow in particular. In the same year, Gatignol [22] independently arrived at the same result. The nowadays most widely accepted form of the Maxey-Riley equations, with corrections by Auton et al. [2], is

$$\frac{1}{R}\dot{\mathbf{v}} = \frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t} - \frac{1}{S}(\mathbf{v} - \mathbf{u}) + \sqrt{\frac{3}{\pi}\frac{1}{S}}\frac{\mathrm{d}}{\mathrm{d}t}\int_0^t \frac{\mathbf{v} - \mathbf{u}}{\sqrt{t-s}}\,\mathrm{d}s, \tag{3.1.8}$$

where $R = \frac{3\rho_f}{\rho_f + 2\rho_p}$ is the density parameter and $S = \frac{1}{3}\frac{a^2/\nu}{T}$ is the scaled ratio of the particle's viscous relaxation time and the characteristic time of the flow [16]. Note that the so-called Faxén corrections derived in [22, 36] as well as the gravitational force have been omitted.

The Stokes number is defined as

$$\mathrm{St} = \frac{\tau_p}{T} = \frac{S/R}{T}.$$

It indicates the importance of the particle's inertia [17]. Note that some authors may instead define S as the Stokes number.

The derivation of (3.1.8) relies on the assumption that the particle's Reynolds number is much smaller than 1, that is,

$$\mathrm{Re}_p = \frac{|\mathbf{v} - \mathbf{u}|a}{\nu} \ll 1.$$

3.1.3 The History Term

The Basset-Boussinesq force, or history term, in the MRE accounts for the fact that the particle is actually subjected to a slightly different flow field than $\mathbf{u}$, as the particle movement modifies the flow as well. An intriguing property of the history term is its nature as a time-derivative of the slip velocity of fractional order $1/2$. Much like integer exponentiation, usually introduced as repeated multiplication, can be extended to arbitrary real (and also complex) numbers using the exponential function, the same can be done for integration and differentiation. Define the integration operator $\mathcal{I}$ via

$$(\mathcal{I}f)(t) = \int_0^t f(s)\,\mathrm{d}s.$$

In the Riemann-Liouville approach [33], one utilizes the formula for repeated integration,

$$\mathcal{I}^n f(t) = \frac{1}{(n-1)!}\int_0^t (t-s)^{n-1} f(s)\,\mathrm{d}s. \tag{3.1.9}$$

This can be generalized to real numbers using the Gamma function Γ, the natural extension of the factorial to complex numbers, resulting in

$$\mathcal{I}^\alpha f(t) \coloneqq \frac{1}{\Gamma(\alpha)}\int_0^t (t-s)^{\alpha-1} f(s)\,\mathrm{d}s \tag{3.1.10}$$

for $\alpha > 0$. To define the derivative of order $\alpha > 0$, one can combine the fractional integral with integer-order differentiation. By first "integrating up" to the smallest integer n larger than α and then differentiating n times,

$$\mathcal{D}^\alpha f(t) \coloneqq \frac{\mathrm{d}^n}{\mathrm{d}t}(\mathcal{I}^{n-\alpha} f)(t), \tag{3.1.11}$$

one obtains the definition of the αth derivative of f in the Riemann-Liouville sense. Inserting $\alpha = 1/2$ and knowing $\Gamma(1/2) = \sqrt{\pi}$, it is easy to see that

$$\mathcal{D}^{1/2} f(t) = \frac{\mathrm{d}}{\mathrm{d}t}(\mathcal{I}^{1/2} f)(t) = \frac{\mathrm{d}}{\mathrm{d}t}\frac{1}{\sqrt{\pi}}\int_0^t \frac{f(s)}{\sqrt{t-s}}\,\mathrm{d}s. \tag{3.1.12}$$

With $f = \mathbf{v} - \mathbf{u}$, one obtains a scaling of the history force in equation (3.1.8).

Fractional-order derivatives have been linked to memory processes [18]. Indeed, while the particle itself does not have any memory, the effect of the particle's past movement on the surrounding flow can be interpreted as a memory-like effect [17].

The history term complicates the equation whilst only having an impact under certain circumstances, and is thus often neglected in applications. The increased complexity stems from the fact that the flow map $F_{t_0}^t$, which maps a particle position at time t_0 to its position at time t, no longer satisfies the semigroup property $F_{t_0}^t = F_{t_1}^t \circ F_{t_0}^{t_1}$ for all $t_0 < t_1 < t$ [30]. Two particles that are in the same position at t_1 may end up in different positions at t, simply because their history was different, as they started in different positions at t_0, see Figure 3.2. Hence, the Maxey-Riley equations with the history term do not define a dynamical system. This increases the difficulty of both the analysis as well as the numerical solution of the MRE. Regarding the first point, existence and uniqueness of solutions was only recently examined [20]. As for the second point, numerical methods require an evaluation of the history force in each

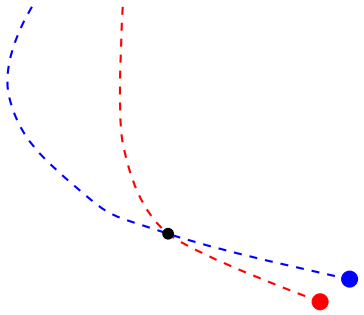


Figure 3.2: Two particle trajectories with different histories share a single point and follow different paths thereafter. This scenario is possible in the Maxey-Riley model because of the history term. Thus, the dynamics do not only depend on the spatial position and the model is not a dynamical system.

time step. This requires the entire particle history to be stored in memory on the one hand, and causes the computation time to grow quadratically in the number of time steps on the other.

Because of this, the history force is often omitted in applications. If the contribution of the history force is small compared to that of the other forces, this is a reasonable approach. In recent years, there have been several publications dedicated to figuring out the importance of this force. In one such publication [17] that considered turbulent flows, it was hypothesized and experimentally validated that the contribution of the history force (in relation to the Stokes drag) is predominantly a function of the particle radius relative to the length scale of the fluid. In particular, the density ratio ρ_p/ρ_f had only a weak influence on the contribution. Denoting the magnitudes of the history force and Stokes drag by a_H and a_S, respectively, the relationship is described by

$$\frac{a_H}{a_S} \approx \alpha \frac{r}{\eta} \tag{3.1.13}$$

for some flow-dependent parameter $\alpha \in \mathbb{R}$. The author concluded that the history force is negligible, if r/η is around 10^{-2} and only starts playing a major role once $r/\eta \approx 1$. Furthermore, the effect of the history force on the slip velocity was analyzed, and the relationship

$$\frac{|\mathbf{v}-\mathbf{u}|_{\text{memory}}}{|\mathbf{v}-\mathbf{u}|_{\text{nomemory}}} \approx \frac{1}{1+\alpha\frac{r}{\eta}}$$

hypothesized and experimentally validated. Once again, the particle radius was the main determinator for the effect of the history force, while the density ratio had only a small influence. In general, the author summarizes the effect of the history force as causing the inertial particles to behave more like tracers, that is, remain closer to the flow.

Figure 3.3 shows an example of a particle trajectory, computed with and without the history term. The velocity field is a simple vortex, and the equation parameters are the same as in Section 4.3, given by Table 4.2. Evidently, the history-free solution experiences a much faster expansion than the one with the history force. If one looks closely at the arrows, one sees that the trajectory computed with the full MRE stays relatively close to the velocity field, while the solution computed without history force follows the arrows to a lesser extent. This is in agreement with Daitche's findings mentioned in the previous paragraph.

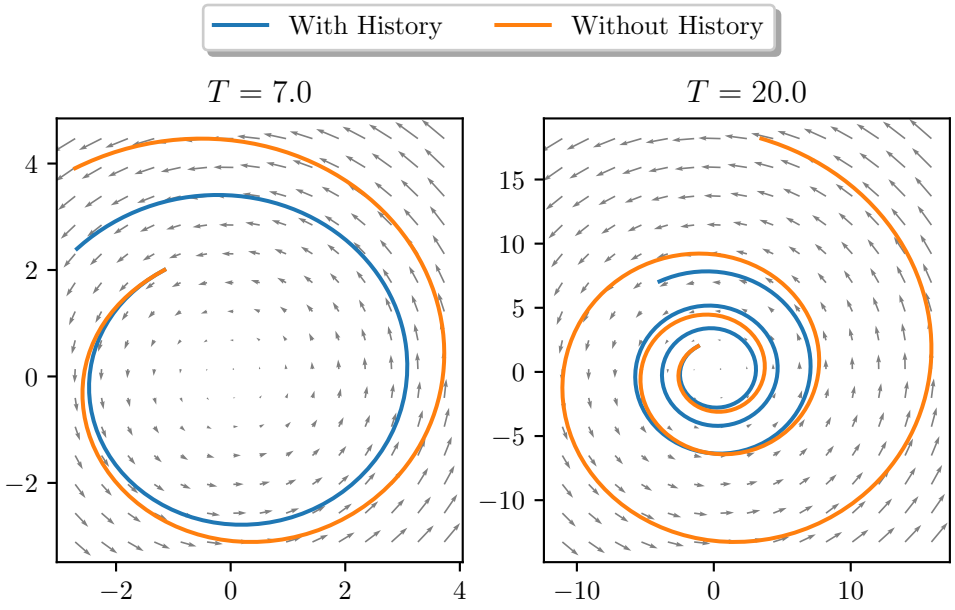


Figure 3.3: Solutions of the Maxey-Riley equation for the vortex velocity field, both with and without the history term. The velocity field is shown in gray in the background. Two different final times $T = 7.0$ and $T = 20.0$ are plotted. The solution without history term has a faster expansion rate than the solution of the full MRE.

3.2 Sparse Identification of Nonlinear Dynamics (SINDy)

SINDy has been introduced in 2016 by Brunton et al. [7] as a novel method for the data-driven discovery of differential equations. In the paper, the authors showcase its viability using a variety of examples, ranging from a simple two-dimensional damped harmonic oscillator all the way to the proper orthogonal decomposition of the Navier-Stokes equations. Several extensions to the SINDy framework have been proposed since, with the aim of either supporting other types of differential equations or improving the robustness. A small number of them are briefly discussed in Section 3.5.

Let $\mathcal{T} = [t_{\text{start}}, t_{\text{end}}]$ be a time interval. We consider a function $\mathbf{x}\colon \mathcal{T} \to \mathbb{R}^d$ described by a dynamical system $\dot{\mathbf{x}}(t) = F(\mathbf{x}(t))$ for an unknown $F\colon \mathbb{R}^d \to \mathbb{R}^d$. In this setting, $\mathbf{x}$ is termed the *state* of the dynamical system. The system is uniquely identified by F and the task of finding the unknown F is termed *system identification*. In the following, we use *system* as short-hand for *dynamical system*. We make the assumption that it is possible to express each component of F in terms of a linear combination of simpler functions, termed *candidate functions*, contained in a pre-selected *feature library* $\{f_i\colon \mathbb{R}^d \to \mathbb{R} \mid i = 1, \dots, m\}$,

$$F = \begin{pmatrix} \sum_{i=1}^m \xi_{i,1} f_i \\ \vdots \\ \sum_{i=1}^m \xi_{i,d} f_i \end{pmatrix} = \begin{pmatrix} \xi_{1,1} & \cdots & \xi_{m,1} \\ \vdots & \ddots & \vdots \\ \xi_{1,d} & \cdots & \xi_{m,d} \end{pmatrix} \begin{pmatrix} f_1 \\ \vdots \\ f_m \end{pmatrix} =: \boldsymbol{\Xi}^\top \mathbf{f}.$$

The validity of this assumption is of course dependent on the choice of the feature library. If the components of F happen to be contained in it, $\boldsymbol{\Xi}$ is trivial. More importantly, it is sparse, which may serve as a good guideline to follow for non-trivial solutions. In practice, one

selects the feature library based on already-known parts of the equation, educated guesses and generic function bases such as (trigonometric) polynomials. In this scenario, for a given feature library, the system identification problem can be formulated as finding a sparse coefficient matrix $\boldsymbol{\Xi} \in \mathbb{R}^{m\times d}$, such that

$$\dot{\mathbf{x}}(t) = F(\mathbf{x}(t)) = \boldsymbol{\Xi}^\top \mathbf{f}(\mathbf{x}(t)) \tag{3.2.1}$$

for all $t \in \mathcal{T}$.

Remark. For ease of notation, we will often omit the time argument when describing a dynamical system, that is, $\dot{\mathbf{x}} = F(\mathbf{x})$ will be shorthand for $\dot{\mathbf{x}}(t) = F(\mathbf{x}(t))$.

In practice, we have access to one or several sampled trajectories $(\mathbf{x}(t_i))_{i=1}^N$, where $t_1 = t_{\text{start}} < t_2 < \cdots < t_N = t_{\text{end}}$ is a sequence of time steps. In general, these do not have to be equidistantly sampled. In case they are, we denote by $\Delta t \coloneqq t_2 - t_1$ the *time step*. In this context, the system identification problem can be considered as a system of linear equations

$$\dot{\mathbf{X}} = \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi}, \tag{3.2.2}$$

where

$$\mathbf{X} = \begin{pmatrix} — & \mathbf{x}(t_1)^\top & — \\ & \vdots & \\ — & \mathbf{x}(t_N)^\top & — \end{pmatrix} \in \mathbb{R}^{N\times d} \quad \text{and} \quad \dot{\mathbf{X}} = \begin{pmatrix} — & \dot{\mathbf{x}}(t_1)^\top & — \\ & \vdots & \\ — & \dot{\mathbf{x}}(t_N)^\top & — \end{pmatrix} \in \mathbb{R}^{N\times d}$$

are the matrices containing the trajectory and derivative data, respectively, at the different time points, and

$$\boldsymbol{\Theta}(\mathbf{X}) \coloneqq \begin{pmatrix} — & \mathbf{f}(\mathbf{x}(t_1)) & — \\ & \vdots & \\ — & \mathbf{f}(\mathbf{x}(t_N)) & — \end{pmatrix} \in \mathbb{R}^{N\times m}.$$

Note that this equation is transposed compared to the previous formulation.

If there is no derivative data available, there are at least two ways to deal with this. The first alternative is to approximate the derivatives numerically with the use of a finite difference scheme. It should however be noted that finite difference methods tend to amplify noise, hence it is preferable to measure the derivatives directly whenever possible, or to use something like total variation regularization, which can be found in [12]. The other alternative is an integral-based approach, outlined in [48, 9], which we briefly recite. First, use the fundamental theorem of calculus to obtain

$$\mathbf{x}(t) = \mathbf{x}(t_1) + \int_{t_1}^{t} \dot{\mathbf{x}}(\tau)\,\mathrm{d}\tau \overset{(3.2.1)}{=} \mathbf{x}(t_1) + \boldsymbol{\Xi}^\top \int_{t_1}^{t} \mathbf{f}(\mathbf{x}(\tau))\,\mathrm{d}\tau,$$

Inserting t_i with $i = 2, \ldots, N$ for t, yields $N-1$ systems of equations, which can altogether be expressed as

$$\begin{pmatrix} | & & | \\ \mathbf{x}(t_2) - \mathbf{x}(t_1) & \cdots & \mathbf{x}(t_N) - \mathbf{x}(t_1) \\ | & & | \end{pmatrix} = \boldsymbol{\Xi}^\top \underbrace{\begin{pmatrix} \int_{t_1}^{t_2} f_1(\mathbf{x}(\tau))\,\mathrm{d}\tau & \cdots & \int_{t_1}^{t_N} f_1(\mathbf{x}(\tau))\,\mathrm{d}\tau \\ \vdots & \ddots & \vdots \\ \int_{t_1}^{t_2} f_m(\mathbf{x}(\tau))\,\mathrm{d}\tau & \cdots & \int_{t_1}^{t_N} f_m(\mathbf{x}(\tau))\,\mathrm{d}\tau \end{pmatrix}}_{=:\,\Gamma(\mathbf{X})}. \tag{3.2.3}$$

Once again, unless measurements of the integrals are available, numerical methods such as quadrature have to be employed. But depending on the application, an approximation to $\Gamma(\mathbf{X})$ may be more accurately computable than an approximation to $\dot{\mathbf{X}}$. For more information on the integral approach, consult [48]. From now on, we will solely focus on the differential approach.

