UNIVERSITY OF CALIFORNIA SAN DIEGO

Construction of Predictive Dynamical Systems from Observed Data Through Data Driven Forecasting

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Physics

by

Randall Clark

Committee in charge:

Professor Massimiliano Di Ventra, Chair
Professor Kamalika Chaudhuri
Professor Duncan Watson-Parris
Professor Yi-Zhuang You

2023

The Dissertation of Randall Clark is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2023

DEDICATION

To my loving parents who made great sacrifices to ensure the success of me and my siblings.

In addition to dedicating this dissertation to my parents, I must also extend this dedication to my advisor, Henry Abarbanel, who tragically passed away days before my dissertation defense. Henry Abarbanel was a brilliant scientist, loving father, an active member of his local community, and a close friend. Henry one time expressed to me that as my advisor he felt responsible for my success during my time in grad school and beyond; he went out of his way to set me up for success beyond graduate school, asking me about my interests and using whatever tools were at his disposal to get me interviews and internships. It was consistent acts of kindness like this that made me realize that Henry's scientific brilliance was matched by his compassion for others. I hope to live my life following the example Henry set with a never ending thirst for knowledge, an undying passion to explore science, and to spread love and generosity as Henry did.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Along this long academic journey, I have received support, encouragement, and inspiration from many people with contributions big and small. I firstly must thank both my parents, Spencer Clark and Kathleen Davis for their unwavering love and support, which made this journey possible. I want to thank my friends whose cooperation and senses of humor got us through rigorous and long sessions of work. I also want to thank my girlfriend, Dr. Kayla Shires, for her love and support throughout the writing of this dissertation.

I want to make a special thanks to my advisor, Henry Abarbanel. Many advisors are brilliant and inspirational, but in addition to this, Dr. Abarbanel has also shown to be patient, kind, and supportive in his role as an academic researcher and advisor.

I want to thank my fellow graduate students in the Abarbanel group (many of which have already graduated) for the support and guidance they have offered along the way. Specifically, I give my thanks to Zheng Fang, Jason Platt, Anna Miller, Adrian Wong, Lawson Fuller, and others for friendship and support.

I'd like to make a special thanks to Dan Margoliash, and all those in the Margoliash lab who spent long hours trying to teach a physicist biology and providing ample test data for all our requests.

This dissertation contains material from published work. Chapter 5 has been adapted from Randall Clark, Lawson Fuller, Jason A Platt, and Henry DI Abarbanel. "Reduced-Dimension, Biophysical Neuron Models Constructed From Observed Data". *Neural Computation*, 34(7):1545–1587, 2022. The dissertation author was the primary investigator and author of this paper.

Chapter 6, has been adapted from a paper that is currently submitted for publication in the journal Physical Review Fluids. The material may appear as Randall Clark, Luke C. Fairbanks, Ramon E. Sanchez, Pacharadech Wacharanan, and Henry D.I. Abarbanel. "Data Driven Regional Water Forecasting: Example Using the Shallow Water Equations".

Submitted to *Physical Review Fluids* 2023. Arxiv: https://arxiv.org/abs/2303.16363. The dissertation author was the primary investigator and author of this paper.

VITA

2016	Associates of Science in Physics, Las Positas Community College
2018	Bachelor of Arts in Physics, University of California Berkeley
2022	Master of Science in Physics, University of California San Diego
2023	Doctor of Philosophy in Physics, University of California San Diego

PUBLICATIONS

Jason A Platt, Adrian Wong, Randall Clark, Stephen G Penny, and Henry DI Abarbanel. "Robust forecasting using predictive generalized synchronization in reservoir computing". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 31.12 (2021), p. 123118

Randall Clark, Lawson Fuller, Jason A Platt, and Henry DI Abarbanel. "Reduced-Dimension, Biophysical Neuron Models Constructed From Observed Data". In: *Neural Computation* 34.7 (2022), pp. 1545–1587

Randall Clark, Henry Abarbanel, Luke C Fairbanks, Ramon E Sanchez, and Pacharadech Wacharanan. "Data Driven Regional Weather Forecasting: Example using the Shallow Water Equations". In: *arXiv preprint arXiv:2303.16363* (2023)

ABSTRACT OF THE DISSERTATION

Construction of Predictive Dynamical Systems from Observed Data Through Data Driven Forecasting

by

Randall Clark

Doctor of Philosophy in Physics

University of California San Diego, 2023

Professor Massimiliano Di Ventra, Chair

The evolution of particles in space, flows on an ocean surface, or orbits of the planets can all be thought of as their own dynamical systems who's forecasts and models are crucial to many scientific disciplines. These dynamical system models depict the physics of what is going on by mathematically describing how each state variable of the system evolves in time. It is our role as computational physicists to find solutions to these complex and often analytically unsolvable dynamical system models to aid in the study of interesting and important physics.

In this dissertation we will go through the development and deployment of a melding

of methods in applied mathematics and machine learning to construct approximate forms to dynamical systems equations for forecasting from data alone in a method known as Data Driven Forecasting (DDF). A theoretical background for the method is first discussed along with a sampling of the different variations of DDF. The utilization of Radial Basis Functions (RBF) to interpolate the behavior of dynamical systems plays a major role approximating the flow of the model dynamics. A breakdown of what dynamical properties like chaos, fractal dimension, Lyapunov exponent, and Jacobian are preserved and under what conditions in reconstructing the model from data.

As DDF builds models from observed data alone, it will contend with the challenge of construction model approximations when fewer than the total dimensions are observed. Through the use of Taken's Embedding Theorem and time delay embedding techniques, the attractor can be reconstructed and forecasting made possible.

This dissertation concludes with a thorough exploration of the method on a Neuro Dynamical system and Fluid Dynamical system where reduced dimensional observations are made and time delay embedding techniques must be used. The results shown in these sections are indicative of the potential for this method to both be expanded upon and applied for modern scientific pursuits.

Chapter 1

Introduction

1.1 Dynamical Systems

The field of Dynamical Systems is the method of analyzing systems as they evolve over time in the form of differential equations or iterated maps; the systems these mathematical objects describe could be swinging pendulums, evolution of stars, orbiting planets, oceanic flows, or even the fractal Mandelbrot set. These models range in their complexity, some are simplifications of the true system because the true system is too hard to understand, some models are best guesses of an unknown perfect model of a complex system. For our purposes, we describe our systems mathematically with systems of ordinary differential equations that model the time evolution of our state variables in state space. These models are described as D dimensional models with parameters p, state variables $\boldsymbol{x}$, and take the following form

$$\frac{dx_i(t)}{dt} = f_i(\boldsymbol{x}(t), p) \qquad\qquad i = 1, ..., D \tag{1.1}$$

The f function in our equation above is what a lot of our research revolves around. It can be built from first principles like the Naiver Stokes equations, or it can be inferred from observational data. The usefulness of the model depends on the model's ability to predict the future behavior of the system.

1.1.1 The History of Dynamical Systems

Many of the properties of dynamical systems have their origin with Newton when he invented differential equations to explain planetary motion, but the beginning of modern dynamical system's theory starts with the work Poincarè published in 1880 when he established the foundation for qualitative analysis of nonlinear differential equations. In this initial framework he invented Poincaré maps (for the study of periodic motion), defined stable and unstable manifolds, introduced perturbation methods, and invented the Poincaré Recurrence Theorem. His ideas were influential because they brought forth qualitative solutions to problems that were very troublesome to solve quantitatively. [4][5][6]

Before the discovery of chaos in 1963, dynamical systems were mostly applied to non-linear oscillators that were very useful to physicists and engineers. Non-linear oscillators were vital in the development of such technologies such as radio, radar, phase-locked loops, and lasers. One example of a non-linear oscillator would be the Colpitts oscillator invented by Edwin Colpitts in 1918 shown in Figure (1).

An analysis of the simple circuit along with some approximations will lead one to the following system of differential equations modelling the voltage across both capacitors and the current through the inductor:

$$\begin{aligned} \frac{dV_{C_1}(t)}{dt} &= \frac{1}{C_1}[I_L - I_S(e^{-V_{C2}/V_{th}} - 1)] \\ \frac{dV_{C_2}(t)}{dt} &= \frac{1}{C_2}[I_L - (V_{C2} + v)/R_E] \\ \frac{dV_{I_L}(t)}{dt} &= \frac{1}{L}[v - R_L I_L - V_{C_2} - V_{C_1}] \end{aligned} \tag{1.2}$$

I_S is the reverse saturation current, v is the constant voltage supplied at the top of the circuit, and $V_{th} = k_B * T/e \approx 0.026 Volts$ is the thermal voltage. There are several forms of the Colpitts Oscillator, each exhibiting complex dynamical behavior; therefore, it is useful to look at the Colpitts Oscillator in a dimensionless form that is unrelated to any

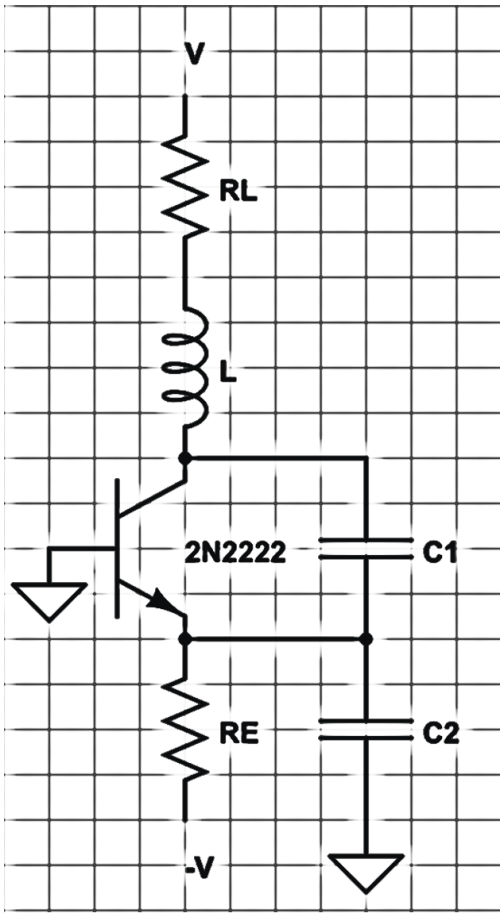

Figure 1.1. Colpitts Oscillator Circuit Design [7]. It was invented in 1918 by Edwin Colpitts and its dynamics are described in equation(1.2) and (1.3). Under certain conditions, the system can exhibit properties of chaos.

specific circuit diagram but still maintains the interesting dynamics. This generic Colpitts Oscillator system of equations takes the following form[7]:

$$\begin{aligned} \frac{dx_1(t)}{dt} &= \alpha x_2 \\ \frac{dx_2(t)}{dt} &= -\gamma(x_1 + x_3) - q x_2 \\ \frac{dx_3(t)}{dt} &= \eta(x_2 + 1 - e^{-x_1}) \end{aligned} \tag{1.3}$$

These equations have no known analytical solution and therefore must be solved numerically. More interestingly, when the equations are solved with a careful choice of parameters, the dynamical system exhibits chaotic behavior, a concept we'll discuss in

more detail in a later section. The chaotic behavior of the Colpitts Oscillator is shown in Figure(1.2).

As it turns out, most of the dynamical systems that we are interested in, like the Colpitts Oscillator, have no known analytical solution. The dynamical systems we are interested in can be very complex and don't lend themselves to finding easy solutions. To overcome this problem and gain a better understanding of the models we want to study, we must solve the dynamical equations numerically.

1.1.2 Solving Dynamical Systems

When analytical solutions to systems of differential equations are not known, our only option to solve these equations numerically by advancing the state parameters forward in time. We are interested to see how the state of our system evolves, and we want to be able to predict the future behavior of our systems; accurately predicting the behavior of a system is helpful because it shows our model is correct which can verify theoretical assumptions about system behavior, it can also be practical to forecast systems like weather to make climate predictions. There are two primary cases that we differentiate, cases of Ordinary Differential Equations (ODE's), and cases of Partial Differential Equations (PDE's).

Systems of Ordinary Differential Equations are the more common system we deal with and are easier to solve. One of the earliest methods of solving ODE's was the Euler Method, invented by Leonhard Euler in 1770[8]. It is a first order method that updates the position of the state variables by moving along the tangent line, the first derivative, a distance h before recalculating the tangent line and moving another distance h. If we know an initial value and the form of the derivative that can be calculated at all points, we can write out the Euler Method as:

$$x(t_0 + h) = x(t_0) + h * \frac{dx}{dt}|_{t=t_0} \tag{1.4}$$

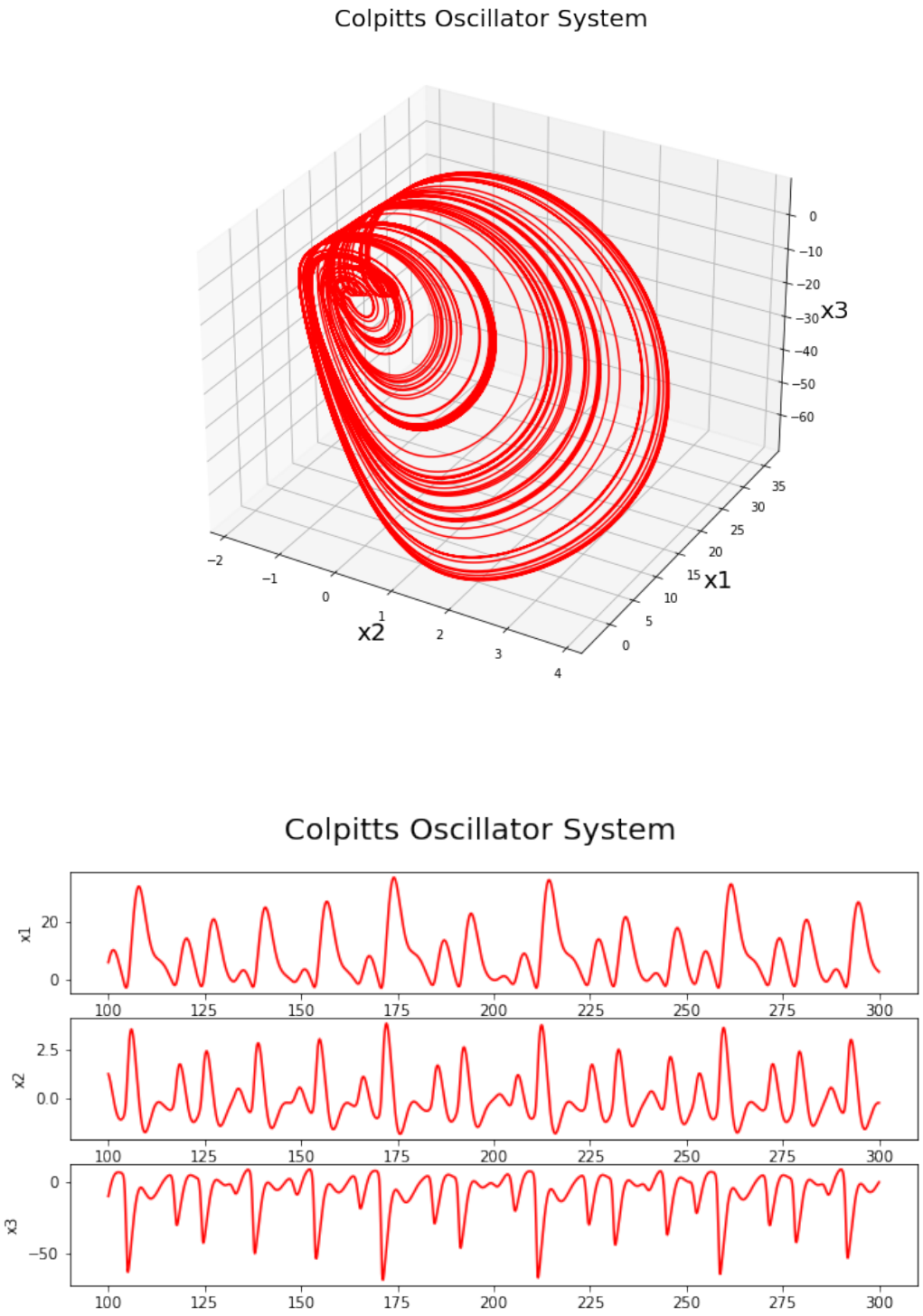

Figure 1.2. Colpitts Oscillator with parameter values $\alpha = 5$, $\gamma = 0.0797$, q = 0.6898, and $\eta = 6.2723$. It's more plainly visible in the 3D graph that the system is chaotic in nature; notice how despite going through many cycles in its phase space, the system is not periodic (for the given parameter values).