Example 3.2.1. Let $t_1, \dots, t_N$ be equidistant time samples with some time step $\Delta t > 0$. Suppose we are given some trajectory $((x_1, y_1), \dots, (x_N, y_N))$, where $x_i = x(t_i)$ and $y_i = y(t_i)$, that behaves according to the dynamics given by the ODE system

$$\begin{aligned} \dot{x} &= -0.1x + 2y \\ \dot{y} &= -2x - 0.1y, \end{aligned} \tag{3.2.4}$$

which describes a damped harmonic oscillator in two-dimensional space, and wish to recover equation (3.2.4) from this data. As a feature basis, we choose all polynomials p in x and y of degree $\deg(p) \leqslant 2$, thus

$$\begin{aligned} f_1(x,y) &= 1, & f_2(x,y) &= x, & f_3(x,y) &= y, \\ f_4(x,y) &= x^2, & f_5(x,y) &= xy, & f_6(x,y) &= y^2, \end{aligned}$$

and thus we obtain the multivariate Vandermonde matrix

$$\boldsymbol{\Theta}(\mathbf{X}) = \begin{pmatrix} 1 & x_1 & y_1 & x_1^2 & x_1y_1 & y_1^2 \\ 1 & x_2 & y_2 & x_2^2 & x_2y_2 & y_2^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & y_N & x_N^2 & x_Ny_N & y_N^2 \end{pmatrix}.$$

The derivative data $((\dot{x}_t, \dot{y}_t))_{t=1}^N$ is not provided, so we employ second-order central differences with forward/backward differences at the boundary points,

$$\dot{\mathbf{X}}_{\mathrm{FD}} = \frac{1}{2\Delta t} \begin{pmatrix} -2 & 2 & & & \\ -1 & 0 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 0 & 1 \\ & & & -2 & 2 \end{pmatrix} \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_N & y_N \end{pmatrix},$$

resulting in an approximation of the derivative matrix up to order $\mathcal{O}((\Delta t)^2)$ in the interior and $\mathcal{O}(\Delta t)$ at the boundary. Here, "FD" stands for "Finite Differences". We now have all ingredients to set up equation (3.2.2) for this example. ❖

The S in SINDy stands for "Sparse", but so far, the formulation (3.2.2) does not reflect that. Sparsity of a vector or matrix can be measured as the number of its nonzero entries, which is commonly denoted by $\|\cdot\|_0$. The notation is analogous to that of the ℓ_p norms for $p > 0$, which is motivated by the fact that $\|v\|_0 = \sum_{i=1}^n |v_i|^0$ for any $v \in \mathbb{R}^n$, at least when using the convention $0^0 = 0$. Despite this, it is not correct to term it the "ℓ_0 norm". It can be seen with a simple counterexample that it violates homogeneity: $\|2 \cdot (1,0)^\top\|_0 = 1 \neq 2 = 2\|(1,0)^\top\|_0$. We will henceforth refer to it as the ℓ_0 *quasinorm*.

In general, a solution to (3.2.2) may not exist. This could either be because the dynamics behind the data cannot be expressed in the chosen feature library, or because of measurement

noise, or – most likely in practical applications – both. To relax the problem, we instead require that the residual $r(\boldsymbol{\Xi}) \coloneqq \dot{\mathbf{X}} - \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi}$ is small enough in the sense that $\|r(\boldsymbol{\Xi})\| \leqslant \varepsilon$ for some small $\varepsilon > 0$ and some matrix norm $\|\cdot\|$. Following this approach and choosing the Frobenius norm, the new problem formulation [9] is given by

$$\underset{\substack{\|r(\boldsymbol{\Xi})\|_F \leqslant \varepsilon \\ \boldsymbol{\Xi} \in \mathbb{R}^{m \times d}}}{\arg\min} \|\boldsymbol{\Xi}\|_0. \tag{3.2.5}$$

Unfortunately, this problem is combinatorial in nature and, to the best of our knowledge, seems to require the consideration of all subsets of the function library, the number of which grows exponentially. A proof of NP-hardness has been published in [38].

Evidently, it is infeasible to solve the exact problem we wish to solve. However, it is still an option to tackle a furthermore relaxed version of the problem instead. The next section outlines three approaches. We end this section with two definitions. The first formalizes the notion of a trained SINDy model, while the second provides us with a compact notation to talk about model predictions.

Definition 3.2.2 (SINDy model)**.** A SINDy model $\mathcal{M}$ is a triple $(\mathcal{F}, \boldsymbol{\Xi}, \mathbf{t})$, where $\mathcal{F} = \{f_i \colon \mathbb{R}^d \to \mathbb{R}\}_{i=1}^m$ is a finite set of ansatz functions, $\boldsymbol{\Xi} \in \mathbb{R}^{m \times d}$ is a coefficient matrix, and $\mathbf{t} = (t_1, \ldots, t_N)$ is a time vector.

Each SINDy model induces a dynamical system $\dot{\mathbf{x}}(t) = F(\mathbf{x}(t))$ for $t \in [t_1, t_N]$, where the ith component of $F = (F_1, \ldots, F_d)^\top$ can be expressed as a linear combination of ansatz functions $F_i = \sum_{j=1}^m \boldsymbol{\Xi}_{j,i} f_j$.

Definition 3.2.3 (Prediction function)**.** For a SINDy model $\mathcal{M}$, we denote the solution of the associated dynamical system with respect to the initial condition $\mathbf{x}(t_1) = \mathbf{x}^{\text{ini}}$ by $\mathbf{x}_{\mathcal{M}}(t; \mathbf{x}^{\text{ini}})$. We then define the *prediction function* $\mathsf{pred}_{\mathcal{M}} \colon \mathbb{R}^d \to (\mathbb{R}^d)^N$ via

$$\mathbf{x} \mapsto \mathsf{pred}_{\mathcal{M}}(\mathbf{x}) = (\mathbf{x}_{\mathcal{M}}(t_i; \mathbf{x}))_{i=1}^N.$$

3.3 On the Choice of the Feature Library

As discussed in the previous section, the success of the SINDy approach is largely grounded in the choice of a feature library that is well-suited to the problem at hand. We briefly discuss some useful heuristics to go by when constructing a feature library.

A good starting point is to add any known terms to the feature library. Often, as in the case of the MRE, only a part of the equation is of interest to us. Hence, the parts that are already known or not of interest can be added to the feature library, making it easier for SINDy to find an accurate model.

Remark. In the following, we perform a slight abuse of notation. When constructing sets of functions $f \colon \mathbb{R}^d \to \mathbb{R}$, instead of writing the explicit mapping $(x_1, \ldots, x_d) \mapsto f(x_1, \ldots, x_d)$, we just write $f(x_1, \ldots, x_d)$ instead. For example, $\sin(kx_i)$ refers to the function defined by the mapping $(x_1, \ldots, x_d) \mapsto \sin(kx_i)$.

Let

$$\mathsf{Poly}_\delta(x_1, \ldots, x_d) \coloneqq \{x_1^{\alpha_1} \cdots x_d^{\alpha_d} \mid \alpha_i \in \mathbb{N}\ \forall i \in \{1, \ldots, d\}, \alpha_1 + \ldots \alpha_d \leq \delta\}$$

denote the library of polynomials in $x_1, \dots, x_d$ of degree up to $\delta \in \mathbb{N}$. Similarly, let

$$\mathsf{Trig}_\delta(x_1, \dots, x_d) \coloneqq \{\sin(kx_i), \cos(kx_i) \mid k = 1, \dots, \delta \text{ and } i = 1, \dots, d\}$$

denote the trigonometric polynomials up to frequency $\delta \in \mathbb{N}$. Generic functions such as these can be used to approximate a variety of functions, via a Taylor or Fourier series, and thus provide useful terms. However, the mentioned approximations are usually not sparse and are thus in conflict with SINDy's emphasis on sparsity. It further should be noted that, while the trigonometric library scales linearly in δ, the polynomial library scales exponentially in δ for fixed n. We therefore recommend incorporating only lower order polynomials. The 0th order polynomial $p \equiv 1$ is particularly important to capture any offset in the system.

3.4 Sparse Regression Techniques

In the following we familiarize ourselves with several techniques to solve (3.2.2) with appropriate sparsity constraints. As we learned, finding the solution with minimal ℓ_0 quasinorm is infeasible, so we have to accept a compromise. At the heart of all of the algorithms we consider is the solution of a regularized least squares problem

$$\min_{\boldsymbol{\Xi}} \frac{1}{2}\|\dot{\mathbf{X}} - \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi}\|_2^2 + \lambda R(\boldsymbol{\Xi}), \tag{3.4.1}$$

where $\lambda > 0$ is a *regularization parameter* and R a *regularizer*. Common choices for R are the ℓ_1 and ℓ_2 norms and the ℓ_0 quasinorm. The case of $R(\cdot) = \|\cdot\|_2$ is commonly called *Tikhonov regularization*, whereas the $R(\cdot) = \|\cdot\|_1$ case is known as the *Least Absolute Shrinkage and Selection Operator* (LASSO) [56]. The case $R(\cdot) = \|\cdot\|_0$ is most interesting to us, as a low ℓ_0 quasinorm directly signals sparsity.

3.4.1 Sequentially Thresholded Least Squares

As pointed out in the previous section, the underlying assumption for SINDy to work is the sparsity of the solution. Therefore, it is sensible to choose $R(\cdot) = \|\cdot\|_0$.

A straightforward method of achieving sparsity is by thresholding. After obtaining an initial least-squares solution, values that are below a certain threshold $0 < \tau \ll 1$, are set to zero. The motivation behind this is that functions with small coefficients have only little contribution to the overall model, and removing them entirely improves sparsity while only sacrificing little accuracy. This process is repeated until convergence.

The procedure is stated formally in Algorithm 1. The notation $\xi[|\xi| > \tau]$ refers to the vector obtained from ξ by removing those components ξ_i with $|\xi_i| \leqslant \tau$. Similarly, $\boldsymbol{\Theta}[:, \xi > \tau]$ refers to the matrix obtained from $\boldsymbol{\Theta}$ by removing the columns corresponding to the indices i for which $|\xi_i| \leqslant \tau$.

Once a component has been discarded, it can not be revived. For this reason, the while loop in Algorithm 1 runs for at most m iterations before the entire vector is thresholded to 0. If this does not happen, the iteration reaches a fixed-point. Therefore, the algorithm always converges. This argument is formalized in Theorem 2.1 of [59].

Despite its simplicity and its rather heuristic nature, it turns out that STLSQ indeed converges to a local minimum of (3.4.1) with $R(\cdot) = \|\cdot\|_0$ and $\lambda = \tau^2$. This was established in

Algorithm 1 Sequentially Thresholded Least Squares (STLSQ)

Require: $\boldsymbol{\Theta} \in \mathbb{R}^{N\times m}$; $\dot{x} \in \mathbb{R}^N$; $\tau, \alpha \geqslant 0$
  $\xi \leftarrow \arg\min_\xi \|\boldsymbol{\Theta}\xi - \dot{x}\|_2^2 + \alpha\|\xi\|_2^2$
  while $\xi[|\xi| > \tau] \neq \xi$ **do**
    $\xi \leftarrow \xi[|\xi| > \tau]$
    $\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta}[:, \xi > \tau]$
    $\xi \leftarrow \arg\min_\xi \|\boldsymbol{\Theta}\xi - \dot{x}\|_2^2 + \alpha\|\xi\|_2^2$
  end while
  return ξ

Theorems 2.4 and 2.5 of [59] for the non-regularized case, i.e., $\alpha = 0$. However, the $\alpha > 0$ case can easily be transformed into the first case by utilizing

$$\widetilde{\boldsymbol{\Theta}} \coloneqq \begin{pmatrix} \boldsymbol{\Theta} \\ \alpha I \end{pmatrix} \in \mathbb{R}^{(N+m)\times m}, \quad \tilde{\dot{\mathbf{x}}} \coloneqq \begin{pmatrix} \dot{\mathbf{x}} \\ 0 \end{pmatrix} \in \mathbb{R}^{N+m}. \tag{3.4.2}$$

A summary of the algorithm's properties is given here without proof.

Theorem 3.4.1 ([59], Corollary 2.11)**.** *Let $\boldsymbol{\Theta} \in \mathbb{R}^{N\times m}$ with $N \geqslant m$ and, without loss of generality, $\|\boldsymbol{\Theta}\|_2 = 1$. Further, let $\dot{\mathbf{x}} \in \mathbb{R}^N$ and $\tau > 0$. Let ξ^k be the sequence generated by Algorithm 1 with $\alpha = 0$ and define the objective function F by*

$$F(\xi) \coloneqq \|\boldsymbol{\Theta}\xi - \dot{\mathbf{x}}\|_2^2 + \tau^2\|\xi\|_0.$$

Then,

(i) ξ^k converges to a fixed point of the STLSQ scheme in at most N steps,
(ii) a fixed point of the scheme is a local minimizer of F,
(iii) a global minimizer of F is a fixed point of the scheme,
(iv) the sequence $(F(\xi^k))_{k=1}^N$ is strictly decreasing unless the iterates are stationary.

Point (iii) implies that the scheme is able to converge to the global minimizer.

STLSQ performs well in practice. Its main drawback is the lack of a true optimization cost function that is reflected in the structure of the algorithm. This implies that it needs to be reformulated in order to introduce adaptations, such as the option to incorporate or trim corrupt data [10]. For this reason, the following algorithm has been developed.

3.4.2 Sparse Relaxed Regularized Regression

Instead of solving (3.4.1) directly, a relaxed variant is considered. This relaxation is expressed as

$$\min_{\boldsymbol{\Xi},\mathbf{W}} \frac{1}{2}\|\dot{\mathbf{X}} - \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi}\|_2^2 + \frac{1}{2\nu}\|\boldsymbol{\Xi} - \mathbf{W}\|_2^2 + \lambda R(\mathbf{W}), \tag{3.4.3}$$

where $\nu > 0$ controls the proximity of the relaxation $\mathbf{W}$ to $\boldsymbol{\Xi}$ and $\lambda > 0$ the amount of regularization. This optimization problem searches for an accurate coefficient matrix $\boldsymbol{\Xi}$ that is simultaneously close to a regularized – in our case this usually means *sparse* – coefficient matrix $\mathbf{W}$.

To translate the problem (3.4.3) into algorithmic form, one employs an iterative procedure similar to STLSQ, see Algorithm 1. However, to enforce the regularization of $\mathbf{W}$ while simultaneously keeping it close to the least-squares solution $\boldsymbol{\Xi}$, a proximal operator is used.