It is first order because the global error scales with the step size h, which means we would need to take very small steps to integrate accurately; this is a computationally slow process, so it is better to use a higher order method. In 1901 the commonly used method of integrating ODE's was developed, called the Runge-Kutta Methods. The idea of the Runge-Kutta methods is that we want to make use of the information within the interval of h, the next improvement to Euler's method, the trapezoidal rule, does exactly that and is second order. But the most commonly used form is the fourth order Runge-Kutta method:

$$k_1 = hf(t_n, x_n) \tag{1.5}$$

$$k_2 = hf(t_n + \frac{1}{2}h, x_n + \frac{1}{2}k_1) \tag{1.6}$$

$$k_3 = hf(t_n + \frac{1}{2}h, x_n + \frac{1}{2}k_2) \tag{1.7}$$

$$k_2 = hf(t_n + h, x_n + k_3) \tag{1.8}$$

$$x_{n+1} = x_n + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 + O(h^5) \tag{1.9}$$

$$t_{n+1} = t_n + h \tag{1.10}$$

With the use of the fourth order method, we are now accurate to order $\mathcal{O}(h^4)$ with a truncation error of $\mathcal{O}(h^5)$, but this still may not be enough. Higher order is important, but we can do more to make sure our integration is accurate, and that can be done using an adaptive step size. A good ODE integrator should exert some adaptive control over its own step size to accommodate regions of large truncation error (the contribution to the step we don't account for from higher order h terms). In regions where large truncation error is detected, usually by comparing the difference in fourth and fifth order Runge-Kutta, the step size is forced to decrease until the truncation error reaches a small enough value to proceed, and increase the step size where truncation error is small to decrease the computational time. Smooth regions often have low truncation error because

there isn't much change in the landscape. But, regions of rapid change and non smoothness increase the truncation error, and introduce a lot of inaccuracies to the integration if smaller step sizes aren't taken. The python library, scipy, has a tool called ODEINT which incorporates 4^{th} Order Runge Kutta with adaptive step sizes and is our primary tool for quickly integrating ODE's. [9]

Solving systems of partial differential equations requires a different approach. Among the methods of solving PDE's there are finite differencing, Monte Carlo, Spectral, and Variational Methods. In our work, we've only used the finite differencing method, so we'll describe its properties and uses here. The idea of finite differencing is to view our state space as a finite incremental grid. The value of each state variable in different locations in space will rely on their variations in nearby points to approximate their spatial derivative. This will also require the information on how to handle boundaries for points located on boundaries with no neighbor on one or more sides of them. For a set of points on a grid with locations $x_0, ..., x_{D_x}$ and $y_0, ..., y_{D_y}$ we can break up partial spatial derivatives of a state variable v (where v could be any state variable in space, like fluid speed in an ocean) in the following way:

$$\frac{\partial v_{i,j}}{\partial x} \approx \frac{v_{i+1,j} - v_{i,j}}{\Delta x} \tag{1.11}$$

$$\frac{\partial^2 v_{i,j}}{\partial^2 x} \approx \frac{(v_{i+1,j} - v_{i,j}) - (v_{i,j} - v_{i-1,j})}{(\Delta x)^2} \tag{1.12}$$

These are a multitude of ways of breaking up the finite difference, the top equation is known as the forward Euler differencing as we are comparing our value at the (i,j) point to the (i+1,j) point in front of it, and the bottom equation is for 2nd derivatives. These are both first order methods and higher order methods exist, but that goes beyond the scope of this dissertation. One final point of significant importance in the domain of PDE's is the stability of the finite difference method. There is no guarantee that any well done finite difference scheme of PDE's will generate a stable result, for this we have to perform

the Von Neumann Stability Analysis or other test to confirm that the finite differencing scheme we choose will be stable. [9]

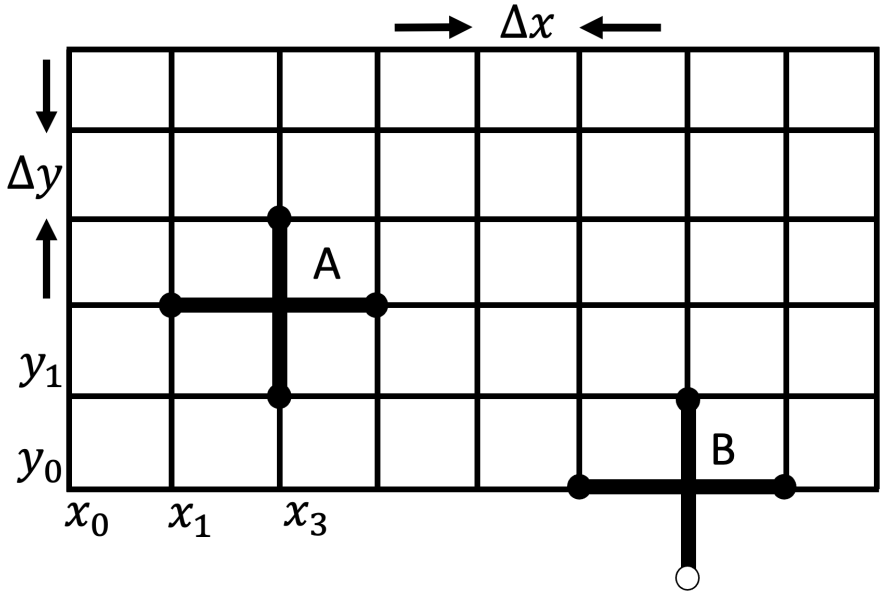

Figure 1.3. This example grid shows how one should perceive finite differencing methods and how at each point, say A, we must consider the points around it to calculate its spatial derivatives. Additionally, should a point, like B, be on the boundary, boundary conditions must be taken into consideration here. Higher order finite differencing methods will go beyond this to include points not just 1 space away but two or more; some higher order methods may even include the diagonal components in their schemes.

1.1.3 Chaos in Dynamical Systems

There is one critical concern that can come when studying dynamical systems, and that is the topic of chaos and what dynamical systems exhibit chaos [10]. The invention of computers gave rise to the ability to solve Dynamical Systems numerically so rapidly that it quickly led to the discovery of chaos by Edward Lorenz in 1963 [11]. Edward Lorenz derived a simplified model of convection rolls in the atmosphere that at a glance appears

simple and unassuming (while too analytically intractable to solve):

$$\begin{aligned} \frac{dx(t)}{dt} &= \sigma(y - x) \\ \frac{dy(t)}{dt} &= rx - y - xz \\ \frac{dz(t)}{dt} &= xy - bz \end{aligned} \tag{1.13}$$

For a wide range of parameters, but not all parameters, Lorenz discovered that the numerical solution gave very erratic answers. The system wouldn't settle to fixed points (points where the system will converge to and stabilize), limit cycles (infinitely repeating loops in space systems stabilize to), or blow up to infinity, but rather, it would oscillate irregularly, never exactly returning to the same place, but staying within a bounded region of space. For fixed points and limit cycles, we say that there exists an attractor for which the trajectories in local space around it all converge to and stays in for all time. For chaotic systems, though, we define the strange attractor that systems like the Lorenz system operate under (with the correct choice of parameters) as an attractor that exhibits sensitive dependence on initial conditions. We can see in Figure(1.5) how taking two nearby points in the Lorenz system causes them to rapidly diverge over time.

We're interested in being able to build Data Driven Forecasting models that can reconstruct dynamical systems. It's important that we study the ability of our tools to handle chaotic dynamical systems when they come up in our research. To understand chaotic systems better and to know if the dynamical system we are studying is actually chaotic, we look to the Lyapunov Exponent.

1.1.4 Lyapunov Exponent

An important quality of the dynamical systems we study are the stability metrics of them and their response to perturbations along their state space. We've addressed the topic of chaotic systems and how they are highly unstable and even slight perturbations

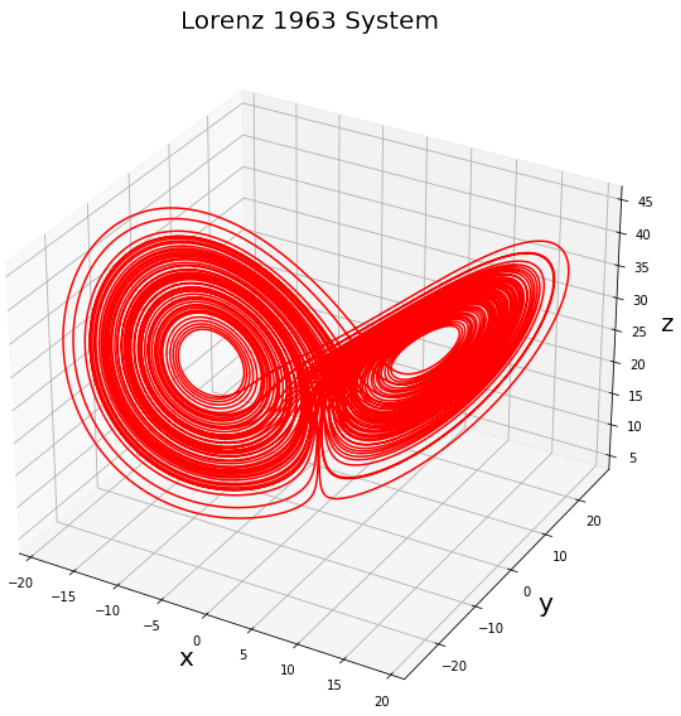

Figure 1.4. This 3D graph of the Lorenz63 system shows how the path of the system remains confined to some volume in space, but never intersects with itself, never repeating its path. There are a few properties of chaotic systems that dictate the behavior shown here. Nearby points diverge away exponentially quickly, the state vector never returns to the same point as the system will move through its state space infinitely while at the same time being confined to some set volume in continuous space.

result in wildly different outcomes, we would like to describe a way we can identify and quantify the stability (or lack thereof) of a dynamical system. To do this we begin with observing a dynamical system transition from $\boldsymbol{x}$(n) to $\boldsymbol{x}$(n+1) = $\boldsymbol{F}(\boldsymbol{x}(n))$ and how it responds to perturbation $\Delta(n)$ (it is important to note that $\boldsymbol{F}$ in this context is not our system of differential equations, but the update rule for such a system). The linearized dynamics reveal the nature of the system's stability:

$$\boldsymbol{x}(n+1) + \Delta(n+1) = \boldsymbol{F}(\boldsymbol{x}(n) + \Delta(n)) \approx \boldsymbol{F}(\boldsymbol{x}(n)) + \boldsymbol{T}(\boldsymbol{x}(n)) \cdot \Delta(n) \tag{1.14}$$

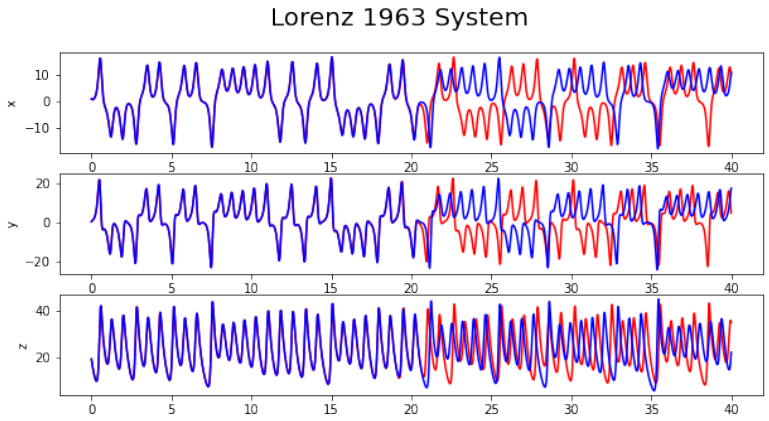

Figure 1.5. These graphs depict how two nearby trajectories with initial conditions [1,0.2,19](red) and [1.00000001,0.2,19](blue) can start so close together but rapidly diverge apart given some time. This graph aims to depict one of the key features of chaotic systems that we've previously mentioned, and that is that nearby points diverge away from each other exponentially quickly at a rate determined by the largest positive lyapunov exponent (See the section on Lyapunov exponents for more discussion on this topic).

For sufficiently small perturbations:

$$\Delta(n+1) = \boldsymbol{T}(\boldsymbol{x}(n)) \cdot \Delta(n) \tag{1.15}$$

$\boldsymbol{T}$ is the Jacobian matrix of our system defined as:

$$\boldsymbol{T}(\boldsymbol{y})_{ab} = \frac{\partial F_a(\boldsymbol{y})}{\partial x_b} \tag{1.16}$$

Where $\boldsymbol{y}$ is an arbitrary vector. Now let's consider what happens to the separation distance Δ after multiple iterations, N, of the update rule, $\boldsymbol{F}$:

$$\Delta(n+N) = \boldsymbol{T}(\boldsymbol{x}(n+N-1)) \cdot \boldsymbol{T}(\boldsymbol{x}(n+N-2)) \cdots \boldsymbol{T}(\boldsymbol{x}(n)) \cdot \Delta(n) \tag{1.17}$$

$$\Delta(n+N) = \boldsymbol{T}^N(\boldsymbol{x}(n)) \cdot \Delta(n) \tag{1.18}$$

Where $\boldsymbol{T}^N$ is the composition of N Jacobians. Now the way to interpret the meaning of this term, $\boldsymbol{T}^N \cdot \Delta(n)$ is in the eigenbasis of the dynamical systems state space, then depending on the eigenvalues of the $\boldsymbol{T}^N$, our separation distance will either decrease (indicating stability) or increase (indicating instability and chaos). More specifically, the eigenvalues of $\boldsymbol{T}^N$ behave as $e^{N\lambda}$ where if lambda is greater than 0 indicates instability and chaos.

The Russian mathematician Oseledec invented a theorem known as the multiplicative ergodic theorem to calculate the lyapunov exponents [12]. This theorem states that if we form the Oseledec matrix:

$$\boldsymbol{OSL}(\boldsymbol{y}, N) = ([\boldsymbol{T}^N(\boldsymbol{y})]^T \cdot \boldsymbol{T}^N(\boldsymbol{y})) \tag{1.19}$$

then as N goes toward infinity, the log of the eigenvalues of the Oseledec matrix divided by twice the total time, 2N, become the global lyapunov exponents, $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_D$. It is important to note that the reason for requiring that N go to infinity is that the Oseledec matrix, which is composed of Jacobians, will vary with changing location in state space, $\boldsymbol{y}$, but by allowing a large N many iterations, the Oseledec matrix will fully explore the state space and give a consistent, $\boldsymbol{y}$ independent, result for the lyapunov exponents.

The lyapunov exponents, $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_D$, can be thought of as an expansion and shrinkage of the various dimensions of the state space along the eigenvectors which are spatially dependent (otherwise our result would diverge off to infinity in the direction of greatest λ). Should any of the λ be positive, that is proof of the existence of chaos. For continuous time dynamical systems, one of the lyapunov exponents must be zero, and that the sum of lyapunov exponents must be less than or equal to zero, $\sum_i^D \lambda_i \leq 0$. In Hamiltonian systems the spectrum of lyapunov exponents is symmetric about zero, and thusly their sum is zero, indicating their state space volume is conserved.

The lyapunov exponents characterize and describe the nature of our dynamical systems. While we use the Oseledec matrix to calculate lyapunov exponents in this

dissertation, the practical use of it is more complicated than indicated here and is left for an extended discussion in the appendix. [13]

1.1.5 Fractal Dimension

In our study of dynamical systems we seek out invariants of the motion to better understand the dynamics we study and to characterize their behavior, one such invariant of the motion are the global lyapunov exponents we just discussed, another is the fractal dimension. Dynamical systems follow trajectories according to their dynamical equations that converge to the attractor (the set of states upon which the system converges to over time); for chaotic dynamical systems, they converge to what is known as the strange attractor, one that the trajectories of the system converge to but follows the rules of chaotic systems. These rules are that close points diverge away from each other exponentially fast (in the short term), the strange attractor is aperiodic never returning to the same spot, but the trajectories are contained to a set volume in phase space. The shape and geometry of this attractor in phase space is significant, as it can be characterized by the dynamical system's fractal dimension.

To inspire the idea of the fractal dimension, we take the example of some density function f(x) that measures the fractional number of points in a sphere of radius r in comparison with the total number of recorded data points for small r. In D dimensions, the volume of this sphere would scale as r^D, extending this to our function, f(x), we can measure how the density of our data scales with volume (up to a limiting point as we work with a finite amount of data). We can then describe our function f(x) as:

$$f(x) \approx r^{d(X)} \tag{1.20}$$

where X is the data set in question and d would be the fractal dimension describing how the quantity of data scales with radius r (for small r, much smaller than the size of the

attractor, but not so small that it disregards the finite data and captures no data points in its sphere). Note that the fractal dimension could be a non integer value.

What makes the fractal dimension significant to the study of a dynamical system is that it defines the effective degrees of freedom expressed in the attractor, as opposed to the nominal degrees of freedom from the original system's dimension. It quantifies the complexity of the attractor. While we use it as one more way to characterize the attractors we study (as we will show later, it will be used to characterize our results in comparison to the original attractor DDF has to learn), it is not known what set of invariants constitutes a complete set to fully define an attractor. In the appendix, we'll describe one of the methods we prefer to use to calculate the fractal dimension (of which there are many).[13, 14]

1.2 Methods of Studying Dynamical Systems

In the study of Dynamical Systems, there have been a multitude of machine learning tools that try to work with models and data sets to extract knowledge of the system through modelling or to predict the future behavior of the system. There are two well known methods of approaching this problem of piecing knowledge of dynamical systems. The first is Data Assimilation methods that try to find a suitable selection of model dependent parameters that will allow the known model to be forecasted accurately from known initial conditions. The second method are data driven methods that use observed data to construct approximate models to forecast dynamical systems, these methods include Reservoir Computing and our own invention Data Driven Forecasting.

1.2.1 Data Assimilation

Dynamical models of a system can be described by a set of first order ordinary differential equations whose time derived state variables are equal to functions of the current state and of parameters, p. These p parameters are not necessarily constant

and defined physical properties; for example, in the context of neuroscience, a neuron's parameters are specific to the neuron that is measured. The neurons are different from organism to organism because of their different genetic make up and upbringing. The neurons are different from different parts of the brain, even within the same part of the brain there are differences from neighboring neurons. When studying fluid dynamics of oceans there are unknown parameters of quantities such as viscous and drag forces that vary depending on the local region being studied; different temperatures, salines, and pollution levels can all have an effect on that parameter which, if unaccounted for, would derail the forecast.

$$\frac{dx_i(t)}{dt} = f_i(\boldsymbol{x}, \boldsymbol{p}) \qquad\qquad i = 1, ..., D \tag{1.21}$$

The goal of data assimilation [15][16] is to take a known set of differential equations, initial conditions, and recorded noisy data sets to estimate the unknown parameters p. One example of a data assimilation method is the Precision Annealing Hamiltonian Monte Carlo technique (PAHMC)[16]. PAHMC is a Bayesian method that receives F observed dimensions from a D dimensional system where $F \leq D$ and tries to answer the question, what is the most likely set of unobserved data and parameters p for the given observed dimensions. Through a process of mathematical operations, the solution to this Bayesian problem is through the minimization of a term known as the Action:

$$\begin{aligned} A(\boldsymbol{x}, \boldsymbol{p}) = &\sum_{k=0}^{F} \sum_{l=1}^{L} R_m/(2F+2)[x_l(t_k) - y_l(t_k)]^2 \\ &+ \sum_{m=0}^{F-1} \sum_{a=1}^{D} R_f/(2F)[x_a(t_{m+1}) - f_a(\boldsymbol{x}(t_m), \boldsymbol{p})]^2 \end{aligned} \tag{1.22}$$

This is the action we must minimize where $\boldsymbol{y}$ is the observed data, $\boldsymbol{x}$ is the estimated data, F is the full length of the observed data set, t_k is the k_{th} time in the data set. Minimizing the action is a very difficult task, as the action is incredibly nonlinear and high dimensional. To minimize the action the precision annealing method is performed, this is done by first

setting R_f equal to zero, this sets the 2nd term in the action, the model error to zero, the minimization is then only performed on the first term of the action, the measurement error, which has a trivial solution $x_l(t_k) = y_l(t_k)$ (all unobserved dimensions of x are set to randomness in a predetermined window). Then the precision annealing process starts, by gently turning on the model error, the global minimum that we were once at has now shifted, the Hamiltonian Monte Carlo method of sampling the local space is performed to find where the global minimum has shifted to. Repeat this process enough times to follow the global minimum as the model error becomes appreciable; however, this does rely on the hope that the global minimum you follow stays the global minimum.

If this process is performed successfully, we will have a good estimation of a model's parameters, which gives us knowledge of the system's intricacies as well as a method of predicting its future behavior. The draw back to this method is that it can be an extremely difficult problem to solve as it deals with very high nonlinearity, high dimensions, takes a very long computation times (on the order of hours, sometimes days), and relies on the model to be accurate. This leads us to the next tools of focus, reservoir computing, which is a data driven model that assumes no knowledge of a model.

1.2.2 Reservoir Computers

A reservoir computer [17][18][19][20] is a type of simplified recurrent neural network architecture that takes input data, scales it up to a large dimension, mixes it in a recursive "reservoir", and finally outputs it back into the lower, original, dimension. There's a large body of work that has gone into reservoir computers and their effectiveness in studying nonlinear dynamical systems and chaotic dynamical systems.

The mathematical form of the reservoir can be depicted in the following way. The RC takes in input $W_{in}\boldsymbol{u}(t)$ which mixes with the recurrent reservoir, the reservoir has its own state, $\boldsymbol{r}(t)$, and that state will evolve in time with the input vector $\boldsymbol{u}(t)$ in time and is the output of the reservoir. This output is then matrix multiplied (thus scaling

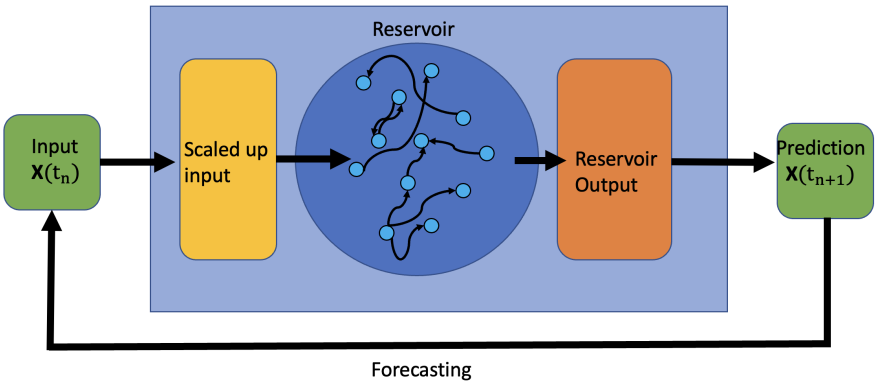

Figure 1.6. This is a graphical interpretation of what a reservoir computer is doing, how it manipulates input data into getting an output which can then be sent back to the input to generate an arbitrarily long forecast. As shown above, the reservoir computer takes an input from our data set and puts it through a scaling process; typically this process will do something like create a much larger matrix of 0's and 1's and perform a matrix multiplication to propagate the 1's and 0's. From here another matrix multiplication is performed with the adjacent matrix to mix and match all the components of the scaled up input, this operation increases the nonlinearity and complexity of the input data before being outputted out of the reservoir. The final matrix multiplication is performed to scale the reservoir data to the predicted output; this final matrix is what is trained linearly with Ridge Regression, as the reservoir itself is too complex to train easily.

it back down to the original dimension of $\boldsymbol{u}(t)$ by the W_{out} matrix; this W_{out} matrix is the only matrix that is trained as W_{in} and the recurrence of the reservoir is randomly generated. The benefit of this is that training a recurrent network would be a terribly difficult process, but by only having to train W_{out} it allows us to use the very simple and easy Ridge Regression to fit W_{out} to the training data. The differential form of the reservoir looks the following way:

$$\frac{d\boldsymbol{r}(t)}{dt} = \gamma[-\boldsymbol{r}(t) + \tanh(\boldsymbol{A}\boldsymbol{r}(t)) + W_{in}\boldsymbol{u}(t) + \sigma_b I] \tag{1.23}$$

This is the hyperbolic tangent form of the reservoir that is commonly used, but other options exist. The $\boldsymbol{A}$ matrix is the adjacency matrix and is responsible for the recurrence in the network. γ and σ are hyperparameters of the model. When the RC receives an

input data, it uses the above equation to update the state of the reservoir by 1 step in time. Then the forecast of the RC is $W_{out}\boldsymbol{r}(t+1) = \boldsymbol{u}(t+1)$.

1.2.3 Data Driven Forecasting

Like Reservoir Computing, Data Driven Forecasting (DDF) is a function approximation tool that reconstructs an approximation to the dynamical system to forecast with, however DDF is a more physically inspired machine learning model that utilizes interpolation tools with a standard update rule to model dynamical systems. To contrast Reservoir Computing's black box behavior in its reservoir, DDF tries to take advantage of the characteristics of the dynamical system (if they are known). A common and powerful choice is to incorporate forcing terms into the interpolation and update rule, as will be shown in both Neurodynamics and Fluid Dynamics problems we solve later. Without further delay we will now dive into the bulk of this dissertation which will be our exploration into what DDF is, how it works, why it's a useful and powerful tool in nonlinear dynamics, and the various applications we've had success with.

Chapter 2

Data Driven Forecasting (DDF)

In the study of dynamical systems it is not always the case that we know the exact form of the dynamical equations or the parameters within them, we argue that there are many cases where a model built from data alone serves better than to use an imperfect model or imperfect parameter estimation (or both). But rather than favor the black box behavior of reservoir computers, we argue that a data driven model (DDF) that utilizes the known form of dynamical equations and allowing an interpolation model to capture the rest of the unknown behavior to be a more insightful model than RC and free of model errors than data assimilation. In this section, we'll go through the inspiration for how DDF was created and the mechanics of how it works.

2.1 What DDF is and how it works

As DDF is a data driven model reconstructor, it naturally starts by obtaining some, possibly noisy, data from either an observation in a laboratory or simulated data generated on a computer. With this data, we seek to piece together the dynamical equations in an approximate form to predict the future behavior of the system. Just as we did in the introduction, we will describe our system of differential equations, but we must convert it from its continuous form to a discrete one as our data can only exist in discrete form. The

process goes as follows for a time step of length h:

$$\frac{dx_i(t)}{dt} = f_i(\boldsymbol{x}(t), p) \qquad\qquad i = 1, ..., D \tag{2.1}$$

$$x_i(n+1) = x_i(n) + h * f_i(\boldsymbol{x}(n), p) \tag{2.2}$$

Using the Euler method, we've provided a very intuitive form for us to view our differential function f_i as. Our goal is to reconstruct this illusive form f_i, the time derivative vector space, using the data. To do this, we must choose a functional representation of f; in this section, we will describe the Taylor Series Representation (TSR) of DDF. We will expand $f_i(\boldsymbol{x}(t), p)$ into its TSR:

$$f_i(\boldsymbol{x}(t), p) = K_i + x_A M_{iA} + x_A x_B J_{iAB} + \mathrm{O}(x^3) \tag{2.3}$$

Note that all repeat indices are summed over for all observed dimensions. This specifically is the Maclaurin Series of the Taylor expansion for the i_{th} dimension. Our goal will be to take the constants in the K, M, J, etc. matrices and to fit them to the data to create our reconstructed state space. Note that we will have to make a choice as to how many orders of the Taylor Series we must keep, usually this would be 2nd or 3rd order and this choice is heavily influenced by the data set; the choice is made clear when working with dynamics that are linear and the Taylor Series expansion is finite because we would simply match the Taylor order to that of the known dynamics, otherwise, some testing must be done if the Taylor Series is infinite for a system under study. This touches on a very important detail that we will come back to multiple times throughout this dissertation and that is that the choice of interpolation should match the characteristics of the dynamical equations of the system in study, this has shown to produce the best results time and time again. Now let's plug our TSR into the discrete update rule to show how we can create a

cost function to fit our TSR to the data:

$$x_i(n+1) = x_i(n) + h[K_i + x_A M_{iA} + x_A x_B J_{iAB} + \mathrm{O}(x^3)] \tag{2.4}$$

$$\begin{aligned}\text{Minimize} \sum_{n=0}^{N} [(x_i(n+1) - x_i(n)) - [K_i + x_A(n) M_{iA} \\ + x_A(n) x_B(n) J_{iAB} + \mathrm{O}(x^3)]]^2\end{aligned} \tag{2.5}$$

Where N+1 is the number of data points in the training window and h has been absorbed into the training coefficients. This minimization is for the i^{th} dimension, the minimization will need to be done D times for a D dimensional system (assuming all D dimensions are observed, we will discuss how to handle observations of a limited number of dimensions from the whole system in the section on Takken's Embedding Theorem and how to reconstruct the state space). Now to perform this minimization we will use Tikhonov-Miller Regularization, also known as Ridge Regression, but first let's manipulate the form of the minimization to something that looks more appropriate. For the case of Taylor Order 2 (it is trivial to scale this up, but it is conducive to show this simpler case) the minimization goes as follows, first define a couple new terms:

$$y_i(n) = x_i(n+1) - x_i(n) \tag{2.6}$$

$$\boldsymbol{Y}_i = [y_i(1), y_i(2), ..., y_i(N)] \tag{2.7}$$

$$\begin{aligned}\boldsymbol{X}(n) = [1, x_1(n), x_2(n), .., x_D(n), x_1(n)x_1(n), x_1(n)x_2(n), \\ ..., x_D(n)x_D(n)]^T\end{aligned} \tag{2.8}$$

$$\hat{\boldsymbol{X}} = [\boldsymbol{X}(1), \boldsymbol{X}(2), ..., \boldsymbol{X}(N)] \tag{2.9}$$

$$\boldsymbol{P_i} = [K_i, M_{i1}, M_{i2}, ..., M_{iD}, J_{i11}, J_{i12}, ..., J_{iDD}]^T \tag{2.10}$$

$$\text{Minimize} \sum_{n=1}^{N} [y(n) - \boldsymbol{P}_i * \boldsymbol{X}(n)]^2 \tag{2.11}$$

Now that the form of the minimization is of the traditional linear form, [y-mx], we perform the Ridge Regression step:

$$\boldsymbol{P}_i = \boldsymbol{Y}_i \hat{\boldsymbol{X}}^T (\hat{\boldsymbol{X}} \hat{\boldsymbol{X}}^T)^{-1} \tag{2.12}$$

It is computationally convenient to note that the $\hat{\boldsymbol{X}}$ matrix is the same for all dimensions and only needs to be constructed once. Ridge Regression is a powerful tool for solving the minimization, but there are two problems with it. The first is that solving this requires $\hat{\boldsymbol{X}} \hat{\boldsymbol{X}}^T$ to be an invertible matrix, which it isn't always is. It also runs into the problem of over fitting, where the fit follows the behavior of the training data set "too well" and doesn't generalize well. The solution to both these problems is actually very easy to implement and only requires us to make a small change to the cost function to include a regularization term:

$$\text{Minimize} \sum_{n=1}^{N} [y(n) - \boldsymbol{P}_i * \boldsymbol{X}(n)]^2 + \beta ||\boldsymbol{P}_i||^2 \tag{2.13}$$

This extra term on the right-hand side is the regularization term and includes a hyperparameter β that we can vary to get the best fitting. Intuitively the idea of this regularization term is to prevent the fitted parameters from becoming too large, if the fitting tries to overemphasize one term in the set of fitting parameters, then it may result in over fitting. However, by the inclusion of this regularization term, we attach a cost (scaled by β) to the magnitude of $\boldsymbol{P}_i$ to dissuade the Ridge Regression algorithm from opting for large values of $\boldsymbol{P}_i$. As β is increased, it will further punish the minimization for choosing larger fitting parameters. It is also very fortunate that this minimization has an exact solution, so we don't have to perform a painful minimization search in a highly nonlinear space. This solution to the regularized Ridge Regression is:

$$\boldsymbol{P}_i = \boldsymbol{Y}_i \hat{\boldsymbol{X}}^T (\hat{\boldsymbol{X}} \hat{\boldsymbol{X}}^T + \beta \boldsymbol{I})^{-1} \tag{2.14}$$

With the training complete the fitting parameters can be plugged into the update rule that can now, given an initial condition, run indefinitely to forecast the behavior of the system with the output of DDF becoming the input for the next DDF update.

2.2 Taylor Series Results DDF

We now want to show the strengths and weaknesses of formulating the interpolation representation in DDF as a Taylor Expansion. For polynomial models, its strength shines, but for non-polynomial models, it struggles greatly to capture the non-polynomial behavior.

2.2.1 Lorenz 1963 Example

The Lorenz 1963 system is our classic example of a chaotic system, that also happens to be a polynomial system. For a Taylor Series Representation of first and second order polynomials, and a careful choice of beta, our regularization parameter, we are able to correctly forecast the behavior of the Lorenz system for 24 Lyapunov Times in Figure(2.1); the Lyapunov Time is a metric for scaling the arbitrary time of a chaotic dynamical system by its largest lyapunov exponent to show a forecast resiliency to the divergent nature of the chaotic system.

2.2.2 Colpitts Oscillator Example

We wish to test the Taylor Series DDF against a non-polynomial model to showcase its difficulties and why we will ultimately choose a different interpolator. The Colpitts Oscillator is a non-polynomial system, one whose Taylor Series expansion is infinite, something we couldn't hope to capture with our finite TSR method. Additionally, attempts to include very high polynomial orders resulted in severe instabilities, as small numerical error to higher power blows up very fast. Figure(2.2) shows the best result extensive grid sweeping could provide.

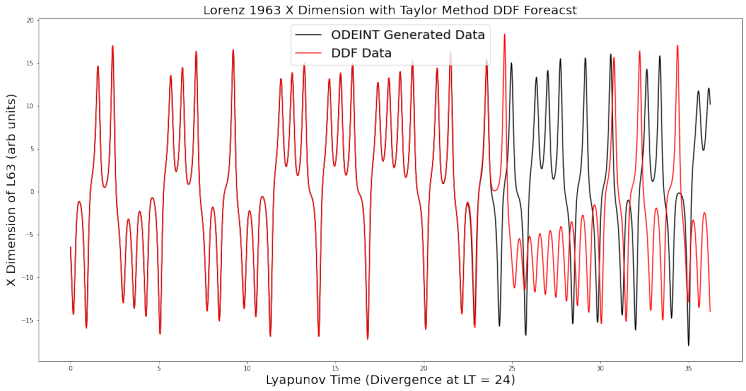

Figure 2.1. With careful grid sweeping we're able to find a small regime where the forecast performed best, from there we used Differential Evolution (see the appendix for more discussion on this method) to pinpoint the exact best beta for forecasting accurately for as many lyapunov times as we could. To be clear, this beta value is over-fitted to this data set, but this example shows the potential for the Taylor Method of DDF. DDF Trained for 10,000 data points, h = 0.002, and $\beta = 9.25874871890625$.

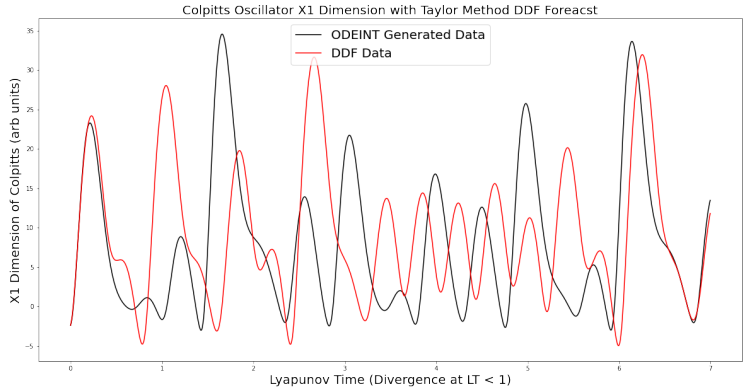

Figure 2.2. This poor result comes from trying to fit the polynomial TSR DDF method on non-polynomial systems with infinitely long TSRs. Training lasted 10,000 data points, and h = 0.02, beta = $10^{-15/2}$. As we will discuss in a future section, a more robust and flexible method must be used to handle greater nonlinearities that will show up in more complex and interesting data sets. The radial basis function is what we will discuss as our solution to this problem.