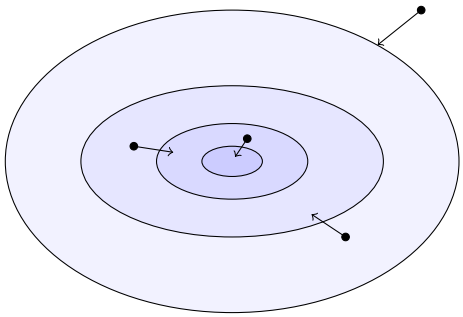


Figure 3.4: Rough behavior of the proximal operator of a function. Points are mapped to nearby points that are closer to the function's minimum. Points outside of the domain are mapped to the closest point on the boundary.

Definition 3.4.2 (Proximal operator [39])**.** Let X be a Hilbert space and $f\colon X \to \overline{\mathbb{R}}$ a lower semi-continuous convex function. Then, the proximity operator $\mathsf{prox}_f\colon X \to X$ is defined via

$$\mathsf{prox}_f(v) = \operatorname*{arg\,min}_{x \in X} f(x) + \frac{1}{2}\|x - v\|_X^2.$$

Remark. While the definition above calls for semi-continuity and convexity of f, the proximal operator is well-defined, as long as a unique minimizer exists. This permits us to use prox operators of certain non-convex functions, such as the ℓ_0 quasinorm, as well.

In colloquial terms, prox_f maps v to a point close to itself that also minimizes f as much as possible. See Figure 3.4 for an illustration. Frequently, a regularized proximal operator $\mathsf{prox}_{\lambda f}$ is considered, where $\lambda > 0$ controls how much x is allowed to deviate from v in order to favor the goal of minimization. For further information on proximal operators, consult [39].

We consider some proximal operators for different functions f. Let $X = \mathbb{R}^n$. The ith component of the proximal operator for the ℓ_1 norm is given by

$$\mathsf{prox}_{\lambda\|\cdot\|_1}(\mathbf{x})_i = \operatorname{sgn}(x_i)\max(|x_i| - \lambda, 0) \tag{3.4.4}$$

and called the *soft-thresholding* operator [60]. For the ℓ_0 quasinorm, the proximal operator turns out to be equivalent to *hard thresholding*,

$$\mathsf{prox}_{\lambda\|\cdot\|_0}(\mathbf{x})_i = \begin{cases} 0, & |x_i| \leqslant \sqrt{2\lambda}, \\ x_i, & |x_i| > \sqrt{2\lambda}, \end{cases} \tag{3.4.5}$$

as performed in STLSQ. Both operators can yield component-wise sparsity. This is not the case for the proximal operator of the ℓ_2 norm, which is given by

$$\mathsf{prox}_{\lambda\|\cdot\|_2}(\mathbf{x}) = \begin{cases} \frac{\|\mathbf{x}\|_2 - \lambda}{\|\mathbf{x}\|_2}\mathbf{x}, & \|\mathbf{x}\|_2 > \lambda \\ \mathbf{0}, & \|\mathbf{x}\|_2 \leqslant \lambda \end{cases}. \tag{3.4.6}$$

Hence, the former two penalties are preferred for sparsity promotion.

The resulting algorithm, called Sparse Relaxed Regularized Regression (SR3) [10], is shown in Algorithm 2. It can be considered a variation of the standard proximal gradient algorithm [39]. To analyze its convergence, we thus turn to the theory of proximal gradient methods. As R may not be differentiable, we need a generalized notion of a differential, called subdifferential.

Algorithm 2 Sparse Relaxed Regularized Regression

Require: $\varepsilon, \mathbf{W}^0$
  $k \leftarrow 0$
  $\mathrm{err} \leftarrow 2\varepsilon$
  while $\mathrm{err} > \varepsilon$ **do**
    $k \leftarrow k + 1$
    $\boldsymbol{\Xi}^k \leftarrow \arg\min_{\boldsymbol{\Xi}} \frac{1}{2}\|\dot{\mathbf{X}} - \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi}\|_2^2 + \frac{1}{2\nu}\|\boldsymbol{\Xi} - \mathbf{W}^{k-1}\|_2^2$
    $\mathbf{W}^k \leftarrow \mathsf{prox}_{\lambda\nu R}(\boldsymbol{\Xi}^k)$
    $\mathrm{err} \leftarrow \|\mathbf{W}^k - \mathbf{W}^{k-1}\|_2/\nu$
  end while

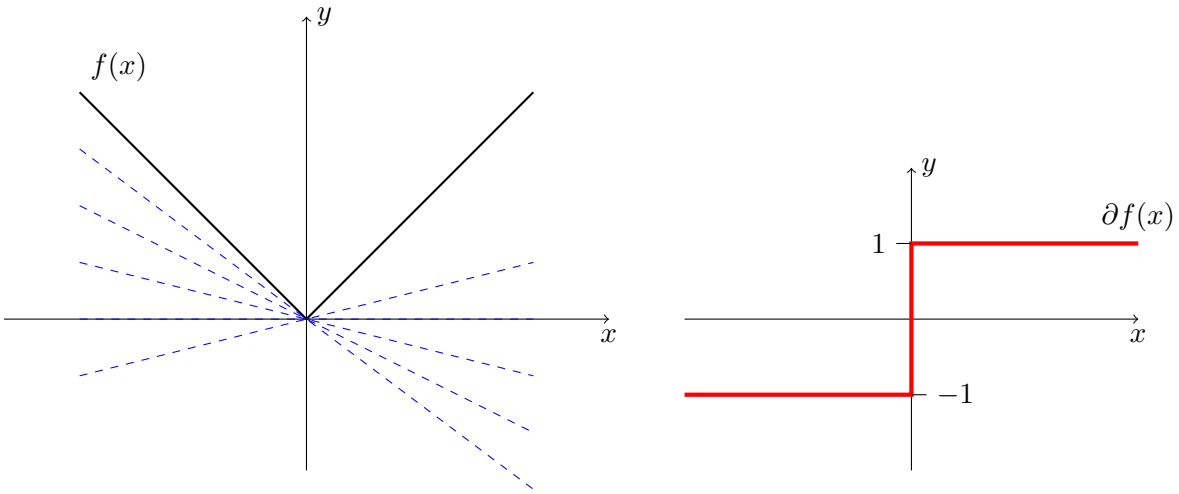


Figure 3.5: Left: Illustration of the subdifferential of $x \mapsto f(x) = |x|$ at 0. The dotted blue lines are examples of subspaces through 0 whose slopes correspond to elements in $\partial f(0) = [-1, 1]$. **Right:** A plot of the graph $\{(x, y) \mid y \in \partial f(x)\}$ of the subdifferential $\partial f(x)$ over an interval around 0.

Definition 3.4.3 (Subdifferential)**.** Let $f\colon \mathbb{R}^n \to \overline{\mathbb{R}}$ be convex. The *subdifferential* of f at $\overline{\mathbf{x}} \in \mathsf{dom} f$ is defined as

$$\partial f(\overline{\mathbf{x}}) := \{\mathbf{v} \in \mathbb{R}^n \mid f(\mathbf{x}) \geqslant f(\overline{\mathbf{x}}) + \langle \mathbf{v}, \mathbf{x} - \overline{\mathbf{x}}\rangle \quad \forall \mathbf{x} \in \mathbb{R}^n\}.$$

Intuitively, the subdifferential of f at $\overline{\mathbf{x}}$ contains all slopes $\mathbf{v}$ such that the hypersurface with slope $\mathbf{v}$ passing through $\overline{\mathbf{x}}$ is below the graph of f, see Figure 3.5. For a global minimum $\mathbf{x}^*$ of f, one has $f(\mathbf{x}) \geqslant f(\mathbf{x}^*)$ for all $\mathbf{x} \in \mathbb{R}^n$, thus yielding the necessary minimality condition $\mathbf{0} \in \partial f(\mathbf{x}^*)$.

Theorem 3.4.4 (Convergence of SR3 ([10], Theorem 1))**.** *Let*

$$p(\mathbf{W}) := \min_{\boldsymbol{\Xi}} \frac{1}{2}\|\dot{\mathbf{X}} - \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi}\|_2^2 + \frac{1}{2\nu}\|\boldsymbol{\Xi} - \mathbf{W}\|_2^2 + \lambda R(\mathbf{W})$$

and suppose that p is bounded from below, i.e., $-\infty < p^* := \inf_{\mathbf{W}} p(\mathbf{W})$. Then, the iterators from Algorithm 2 satisfy

$$\frac{1}{N}\sum_{k=1}^{N} \|g^k\|_2^2 \leqslant \frac{1}{\nu N}(p(\mathbf{W}^0) - p^*),$$

where $g^k \in \partial p(\mathbf{W}^k)$.

With $R(\cdot) = \|\cdot\|_0$, SR3 performs similarly to STLSQ, both in terms of accuracy and sparsity [10], This is to be expected, as the proximal operator for the ℓ_0 quasinorm is hard thresholding. Yet, SR3 has the advantage of being easily extendable, due to the proximity of the algorithmic formulation to the objective function (3.4.3). We briefly discuss some potential extensions. For more details on each of these, consult [10].

Constraints. Physical systems often adhere to certain linear constraints such as symmetries or conservation laws. It is sensible to then restrict the solution space to only those solutions adhering to said constraints. Let $\boldsymbol{\xi} \coloneqq \mathsf{vec}(\boldsymbol{\Xi})$, where $\mathsf{vec}_{m,d}\colon \mathbb{R}^{m\times d} \to \mathbb{R}^{m\cdot d}$ is the *vectorization operator* that flattens a matrix in a column-wise fashion, i.e. the resulting vector consists of the matrix's columns stacked on top of each other. The subindices m, d are omitted whenever they can be inferred from the context, so that we write vec instead of $\mathsf{vec}_{m,d}$. A set of n_c linear constraints can then be expressed as a linear equation $\mathbf{C}\boldsymbol{\xi} = \mathbf{d}$, where $\mathbf{C} \in \mathbb{R}^{n_c\times(d\cdot m)}, \mathbf{d} \in \mathbb{R}^{n_c}$ make up the left- and right-hand sides of the constraints respectively. The optimization problem considered is thus

$$\begin{aligned} &\min_{\boldsymbol{\Xi},\mathbf{W}} \frac{1}{2}\|\dot{\mathbf{X}} - \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi}\|_2^2 + \frac{1}{2\nu}\|\boldsymbol{\Xi} - \mathbf{W}\|_2^2 + \lambda R(\mathbf{W}), \\ &\text{s.t. } \mathbf{C}\boldsymbol{\xi} = \mathbf{d}. \end{aligned}$$

This can be solved by modifying Algorithm 2. Before mentioning the modification, we need to define a special type of matrix product.

Definition 3.4.5 (Kronecker product)**.** Given two matrices $\mathbf{A} \in \mathbb{R}^{i\times j}$, $\mathbf{B} \in \mathbb{R}^{k\times \ell}$, their *Kronecker product* $\mathbf{A} \otimes \mathbf{B} \in \mathbb{R}^{ik\times j\ell}$ is defined via

$$\begin{pmatrix} a_{11}\mathbf{B} & \dots & a_{1j}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{i1}\mathbf{B} & \dots & a_{ij}\mathbf{B} \end{pmatrix}. \tag{3.4.7}$$

The modification relies on the so-called vec-kron identity

$$\mathsf{vec}(\boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi}) = (\mathbf{I} \otimes \boldsymbol{\Theta}(\mathbf{X}))\boldsymbol{\xi},$$

which relates vectorization and the Kronecker product.

Constraints can also be helpful when trying to incorporate functions with multiple outputs into the feature library, by requiring that the coefficients in front of the component functions be the same.

Parameterized library functions. So far, the function library we considered consisted of functions of the input features. In some cases, it makes sense for a candidate function to depend on other further parameters. For example, if it is known or suspected that the dynamics contain an exponential term of the form $\exp(\alpha x)$ for some unknown $\alpha \in \mathbb{R}$, the classical approach would require including this function for several values of α in the function library. This is cumbersome and likely infeasible without repeated manual adaptation of the function library. Instead, one can consider the nonlinear least squares problem

$$\min_{\boldsymbol{\Xi},\mathbf{W},\boldsymbol{\alpha}} \frac{1}{2}\|\dot{\mathbf{X}} - \boldsymbol{\Theta}(\mathbf{X},\boldsymbol{\alpha})\boldsymbol{\Xi}\|_2^2 + \frac{1}{2\nu}\|\boldsymbol{\Xi} - \mathbf{W}\|_2^2 + \lambda R(\mathbf{W}), \tag{3.4.8}$$

in which $\boldsymbol{\alpha}$ is included as an optimization variable, which the feature library depends on. To solve (3.4.8), Algorithm 2 can be adapted, such that each prox-step is followed by a Newton step for $\boldsymbol{\alpha}$. The gradients and Hessians required for this can be computed using automatic differentiation. An initial parameter guess $\boldsymbol{\alpha}^0$ needs to be provided as input.

Data Trimming. Outliers in the data can decrease the quality of results obtained by SINDy. By additionally optimizing over the choice of time samples, one can potentially obtain better results, as the outliers will simply be excluded. To make this possible, one needs an estimate of the number of correct samples, h. If derivatives are numerically approximated, that number is likely higher, because neighboring samples are affected as well. If $\mathbf{u} \in \{0,1\}^N$ is a vector such that $u_i = 1$ if, and only if, the ith sample should be included, then $\sum_{i=1}^N u_i = h$ is a reasonable constraint. In practice, optimization over real numbers is easier than over integers, hence we optimize over $\mathbf{v} \in [0,1]^N$ instead, while keeping the constraint. The modified objective

$$\min_{\boldsymbol{\Xi},\mathbf{W},\mathbf{v}} \sum_{i=1}^N \frac{v_i}{2} \|\dot{\mathbf{X}} - \boldsymbol{\Theta}(\mathbf{X},\boldsymbol{\alpha})\boldsymbol{\Xi}\|_2^2 + \frac{1}{2\nu}\|\boldsymbol{\Xi} - \mathbf{W}\|_2^2 + \lambda R(\mathbf{W}),$$
$$\text{s.t. } 0 \leqslant v_i \leqslant 1,\ \sum_{i=1}^N v_i = h,$$

can be solved by performing gradient descent over $\mathbf{v}$ after each prox-step, and projecting onto the feasible region $\{0 \leqslant v_i \leqslant 1 : \sum_{i=1}^N v_i = h\}$.

3.5 Further Extensions

Since its original publication in 2016, numerous extensions to the SINDy framework have been proposed. We briefly discuss a few of them.

SINDy for Partial Differential Equations. The extension to partial differential equations of the form

$$u_t = F(u, u_{x_1}, u_{x_2}, \dots, u_{x_n}, u_{x_1x_2}, \dots)$$

is straightforward and does not require an adaptation to the training algorithm. The left-hand side u_t is a column vector of the time-derivative of u at the different time steps, while the function library $\boldsymbol{\Theta}$ contains the different partial derivatives of u. The paper [46] goes into more detail and showcases a number of examples.

Parallel Implicit SINDy. To discover implicit differential equations of the form

$$f(\mathbf{x}, \dot{\mathbf{x}}) = 0,$$

it is possible to incorporate the time-derivatives as features, resulting in the problem of finding sparse solutions to $\boldsymbol{\Theta}(\mathbf{X}, \dot{\mathbf{X}})\boldsymbol{\Xi} = \mathbf{0}$ [35]. This approach, however, suffers from ill-conditionality due to nullspace computations.