2.2.3 Parameter Fitting of Polynomial Models

The success of the TSR method on polynomial systems merits a quick discussion on the accuracy of the fitting of the coefficients. Choosing the order of the TSR to match the

finite order of a polynomial system, in the case of the Lorenz 1963 system, has shown to provide an accurate fitting to the actual Taylor Series (or more specifically, the Maclaurin Series) coefficients for sufficiently long training. Figure(2.3) shows the fitted coefficients for the x equation of the Lorenz 1963 system as a function of training time.

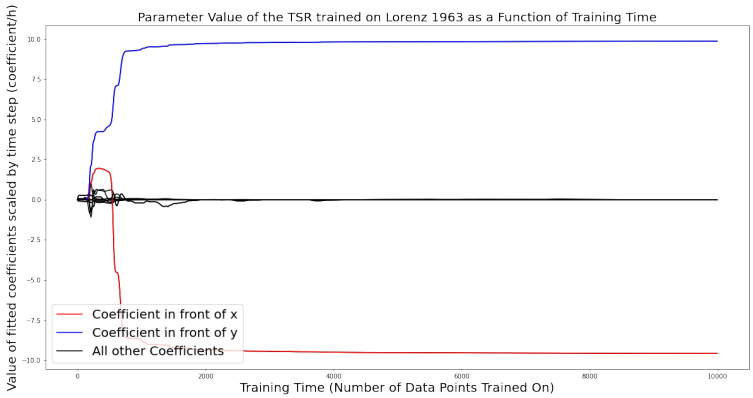

Figure 2.3. This is a graph of the DDF coefficients of the X equation for the Lorenz 1963 system. With sufficient training, the coefficients converge on the true value for the polynomial equation. There have also been tests with the Lorenz 1996 system (a polynomial system) and have shown to converge to correct coefficients as well; while this isn't a rigorous proof of DDF's ability to correctly obtain coefficients for polynomial systems, it does give intuition as to what TSR DDF is doing and is not simply a black box with seemingly random coefficients that just happen to work. The foundation of DDF is rooted in trying to formulate a method that isn't purely a machine learning method, but one that recreates the physical equations.

2.3 Radial Basis Functions

Alternatively, instead of using the TSR, which is less effective outside of polynomial dynamical systems, we can use Radial Basis Functions (RBF) to interpolate the time derivative state space.

2.3.1 Radial Basis Function as Tools for Interpolation

In our search for a powerful interpolator that could handle nonlinear systems, we came across the body of work for Radial Basis Function Expansions and their uses in

field of function interpolation [21][22][23]. Interpolation is the method of estimating the values of nearby points based off of the known values in local space; this is in contrast to extrapolation which is the method of estimating values of far away points outside the local space our known points occupy and is a much harder problem that DDF doesn't deal with. This is why it is important when analyzing a data set that the set of data be flush with data points in all relevant areas in space. Radial Basis Functions are no different, they operate by placing objects in space called "centers" and they estimate the value of our current location in space by comparing it with the distance of our current point to all the centers. In practice, these centers will be chosen to be in locations based on the training data set and the radial distance from the center to a location in question will go into the Radial Basis Function, of which there are a large selection. This is the basic concept for the RBF that was invented by Broomhead and Lowe in 1988.

The mathematical form of the RBF expansion is depicted by a summation over weighted RBF's whose functions are user defined and each RBF is defined by a location in space, $\boldsymbol{c}(q)$ (where q acts as a label for each individual center and will be summed over for all RBF's), that will be compared with the state's current location in space, $\boldsymbol{x}$:

$$f(\boldsymbol{x}) = \sum_{q=1}^{N_c} \omega_q \phi(||\boldsymbol{x} - \boldsymbol{c}(q)||) \tag{2.15}$$

The ω_q is the training coefficient that will be fitted to the data similarly to the TSR method, the ϕ is the RBF, and N_c is the number of centers. Now with this representation for the unknown function we can create a new update rule as we did before with the TSR:

$$x_i(n+1) = x_i(n) + h \sum_{q=1}^{N_c} \omega_{iq} \phi(||\boldsymbol{x} - \boldsymbol{c}(q)||)(n) \tag{2.16}$$

We'll take this form for our new representation in trying to model the flow of the data. The unknown function $f_i(\boldsymbol{x}, n)$ is represented by an RBF Expansion. We'll create a similar

cost function to minimize as we did in the TSR; fortunately, the RBF representation operates in a linear function space and lends itself to being solved by Ridge Regression. If we weren't able to use that, we would have to rely on simulated annealing techniques, which can be quite challenging for the type of large cost functions we are dealing with in DDF. For a D dimensional system with N training data, the cost function we seek to minimize takes the following form (note that there will be D of these minimizations to perform, 1 for each dimension):

$$\text{Minimize} \sum_{n=1}^{N} [(x_i(n+1) - x_i(n)) - \sum_{q=1}^{N_c} [\omega_{iq} \phi(||\boldsymbol{x} - \boldsymbol{c}(q)||)]]^2 \tag{2.17}$$

Where h has been absorbed into the training coefficients. Just like before, we will rewrite this equation into something simpler that more easily fits into the Ridge Regression formula.

$$y(n) = x_i(n+1) - x_i(n) \tag{2.18}$$

$$\boldsymbol{Y}_i = [y_i(1), y_i(2), ..., y_i(N)] \tag{2.19}$$

$$\Omega = [\omega_1, \omega_2, \omega_3, ...\omega_{N_c}] \tag{2.20}$$

$$\Phi(n) = [\phi(||\boldsymbol{x}(n) - \boldsymbol{c}(1)||), ..., \phi(||\boldsymbol{x}(n) - \boldsymbol{c}(N_c)||)]^T \tag{2.21}$$

$$\boldsymbol{\Phi}(n) = [\Phi(1), \Phi(2), ..., \Phi(N)] \tag{2.22}$$

$$\text{Minimize} \sum_{n=1}^{N} [y(n) - \Omega * \Phi(n)]^2 \tag{2.23}$$

Now we perform Ridge Regression:

$$\Omega = \boldsymbol{Y}\boldsymbol{\Phi}^T(\boldsymbol{\Phi}\boldsymbol{\Phi}^T)^{-1} \tag{2.24}$$

Where $\boldsymbol{Y}$ is a 1 by N dimensional matrix of the y(n) values and $\boldsymbol{\Phi}$ is an N_c by N

dimensional matrix of the $\phi(||x(n) - c(q)||)$ values. We can also include an additional term to regulate the size of the fitted ω values to prevent over fitting using the same method described in the TSR section:

$$\Omega = \boldsymbol{Y}\boldsymbol{\Phi}^T(\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \beta * I)^{-1} \tag{2.25}$$

Now we have our method of training for RBF's, we need to choose our Radial Basis Function and how we will choose centers for it from the data.

2.3.2 Choosing a Function

The RBF's job in DDF is to capture the behavior of the vector space that isn't already modelled in the DDF update rule (we'll see later how to incorporate extra terms into the DDF update rule outside pure RBF's). In doing this, we need to choose a function that the RBF will use to capture the behavior of our dynamical system. There are many choices of radial basis function, and much research has gone into finding better radial basis functions; here are some examples of common radial basis functions[23]:

$$\begin{aligned}
&\phi(r) = e^{\frac{-r^2}{R}}, \text{Gaussian} \\
&\phi(r) = (\sigma^2 + r^2)^{\alpha}, 0 < \alpha < 1, \text{MultiQuadric} \\
&\phi(r) = \frac{1}{(\sigma^2 + r^2)^{\alpha}}, \alpha > 0, \text{Inverse MultiQuadric} \\
&\phi(r) = r, \text{Linear} \\
&\phi(r) = r^2 ln(r), \text{Thin-Plate Spline} \\
&\phi(r) = \frac{1}{1 + e^{\frac{r}{\sigma^2} - \theta}}, \text{logistic function}
\end{aligned} \tag{2.26}$$

The r in these equations is equal to the radius term, $||x(n) - c(q)||$, used previously. Among these RBF options, we have performed tests with the Gaussian and MultiQuadric forms [24][25]. As we will show in our results section, there will be data sets that different

Radial Basis Functions are better suited for. It is useful to have a handful of options for Radial Basis Functions to keep ready when testing challenging data sets. In the sections on Neurodynamics and Fluid Dynamics in this dissertation, only Gaussian RBF's were used, as they were found to be a very good general use RBF that apply well to a wide range of problems. The Gaussian RBF prioritizes the influence of nearby centers and diminishes the input from centers that are far away, which is an attractive feature about it. We've used this knowledge to intuit how to place our centers and this is to saturate all regions of space, particularly regions of large second derivatives, to properly capture the behavior of the system.

2.4 Choosing a Center Scheme

An important question when using the RBF's is how many centers do we need and where do we put them. This answer largely be determined by the size and shape of the training data set. A simple but effective solution would be to take every data point in the data set and make that location in state space a center; this would work, but it would also make the DDF program run horribly slow and would be unnecessarily large. In our experience, the number of centers actually doesn't need to be very large, something on the order of 100's will suffice for a lot of jobs. There is also a tendency to get diminishing returns on using more and more centers beyond this limit. We want to get the most use out of these centers as we can, and to do this we will use a K-means clustering algorithm to select our center location.[23][26]

To understand why we use the K-means clustering algorithm, let's first describe what it's doing. The K-means clustering algorithm takes a given data set and creates K clusters that then assigns each data point in our data set to one of the K clusters (the clusters need not be the same size). Each cluster is given a cost value that is the sum of the distances from each data point, $\boldsymbol{x}$, within that cluster to a defined center, $\boldsymbol{c}$. The goal

is to minimize a cost function that is created by averaging the cost of each cluster:

$$\text{Minimize} \qquad E(c_1, ..., c_K) = \frac{1}{N} \sum_{k=1}^{K} \sum_{x_n \epsilon C_k} ||\boldsymbol{x} - \boldsymbol{c}_k||^2 \tag{2.27}$$

To minimize this cost function, we will shift around all the centers within each cluster and data points will move around to the most appropriate cluster to minimize this cost function. The ideal center has been proven to be the average of each data point's location within the cluster, so the only challenge is choosing the best grouping of data points. Intuitively, we can imagine how this will result in data being both smoothly distributed across regions of dense centers, but also giving emphasis to sparse areas in space where more attention might be needed as there is lacking data there. This is the ideal spread of centers in RBF's, so we choose the centers in the K-means clustering algorithm to be the centers in the RBF (which unfortunately share the same name). There are a multitude of solutions to this minimization, since we use the sci-kit learn library for their K-means algorithm, we use the Generalized Lloyd algorithm. Using the sci-kit learn library, we're able to make Figure(2.4) to exemplify how the centers would come out for a training data set from the Lorenz 1963 system.

Another prominent method in the RBF community is to use the 2nd derivative as an indicator for where to densely and sparsely put one's centers [27]. While we don't use this method, it has gained traction within the RBF community and is worth mentioning.

2.4.1 RBF Plus Polynomial

Now we begin our discussion on one of the simplest adaptations we can make to the RBF representation of DDF for a dynamical system that has additive polynomial terms in its dynamical equations[21]. This is a useful tool in our toolkit and is well worth going over as it shows up commonly in forcing terms as they are often additive in the dynamical equations (in our cases linear too); it will also act as a showcase for how to seamlessly

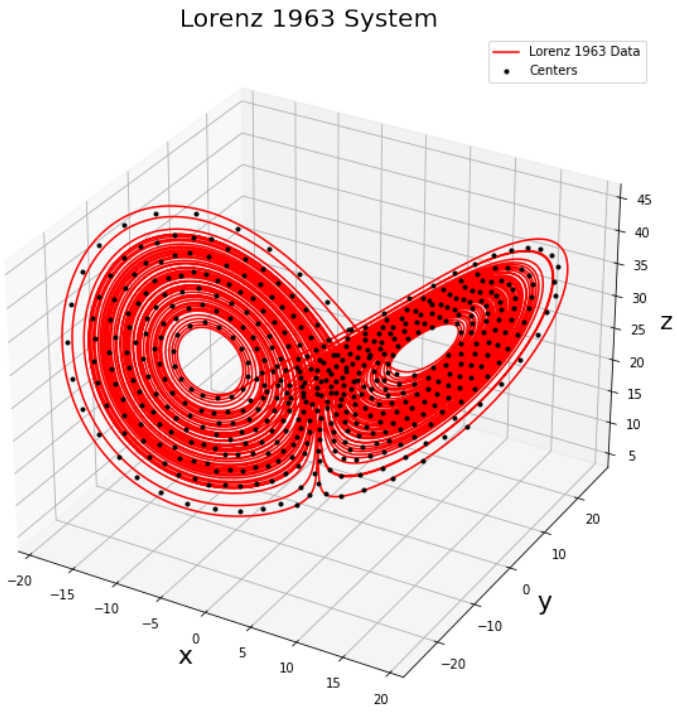

Figure 2.4. The Lorenz 1963 System with the centers we calculated from the K-means algorithm from sci-kit learn. Take note of how there is a smooth distribution of centers, with a tendency for centers to bunch up a bit more in regions where the lines are densely packed in. We decided on using the K-means algorithm as a way to easily and cleverly choose centers in comparison with an older and more brute force method of putting a center every n^{th} data point in the time series data. The algorithm itself is quick to run and can be found in sci-kit learns python library.

make additions to the DDF interpolation of the dynamical equation's vector space to match as many known characteristics as we can. There is a powerful rule of thumb we've learned in studying DDF, the closer we can match our interpolation to the known features in the dynamical models, the more effective the forecast will be. The intuitive argument for why this is the case is that the more complexity there is in a dynamical equation that we can strip away, the less there will be for the RBF's to have to interpolate; there will be limits to this as highly nonlinear models won't lend themselves to much adaptation, and

we'll have to fall back on RBF's to interpolate their behavior. For simpler cases, however, we can take advantage of this rule of thumb, and one of the most common ones we've come across is the additive polynomial. We let p_j, j = 1,2,...,$\widehat{m}$ be a basis of polynomials and use this to define our new vector space interpolator:

$$f_i(\boldsymbol{x}) = \sum_{q=1}^{N_c} \omega_{iq} \phi(||\boldsymbol{x} - \boldsymbol{c}(q)||) + \sum_{j=1}^{\widehat{m}} \alpha_{ij} p_j(\boldsymbol{x}) \tag{2.28}$$

This new form for the vector space still maintains the linear function space needed to perform Ridge Regression. The function is still a sum of unknown parameters times some function of time, and they can be grouped together similarly to what was done prior in the training sections of this dissertation. In typical DDF fashion, we don't assume any knowledge of the parameters that would exist in front of the polynomial term and rely on the fit to the data to tell us what that coefficient, in this case α_{ij}, should be. We try to match the polynomial order to the polynomial order that exists in the dynamical equations, which is commonly first order.

This method isn't limited to polynomial, they're just common additives in the dynamical equations we've come across, but this method would work for any additive feature of a fitting coefficient times a function of time and/or state space variables.

2.5 Radial Basis Function DDF Results

The RBF approach to DDF is a more general approach to modelling dynamical systems that is applicable for nonlinear models, unlike the TSR method. What we will see is that while the RBF method doesn't achieve the same level of predictive power as the TSR method does in polynomial systems, it will still perform well, and it will perform well in non-polynomial chaotic systems. The strength of DDF comes from the flexibility of choice in its interpolation scheme to best tackle a difficult data set.

2.5.1 The Lorenz 1963 System

The Lorenz 1963 model as described in equation(1.13) was very effectively modelled by the TSR method, achieving an accurate forecast for up to 24 lyapunov times, the more general RBF method performs slightly worse in exchange for a more general interpolation scheme. Using the standard Gaussian RBF, we generate the forecast shown in Figure(2.5).

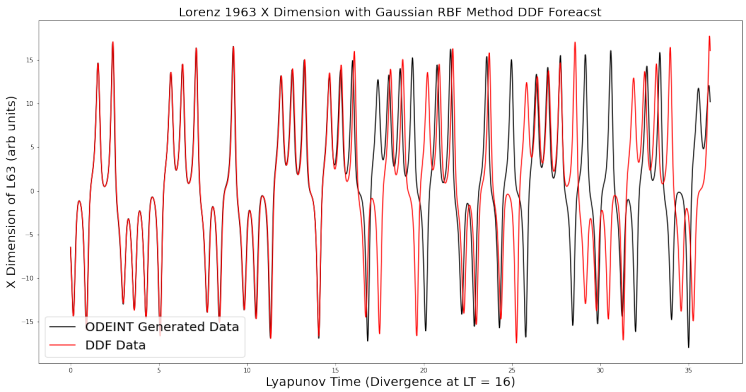

Figure 2.5. Now using the RBF method of DDF we perform a new forecast of the Lorenz 1963 system. With careful grid sweeping we're able to find a small regime where the forecast performed best, from there we used Differential Evolution to pinpoint the exact best beta for forecasting accurately for as many lyapunov times as we could. This method stays accurate for about 16 lyapunov times. DDF Trained for 10,000 data points, h = 0.002, $\beta = 0.011350091952587656$, and R = 0.010720697084284608

2.5.2 The Colpitts Oscillator

The Colpitts Oscillator is a unique problem in our testing because the performance of it is greatly improved by switching from Gaussian RBFs to MultiQuadric RBFs, this is something we have only come across with this dynamical model. We speculate that the reason for this is due to the explosive nature of the exponential in the Colpitts Equations. The Multiquadric RBF as shown in equation(2.26) has two hyperparameters that we sweep across instead of the usual 1. The Multiquadric is special in that each center's influence doesn't drop off with distance from the center as the Gaussian RBF does, rather the

farther one travels from the center, the more influence it has! Generally, we prefer the Gaussian, we take the approach that centers should have influence over their local space and that centers far away shouldn't as their far distance from the points of interest should dull their effectiveness to the nuance of far away local space. The Colpitts Oscillator and MultiQuadric combination defies this approach for reasons that we could only speculate having to do with the exponential behavior of Colpitts. Nonetheless, the Multiquadric results are superior from the Colpitts Oscillator and are shown in Figure(2.6).

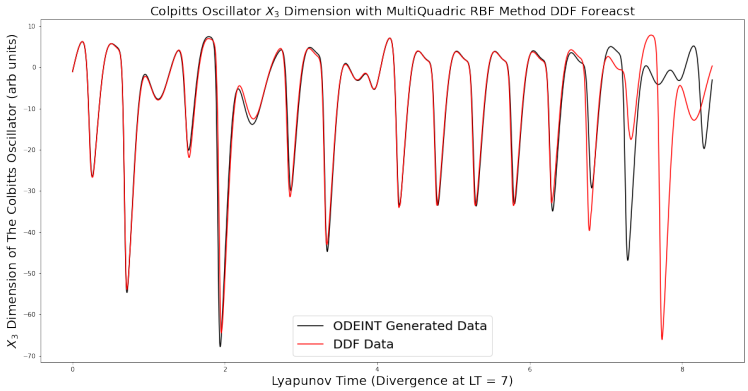

Figure 2.6. The Taylor Method struggled with the nonlinear behavior and exponential terms in the Colpitts Oscillator, as shown in Figure(2.2). The RBF method, specifically the Multiquadric RBF, handles this system much better, achieving a near 7 Lyapunov Times of accurate prediction, which we think is a good result for the nonlinear and exponential Colpitts Oscillator. We again used grid sweeping followed by differential evolution to find the best parameters. For this result we used $\beta = 0.018772874248566572$, $r^{\frac{1}{2}} = 0.013545674322853224$, and $\alpha = 0.7$

2.6 Other Interpolators

For the remainder of this paper we will only concern ourselves with the Radial Basis Function Representation of the dynamical vector space, but an important point needs to be made that the future work of DDF should not be limited to only the Radial Basis Function. DDF is not an RBF method, rather it is a method of reconstructing the dynamical model (in an approximate form) from data and utilizes interpolators to do this

(it just happens that we have found great success with the RBF, so we've focused a lot of our work on it). At it's very core, DDF cares about creating a mapping based on a data set that can take an input, $\boldsymbol{x}$, and forecast the next step based off a time series data set of (x,y) data in the form $[(\boldsymbol{x}(1), \boldsymbol{x}(2) - \boldsymbol{x}(1)), ..., (\boldsymbol{x}(N), \boldsymbol{x}(N+1) - \boldsymbol{x}(N))]$; Figure(2.7) captures the essence of this idea graphically. We stand by the RBF formulation in this paper and its nice properties, but by the construction of DDF, any interpolator can more or less solve this problem of mapping x to y.

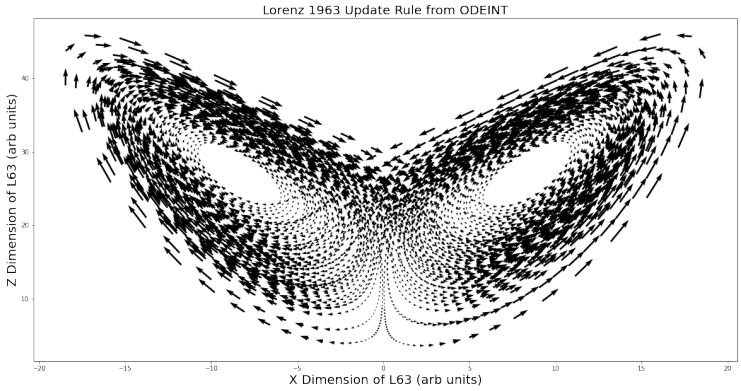

Figure 2.7. This graph of the Lorenz 1963 system illustrates the basic idea of what DDF is trying to capture. DDF is trying to capture the update rule for how the trajectory in state space should look for the entire volume on which the attractor of the dynamical system is confined to (although these updates are from ODEINT, not DDF). As part of the illustration, arrows that represent updates of larger magnitude are represented by large arrow, whereas areas of smaller updates are represented by smaller arrows. The size of these steps will be directly influenced by the time step, h, as smaller time steps will result in smaller spatial steps in state space.

Chapter 3

Takken's Embedding Theorem and Reconstructing the State Space

It is often the case that experimental observation of a system does not provide us with knowledge of all the states in the system, this is a problem because we are now lacking the information needed to model this system. Without the full array of data of each state, we couldn't even advance the dynamical equations (if they were known) forward in time with numerical methods like Runge-Kutta. The geometry of the state space and the attractor would be incomplete, but the structure of the multivariate dynamics can be unfolded using Takken's Embedding theorem.

3.1 Takken's Embedding Theorem

The problem of reconstructing the state space of limited observation (such that the observed dimensions, $\boldsymbol{y}(n)$, are fewer than the total dimension of a system, $\boldsymbol{x}(n)$), was solved by the embedding theorem attributed to Takens and Mañé[10, 28–30]. Takken's Embedding Theorem tells us that if we are able to observe a single scalar quantity $h(\bullet)$ of some vector function of the dynamical variables $\boldsymbol{g}(\boldsymbol{x}(n))$, then we can unfold the geometric structure from this set of scalar measurements $h(\boldsymbol{g}(\boldsymbol{x}(n))$. This new vector space is made out of components consisting of $h(\bullet)$ applied to higher powers of $\boldsymbol{g}(\boldsymbol{x}(n))$. This vector

would appear as:

$$\boldsymbol{s}(n) = [h(\boldsymbol{x}(n)), h(\boldsymbol{g}^{T_1}(\boldsymbol{x}(n))), ..., h(\boldsymbol{g}^{T_{D_N-1}}(\boldsymbol{x}(n)))] \tag{3.1}$$

This $\boldsymbol{s}(n)$ vector would be our new operating space, and we will perform all our operations not with the observed D_0 dimensions, but $D_0 * D_N$ dimensions. According to Takken's Embedding Theorem, we get to choose our function $\boldsymbol{g}$, so we'll choose it to transform the state to a state at a previous time. Each power of $\boldsymbol{g}^T$ will transform the state vector back an additional time, τ, such that the time delays will all be an integer τ apart. We apply the $h(\bullet)$ function to the time delayed state of the full system and outputting only the observed dimensions, our data set. Now our new state vector, $\boldsymbol{s}(n)$ takes this new form:

$$\boldsymbol{s}(n) = [\boldsymbol{y}(n), \boldsymbol{y}(n-\tau), ..., \boldsymbol{y}(n-(D_E-1)\tau] \tag{3.2}$$

We will make use of as many observations in this formulation as we can, and we will have time delayed duplicates of the D_0 observed dimensions.

The proper interpretation to take is that the time delays inform the dynamical system of what is happening outside of the observed variables and how the unobserved variables affect the observed variables. This information is critical to developing a complete view of the model dynamics.

The reconstruction of the original state space is made easy by the simple time delay transformations, we'll go through the theory of what constitutes the best choice for the number of time delays and the length of those time delays next, and after that we'll show how this looks in practice in a DDF setting.[13, 14]

3.1.1 Choosing the length of Time Delays

With the formulation set up for us to start using time delays to unfold our state space, we need to make a choice of how long the time delays will be, but the embedding theorem allows all time delays as valid reconstructors (except for those that match the precise period of a periodic signal). In theory, we can choose any time delay as long as we have an infinite amount of infinitely accurate data, unfortunately, this isn't usually, or ever, the case. Without any guidance from the embedding theorems on what choice of time delay to make, we'll make use of the rules described by Abarbanel in 1996 [13][31].

Firstly, the time delay, τ should be some integer multiple of the time step, h, of the recorded data so that the delayed time is also an observed time.

Secondly, we to choose times that are not so close together such that the original point in time and the delayed point in time have exactly no independence from one another. The usual example for this would be in the case of studying the weather, if we are sampling the temperature of the atmosphere with time delays on the order of milliseconds, there isn't much appreciable difference in the weather until minutes have gone by, and we have over sampled the system.

The third and final rule is that we don't want to time delay too long, especially in chaotic systems. This is because after a significant amount of time has passed in the system the relationship between $\boldsymbol{y}(n)$ and $\boldsymbol{y}(n-\tau)$ is completely random and independent of each other, this offers us no insight into how the system evolves with time. Our goal now is to choose a criterion that allows us to choose a time delay that follows these rules, where the time delay is large enough such that an appreciable amount of change has happened in the system, but not too much that the state of the system is independent of its time delayed state. Our choice for this criteria comes in the form of the average mutual information:

$$I_{AB} = \sum_{a_i,b_i} P_{AB}(a_i,b_i) log_2[\frac{P_{AB}(a_i,b_i)}{P_A(a_i)P_B(b_i)}] \tag{3.3}$$

We can rewrite this in terms of our data point $\boldsymbol{y}(n)$ and its first time delay $\boldsymbol{y}(n-\tau)$

$$I_{AB} = \sum_{\boldsymbol{y}(n),\boldsymbol{y}(n-\tau)} P(\boldsymbol{y}(n), \boldsymbol{y}(n-\tau)) log_2[\frac{P(\boldsymbol{y}(n), \boldsymbol{y}(n-\tau))}{P(\boldsymbol{y}(n))P(\boldsymbol{y}(n-\tau))}] \tag{3.4}$$

Using a special criterion invented by Fraser in 1985 we will take advantage of the average mutual information to satisfy our three rules that we want to follow. The average mutual information between time delayed data sets tends towards zero as the time delay grows larger, which is in line with our third rule that they must become independent with increasing time. We want to find the sweet spot of not too close, but not so far away that average mutual information goes to zero to follow our second rule, so according to Fraser, we choose the first minimum of the average mutual information.

The challenge now is in calculating the average mutual information to find the first minimum. We'll make use of our observed data and put it into histograms; there will be some deviation in the result based on how binning is done, but assuming it is done in a standard way, we can calculate the average mutual information by summing over the bins and using their ratios of present data points/total data points to calculate P. Using this process the average mutual information will be calculated for many τ's, starting with $\tau = h$ and incrementally increasing it by factors of h.

3.1.2 Choosing the Number of Time Delays

With the knowledge of what our time delay length will be, we must then ask ourselves how many time delays dimensions do we want. For a D dimensional system with D_0 observed dimensions ($D_0 \leq D$), we will have a total of $D_E * D_0$ dimensions where D_E is the number of sets of delayed dimensions plus the original leading set. In choosing the value of, D_E what we are really asking is how many dimensions are necessary to unfold the observed orbits from self overlaps arising from the projection of the attractor to a lower dimensional space. We call the lowest dimension that exists such that no more

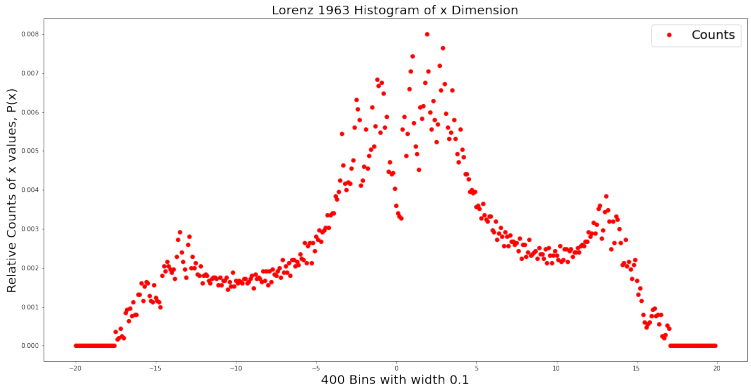

Figure 3.1. We took 25,000 data points from the x dimension of the Lorenz 1963 data set and put it in a histogram with a bin size of 0.1 with a total of 400 bins. This is an approximate form for the discrete probability distribution for the x dimension of the Lorenz 1963 model. This approximate probability distribution, in the form of a histogram, will be used to solve the discrete form of the average mutual information described in equation(3.4) which would be used to find the best time delay length parameter τ.

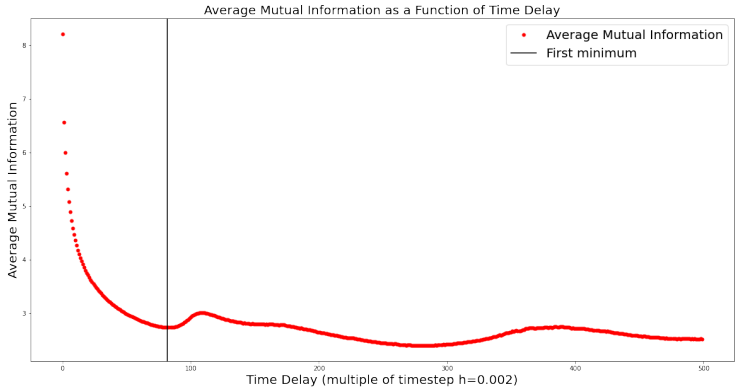

Figure 3.2. With the probability distribution we made above in Figure(3.1) and another similar one we made for the time delayed Lorenz 1963 data of the x dimension, we calculate the average mutual information from equation(3.4) for many different time delays, τ, to find our optimal time delay to be at $\tau = 87$*(h) where h in this case is 0.002.

of the overlapping points remain the embedding dimension. Note that the embedding dimension need not be the same as the original dimension of the system, fortunately the theorem states that if our total dimensions, $D_0 * D_E$, is greater than twice the fractal dimension of the original system, D_A, that would be sufficient to unfold all the overlapping

points.[32][13]

Now, while this condition of only needing more than twice the original dimension is a convenient theorem to fall back on, we may not want to rely on it entirely. The first obvious problem is that we may want to use fewer dimensions to reduce the computational overhead of our experiments. Additionally, we may not know the original system's fractal dimension if we only have the capability to observe some subset of the dimensions. For these problems, there exists a test that has been designed to test the overlapping of the system by looking at the number of the false nearest neighbors. False nearest neighbors arise when far away points in state space are projected onto each other through the process of observing reduced dimensions, so by creating time delay duplicates, we can tell when we have sufficiently unfolded the space when the number of false nearest neighbors drops to zero. The test we conduct is to take each point in our data set, $\boldsymbol{y}(n)$, as well as it's first time delay according to the average mutual information, and to calculate its nearest neighbor, $\boldsymbol{s}^{NN}(n)$. If the insertion of an additional time delay dimension takes the nearest neighbor out of the local space of the data point in question, then that nearest neighbor was a false nearest neighbor as opposed to a true neighbor that exists next to our $s(n)$ though dynamical origins. Once we increase the dimension to the point where no more false nearest neighbors exist, we have identified the embedding dimension, and we are done.

There is one last criteria that we need to address and that is identifying how far is far enough to be considered a false nearest neighbor, in practice it will be a subjective task, but we can define some terms to make it easier. We now define the Euclidean distance for the D_E dimension and the D_E+1 dimension for nearest neighbors at time n:

$$R_{D_E}(n)^2 = \sum_{d}^{D_E * D_0} [s_d(n) - s_d^{NN}(n)]^2 \tag{3.5}$$

$$R_{D_E+1}(n)^2 = \sum_{d}^{(D_E+1)*D_0} [s_d(n) - s_d^{NN}(n)]^2 \tag{3.6}$$

$$R_{nD_E} = \frac{R_{D_E+1}(n)}{R_{D_E}(n)} \tag{3.7}$$

R_{nD_E} is our rating for the false nearest neighbor at the n^{th} data point in time and for the D_E number of time delay dimension sets. We need to choose a cut-off point for which any distance greater is labelled a false nearest neighbor; this value will vary from data set to data set, but a standard initial choice for labelling a point a false nearest neighbor is $R_{nD_E} > 15$.

We'll go through an example of this process for the Lorenz 1963 system. We will start by calculating the nearest neighbors for $D_E = 1$, no time delays, and comparing their distances before and after adding an additional time delay dimension, $D_E = 2$. After recording the number of false nearest neighbors detected, we'll recalculate new nearest neighbors for the next data set with $D_E = 2$ and recursively repeating this process until D_E is large enough that no false nearest neighbors appear.

3.2 Applying Time Delays to DDF

Let's bring this discussion back into focus of DDF and the problem we are trying to solve; we have a data set, $\boldsymbol{y}(n)$, that has D_0 dimensions in a system with D total dimensions, such that $D_0 < D$, and we want to forecast its future behavior. To do this, we need to use the time delay embedding tools to reconstruct the attractor and state space statistics. Without reconstructing the state space, we'll never be able to properly forecast the model dynamics, as the dynamics are geometrically folded up as a result of observing fewer than the total number of dimensions. Fortunately the process of unfolding them with time delays is a relatively painless one, we start by finding the first minimum of the average mutual information to determine our time delay length, and after that, we find the embedding dimension, the smallest dimension free of false nearest neighbors. With

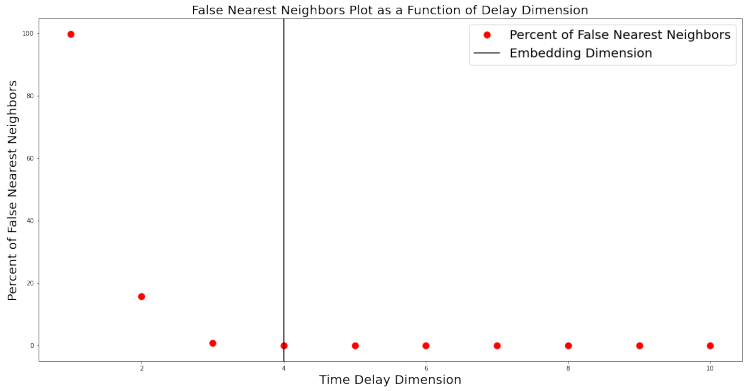

Figure 3.3. This test is conducted using an observation of only the x dimension of the Lorenz 1963 system, which looks at the percentage of false nearest neighbors. The percentage of nearest neighbors being false nearest neighbors drops to zero when $D_E = 4$ for a system that is originally 3 dimensions. This is a slightly subjective result, as we chose the cutoff R to be 25. Additionally, the percentage of false nearest neighbors was so low for $D_E = 3$ that it could be argued that we were too conservative with our choice of threshold for R. Regardless of some subjectivity for thresholds, we have gained an insight into what number of dimensions we'll need (say D_E equal to 3 or 4) to unfold the geometry of our data set to reconstruct the state space.

these tools, we can start modelling the dynamical system with DDF.