To circumvent this, the authors of [27] observe that, if there exists a sparse model of the dynamics and at least a single term $f_j(\mathbf{x}, \dot{\mathbf{x}})$ of the dynamics is known, then the problem

$$f_j(\mathbf{X}, \dot{\mathbf{X}}) = \boldsymbol{\Theta}(\mathbf{X}, \dot{\mathbf{X}} \mid f_j)\boldsymbol{\Xi} \tag{3.5.1}$$

has a sparse solution. Here, the notation $\boldsymbol{\Theta}(\mathbf{X}, \dot{\mathbf{X}} \mid f_j)$ indicates the removal of the column corresponding to f_j. A training algorithm can thus be designed that test each individual library function and checks whether a sparse solution of (3.5.1) is found. This approach avoids nullspace computations and can be performed in parallel.

Weak SINDy. Weak SINDy (WSINDy) [37] relies on the weak formulation of the system dynamics, which, considering some smooth test function $\phi\colon \mathbb{R} \to \mathbb{R}$ and a time interval $(a, b) \subset [t_1, t_N]$, reads

$$\phi(b)\mathbf{x}(b) - \phi(a)\mathbf{x}(a) - \int_a^b \phi'(s)\mathbf{x}(s)\,\mathrm{d}s = \int_a^b \phi(s)\mathbf{F}(\mathbf{x}(s))\,\mathrm{d}s. \tag{3.5.2}$$

Choosing $\phi \equiv 1$ leads to the integral approach discussed in 3.2. If ϕ is compactly supported, the evaluations at the boundary points a and b vanish.

The next step is to construct a family of test functions $(\phi_k)_{k=1}^K$ of size equal to the number of subintervals of the form (t_i, t_j) with $i < j$, thus $K \coloneqq \frac{N(N-1)}{2}$. Equation (3.5.2) can then be expressed as the linear problem $\mathbf{G}\boldsymbol{\Xi} = \mathbf{b}$, where $\mathbf{G}_{i,j} \approx \int \phi_i(s) f_j(\mathbf{x}(s))\,\mathrm{d}s$ and $\mathbf{b}_{i,j} \approx \int \phi_i'(s)\mathbf{x}_j(s)\,\mathrm{d}s$. These quantities can be approximated using numerical quadrature. Finally, the generalized ℓ_2-regularized least-squares problem

$$\boldsymbol{\Xi}^* = \arg\min_{\boldsymbol{\Xi}} \left\{ (\mathbf{G}\boldsymbol{\Xi} - \mathbf{b})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{G}\boldsymbol{\Xi} - \mathbf{b}) + \alpha^2 \|\boldsymbol{\Xi}\|_2^2 \right\},$$

where $\boldsymbol{\Sigma}$ is a suitably chosen error covariance matrix, is solved using STLSQ.

Compared to traditional SINDy, WSINDy avoids point evaluations of the (usually approximated and thus error-ridden) derivatives and is therefore expected to perform more robustly in the presence of noise.

Autoencoder SINDy. The main appeal of SINDy is its capability to discover *parsimonious* models. However, not all systems allow for a simple expression of their dynamics – at least not in the coordinate system of the input $\mathbf{x}$. The work by Champion et al. [11] tackles this problem by concurrently training a SINDy model and an autoencoder, such that the dynamics of the former are parsimonious in the latent space of the latter.

Let $\varphi\colon \mathbb{R}^d \to \mathbb{R}^{d'}$ with $d' \ll d$ be the encoder network, $\psi\colon \mathbb{R}^{d'} \to \mathbb{R}^d$ the decoder network, and $\mathbf{z} = \varphi(\mathbf{x})$. The weights of the SINDy model and the autoencoder are learnt during a joint training procedure, for which the objective function is

$$\|\mathbf{x} - \psi(\mathbf{z})\|_2^2 + \lambda_1 \mathcal{L}_{\mathrm{d}\mathbf{x}/\,\mathrm{d}t} + \lambda_2 \mathcal{L}_{\mathrm{d}\mathbf{z}/\,\mathrm{d}t} + \lambda_3 \|\boldsymbol{\Xi}\|_1. \tag{3.5.3}$$

The loss terms $\mathcal{L}_{\mathrm{d}\mathbf{x}/\,\mathrm{d}t}$ and $\mathcal{L}_{\mathrm{d}\mathbf{z}/\,\mathrm{d}t}$ in the center, which we have purposely left undefined here, penalize the prediction error of the dynamics in the latent space (i.e. of $\dot{\mathbf{x}}$), and the reconstruction error of $\dot{\mathbf{x}}$, respectively. The regularization parameters $\lambda_i \geqslant 0$ for $i = 1, 2, 3$ control the relative importance of each of the three regularization objectives. Along with the ℓ_1 norm in the loss function, sequential thresholding is used during training as well.

This approach can be seen as a way to perform SINDy with a trainable feature library: While, technically speaking, the library itself is fixed before the training procedure, the variables fed into the library functions lie in the latent space, which is trainable.

Chapter 4

Application and Results

The following section details the experiments that were performed. All code was written in Python 3.10.8 and made use of the `PySINDy` package [50, 28] version 1.7.5. The experiments were performed on an Apple M1 Pro chip, and the mentioned computation times refer exclusively to this chip. The default ODE solver we used was `LSODA` [40], as implemented in the `solve_ivp` method from the `SciPy` package [57] version 1.9.3.

4.1 Data Generation and Evaluation Metrics

To apply SINDy to the Maxey-Riley equations, we first generated numerical solutions to be used as training data. The solution method is based on a transformation of the Maxey-Riley equations into a heat equation on a semi-infinite domain [41]. We will not go into further detail on the way the solutions were obtained.

Because both the spatial and temporal domains are infinite, we have to make choices as to which finite regions to consider and which to neglect, which may very well have an impact on the resulting model. In the spatial case, there are two obvious choices:

1. An equidistant local grid. If there is a particular area or point of interest, choose a grid around that point. If the equation is simple enough that it does not produce local behavior, the precise region should not matter, so you might as well choose a grid around the origin for simplicity.

2. A grid with exponentially growing gaps. This approach allows to somewhat represent infinite domains in a finite setting. The obvious advantage here is that different local behaviors may be captured in the data, possibly yielding a more accurate model. A drawback is the fact that most points are still going to be centered around the origin, with only a small number being placed at points further outside. Depending how SINDy is trained with multiple trajectories, there may simply not be enough emphasis on those outside points.

For the temporal grid, the most obvious choice is to pick start and final times t_1, t_N and create an equidistant grid between them. A non-equidistant grid is also possible, but the quality of the computed time-derivatives will suffer, if the gaps are taken too large. For this reason, the exponential-gap approach is not a good idea.

Naturally, the data depends on the chosen velocity field $\mathbf{u}$. Because SINDy should be able to detect the coefficients of the components pertaining to the velocity field, training data for a single model never contained data generated from different velocity fields. Therefore, every SINDy model is velocity field-dependent.

A similar approach could be taken for the equation parameters ρ_f, ρ_p, a and ν. If SINDy is not instructed to consider them as equation variables, they should be kept constant across a training data set. This results in multiple SINDy models that are able to handle different cases of the equations with good precision. An alternative approach would be to introduce each equation parameter, say a, as an additional variable with defining equation $\dot{a} = 0$. In that case, different values for the parameters can be mixed in a single training set, because SINDy is allowed to make the resulting equations dependent on those parameters. However, in practice we found this approach to yield worse results, see Section 4.3.2.

For all experiments, the data was split into a training set and an evaluation set, respectively, with the former containing 80% of the overall data. Good performance on the evaluation set – which the model has not seen before – indicates good generalization, which we desire. Additionally, to test the model's ability to extrapolate, initial data outside of the sampling interval used to obtain the training data was added to the evaluation set.

Each model $\mathcal{M}$ is evaluated on error and complexity. Error is computed as the average relative ℓ_2 error across all trajectories in the evaluation set. More precisely, if $(\mathbf{x}_t^i)_{t=1}^N$ is the true ith trajectory, the model error is given by

$$\mathsf{err}(\mathcal{M}) \coloneqq \frac{1}{N}\sum_{i=1}^{N} \frac{\|\mathbf{x}^i - \mathsf{pred}_{\mathcal{M}}(\mathbf{x}_1^i)\|_2}{\|\mathbf{x}^i\|_2}.$$

Relative errors are easier to interpret than absolute errors. For instance, a relative error of 1 is quite bad, because the error is as large as the trajectory itself.

Meanwhile, model complexity is measured as the number of non-zero entries in the coefficient matrix, given by its ℓ_0 quasinorm $\|\boldsymbol{\Xi}\|_0$. While simple, this is enough to find a model with a desired trade-off between accuracy and sparsity. The trade-off can be achieved by minimizing the objective, or loss term,

$$\mathcal{L}_{\text{traj}}(\mathcal{M}; \alpha) \coloneqq \mathsf{err}(\mathcal{M}) + \alpha\|\boldsymbol{\Xi}\|_0 \tag{4.1.1}$$

for some regularization parameter $\alpha > 0$. This is a kind of *hyperparameter search*, because the optimization variables are not the model's coefficients, but the parameters controlling the learning algorithm, such as λ for regularization or τ for thresholding. A classic approach is to use a grid search or random search instead of a gradient-based optimization method, as the latter would require differentiating the training algorithm, which is challenging in practice. It has been demonstrated both empirically and theoretically that random search is the preferred method over grid search, at least for neural networks [4].

As an alternative to (4.1.1), it is sometimes more convenient to minimize

$$\mathcal{L}_{\text{deriv}}(\mathcal{M}; \alpha) \coloneqq \|\dot{\mathbf{X}} - \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi}\|_2^2 + \alpha\|\boldsymbol{\Xi}\|_0, \tag{4.1.2}$$

instead, which appears more natural from the SINDy perspective anyway. While it is further away from the true goal – a good approximation of the *trajectory* –, it is closer to the problem solved by the SINDy algorithm. In particular, no model predictions, which involve calls to an ODE solver, have to be made. Such calls take time and have several times led to crashes during our experiments.

Algorithm 3 Hyperparameter Search

Require: $\dot{\mathbf{x}} \in \mathbb{R}^{N \times d}$, $\{\boldsymbol{\Theta}_i\}_{i=1}^{k}$
  $\alpha \leftarrow \mathsf{cond}(\boldsymbol{\Theta}_0)$
  $\boldsymbol{\Xi}^* \leftarrow \mathsf{STLSQ}(\boldsymbol{\Theta}_0, \dot{\mathbf{x}}, 0, 0)$
  $\mathcal{L}^* \leftarrow \mathcal{L}_{\text{deriv}}(\boldsymbol{\Xi}^*; \alpha)$
  for $i \in [k]$ **do**
    $j \leftarrow 0$
    while $(\tau \leftarrow \mathsf{next\_tau}(j)) < \infty$ **do**
      $\boldsymbol{\Xi} \leftarrow \mathsf{STLSQ}(\boldsymbol{\Theta}_i, \dot{\mathbf{x}}, \tau, 0)$
      $\mathcal{L} \leftarrow \mathcal{L}_{\text{deriv}}(\boldsymbol{\Xi}; \alpha)$
      if $\mathcal{L} \leqslant \mathcal{L}^*$ **then**
        $\mathcal{L}^* \leftarrow \mathcal{L}$
        $\boldsymbol{\Xi}^* \leftarrow \boldsymbol{\Xi}$
      end if
      $j \leftarrow j + 1$
    end while
  end for
  return $\boldsymbol{\Xi}^*$

Algorithm 3 shows an outline of such a hyperparameter search. A collection of given feature libraries is considered in order, and for each library, several threshold values τ, dictated by next_tau are tried. The overall best model is returned. The function next_tau may just return equidistantly spaced values in a certain range, but can also be more refined. For example, in the supplementary material of [46], the following is suggested: If the best model was updated in the last step, increment the threshold by some value $\Delta\tau$, which was initially provided by the user. If the best model was not updated (i.e. the loss worsened), set $\tau = \max(0, \tau - 2\Delta\tau)$, update $\Delta\tau \leftarrow \frac{2\Delta\tau}{max\_iter - j}$ and finally increment τ by the new $\Delta\tau$. Here, max_iter is another value provided by the user, and next_tau returns ∞ once $j > max\_iter$. This algorithm is similar to gradient descent in principle. Although it does not rely on derivatives, it finds a local minimum of the loss function.

4.2 Velocity Fields

In this section, we give a brief overview over the different velocity fields that the SINDy experiments were run on.

4.2.1 Analytical Vortex

The analytical vortex was first studied in [8], in which an analytical solution was also derived, hence the name. The velocity field $\mathbf{u}_{\text{ana}}$ is given by the mapping

$$\begin{bmatrix} x \\ y \end{bmatrix} \mapsto \begin{bmatrix} -y \\ x \end{bmatrix}.$$

Its material derivative $\frac{D\mathbf{u}_{\text{ana}}}{Dt}$ maps a point $[x, y]^\top$ to

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} - y \cdot \partial_x \mathbf{u}_{\text{ana}} + x \cdot \partial_y \mathbf{u}_{\text{ana}} = -y \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} + x \cdot \begin{bmatrix} -1 \\ 0 \end{bmatrix} = \begin{bmatrix} -x \\ -y \end{bmatrix}.$$

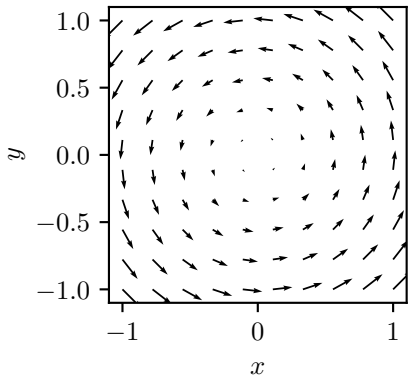


Figure 4.1: A quiver plot of the vortex velocity field in the region $[-1, 1]^2$.

Param.	Value
U	5.414
L	1.770
$\mathbf{a}$	$(0.0075, 0.15, 0.3)^\top$
$\mathbf{k}$	$(6.371)^{-1}(2, 4, 6)^\top$
$\mathbf{c}$	$U(0.1446, 0.205, 0.461)^\top$
$\mathbf{s}$	(k_1c_1, k_2c_2, k_3c_3)

Table 4.1: Parameters for the Bickley Jet.

A quiver plot of the velocity field is shown in Figure 4.1. It is steady and rotationally invariant. Evidently, this is a very simple velocity field with a predictable structure. Thus, it serves as a good starting point for the application of SINDy, before moving on to more complex velocity fields, such as the Bickley Jet, which is unsteady and has no rotational symmetry.

4.2.2 Bickley Jet

The Bickley Jet $\mathbf{u}_{\text{bick}}$ is a time-dependent velocity field. It is given by

$$\begin{bmatrix} x \\ y \\ t \end{bmatrix} \mapsto \begin{bmatrix} U \cdot (1 - \tanh(y/L)^2) + 2U\operatorname{sech}(y/L)^2 \tanh(y/L) \cdot \sum_{i=1}^{3} a_i \cos(k_i x - s_i t) \\ -UL\operatorname{sech}(y/L)^2 \cdot \sum_{i=1}^{3} a_i k_i \sin(k_i x - s_i t) \end{bmatrix},$$

with parameters $U, L \in \mathbb{R}$, $\mathbf{a}, \mathbf{k}, \mathbf{s} \in \mathbb{R}^3$. The particular parameter values we exclusively used are found in Table 4.1.

A quiver plot of the velocity field at four different times is shown in Figure 4.2. The general structure of the flow is that of a sine wave. At each "bump", there is a small vortex.