We will no longer be training and outputting the $\boldsymbol{y}(n)$ data, but rather the time delay data with D_E sets of $\boldsymbol{y}(n)$ time delay sets, with the length of the time delay being τ.

$$\boldsymbol{s}(n) = [\boldsymbol{y}(n), \boldsymbol{y}(n-\tau), ..., \boldsymbol{y}(n-(D_E-1)\tau)] \tag{3.8}$$

$$\begin{aligned}\boldsymbol{s}(n) = [y_1(n), ..., y_D(n), y_1(n-\tau), ..., y_D(n-\tau), ..., y_1(n-(D_E-1)\tau),\\ ..., y_D(n-(D_E-1)\tau)]\end{aligned} \tag{3.9}$$

The way we treat this data with DDF is slightly different from before, we'll take our newly created $\boldsymbol{s}(n)$ data and will train DDF on all $D_E * D_0$ dimensions of it. However, we will not forecast the behavior of all of its dimensions, rather, we will only forecast the leading D_0 dimensions. All the time delay dimensions will follow the rule defined in the equations above, they ***must*** be the value of the leading term delayed by some integer

multiple of τ in the past. So as we are updating the leading terms with DDF, we will update all the delayed terms with what DDF predicted for the leading term some time in the past. Our DDF update will appear the following way:

$$y_i(n+1) = y_i(n) + \sum_{q=1}^{N_c} \omega_{iq} \phi(||\boldsymbol{s}(n) - \boldsymbol{c}(q)||) \qquad i = 1, 2, .., D_E \tag{3.10}$$

There is a choice that will go on a case by case basis in terms of which values in the interpolation (RBF's, polynomials, etc.) of the update rule are inputted with the full time delay vector $\boldsymbol{s}(n)$ and which term is inputted with only the leading term $\boldsymbol{y}$. In practice, the RBF always receives the time delay term, $\boldsymbol{s}(n)$, but the polynomial terms have had empirical success only receiving the leading term, $\boldsymbol{y}(n)$ (when used in conjunction with the time delayed RBF's). Additionally, external forces are considered not part of the dynamics, as they are external, and thusly, are not time delayed; this case will be obvious in the Neurodynamics section when the topic of external stimulus to the neurons is discussed.

3.3 Time Delay DDF Examples

To give a nice illustration of what the time delays look like, we include Figure(3.4); it takes only the x dimension from the Lorenz 1963 system and shows the leading term followed by three of the same dimensions with their own respective time delays. The time delays in this figure are a little larger to better illustrate the concept as the τ is chosen to be 100 time steps, but the more effective τ is closer to about 87 time steps (for a time step length of 0.002 arbitrary Lorenz 1963 time units).

Our next figure, Figure(3.5), shows what happens when one tries to perform DDF on a single observed dimension. Here we see that the best one can hope for in a 1 dimensional system is a convergence to a single value (due to the decaying nature of the Gaussian RBFs, blowing up to infinity isn't an option); for systems of many dimensions, observing multiple dimensions, but not all, will still result in a poor forecast as the state space needs

to be reconstructed and reduced dimensional observation will project the properties of the state space to fewer dimensions.

The final figure, Figure(3.6), shows that with time delay embedding, we can only observe the x dimension of the Lorenz 1963 system, and with careful choice of time delay and time delay dimension, we can reconstruct the state space of the Lorenz 1963 system so well it can forecast for 13 lyapunov times.

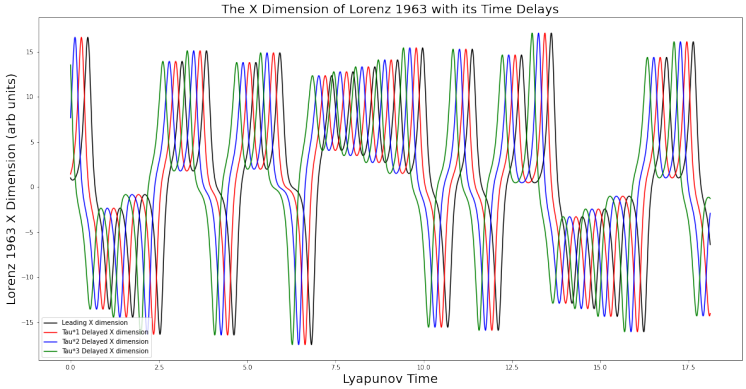

Figure 3.4. This graph is an illustration of what time delay embedding would look like, albeit for a one dimensional observation. If we were to use this observation of the x dimension for DDF, the data set would be broken up into four dimension, all copying the same data from the same X dimension observation, but with a time delay τ according to this graph. In this case, τ is set to 100 and the time delay dimension is set to 4, but this is just for show (τ equal to 100 looked a lot nicer than a smaller better performing τ). In practice, the best performing combination was found to be 13 τ with 11 time delay dimensions (as shown in Figure(3.6)).

3.4 Why use Time Delay Embedding?

Finally, we want to end on the discussion of why should we bother with time delays at all, and address the question of what would happen if we just trained DDF of the limited data, $\boldsymbol{y}(n)$, alone without time delays. By using time delays, we reconstruct the state space and thus the attractor that we need to forecast forward in time properly. But if we ignore the time delay embedding, and allow the state space geometry to remain

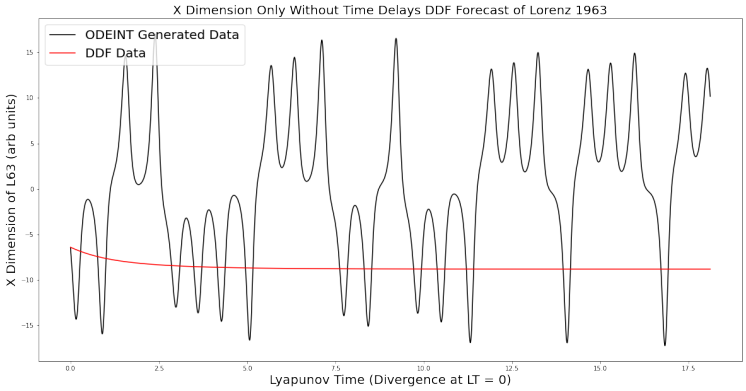

Figure 3.5. This figure shows the necessity for the time delay embedding technique when forecasting reduced dimensional observations. This one dimensional update rule results in convergence to a single value, which is both uninteresting and useless. Even if one had multiple observations from a larger system, if they do not have observations of the entire system, TDE must be used to reconstruct the state space, or they will suffer the penalty of forecasting with an incomplete state space/attractor, and their forecast will most likely be incorrect.

projected and folded up, then the model dynamics will be at best extremely difficult to piece together with DDF alone, and at worst, completely impossible to reconstruct without Time Delay Embedding.

To give an example of why this would be impossible, imagine the case of the Lorenz 1963 system, it is a three-dimensional and chaotic system (when the correct parameters are chosen); let's say we only observe one dimension of this three-dimensional chaotic system, and if we were to train DDF on this one dimension alone, we would never replicate the model behavior. This is because DDF constructs an approximate dynamical system to the differential equation:

$$\frac{dx_i(t)}{dt} = f_i(\boldsymbol{x}(t)) \qquad i = 1, 2, .., D \tag{3.11}$$

Well let's say this DDF constructed dynamical system is 1 dimensional, D=1, based off of the single observation, then there are only two possible outcomes, convergence to a

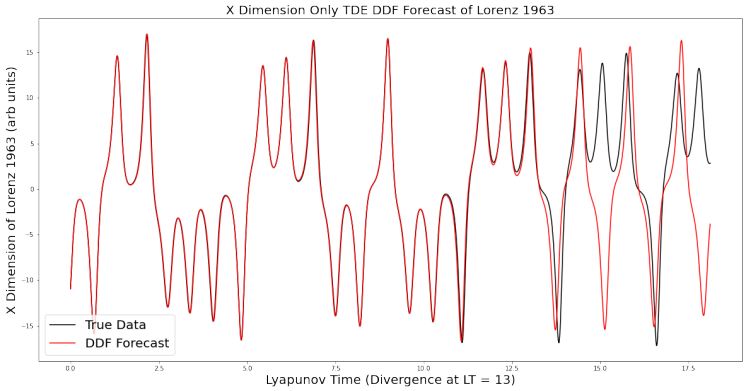

Figure 3.6. Here we can see that using time delay embedding, we have reconstructed the state space, or in other words, we have reconstructed the statistical properties of the state space and the attractor to appropriately update the forecast of the x dimension in time. While we reconstruct the information about the system to forecast well, we still don't have any information about the unobserved dimensions, and this will not change. TDE will not tell us anything about the unobserved dimensions, only aid us in forecasting the observed ones. This forecast stayed accurate for 13 lyapunov times with the use of Gaussian RBFs and TDE, only losing about 3 lyapunov times from the Gaussian RBF use with the complete data set and no TDE. The hyperparameters for this forecast are β = 6.680359280320344e-08, R = 0.005041444147767352, Time Delay Dimension = 11, τ = 13 (Differential Evolution was used to pinpoint the best forecast). Note that this tau value is much smaller than what was predicted using the average mutual information, and the dimension size is much larger than the 3 or 4 that the false nearest neighbors recommended. These results showed up simply because differential evolution found them to work well and found that a trade-off of shorter time delays and greater dimensions provided strong predictive power. The best combination of time delay length and number of dimensions will always involve some guess work, but the false nearest neighbor and average mutual information methods will always act as your starting point in this search.

fixed point and divergence to infinity (technically there are unstable fixed points that are stable as a single point in space, but any noise in a physical system would deny the system any long term convergence to one). Well, clearly this completely negates DDF's ability to replicate the chaotic behavior of the full three-dimensional system. Empirical tests also support this idea, as DDF consistently converges to fixed points when trained on 1D sets of data. View Figure(3.7) to see the limitations in which 1D systems operate.[5]

Through time Delay Embedding we are able to reconstruct the state space and

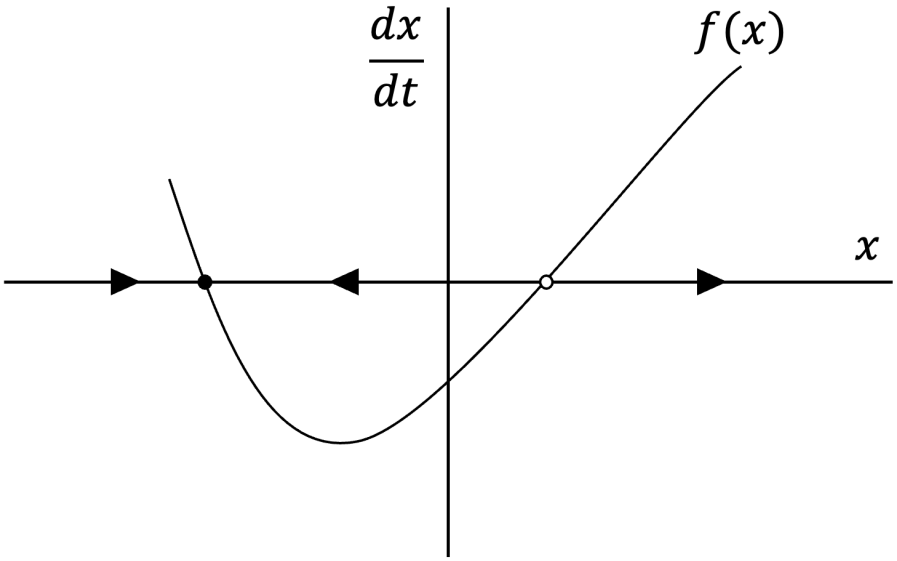

Figure 3.7. 1 Dimensional system showing a stable fixed point (filled circle), an unstable fixed point (hollow circle), and a divergence to $+\infty$ for value sufficiently large and positive enough. This is significant because if we try to take data from an observation of a single dimension, any attempt to construct a 1D dynamical system is doomed to fail as the outcome will always be convergence to a fixed point or divergence to $+/-\infty$. 2D systems can exhibit limit cycles in addition to fixed points and infinite divergence, and 3D systems and larger can exhibit chaos. It's also important to note that even if we observe 3 or more dimensions, but not all the dimensions of a system, we will fail to reconstruct the model dynamics without time delay embedding techniques, and this example is just to show an obvious and extreme case of why TDE is so important.

even in higher dimensions with a large (greater than two dimensions such that chaotic behavior is possible) but less than complete observation of a physical system, Time Delay Embedding consistently shows to be a boon to the predictive power of DDF.

Chapter 4

The Dynamical Theory of DDF

We have discussed previously in the introduction the different tools that mathematicians have used to characterize dynamical systems, what we seek to do now is to use these same tools and compare the characteristics of dynamical systems to the ones the DDF approximates in replicating those systems. The fractal dimensions, the lyapunov exponents, and the Jacobian of the dynamical system (which is used to calculate the lyapunov exponents) are all fundamental descriptors of a dynamical system; while it is not known what a complete set of characterizations for a dynamical system may be, we do have these characterizations that can at least be used to assess the similarities and differences that exist between systems. In the case of a DDF set of learned equations from observed data, we seek to calculate the fractal dimension, Jacobian, and lyapunov exponents of the DDF equations to learn how well DDF has learned the dynamical system. While the ultimate goal of Data Driven Forecasting is to forecast, we do so by reconstructing an approximate form of the dynamics and by comparing the properties of the DDF dynamics to the true dynamics, we can gain some trust in the predictions DDF makes and better understand what DDF is doing under the hood.

4.1 Fractal Dimension Preservation

The first test we will conduct will be to verify the preservation of the fractal dimension of the forecasted data set generated by DDF. The attractor that arises from the DDF fit to observed data will have its own fractal dimension (its own shape and geometry in the phase space it was trained in), the data set that DDF generates through forecasting is evident of this. The DDF update rule follows this attractor, or strange attractor for chaotic systems, and should the fit of DDF to observed data be successful, the DDF attractor should behave very closely to the original as verified by the accuracy of the forecast and the consistency of the fractal dimension.

It is easy to see how important the fractal dimension is to the forecast as the fractal dimension is a measure of shape and volume in phase space; if the fractal dimension is not consistent, the DDF attractor will either lead the forecast out of the constrained volume of the original dynamics (too large of a fractal dimension) or not fully explore the phase space of the original data (too small of a fractal dimension). Of course, it is possible to preserve the fractal dimension but get the geometry wrong as the fractal dimension is simply a number that a wrong forecast could get lucky and match; we can, however, be more confident in our DDF attractors accuracy with the calculation of the correct fractal dimension if the forecast is accurate as well.

Now that we've established the importance of preserving the fractal dimension with DDF, we show two cases of DDF preserving the fractal dimension. For DDF forecasts that are sufficiently accurate, it will preserve the fractal dimension so long as long term instabilities don't take over; anecdotally we have found that instabilities are more likely to occur in the case when the forecast is either poor or too short. We have also found that the fractal dimension of the DDF attractor is more accurate when the forecast is more accurate. This intuitively makes sense as capturing the correct dynamics is a requirement for good forecasting which also necessitates operating in the correct volume of phase space,

hence the matching fractal dimension. Figure(4.1) depicts the fractal dimension result for the Lorenz 1963 system for both the case of DDF learning the full system and learning a single dimension and using time delay embedding.

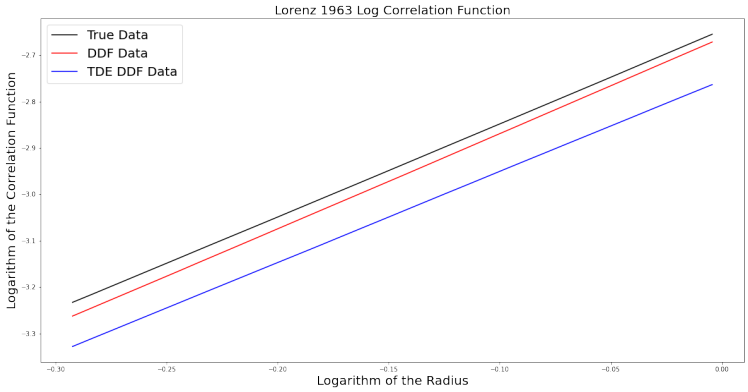

Figure 4.1. Using the method of Fractal Dimension calculation outlined in the appendix (Using the Correlation function), we calculate the fractal dimension of the Lorenz 1963 system from the slope in the graph to be 2.01 ± 0.03, the DDF data to be 2.06 ± 0.02, and the TDE DDF data to be 1.96 ± 0.02. The values are all in agreement, as the uncertainty in each measurement overlaps. The reported value of Lorenz 1963's fractal dimension is estimated to be 2.05 ± 0.01 [33], the reason for the discrepancy between this value and the one we report is likely do to numerical error on our part; the quantity of data and size of time step we use likely introduced some error, but the result suffices to show that consistency between fractal dimensions between the original system and DDF's reconstruction of that system. It is also worth noting that the TDE DDF data set is the same as in Figure(3.6) implying that even though it had 11 time delay dimensions, it still reconstructs the correct fractal dimension of the original system within its error bounds from a single observed dimension (the x dimension of Lorenz 1963).