4.3 Experiments with the Vortex

Remark. To avoid convoluted sentences, each initial condition will from now on be associated with the trajectory it generates. Thus, it is natural to speak about, say, "the trajectory at $(0, 0)$". It should be clear from the context whether this refers to the trajectory in the data set, or the one predicted by the model.

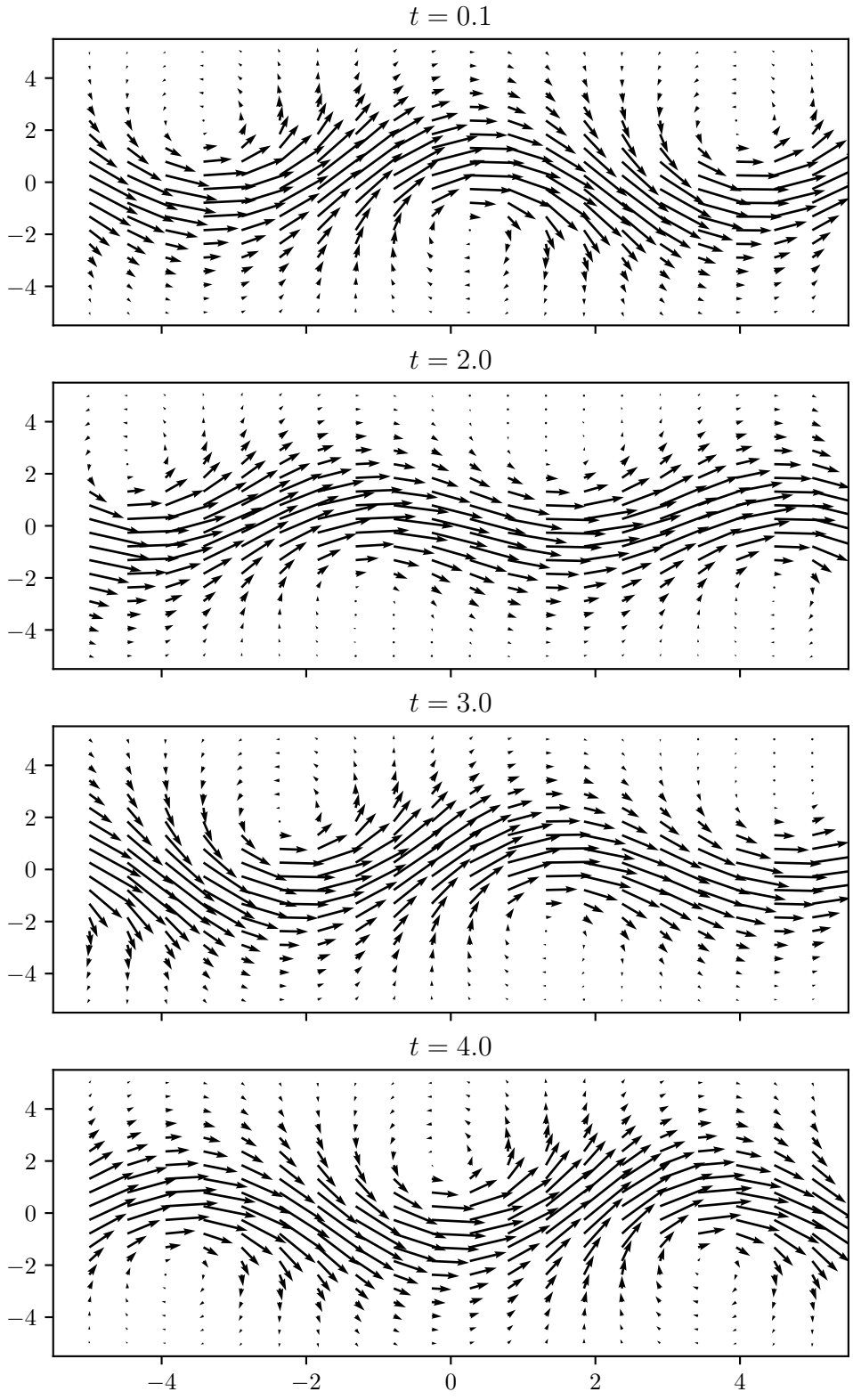


Figure 4.2: Quiver plots of the velocity field belonging to the Bickley Jet over the region $[-5, 5]^2$ at four different time points $t \in \{0.0, 1.0, 2.0, 3.0\}$.

Description	Symbol	Value
Number of particles	n_p	400
Particle density	ρ_p	2.0
Fluid density	ρ_f	1.0
Density constant	R	5/3
Particle radius	a	3.0
Kinematic viscosity	ν	1.0
Time scale	T	10.0
Initial time	t_{ini}	0.0
Final time	t_{fin}	50.0
Time step	Δt	0.025

Table 4.2: An overview of the vortex dataset used for the experiments in this subsection.

4.3.1 Dynamics with fixed equation parameters

In this experiment, the *analytical vortex* with fixed parameters $\rho_p = 2.0$, $\rho_f = 1.0$, $a = 3.0$ and $\nu = 1.0$ was considered. The choice of parameters was mostly arbitrary, but we purposely avoided the trivial dynamics resulting from $\rho_p = \rho_f$. A total of 400 trajectories with equidistantly spaced initial conditions in the region $[-2, 2] \times [-2, 2]$ were generated numerically. Of those in the subregion $[-1, 1] \times [-1, 1]$, 80% were randomly selected for the training set, while the rest of the data was included in the evaluation set. Table 4.2 summarizes all relevant information about the data set.

Because both $\mathbf{u}_{\text{ana}}$ and its material derivative are linear, it appeared sufficient to use a polynomial library of degree at most 1. The set of variables used was $\{x, y, \dot{x}, \dot{y}, t\}$. As SINDy only finds first-order systems, the inclusion of $\dot{x}$ and $\dot{y}$ was necessary in order to involve the second derivatives. The inclusion of the time variable t is not strictly necessary in this case, as the velocity field is steady, but time-dependent terms may still show up, for example due to the history term. The trivial equations

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \dot{x}, \quad \frac{\mathrm{d}y}{\mathrm{d}t} = \dot{y}, \quad \frac{\mathrm{d}t}{\mathrm{d}t} = 1$$

are usually discovered correctly from the data, but can alternatively be enforced using constraints, see Paragraph 3.4.2. We remark that the coefficients for the trivial equations are not taken into consideration when computing model complexity.

Using the STLSQ algorithm with a threshold of $\tau = 0.5$, SINDy found the model $\mathcal{M}$ with nontrivial equations

$$\begin{aligned} \ddot{x} &= -0.795x - 1.442y - 1.408\dot{x} \\ \ddot{y} &= 1.432x - 0.795y - 1.397\dot{y}. \end{aligned} \tag{4.3.1}$$

Error and complexity are given by $\mathsf{err}(\mathcal{M}) \approx 0.006$ and $\|\boldsymbol{\Xi}\|_0 = 6$, respectively. The computation took around 200 milliseconds on an Apple M1 Pro chip. An example trajectory from the dataset is shown in Figure 4.3. Meanwhile, the the corresponding prediction of the model is drawn on top. The prediction is remarkably close to the ground truth. There is a small deviation, which becomes more noticeable near the end, resulting in a slightly less expansive vortex.

Extrapolation. Figure 4.4a showcases the relative ℓ_2 error for the different trajectories in both the training and evaluation set. As one can see, the error hardly varies, even when

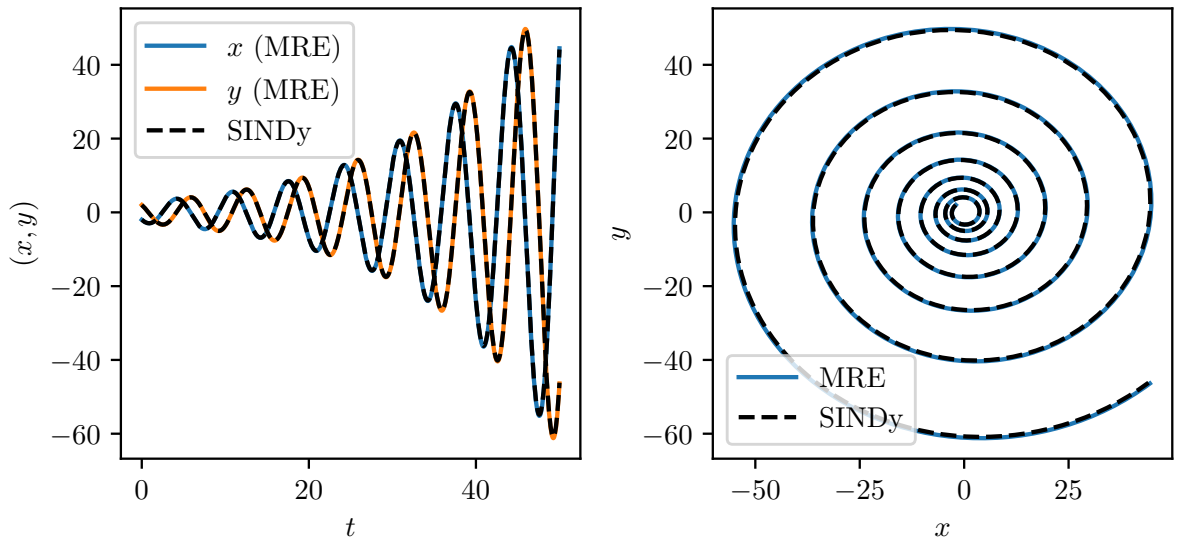
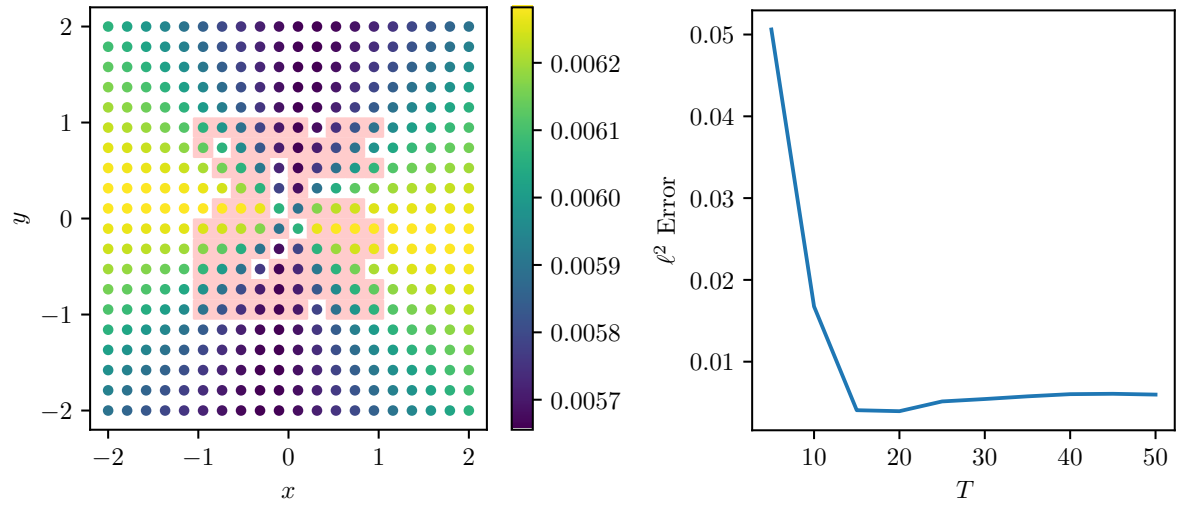


Figure 4.3: The trajectory with initial condition $(-2, 2)$ plotted in the form of both components with respect to time (left) and in the phase space (right). In both plots, the model prediction is overlaid as a black dashed line.



(a) Spatial inter- and extrapolation capabilities of the model. Each dot corresponds to an initial condition with the color indicating the ℓ_2 of the corresponding trajectory. Highlighted in red are the initial conditions corresponding to the trajectories contained in the training set.

(b) Behavior of the ℓ_2 error as the trajectory cut-off T during training increases. $T = 15$ appears to be the most efficient option, which corresponds to 30% of the overall trajectory length.

Figure 4.4: Spatial and temporal extrapolation capabilities of the first vortex model.

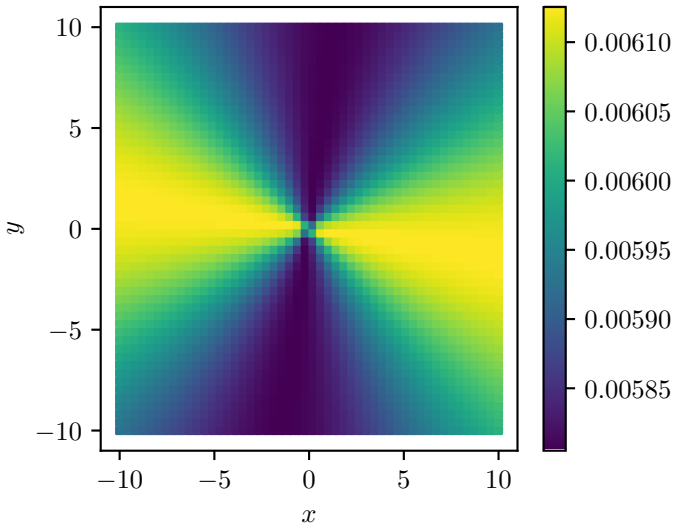


Figure 4.5: Relative error in $[-10, 10]^2$ of the SINDy model trained on $[-1, 1]^2$. The plot looks identical, both in terms of shape and error values, when considering larger regions $[-100, 100]^2, [-1000, 1000]^2$ and also a smaller region $[-0.1, 0.1]^2$.

extrapolating to the larger rectangle. This indicates good spatial generalization of the model. To further investigate whether this generalization is only a local phenomenon or extends globally across the entire two-dimensional space, we repeated the experiment with grids over regions of the form $[-10^c, 10^c]^2$ with $c \in 1, 2, 3$. Each grid was sampled with 50 points in each dimension. While this reduces the density for larger grids, we believe that this is sufficient to get an idea of the large-scale behavior. The result for $c = 1$ can be seen in Figure 4.5. The results for the other values of c look identical. Combined with the simple dynamics of the vortex, this suggests the existence of a global upper bound on the relative error, which is on the order of 7×10^{-2}. The particular landscape visible in the figure indicates that the model is marginally better at predicting trajectories whose initial conditions have a large y component in magnitude, and a relatively small x component.

Figure 4.4b shows the performance of temporal extrapolation. If the model was trained only on the first 10% of the available data of each trajectory, the error stands at approximately 0.05. A significant decrease in error is observed until $T = 15$, corresponding to 30% of the overall trajectory length. However, beyond this point, there is little reduction in error.[1] An explanation for this can be derived by looking at how much information is contained in a vortex trajectory that has been cut off at T. For $T = 5$, no full revolution is visible. For $T = 10$, about 1.5 revolutions are visible, and one can see that there are gaps in-between revolutions. Only starting at $T = 15$, one can make out the ratio with which gaps are growing, i.e. the vortex is expanding. Beyond this point, there is no new information to be gained from considering larger time horizons. Supporting this, we found that even across a longer time horizon, the performance still remains good, although there is a slight increase in error. For $T = 200.0$, the model trained on data until $T = 50.0$ has a relative error of approximately 0.0126.

[1] Notably, there is a slight uptick in error after 30%, which may be attributed to sampling variations rather than a genuine decline in performance.