4.2 Jacobian Reconstruction and Lyapunov Exponent Consistency

Before we can get into a discussion on the lyapunov exponents, we must discuss the Jacobian of the DDF system, as that is needed to perform the calculations of the lyapunov exponents. The Jacobian is a direct feature of the model equations, one that can easily be calculated from simple partial derivatives; the Jacobian of the DDF is a simple task to

calculate as well, the sum of radial basis functions allow for easy calculations.

We've seen in Figure(2.3) how the TSR DDF method can learn the Taylor series values for polynomial system and thus the Jacobian, but we are more interested in the non-polynomial models and the RBF representation. We want to know if the RBF method using Gaussian RBFs can accurately capture the Jacobian of the original model equations. From tests that we have conducted, we have found that they do **not**.

A Method has been invented to use RBFs to calculate Jacobians and the derivatives [34], but the way DDF is implemented does not allow for the calculation of the Jacobian of the dynamics that updates the data, at least not with the current usage of DDF. This means that the lyapunov exponents that could be calculated will also not match those of the original dynamics. While this shortcoming of reconstructing the model dynamics (which is what DDF attempts to do) may seem disappointing, it lays the groundwork for future updates to DDF to expand on itself, and it doesn't take away from the forecasting power already displayed by DDF.

Finally, there is one last point of interest to go into, and that is the calculation of the lyapunov exponents of the trained DDF system. We've already discussed how DDF can't capture the lyapunov exponents of the system that it trains on due to its difficulty calculating the Jacobian, but there is still insight the lyapunov exponents can offer us on the trained DDF system. Figure(4.2) shows our result for calculating the lyapunov exponents of the DDF system trained on Lorenz 1963 ($\lambda_1 = 0.89$, $\lambda_2 = 0.003$, and $\lambda_3 = -6.8$; for reference, the true lyapunov exponents of the Lorenz 1963 system are $\lambda_{1T} = 0.9$, $\lambda_{2T} = 0$, and $\lambda_{3T} = -14.7$ for the chosen parameters of the system); in this figure we see that the leading lyapunov exponent is both positive and similar to the true value of the Lorenz 1963 system. The closeness in leading LE is likely due to the fact that it is the dominant LE on the behavior of the dynamics and DDF picked up on that, and as one might expect the other LE is quite different (not counting the zero LE, as that is expected of a continuous time dynamical system). We should point out that as a

result of the DDF having a positive lyapunov exponent, it satisfies the criteria for being a chaotic system and in fact does exhibit chaotic properties like extreme sensitivity to initial conditions.

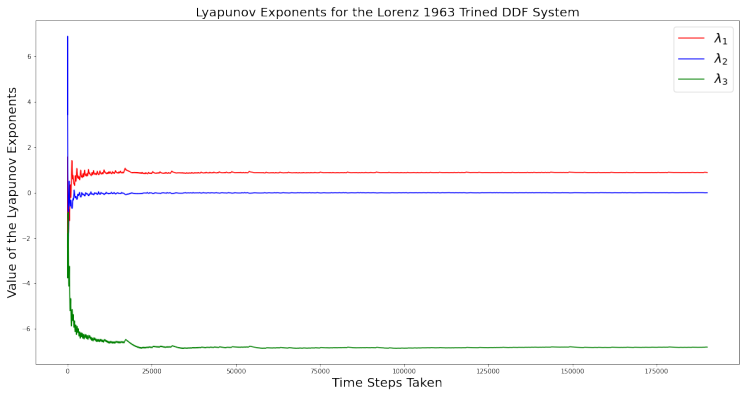

Figure 4.2. This graph shows the convergence of the lyapunov exponents to their estimated value as the number of time steps increases using the method of LE calculation listed in the appendix. The values converge to about $\lambda_1 = 0.89$, $\lambda_2 = 0.003$, and $\lambda_3 = -6.8$. For reference, the true lyapunov exponents of the Lorenz 1963 system are $\lambda_{1T} = 0.9$, $\lambda_{2T} = 0$, and $\lambda_{3T} = -14.7$ for the chosen parameters of the system. As all time series dynamical systems, the trained DDF system has one lyapunov exponent that is at zero. The maximal LE value λ_1 is remarkably close to the maximal LE of the original Lorenz system, which makes sense as this is the dominating lyapunov exponent of the system and the two dynamics behave very closely as we trained DDF to do. Another important thing to note is the existence of the positive lyapunov exponent implying that the trained DDF dynamics are chaotic, replicating a very significant behavior that existed in the original Lorenz dynamics; this DDF system, three separate sums of Gaussian RBFs (because it's training on a three-dimensional system) is capable of learning and exhibiting chaos (In testing we have also verified the extreme sensitivity to initial conditions to exist).

4.3 Concluding Remarks

Ultimately what makes DDF useful is not what was discussed in this section, DDF's utility comes from its ability to take observed data and to forecast future behavior of that system under the same or different conditions. We include this section to flesh out the lengths to which DDF reconstructs model dynamics, where it succeeds and where it falls short. We've learned that DDF can match fractal dimension for sufficiently accurate

fitting, we've learned that it can't reconstruct Jacobians, but it can learn chaos and have positive lyapunov exponents. These lessons are important because DDFs forecasting power comes from its reconstruction of the model dynamics, as evident here, a near perfect reconstruction is not necessary as DDF has been shown already to forecast with great accuracy despite the shortcomings discussed already. With the groundwork laid out here in this section, future work may go into adapting the DDF method to learn more about the model dynamics that it currently does and to study the inner workings of DDF more deeply, either to continue improving the forecasting power of DDF or to develop a tool that can give us estimates of model characteristics from observed data alone.

Chapter 5

DDF in Neurodynamics

DDF constructions have been shown to be capable of replicating both theoretically and experimentally generated data of the neuron membrane voltage. The theoretical data comes from the Hodgkin Huxley Neuron (more specifically, its numerical integration to generate time series data sets), and the experimental data comes from a laboratory doing current clamp experiments. As the experiments are only able to record the voltage, we use our time delay embedding techniques to generate an appropriate space in which the full dynamics can be realized; all the knowledge of the ion channels, gating variables, and their many parameters are not available to us in these tests. The current applied to the neuron is known to us ahead of time as the experimenter chooses the current ahead of time before applying it to the neuron, we will treat this current as a known driving force. Finally, after analyzing both Hodgkin Huxley neurons and experimentally measured neurons, we'll construct networks of neurons and show the capacity for DDF to create what we call "DDF Neurons" which can be duplicated in a network and connected to each other through ligand gated synaptic connections.[2]

5.1 Hodgkin Huxley Neurons

The Hodgkin Huxley model was the first quantitative model of active membrane properties developed in 1952[35]. It was originally used to calculate the action potentials of

the squid giant axon. Hodgkin and Huxley used voltage clamp experiments that produced the experimental data that guided them in their mathematical constructions of the sodium, potassium, and leak currents descriptions and the relationship ions had to the membrane potential. The Hodgkin Huxley model, also known as the NaKL model, is an effective model that has been shown to produce action potentials similar to what has been found experimentally and acts as a useful practice step before tackling the challenging task of using real noisy data.[36]

5.1.1 Hodgkin Huxley Structure for Driven DDF

Before jumping into the equations of Hodgkin Huxley, we must first address the structure of the equations and how we'll handle the stimulating current $I_{stim}(t)$. We'll break down the form of the Hodgkin Huxley equations in a way that will be easy to view in a DDF context:

$$C\frac{dV(t)}{dt} = F_{intrinsic}(V(t), \boldsymbol{A}(t)) + I_{stim}(t) \tag{5.1}$$

$$\frac{d\boldsymbol{A}(t)}{dt} = F_{\boldsymbol{A}}(V(t), \boldsymbol{A}(t)) \tag{5.2}$$

$F_{intrinsic}(V(t))$ represents the intrinsic behavior of the Hodgkin Huxley neuron, all of its ion channels, gating variables, and parameters exist in this function which sum up the electrophysiological behavior of biophysical neuron. The ions in this equation are the Sodium (Na), Potassium (K), and Leak (L) currents (The leak current isn't actually an ion channel, it acts as a fudge factor to account for error in the ion channels). The gating variables, which will later be denoted as m, h, and n, are labelled $\boldsymbol{A}(t)$ and range from 0 to 1 indicating an either closed or open state.

The Hodgkin Huxley formalism, as well as the true dynamics of real bio physical neurons, includes the external stimulus as an additive forcing term, I_{stim}, which we use to our benefit in the DDF protocol as DDF is particularly capable of handling additions of terms (referencing our prior discussion of the linear functionality of training weights in

the RBF's). The solutions, or discretization, of the Hodgkin Huxley equations take the following form as we manipulate them into a DDF formulation:

$$V(n+1) = V(n) + \int_{t_n}^{t_{n+1}} dt' \frac{F_{intrinsic}(V(t'), \boldsymbol{A}(t'))}{C} + \int_{t_n}^{t_{n+1}} dt' \frac{I_{stim}(t')}{C} \tag{5.3}$$

$$V(n+1) = V(n) + f_V(V(n), \boldsymbol{A}(n)) + \frac{h}{2C}[I_{stim}(n+1) + I_{stim}(n)] \tag{5.4}$$

$$\boldsymbol{A}(n+1) = \boldsymbol{A}(n) + \int_{t_n}^{t_{n+1}} dt' F_{\boldsymbol{A}}(V(t'), \boldsymbol{A}(t')) \tag{5.5}$$

$$\boldsymbol{A}(n+1) = \boldsymbol{A}(n) + f_{\boldsymbol{A}}(V(n), \boldsymbol{A}(n)) \tag{5.6}$$

Here f_V and $f_{\boldsymbol{A}}$ are both the interpolated functions used by DDF, in our case we use Gaussian RBF's. The integrals of the Hodgkin Huxley equation are too intractable to solve, but we can think of DDF approximating what these integrals would be. The stimulus integral is approximated using the 2nd order midpoint rule[37].

5.1.2 The Hodgkin Huxley Neuron Example

Now we want to analyze the DDF representation of the Hodgkin Huxley (HH) model for two cases. The first case will be an unrealistic test that utilizes all four of the dimensions, voltage and the m, h, and n gating variables; this test is unrealistic because in an experimental setting the researcher will only ever be able to measure the voltage in current clamp experiments. This first case is still performed as a confidence building exercise to establish DDF's capability to handle biophysical systems. The second case will be tackling the more realistic problem of only observing the voltage. This means our DDF tool will only train and forecast on the voltage alone; we'll also need to utilize time delay embedding methods in order to unfold the state space that is now projected down to the one voltage dimension.

The first step in this process is to generate our data set for training and forecasting comparisons using the HH equations. The HH model is four dimensional, containing voltage

and the gating variable m, h, and n, as well as three ion channels, Sodium, Potassium, and a leak channel (the leak channel is a fudge factor channel that accounts for errors in this simple model).

The equations for the HH model are:

$$\begin{aligned} C\frac{dV(t)}{dt} = g_{Na}m(t)^3h(t)(E_{Na} - V(t)) + g_K n(t)^4(E_K - V(t)) \\ + g_L(E_L - V(t)) + I_{stim}(t) \end{aligned} \tag{5.7}$$

$$\begin{aligned} \frac{da(t)}{dt} &= \frac{\eta_a(V(t) - a(t)}{\tau_a(V(t))} \\ \eta_a(V(t)) &= \frac{1}{2} + \frac{1}{2}(\tanh\left(\frac{V(t) - V_a}{\Delta V_a}\right)) \\ \tau_a(V(t)) &= \tau_{a0} + \tau_{a1}(1 - \tanh^2(\frac{V(t) - V_a}{\Delta V_a})) \end{aligned} \tag{5.8}$$

Where a should be interpreted as m, h, and n respectively. The parameter values are taken from [38]. The parameter values are listed below:

Then our observation of the four variables, $\boldsymbol{y}(t) = [V(t), m(t), h(t), n(t)]$, would be input into this explicit form for the DDF update rule. For voltage, the update rule is:

$$V(n+1) = V(n) + \sum_{q=1}^{N_c} \omega_{vq} e^{-R*||\boldsymbol{y}(n) - \boldsymbol{c}(q)||^2} + \frac{\omega_c}{2}[I_{stim}(n+1) + I_{stim}(n)] \tag{5.9}$$

Note that we're using Gaussian RBFs, N_c centers, and we've replaced the h/C in front of the stimulus term and replaced it with the ω_c, our fitting constant in front of the stimulus. The addition of the stimulus and its fitting parameter into the Ridge Regression is an easy extrapolation of the method to include one more dimension of parameters and inputs. Now we'll define our DDF update rule for the gating variables:

$$a(n+1) = a(n) + \sum_{q=1}^{N_c} \omega_{aq} e^{-R*||\boldsymbol{y}(n) - \boldsymbol{c}(q)||^2} \tag{5.10}$$

Table 5.1. Our chosen parameter values for each of our Hodgkin Huxley neurons. The values are taken from [38]

Parameter	Input Value
C	1.0
g_{Na}	120.0
E_{Na}	50.0
g_K	20.0
E_K	-77.0
g_L	0.3
E_L	-54.4
V_m	-40.0
ΔV_m	15
τ_{m0}	0.1
τ_{m1}	0.4
V_h	-60.0
ΔV_h	-15.0
τ_{h0}	1.0
τ_{h1}	7.0
V_n	-55.0
ΔV_n	30.0
τ_{n0}	1.0
τ_{n1}	5.0

Where a is to be interpreted as m, h, and n respectively. The task for DDF is to fit the RBF's to the behavior of the HH system that is associated with a user specified external stimulus.

Before we are ready to generate the HH data and perform DDF on it, we must have a discussion of what constitutes an appropriate choice of stimulus for biophysical neurons. There are two common issues that arise that derail our ability to perform DDF on neurons. The first issue comes from the fact that cell membranes act as low pass filters and when external stimuli are chosen to oscillate at high enough frequencies, aspects of $I_{stim}(t)$ are filtered out by the low pass filter and training is likely to be insufficiently well-informed. The test we have found to be useful in verifying whether a stimulus frequency is too high is to look at the Fourier power spectrum of the external stimulus. The Fourier power

spectrum must lie in a range that small enough that the frequency doesn't pass some threshold limit, which will vary depending on the type of neuron in question.

The second common issue that is best avoided, that is favored by many biologists in the field, is the use of step currents; there are two reasons we as practitioners of DDF prefer to avoid step currents. The first reason is that our methods rely on smooth transitions to properly interpolate our data, DDF specifically is a method reliant on capturing the vector space update rules from one position to the next and will not respond well to violent discontinuous jumps. The second reason why using step currents makes for a poor DDF neuron, is that while DDF can learn the behavior of a constant external stimulus, it does very little to explore the state space as the voltage will oscillate repetitively in a cyclical manner; it also poorly trains the DDF to handle much else other than a constant current which isn't very interesting.

Additionally, we must remark on the importance of the amplitude of the stimulating current to be sufficient to induce an action potential, or spiking behavior, in the neuron but not too stimulating that the neuron doesn't spend any meaningful amount of time in the sub threshold regime; this guarantees that the full dynamic range of the neuron is well represented which is very important for our DDF method. DDF is an interpolation method, and as such, must be exposed to the full range of neuron behavior in training to capture the proper behavior for forecasting. These rules apply to both applications of DDF to HH models and experimental data. V(t) data collected with $I_{stim}(t)$ chosen employing these guidelines were regularly successful in using Data Assimilation techniques to estimate the properties of rich HH models from laboratory data [38–40].

What we have found to be a reliable external stimulus has been the chaotic wave forms from the Lorenz 1963 system, we specifically feed into the neuron the x dimension of the Lorenz system and discard the y and z dimensions. The x dimension of the system is scaled both in amplitude and in frequency to bring out an appropriate behavior in the neuron, exploring both the action potential and sub threshold behavior. Now in Figure(5.1)

and Figure(5.2) we can see both our choice of stimulus, how the HH neuron responds to this stimulus, and our DDF prediction based on this stimulus. DDF is presented with the full I_{stim} time series for both the training and predicting times. It is also trained on all four dimensions, but we only show its performance with the voltage (it is pretty standard for DDF to perform equally well, or poorly, in all dimensions).

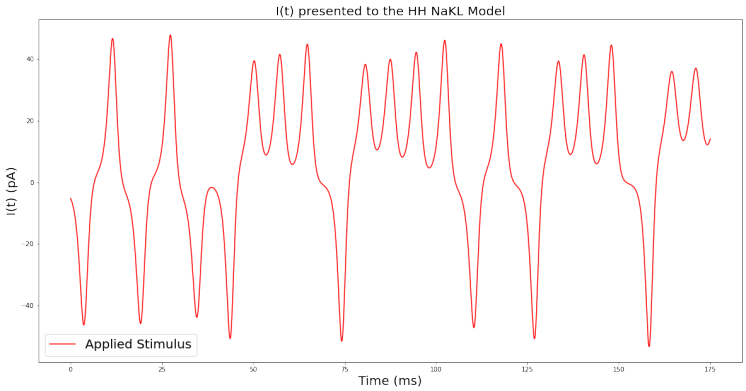

Figure 5.1. This is the x dimension of the Lorenz 1963 dimension, and it has been scaled appropriately to be used as the stimulus for the HH neuron. Using a simple tool like ODEINT we can generate any choice of stimulus to later be input into the HH neurons; later we'll see that it's these same types of Lorenz 1963 x dimension signals that get inputted into real neurons during experiments in the Margoliash lab through the use of wave function generators.