Model		x	y	$\dot{x}$	$\dot{y}$	Error	Compl.
$\mathcal{M}_1$	$\ddot{x}$	0	13.454	-14.168	0	3.4×10^{-2}	4
	$\ddot{y}$	-13.454	0	0	-14.168		
$\mathcal{M}_2$	$\ddot{x}$	-0.795	-1.435	-1.4	0	2.3×10^{-2}	5
	$\ddot{y}$	-13.454	0	0	-14.168		
$\mathcal{M}_3$	$\ddot{x}$	**-0.795**	**-1.435**	**-1.4**	**0**	6.0×10^{-3}	**6**
	$\ddot{y}$	**1.439**	**-0.795**	**0**	**-1.404**		
$\mathcal{M}_4$	$\ddot{x}$	-0.795	-1.435	-1.4	0	7.3×10^{-3}	7
	$\ddot{y}$	1.43	-0.602	0.205	-1.408		
$\mathcal{M}_5$	$\ddot{x}$	-0.62	-1.421	-1.397	-0.186	7.9×10^{-3}	8
	$\ddot{y}$	1.43	-0.602	0.205	-1.408		

Table 4.3: Different models for varying threshold parameter $\tau \in [0, 0.95]$. Marked in bold is the best-performing model.

Hyperparameter Search. For this model, we have chosen the threshold parameter $\tau = 0.5$ by hand. To discover a potentially better model, we performed a hyperparameter search over τ by considering 20 equidistant values in the range $[0, 0.95]$, fitting a SINDy model for each, and computing the respective trajectory loss $\mathcal{L}_{\text{traj}}$ with $\alpha = 0.005$. The results can be seen in Figure 4.6. Due to the nature of the STLSQ algorithm, the landscape is piecewise constant. In total, 5 different models were produced. A particularly parsimonious model, found with $\tau \gtrapprox 0.65$, is given by

$$\begin{aligned} \ddot{x} &= ay + b\dot{x}, \\ \ddot{y} &= -ax + b\dot{y} \end{aligned} \tag{4.3.2}$$

with $a \approx -13.454$ and $b \approx -14.168$. Its relative error across the evaluation set stood at around 0.034. An overview over all five models is given in Table 4.3. The minimum is achieved between 0.25 and 0.6, validating the original choice of 0.5.

Another hyperparameter is the choice of ansatz function space. The amount of options is infinite, as one could, for instance, consider polynomials of arbitrarily high order. We considered polynomial orders between 1 and 4. Note that the number of monomials up to dth degree of n variables is $\binom{n+d}{n}$, which grows exponentially in d for fixed n. For a fixed threshold value $\tau = 0.5$, the same model (4.3.1) was found for each considered library.

Number of Trajectories. After the optimal parameters have been found, we investigated the effect of the number of trajectories in the training set. In many real-world applications, there is not enough training data available. Therefore, a model capable of learning from little training data or even just one sample, commonly referred to as *one-shot learning*, would be preferable. The model that resulted in (4.3.1) was trained on 80 trajectories. Figure 4.7a shows an approximation[2] of the relative error on the evaluation set as a function of the number of trajectories $N_{\text{traj}} \in \{1, 2, 3, 4, 5, 10, 20, 40, 80\} =: I$ for three randomly chosen sequences of training sets $(D_i)_{i \in I}$ with $|D_i| = i$ and $D_1 \subset D_2 \subset \cdots \subset D_{80}$. Naturally, there are slight variations between different sequences, but a general trend is visible. The performance of the model with a training set of size 1 is as good as, if not better, than the performance with the full training set. Interestingly, the error does not decrease with more training data, but sometimes

[2]The error was computed on only 25% of the evaluation set.

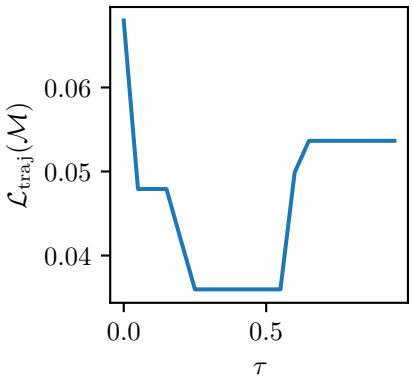
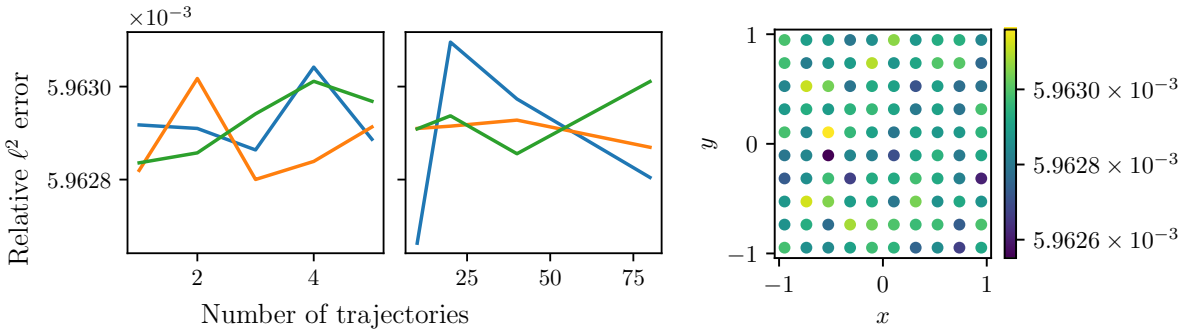


Figure 4.6: Results of the hyperparameter search for the threshold parameter of STLSQ. Regularization is set to $\alpha = 0.005$. The y-axis shows the trajectory loss $\mathcal{L}_{\text{traj}}$ of the model over the evaluation set. The smallest value is achieved between approximately 0.25 and 0.6.

even increases. However, it should be noted that these fluctuations happen on the order of 10^{-3}, implying hardly any substantial change in the error.

The above result suggests a good one-shot performance. It might then be worth investigating whether certain trajectories, used as single training data, are better for learning the vortex dynamics than others. We considered each trajectory in from the training data in $[-1, 1]$ and computed the error across the entire dataset. The results can be see in Figure 4.7b. The variation in error is extremely small (within a range of size 5×10^{-4}), demonstrating that no particular trajectory from this region provides a significantly better or worse one-shot performance. It should be noted that, due to the sampling of the initial data, no trajectories start with $x = 0$ or $y = 0$. In those cases, it follows directly from the definition of velocity field that the particle will be stationary in one of the dimensions, and hence, an insufficient model be learnt.

Noise. While the dataset considered here is numerically generated and thus contains no measurement noise, the latter is usually part of any type of real-world data. Thus, it is worth examining the effect of noise on the performance of SINDy for this dataset. To this end, normally distributed noise $n \sim \mathcal{N}(0, \sigma^2)$ was added to all trajectories in the training set, with $\sigma \in [0, 0.01]$, equidistantly sampled in 20 steps. The STLSQ algorithm with $\tau = 0.5$ was then run on the noisy data, followed by an evaluation over 20% of the evaluation set. The results, seen in Figure 4.8, indicate that even small amounts of noise – as little as $\sigma \approx 0.001$ – have a dramatic effect on the model error. An example trajectory from the (non-noisy) evaluation set along with the prediction of a SINDy model trained on noisy data is shown in Figure 4.9. For reference, the first row of Figure 4.10 shows a noisy trajectory with $\sigma = 0.001$ and the corresponding derivative data, computed using a finite difference method. It might seem surprising that the model discovery already fails in the case of this first row, since the noise in both trajectory and derivative is barely visible. However, one needs to remember that all features, including $\dot{x}$ and $\dot{y}$, are numerically differentiated during training to form $\dot{\mathbf{X}}$, causing a further amplification of noise.



(a) Effect of the number of trajectories on the error for three different random samples of the training data. Note the irregular spacing of the x-axis and the scale of the y-axis. The performance of the model only minimally improves with access to 80 trajectories compared to just one.

(b) One-shot performance of different trajectories in the region $[-1, 1]$. The variance in error is very small, indicating that no particular trajectory is better or worse suited for one-shot learning of the vortex dynamics.

Figure 4.7: Effect of number of trajectories on the error, as well as the choice of trajectory in the case of one-shot learning.

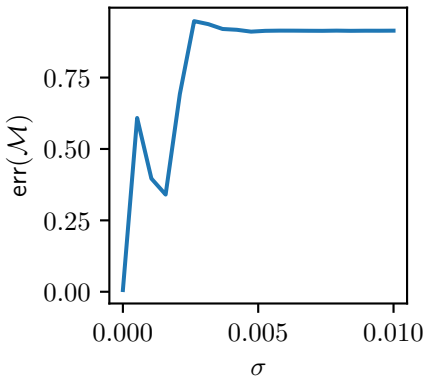


Figure 4.8: The effect of noise on the model error. Normally distributed noise with standard deviation $\sigma \in [0, 0.01]$ is added to each component of each trajectory in the training set. Even for small values of σ, the model error dramatically increases. Starting a bit before the $\sigma = 0.005$ mark, the error stays close to 1.

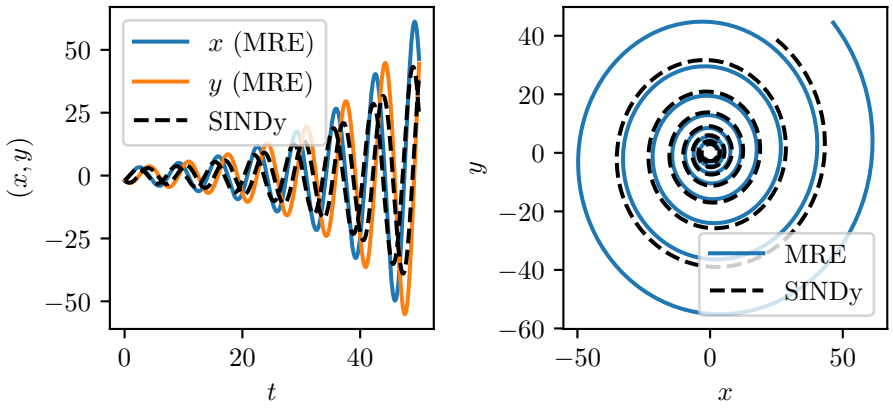


Figure 4.9: Example trajectory and the corresponding prediction of a SINDy model trained on noisy data with $\sigma \approx 0.001$.

Learning the History Term Only

Given that the history term is the main cause for the MRE's difficulty, it seems sensible to train a model only for this particular portion of the equation. So, given solutions $\mathbf{x}$ and $\tilde{\mathbf{x}}$ of the MRE with and without the history term, respectively, we wish to learn a model describing the difference $\mathbf{x} - \tilde{\mathbf{x}}$. However, here we reach the limits of SINDy: It is a tool for discovering dynamical systems, but the history force is the reason that the MRE is not a dynamical system. In this particular case, each trajectory satisfies $\mathbf{x}(0) = \tilde{\mathbf{x}}(0)$, and hence each trajectory to train the SINDy model for the history force would start in the origin.

4.3.2 Dynamics with variable equation parameters

First try. While the dynamical system (4.3.1) obtained in the previous section does a very good job at predicting the dynamics of the vortex field, it still lacks generality due to the presence of numerical parameters. Presumably, these must depend on the parameters of the system, which, in our case are made up of the particle and flow parameters: the density ratio $\rho := \rho_p/\rho_f$, the particle radius a and the kinematic viscosity ν. By including them as equation variables and constraining their time-derivatives to zero, one can, in principle, apply SINDy to discover dynamics that depend on the equation parameters. To this end, 125 different parameter combinations

$$(\rho, a, \nu) \in \{3, 2, 1, 1/2, 1/3\} \times \{0.5, 1.625, 2.75, 3.875, 5\}^2$$

were considered. For each combination, 100 trajectories with initial conditions in the region $[-1, 1]$ were generated.

A naive application of STLSQ with manually chosen hyperparameters proved unsuccessful. For $\tau = 0.05$ and a third degree polynomial library, the algorithm incorrectly identified $\dot{x} = 0.111\dot{x}\rho^2$, analogous for $\dot{y}$, and $\dot{t} = 0$. Furthermore, the model complexity was extremely high at $\|\boldsymbol{\Xi}\|_0 = 108$. The computation took approximately 6 seconds. Altering the threshold value did not yield considerable improvements. Instead, we switched to the SR3 algorithm with ℓ_0 thresholding, which, while yielding equally bad results, had the advantage of admitting constraints. Constraining $\dot{x}$, $\dot{y}$ and $\dot{t}$ to their correct values, however, still yielded an equally complex model ($\|\boldsymbol{\Xi}\|_0 = 72$), which performed poorly on a few example trajectories. To make

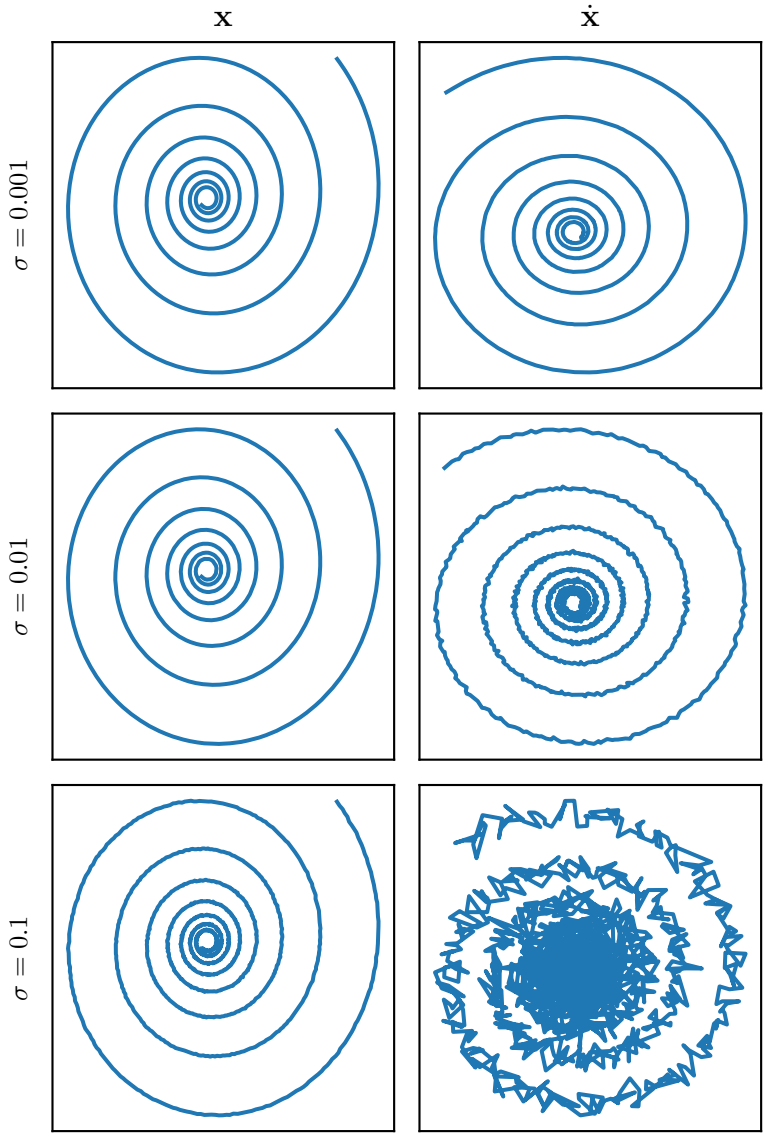


Figure 4.10: Noisy trajectories along with their numerically computed derivatives for different standard deviations $\sigma \in \{0.001, 0.01, 0.1\}$. Evidently, finite difference methods amplify noise: While the noise is not or barely visible in the trajectory, it is quite prominent in the corresponding derivative. This can be remedied by smoothing the data beforehand, or by applying regularized differentiation methods such as in [12].

matters worse, a thorough error evaluation over the entire dataset proved difficult, as the ODE solver would often fail to converge.

Second try. Subsequently, we performed a similar experiment, but on different data. Given that the vortex has an inward flow for $\rho < 1$, and an outward flow for $\rho > 1$, we decided to consider only one case. Hence, data was generated in the region $[2, 5] \times [1, 5]^2$ with 11 samples in each dimension. Using Algorithm 3 with the STLSQ optimizer and $\boldsymbol{\Theta}_i = \mathsf{Poly}_i(x, y, \dot{x}, \dot{y}, \rho, a, \nu)$ for $i \in \{1, 2, 3\}$, the resulting model was given by $\dot{x} = 0.676x - 1.645\dot{y}$ and $\dot{y} = 0.748y + 1.720\dot{x}$ – completely independent of a, ρ and ν. When explicitly setting $\tau = 0$, the model had a small dependence on the parameters, but all the corresponding coefficients were of low order:

$$\ddot{x} = -0.002 + 0.669x + 0.105y + 0.122\dot{x} - 1.649\dot{y} + 0.007\rho + 0.009a + -0.009\nu$$
$$\ddot{y} = -0.001 - 0.176x + 0.712y + 1.704\dot{x} + 0.203\dot{y} + 0.002\rho + 0.001a.$$

The average trajectory error (over the training set) was at around 0.62 and even the smallest trajectory error stood around 0.23. This indicates that the model did not learn the dynamics for any particular set of parameters, but seems to have found a tradeoff.

Using Symbolic Regression to Determine the Bulk Parameters

As the approach to generalizing the model by including the system parameters as target variables proved unsuccessful, we next tried to emulate the approach performed in [49]. The authors referred to these parameters as *bulk parameters* – a terminology we shall adopt. To further determine an expression for those parameters in relation to the constants of the physical system, they used symbolic regression via genetic programming.

Using the same data as in the second try above, we trained a SINDy model for each parameter tuple (ρ, a, ν) with STLSQ. For the training, three different threshold parameters $\tau = 0.005, 0.05, 0.5$ were tried and the model with lowest relative error on the training trajectory was selected. Figure 4.11 shows the relationship of the parameters with one of the coefficients, $\xi_{\ddot{x},x}$. The relationship does not appear continuous and contains several bumps, which can be explained as follows. For each set of parameters, there may be several good models describing the dynamics, and the best model for one set of parameters may not be the best for a different set.

To deal with this, we attempted to enforce a certain structure on the model, in the hopes of obtaining the same model "family" across all sets of parameters. Inspired by the particularly parsimonious model (4.3.2), we enforced the constraints $\xi_{\ddot{x},y} = -\xi_{\ddot{y},x}$ and $\xi_{\ddot{x},\dot{x}} = \xi_{\ddot{y},\dot{y}}$ and trained the model with SR3, once again trying out three threshold parameters $\tau = 5 \times 10^{-i}$ for $i \in \{1, 2, 3\}$ and choosing the model with lowest relative error. Figure 4.12 shows the same relationship for this new collection of models. While the precise values are quite different, the relationship now appears continuous. Furthermore, the largest relative error was very low at around 3×10^{-4}, indicating no major loss of accuracy despite the structural restrictions on the model.

At this point, one can apply symbolic regression methods to obtain expressions describing each of the model coefficients as functions of (ρ, a, ν). This is however beyond the scope of this work and can be viewed a further direction of research.

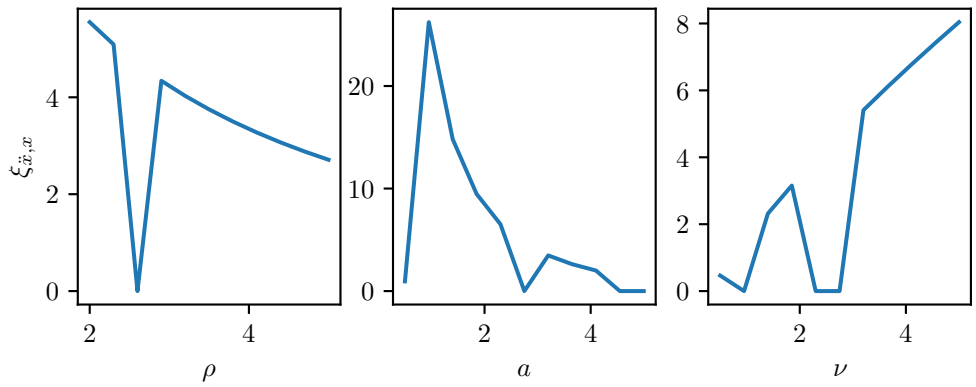


Figure 4.11: Relationship between equation parameters (ρ, a, ν) and the coefficient for x in the equation for $\ddot{x}$. In each of the three plots, the other two variables are fixed. The fixed values are given by $\rho^0 = 2.6, a^0 = 2.75$ and $\nu^0 = 2.75$. While a general trend is visible, there are still discontinuities.

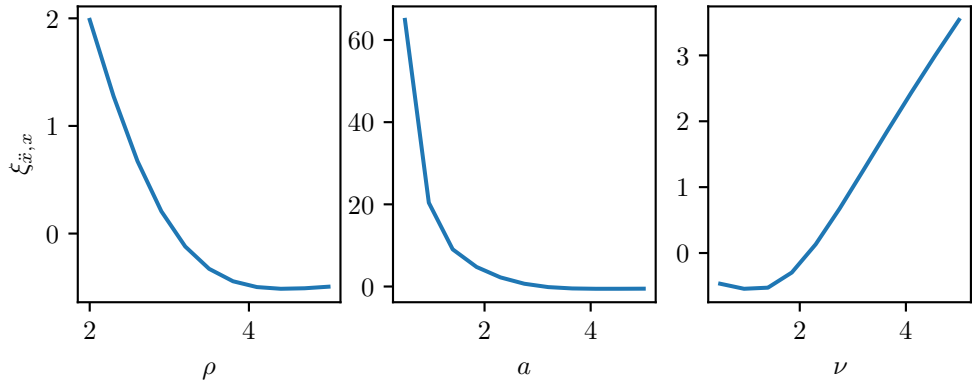


Figure 4.12: Relationship between equation parameters (ρ, a, ν) and the coefficient for x in the equation for $\ddot{x}$ for the constrained model. The fixed values are once again given by $\rho^0 = 2.6, a^0 = 2.75$ and $\nu^0 = 2.75$. The values are different from Figure 4.11, but indicate a smooth relationship.

4.4 Experiments with the Bickley Jet

Similar to the vortex, the trajectory data was sampled from the region $[-1, 1]^2$ with 10 samples per dimension, totaling 100 training trajectories. The equation parameters were the same as in Table 4.2.

4.4.1 Learning the Entire Trajectory

Proceeding similarly to the vortex, we first restricted ourselves to a polynomial function library. We performed a hyperparameter search over the polynomial degree $d \in \{1, 2, 3\}$ and the threshold parameter $\tau \in \{0, 0.05, 0.1, \ldots, 0.95\}$. The derivative loss $\mathcal{L}_{\text{deriv}}$ was used with $\alpha \approx 1.14$. This value was chosen as $10^{-3} \cdot \mathsf{cond}(\boldsymbol{\Theta})$, where $\boldsymbol{\Theta}$ is the system matrix obtained with the degree-1 polynomial library.[3] This choice is based on a heuristic presented in [46]. The best identified model was

$$\begin{aligned} \dot{x} &= 0 \\ \dot{y} &= -24.0 - 7.0y + 4.1\dot{x} \end{aligned}$$

and had a derivative loss of $\mathcal{L}_{\text{deriv}}(\mathcal{M}) \approx 5.45$. The best and worst trajectories are shown in Figure 4.13. The results are arguably unsatisfactory, which is only confirmed by the relative error, standing at around 0.73. The entire computation took around 166 seconds for 60 models in total, so a little under three seconds per model. Applying the same procedure with SR3 and ℓ_0 or ℓ_1 thresholding yielded the same model, but it took only 100 seconds.

The difficulty of the Bickley-Jet is the fact that the particle can "slip out" of the dominant sine wave, as seen in the bad example. This can happen even when $y_0 = 0$. Therefore, it does not seem sensible to only sample trajectories that stay within the sine wave to at least obtain a locally correct model. Instead, we try to aim for a more powerful model by incorporating more functions into the feature library. An obvious choice is given by the velocity field.[4] We added both of its components u and v to the feature library and used constraints to ensure that the coefficient of u in $\dot{x}$ is equal to the coefficient of v in $\dot{y}$. However, this would cause both coefficients to be 0. Taking it a step further and forcing the coefficients to be 1, yielded a model $\mathcal{M}$ with $\mathcal{L}_{\text{deriv}}(\mathcal{M}) \approx 10.8$ and $\mathcal{L}_{\text{traj}} \approx 8.62$. The relative error stood at $\mathsf{err}(\mathcal{M}) \approx 0.67$. While better than the previous model, this was still not satisfying.

Instead of considering the velocity field as a whole, we considered its individual linear components and constructed a library containing all of them, together with polynomial features. The final library $\mathcal{F}$ was

$$\mathcal{F} \coloneqq \mathsf{Poly}_3(x, y, \dot{x}, \dot{y}, t) \cup \mathcal{F}_{\text{trig}} \cup \mathcal{F}_{\text{hyp}} \cup (\mathcal{F}_{\text{trig}} \otimes \mathcal{F}_{\text{hyp}}) \tag{4.4.1}$$

where

$$\mathcal{F}_{\text{trig}} \coloneqq \{\sin(k_i x - s_i t), \cos(k_i x - s_i t) \mid i = 1, 2, 3\}$$

and

$$\mathcal{F}_{\text{hyp}} \coloneqq \{\tanh(y/L), 1 - \tanh(y/L)^2, \operatorname{sech}(y/L)^2, \tanh(y/L)\operatorname{sech}(y/L)^2\}.$$

[3] Since we want to compare libraries of different polynomial degrees, this choice is somewhat arbitrary. It might be more sensible to choose the largest or smallest condition number instead.

[4] Note that, while we did not explicitly do this for the vortex, its velocity field was linear in x and y and therefore contained as part of the polynomial library.

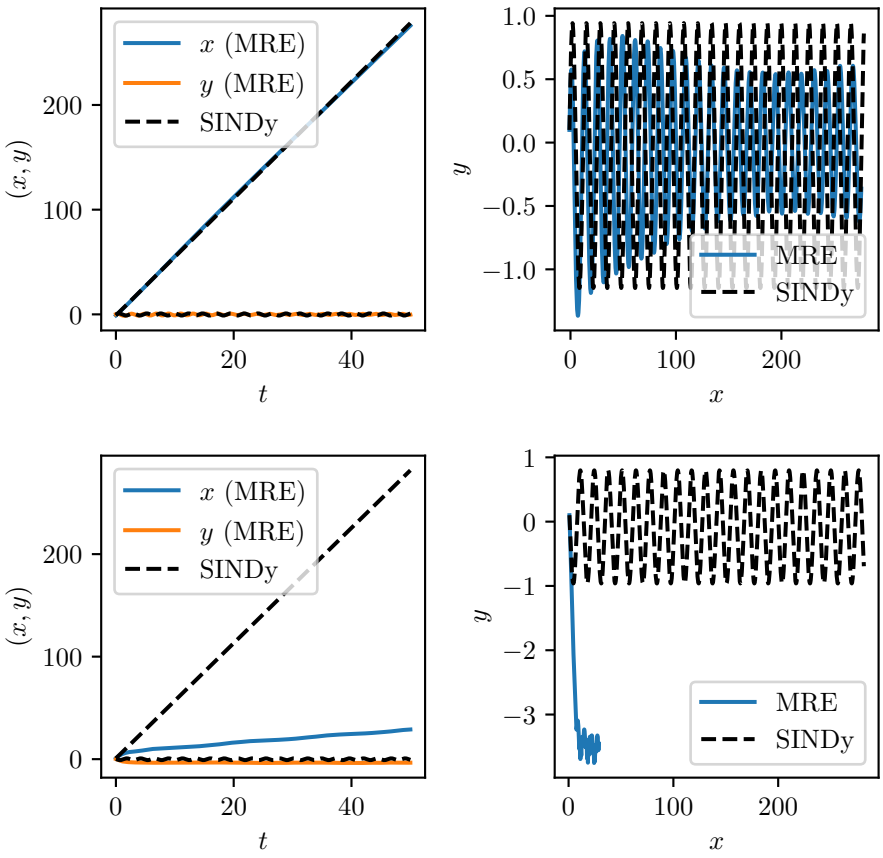


Figure 4.13: Bickley Jet: Best and worst performance of the SINDy model obtained with a polynomial library.

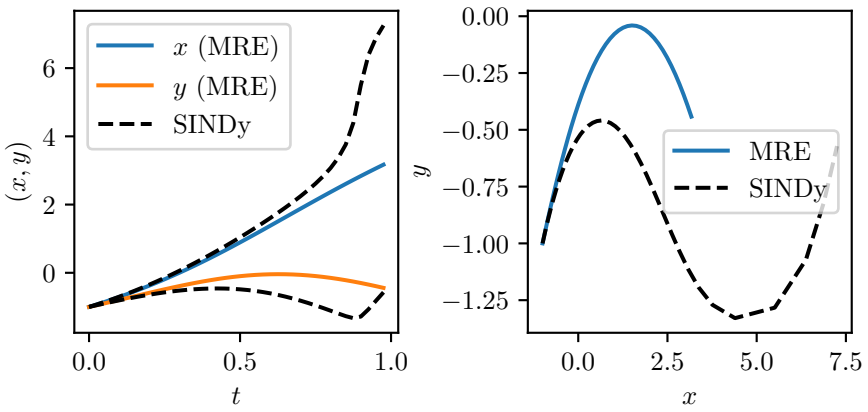


Figure 4.14: An example trajectory from the dataset along with the predictions of the model that was trained with the feature library (4.4.1). Results are only shown until time $T = 1.0$. The model diverges early on and does not perform well.

Here, $\otimes$ denotes pairwise multiplication, i.e. $A \otimes B := \{a \cdot b \mid a \in A, b \in B\}$. We applied Algorithm 3 with the SR3 optimizer on this problem, but manually set $\alpha = 0.05$, as the condition number of the matrix $\boldsymbol{\Theta}$ was extremely large for this problem. Constraints were used to enforce the three trivial equations. The resulting model was dense with a complexity of $\|\boldsymbol{\Xi}\|_0 = 72$, and had a derivative loss of $\mathcal{L}_{\text{deriv}}(\mathcal{M}; 0.05) \approx 8.51$. We were unable to compute the trajectory loss, but the model performed poorly on a randomly chosen trajectory over a time horizon of $[0.0, 1.0]$, see Figure 4.14.

Chapter 5

Conclusion and Outlook

5.1 Summary

In the present thesis, the system identification algorithm SINDy was applied, with the aim of obtaining an approximate system model of the Maxey-Riley equation – an integro-differential equation describing the movement of an inertial particle in a fluid. This search for a simpler model was motivated by the high computational costs of exact solvers.

First, we were successfully able to discover simple dynamics for the equation in the case of a simple velocity field containing a single vortex, and fixed equation parameters. We found an optimal model in the sense that it minimized an objective function that combines model error and complexity. We further investigated the number of necessary training trajectories and found that a single trajectory was enough to train the model to a desirable accuracy, as long as its initial condition did not lie on one of the coordinate axes. In particular, an increase in the number of training trajectories did not yield a decrease in error.

While the above procedure can be repeated to obtain SINDy models for other equation parameters, we attempted to generalize the resulting system. By generating trajectory data for varying equation parameters and training a SINDy model for each set of parameters, we were able to see the relationship between the parameters and the coefficients of the SINDy model. When done without restriction on the form of the model, the relationship was only piecewise continuous and sometimes contained strong fluctuations. After constraining the model via enforcing $\xi_{\ddot{x},y} = -\xi_{\ddot{y},x}$ and $\xi_{\ddot{x},\dot{x}} = \xi_{\ddot{y},\dot{y}}$, a continuous-looking relationship emerged. An application of symbolic regression to obtain mathematical expressions in $(\rho, a\nu)$ for the coefficients is possible at this point, but was beyond the scope of this work.

For a second, more complex velocity field, the Bickley Jet, we initially tried to find a model using purely polynomial features. The resulting model however was one-sided, as it contained the equation $\dot{x} = 0$, which was also reflected in its poor performance. To improve on this, in a first try we extended the feature library to include the entire velocity field. In a second try, we instead included several smaller components of the velocity field. However, both results were unsatisfactory.

5.2 Outlook and Future Work

This work addressed purely the data-driven approach using SINDy. As we observed in Chapter 4, this approach has its limits, particularly for complicated velocity fields and when trying to learn the history force in isolation. This was not possible with SINDy, because the different trajectories in the data set would, by construction, always start in the origin, and hence behave identically according to the model. To circumvent this, more flexible data-driven methods need to be introduced, such as neural networks. An approach worth investigating would be a universal differential equation of the form

$$\ddot{\mathbf{x}}(t) = \mathrm{MRE}(\mathbf{x}(t), \dot{\mathbf{x}}(t), t) + \mathrm{NN}_\theta(\mathbf{x}(t), \dot{\mathbf{x}}(t), t),$$

where MRE denotes the right-hand side of the Maxey-Riley equation without history force, and NN_θ is a neural network parameterized by θ. The recurrent neural network (RNN) architecture [23, Ch. 10] could be particularly suited for this task, due to the recurrent nature of the history term. This has in fact been done in [58], in which they augmented an approximation of the MRE by a long short-term memory network [26] G,

$$\dot{\mathbf{x}}(t) = \mathbf{u}(\mathbf{x}, t) + \frac{S}{R}(R-1)\frac{\mathrm{D}\mathbf{u}(\mathbf{x}, t)}{\mathrm{D}t} + G(\boldsymbol{\xi}(t), \boldsymbol{\xi}(t-\tau), \boldsymbol{\xi}(t-2\tau), \dots), \tag{5.2.1}$$

where τ denotes the time lag and $\boldsymbol{\xi}(t) = \left[\mathbf{u}(\mathbf{x}, t), \frac{\mathrm{D}\mathbf{u}(\mathbf{x},t)}{\mathrm{D}t}\right]^\top$ is a four-dimensional state vector. The authors tested their method on three different flows and achieved good prediction accuracy.

The idea of generalizing the model using symbolic regression set out in Section 4.3.2 seemed promising and could be investigated further. Similar to SINDy, the challenges include the choice of a good library of functions, which are then composed and mutated, as briefly described in Chapter 1.

A further, rather minor, improvement could be made on the concept of model complexity. In this work, only the number of nonzero entries in the coefficient matrix was considered. However, given two models of the form

$$\begin{aligned} \ddot{x} &= ay + b\dot{x}, \\ \ddot{y} &= -ax + b\dot{y} \end{aligned} \qquad \text{and} \qquad \begin{aligned} \ddot{x} &= ay + b\dot{x}, \\ \ddot{y} &= cx + d\dot{y} \end{aligned}$$

for $a, b, c, d \neq 0$, the first one is arguably less complex than the second, despite the fact that both share the same number of nonzero entries. Hence, the number of different nonzero entries of the coefficient matrix might be a more accurate complexity measure.

Finally, advanced machine learning techniques can be leveraged to combine multiple SINDy models into an enhanced composite model which may yield better performance. This might involve training distinct models on different subsets of an extensive feature library, followed by their incorporation into ensemble learning techniques, such as bagging or boosting. However, a notable drawback of this approach is the diminished interpretability, given the absence of a unified dynamical system that comprehensively describes the dynamics.

Concluding this thesis, it can be stated that, while remarkable results have been demonstrated with the SINDy algorithm on suitable data, our results indicate that this does not immediately ring true for the Maxey-Riley equation. From a theoretical perspective, there is an obvious conflict given by the fact that the MRE model does not constitute a dynamical system, yet the SINDy algorithm exclusively learns dynamical systems. That said, the SINDy approach is flexible and depends strongly on the feature library, and it still remains a possibility that a good approximation can be found with the right feature library.

Bibliography

[1] Vladimir I. Arnol'd. *Ordinary Differential Equations, Third Edition.* Springer-Verlag Berlin Heidelberg, 1992. Chap. 1, pp. 43–45.

[2] T. R. Auton, J. C. R. Hunt, and M. Prud'Homme. "The force exerted on a body in inviscid unsteady non-uniform rotational flow". In: *J. Fluid. Mech.* 197 (1988), pp. 241–257.

[3] Alfred Barnard Basset. *A treatise on hydrodynamics: with numerous examples, Vol. 2.* Deighton, Bell and Company, 1888. Chap. 22, pp. 285–297.

[4] James Bergstra and Yoshua Bengio. "Random Search for Hyper-Parameter Optimization". In: *Journal of Machine Learning Research* 13 (2012), pp. 281–305.

[5] F. J. Beron-Vera and P. Miron. "A minimal Maxey–Riley model for the drift of Sargassum rafts". In: *J. Fluid Mech.* 904 (2020).

[6] Joseph Boussinesq. "Sur la resistance quoppose un fluide indefini en repos, sans pesanteur, au mouvement varie d'une sphere solide qu'il mouille sur toute sa surface, quand les vitesses restent bien continues et assez faibles pour que leurs carres et produits soient negligiables". In: *C. R. Acad. Sci. Paris* 10 (1885), pp. 935–937.

[7] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. "Discovering governing equations from data by sparse identification of nonlinear dynamical systems". In: *Proceedings of the National Academy of Sciences* 113.15 (Mar. 2016), pp. 3932–3937. DOI: 10.1073/pnas.1517384113. URL: https://doi.org/10.1073%2Fpnas.1517384113.

[8] F. Candelier, J. R. Angilella, and M. Souhar. "On the effect of the Boussinesq–Basset force on the radial migration of a Stokes particle in a vortex". In: *Physics of Fluids* 16.5 (2004), pp. 1765–1776.

[9] Alejandro Carderera et al. "CINDy: Conditional gradient-based Identification of Nonlinear Dynamics - Noise-robust recovery". In: *Journal of Computational and Applied Mathematics* (2021). under review.

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

[11] Kathleen Champion et al. "Data-driven discovery of coordinates and governing equations". In: *Proceedings of the National Academy of Sciences* 116.45 (2019), pp. 22445–2245.

[12] Rick Chartrand. "Numerical Differentiation of Noisy, Nonsmooth Data". In: *ISRN Applied Mathematics* (May 2011).

[13] Ricky T. Q. Chen et al. "Neural Ordinary Differential Equations". In: *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*. 2018.

[14] S. Corrsin and J. Lumley. "On the Equation of Motion for a Particle in Turbulent Fluid". In: *Appl. Sci. Res.* 6 (1956), pp. 114–116.

[15] C. Crowe et al. *Multiphase Flows with Droplets and Particles, Second Edition.* CRC Press, 2012. Chap. 4, pp. 93–94.

[16] Anton Daitche. "Advection of intertial particles in the presence of the history force: Higher order numerical schemes". In: *Journal of Computational Physics* 254 (2013), pp. 93–106.

[17] Anton Daitche. "On the role of the history force for intertial particles in turbulence". In: *J. Fluid. Mech.* 782 (2015), pp. 567–593.

[18] Maolin Du, Zaihua Wang, and Haiyan Hu. "Measuring memory with the order of fractional derivative". In: *Scientific Reports* 3 (2013).

[19] Lawrence C. Evans. *Partial Differential Equations, Second Edition.* American Mathematical Society, 2010. Chap. 2, pp. 44–65.

[20] Mohammad Farazmand and George Haller. "The Maxey–Riley equation: Existence, uniqueness and regularity of solutions". In: *Nonlinear Analysis: Real World Applications* 22 (2015), pp. 98–106.

[21] Richard P. Feynman, Robert B. Leighton, and Matthew Sands. *The Feynman Lectures on Physics, Volume II.* Originally published 1964. Addison-Wesley, 2013. Chap. 41. URL: `https://www.feynmanlectures.caltech.edu/II_41.html`.

[22] R. Gatignol. "The Faxén formulae for a rigid particle in an unsteady non-uniform Stokes flow". In: *J. Mec. Theor. Appl.* 1 (1983), pp. 143–160.

[23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* The MIT Press, 2016.

[24] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* The MIT Press, 1992.

[25] Daniel Heldt et al. "Approximate computation of zero-dimensional polynomial ideals". In: *Journal of Symbolic Computation* 44.11 (2009), pp. 1566–1591.

[26] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997).

[27] Kadierdan Kaheman, J. Nathan Kutz, and Steven L. Brunton. "SINDy-PI: a robust algorithm for parallel implicit sparse identification of nonlinear dynamics". In: *Proc. R. Soc. A* 476.2242 (2020).

[28] Alan A. Kaptanoglu et al. "PySINDy: A comprehensive Python package for robust sparse system identification". In: *Journal of Open Source Software* 7.69 (2022), p. 3994. DOI: `10.21105/joss.03994`. URL: `https://doi.org/10.21105/joss.03994`.

[29] Mårten Landahl and E. Mollo-Christensen. "Turbulence and Random Processes in Fluid Mechanics, Second Edition". In: Cambridge University Press, 1992, p. 10.

[30] Gabriel Provencher Langlois, Mohammad Farazmand, and George Haller. "Asymptotic Dynamics of Inertial Particles with Memory". In: *Journal of Nonlinear Science* 25 (2015), pp. 1225–1255.

[31] Marcia Levitus. *Mathematical Methods in Chemistry.* Arizona State University. Chap. 4, pp. 61–74.

[32] Xiaoxuan Liu, Livia Faes, and Aditya U Kale et al. "A comparison of deep learning performance against health-care professionals in detecting diseases from medical imaging: a systematic review and meta-analysis". In: *Lancet Digital Health* 1.6 (2019).

[33] Adam Loverro. *Fractional Calculus: History, Definitions and Applications for the Engineer*. `https://web.archive.org/web/20051029113800/http://www.nd.edu/~msen/Teaching/UnderRes/FracCalc.pdf`. 2004.

[34] Alan M. Turing. "Computing Machinery and Intelligence". In: *Mind* 236 (1950), pp. 433–460.

[35] Niall M. Mangan et al. "Inferring Biological Networks by Sparse Identification of Nonlinear Dynamics". In: *IEEE Transactions on Molecular, Biological, and Multi-scale Communications* 2.1 (2016), pp. 52–63.

[36] Martin R. Maxey and James J. Riley. "Equation of motion for a small ridig sphere in a nonuniform fluid". In: *The Physics of Fluids* 26 (1983), pp. 883–889.

[37] Daniel A. Messenger and David M. Bortz. "Weak SINDy: Galerkin-Based Data-Driven Model Selection". In: *Multiscale Modeling & Simulation* 19.3 (Jan. 2021), pp. 1474–1497. DOI: `10.1137/20m1343166`. URL: `https://doi.org/10.1137%2F20m1343166`.

[38] B. K. Natarajan. "Sparse Approximate Solutions to Linear Systems". In: *SIAM Journal of Computation* 24.2 (1995).

[39] Neal Parikh and Stephen Boyd. "Proximal Algorithms". In: *Foundations and Trends in Optimization* 1.3 (2014), pp. 127–239.

[40] Linda Petzold. "Automatic Selection of Methods for Solving Stiff and Nonstiff Systems of Ordinary Differential Equations". In: *SIAM Journal on Scientific and Statistical Computing* 4.1 (1983).

[41] S. Ganga Prasath, Vishal Vasan, and Rama Govindarajan. "Accurate solution method for the Maxey-Riley equation, and the effects of Basset history". In: *Journal of Fluid Mechanics* 868 (2019), pp. 428–460.

[42] John R. Koza. "Genetic programming as a means for programming computers by natural selection". In: *Statistics and Computing* 4 (1994), pp. 87–112.

[43] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, 1992.

[44] Christopher Rackauckas et al. "Universal Differential Equations for Scientific Machine Learning". In: *CoRR* abs/2001.04385 (2020). URL: `https://arxiv.org/abs/2001.04385`.

[45] Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378 (2019), pp. 686–707.

[46] Samuel H. Rudy et al. "Data-driven discovery of partial differential equations". In: *Science Advances* 3.4 (2017).

[47] Mario Sandoval and Omar Vergara. *Drops in the wind: their dispersion and COVID-19 implications*. Preprint. 2021. arXiv: `2102.08277 [physics.flu-dyn]`.

[48] Hayden Schaeffer and Scott G. McCalla. "Sparse model selection via integral terms". In: *Physical Review E* 96.2 (2017).

[49] Michael Schmidt and Hod Lipson. "Distilling Free-Form Natural Laws from Experimental Data". In: *Science* 324.81 (2009).

[50] Brian de Silva et al. "PySINDy: A Python package for the sparse identification of nonlinear dynamical systems from data". In: *Journal of Open Source Software* 5.49 (2020), p. 2104. DOI: 10.21105/joss.02104. URL: https://doi.org/10.21105/joss.02104.

[51] David Silver, Thomas Hubert, and Julian Schrittwieser et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419 (2018), pp. 1140–1144.

[52] Joseph H. Spurk and Nuri Aksel. *Fluid Mechanics, Second Edition.* Springer-Verlag Berlin Heidelberg, 2008.

[53] George Gabriel Stokes. "On the effect of internal friction of fluids on the motion of pendulums". In: *Transactions of the Cambridge Philosophical Society* 9, part 2 (1856), pp. 8–106.

[54] Wenbo Tang et al. "Locating an atmospheric contamination source using slow manifolds". In: *Physics of Fluids* 21 (2009).

[55] Chan Mou Tchen. "Mean value and correlation problems connected with the motion of small particles suspended in a turbulent fluid." PhD thesis. TU Delft, 1947.

[56] Robert Tibshirani. "Regression Shrinkage and Selection via the Lasso". In: *Journal of the Royal Statistical Society Series B* 58.1 (1996), pp. 267–288.

[57] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[58] Zhong Yi Wan and Themistoklis P. Sapsis. "Machine learning the kinematics of spherical particles in fluid flows". In: *J. Fluid Mech.* 857 (2018).

[59] Linan Zhang and Hayden Schaeffer. "On the Convergence of the SINDy Algorithm". In: *Multiscale Modeling & Simulation* 17.3 (2019), pp. 948–972.

[60] Peng Zheng et al. "A Unified Framework for Sparse Relaxed Regularized Regression: SR3". In: *IEEE Access* 7 (2018), pp. 1404–1423.