With this success we have created what we call a "DDF Neuron", or in this case a "DDF HH Neuron", that replicates the biophysical behavior of spiking and sub threshold behavior. This will be a key idea when it comes to building networks of neurons, that the neurons in a constructed network can be made up of DDF Neuron duplicates; this is very simple to do in practice, we simply take our trained DDF model with all its fitted parameters and create duplicates of it to receive different inputs and output its predicted voltage to different locations according to some network grid. Before we can do this, we must now address the issue of recording voltage only

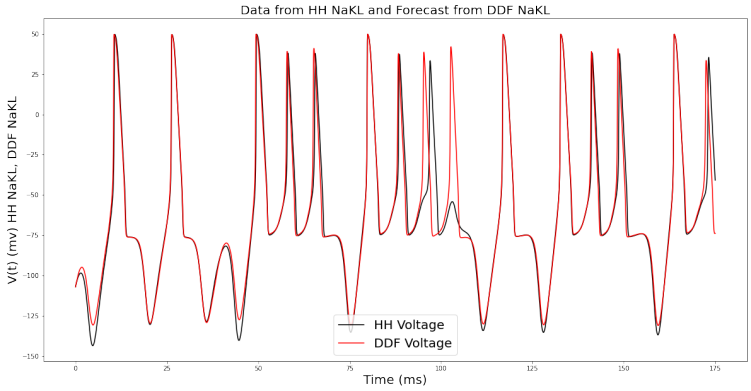

Figure 5.2. This is an effective application of DDF in constructing an HH neuron that replicates both the action potential and sub threshold behavior. The HH data was obtained by using ODEINT on the HH equations. The DDF process included the use of 5000 centers, R = 1e-2, beta = 1e-5, and it trained on 125ms of data (or 25,000 data points) and forecasted for 175 ms. The Neuron in this graph is stimulated by the current shown in Figure(5.1)

5.1.3 Observing V(t) Only

What is actually measured in current clamp laboratory experiments is V(t) alone. The neuron dynamics reside in a higher-dimensional space than the one-dimensional V(t) that is measured. What we observe is the operation of the full dynamics projected down to the single dimension V(t). To proceed, we must effectively "unproject" the dynamics back to a "proxy space" comprising the voltage and its time delays, which is equivalent to the original state space V(t) and the gating variables for the ion channels.

We accomplish this in the way described in the Takken's Embedding Theorem section, we will create a vector of time delays, $\boldsymbol{s}(n)$ from the observed dimensions, $\boldsymbol{y}(n)$. The new time delayed embedded system will include $(D_E - 1)$ sets of time delayed data plus the first original set of data that isn't time delayed, resulting in $D_0 * D_E$ total dimensions of the system. Since we only observe the voltage, $D_0 = 1$, and D_E becomes the total dimension of our time delay embedded system.

$$\boldsymbol{s}(n) = [\boldsymbol{y}(n), \boldsymbol{y}(n-\tau), ..., \boldsymbol{y}(n-\tau(D_E-1))] \tag{5.11}$$

$$\boldsymbol{s}(n) = [V(n), V(n-\tau), V(n-2\tau), ..., V(n-(D_E-1)\tau)] \tag{5.12}$$

The physics behind the time delay construction is that as the observed system moves from time n to n-τ, the dynamics of the system incorporates information about the activity of all other variables beyond the voltage alone. As described in section 4, we will use average mutual information to choose our time delay and the false nearest neighbors test to choose our embedding dimension, D_E.

We need to change the format of how we perform DDF slightly to accommodate these new time delays. We will use our new data set, $\boldsymbol{s}(n)$, with D_E dimensions of time delays of V(n) and time delays that are τ long (where τ is an integer multiple of the sampling rate, h) to construct this new formulation. We will choose our centers based off the new $\boldsymbol{s}(n)$ data set, so our centers will be D_E dimensional, but unlike traditional DDF, we will only forecast one dimension, the leading dimension. All non-leading dimensions, in this case $D_E - 1$ of them, will repeat the forecasted values of the lead term with their respective time delay applied. Note that the external stimulus, which is still using the trapezoidal rule, will not be time delayed as it is not a function of state variables, but is external to the system. The DDF update rule which only applies to one dimension for the case of current clamp experiments takes this form:

$$V(n+1) = V(n) + \sum_{q=1}^{N_c} \omega_{vq} e^{-R*||\boldsymbol{s}(n)-\boldsymbol{c}(q)||^2} + \frac{\omega_c}{2}[I_{stim}(n+1) + I_{stim}(n)] \tag{5.13}$$

5.1.4 HH Results When Only V(t) is Observed

Still using data from the HH model defined in equations(5.7,5.8), we generate a data set and discard all the dimensions of the data set except for the voltage. We now

train a Gaussian RBF with 125 ms of data for V(t) alone to fit the training parameters of the RBF to the data. Then, with this training complete, we forecast on 500ms of voltage data.

As in the earlier (unrealistic) example when all the state variables from the HH neuron model were available, the DDF neuron is able to predict the time course of the observable membrane voltage with significant accuracy while only being provided the voltage. Note that one cannot forecast the gating variables of the HH model when their data sets are not provided, as we have no information on them. Their statistical contributions are captured in the Time Delay Embedding reconstruction, but that only enables us to forecast the unprojected voltage and no other state variable.

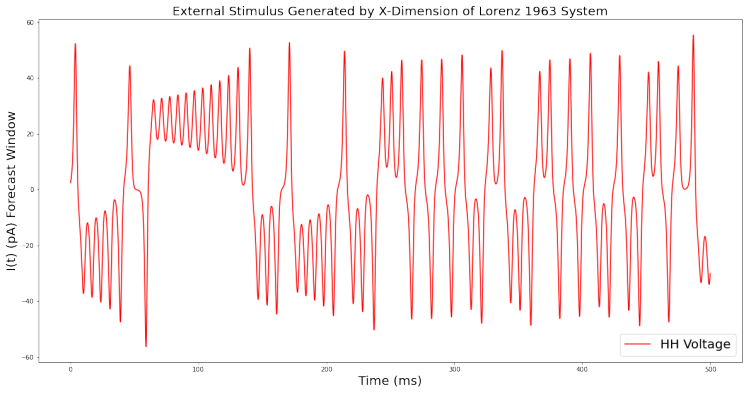

Figure 5.3. Data generated from the Lorenz 1963 system was used as the external stimulus for the voltage only HH DDF experiment. This data that was used in as the stimulus is specifically the x dimension of the Lorenz 1963 system, and the y and z dimensions were discarded.

5.1.5 Comparing Forecasting Times

To assess the effectiveness of using trained DDF to forecast V(t) data, for example, for the efficiency of computational demands on a DDF neuron in a circuit where it replaces an HH model, we compared the computation time for solving our HH model to the forecasting time of a DDF trained on the V(t) from the HH model.

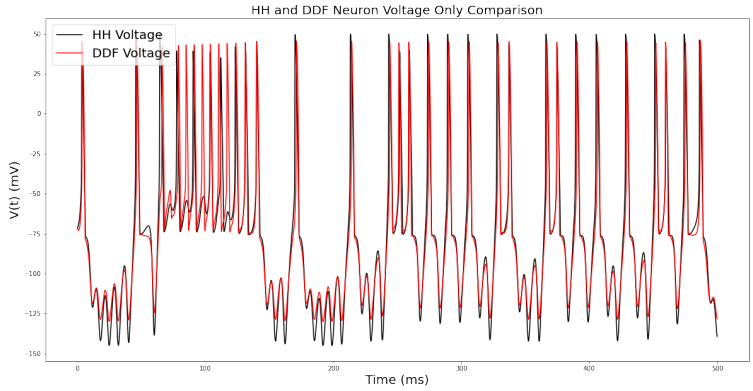

Figure 5.4. Data generated by solving the HH model and by forecasting with the HH trained DDF Neuron. We observe only the membrane voltage, V(t), and train on 125 ms of voltage data. The forecasting window could be arbitrarily long, but was chosen to be 500 ms. This is a numerical calculation, but it corresponds to a realistic current clamp experiment where, given a driving current $I_{stim}(t)$, only V(t) is observed. $h = 5 \cdot 10^{-3}$ ms, $\beta = 100$, $R = 10^{-3}$, $\tau = 8h$, $N_c = 5000$.

We generated HH data by solving equations(5.7,5.8) using a standard fourth order Runge-Kutta method [9] with a time step of h = 0.02 ms. The times taken by the generation of the HH data in a forecast window of 2000 ms (10^5 time steps) were 8.9 s for either CPU time or wall clock time.

We then forecast in the same window using the same $I_{stim}(t)$ as for the HH model but using a trained DDF, trained on V(t) from the HH model and forecasting on only V(t).

The choice of the number of centers N_c in the training and forecasting for the DDF is important to the forecasting time of the DDF. The number of centers used in the results shown in the paper were largely excessive (in the neighborhood of 5000), and were chosen this way to exemplify marginally, if not negligibly, more accurate results; in practice we could have as much success with 500 or even 100 centers. If we choose $N_c = 500$, then the CPU time for forecasting the 2000 ms with h = 0.02 ms is 3 s, while the wall clock time is 2.4 s. If we decreased the number of centers to $N_c = 100$, then the CPU time during forecasting is reduced to 2 s while the wall clock time drops to 1.5 s.

The training time for the DDF with $N_c = 500$, on V(t) from the HH model is 1.1 s for CPU time and 0.63 for wall clock time. This decreases to 135 ms CPU time and 142 ms wall clock time for $N_c = 100$.

We should mention that for these tests $D_E = 4$ as time delay embedding had to be used for the voltage only tests. The number of time delay dimensions used will have a noticeable effect on the computation time, as the increasing number of dimensions increases the number of computations required for training and forecasting.

These times will vary as the complexity of HH model neuron increases from the minimalist NaKL models to a model for observed laboratory observations. One expects V(t) trained and forecasting on DDF to become relatively more efficient than our results on the simple HH model. The training times for a DDF on V(t) data alone are quite fast. In the scenarios where we substitute DDF V(t) neurons for HH neurons in a circuit, the computational efficiency is what will be of central importance.

We will end on one final note that the numbers provided in this section are not an apples to apples comparison, as the programs involved are not professionally optimized; they were built in a laboratory setting with the intention of conducting comparisons and experiments with. Additionally, these tests were performed on a MacBook Air with an M1 Chip and 8 GB of memory, which will also play a role in affecting the outcome of these results. The key takeaway from this section is to exemplify a current standard for the processing speed of DDF and how competitive its speed is in a somewhat subjective manner.

5.2 Real Neurons

Now that we have had success applying DDF to the simple case of the HH model, we seek to push forward with applying DDF to the ultimate goal of using real neuron data from a once living animal. In our effort to study and access real neuron data, we have

collaborated with biologists (as we are a purely theoretical group with no surgical skill) to obtain this data. We worked with biology groups interested in studying the Avian Song system. In the collaboration between our groups, we have provided them with guidance on choosing effective currents, and they have measured the effects of these stimuli on the isolated membrane voltage of HVC neurons from the zebra finch song system.

5.2.1 Real Neuron Data Collection

Current clamp data were collected by C.D. Meliza at the University of Chicago laboratory of Daniel Margoliash from presenting various stimulating currents, $I_{stim}(t)$, to isolated HVC neurons in a zebra finch in vitro preparation. The data were organized into epochs of length about 2 to 6 seconds, observed over several hours. We'll perform two primary tests with these epochs, the first is that we'll take data from a single epoch, specifically epoch 25, train a DDF Neuron based on this data, and forecast the later behavior of this epoch (directly after training has ended). This test will show that we can capture the behavior of the epoch 25 neuron and have a real "DDF Neuron" that can be thought of as an approximate duplicate of a real neuron, but now we can input any stimulus we want into. This leads to our second test, which is to input the stimulus from epoch 26 into our "DDF Neuron" trained on epoch 25 stimulus and voltage data; this will showcase the adaptability of our DDF neuron to have flexibility to stimuli other than what it was trained on (Note that both stimuli are derived from a combination of the Lorenz system and step currents).

It is important to specify the epoch that we train the neuron with because the several hours of testing on the neuron is, to put bluntly, killing the neuron. This dying process of the neuron can be thought to have an altering effect on the intrinsic parameters of the neuron model itself. For example, imagine the neuron following the behavior of the HH neuron, but instead now the parameters are slowly varying functions of time because as the neuron dies, its behavior changes, which can be modelled as the parameters

changing in time. It is for this reason that when we conduct our second test to train and test on two separate epochs, we specifically choose the epochs to be immediately next to each other so that their intrinsic parameters are as similar as possible to keep the model consistent.

Now we will introduce the first set of data provided to us by the Margoliash lab. This is the full data set from epoch 25 including its stimulus and voltage response.

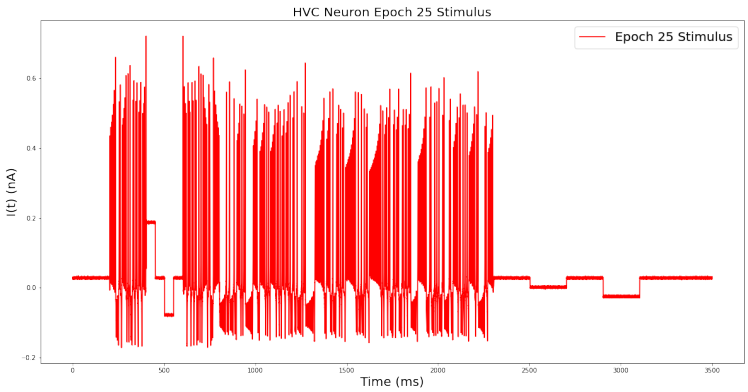

Figure 5.5. Stimulating current $I_{stim}(t)$ for the current clamp experiment at the Margoliash Lab at the University of Chicago. It's a mix of DC signals and Lorenz 1963 x dimension signals. The x dimension of Lorenz 1963 is a nonlinear and chaotic signal that is useful for exploring many different parts of the Neuronal state space, this exploration will give DDF a more flushed out resolution of the neuron's behavior enabling DDF to properly respond to a wider array of future input stimuli upon training on this voltage stimuli pair.

5.2.2 DDF Training and Forecasting on The Same Epoch

Now, using the same methods that were performed in the DDF application to the HH model, we will apply them to the real data from epoch 25 described in the section above. The Gaussian RBF's will now learn the behavior of a real live (although slowly dying) neuron from the zebra finch bird. Note that we again perform time delay embedding techniques on this data set, as the current clamp experiments only measure voltage.

Here we have in Figure (5.8) the successful creation of a DDF neuron that has

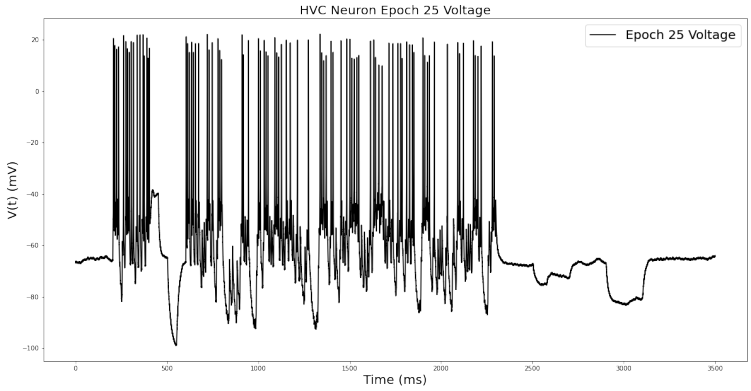

Figure 5.6. Membrane voltage response collected by C.D. Meliza (who is now employed at University of Virginia) at the Margoliash lab. The data were collected in multiple epochs from the same HVC neuron in zebra finch. Between epochs $I_{stim} = 0$. Many epochs of varying length were recorded. These data are, 3500 ms long.

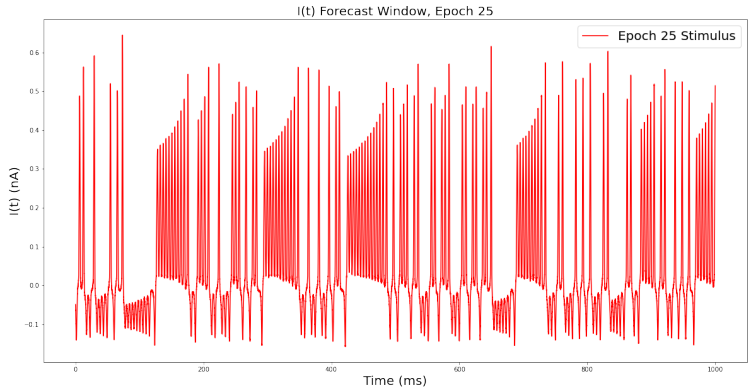

Figure 5.7. The recorded stimulus during the forecasting window in the epoch 25 DDF test. This is the stimulus that is being applied to the neuron during the same window of time as the voltage and DDF voltage recordings/prediction going on in Figure(5.8).

learned both the action potential and sub threshold behavior of the HVC neuron. The intrinsic dynamical properties of this real neuron have been approximated by the sum of radial basis functions (in the form of Gaussians for this case). The choice of R and β were found through a hyperparameter selection that is described in the appendix of this dissertation. Note how, as one would expect, the time delay went from 8h to 2h as the sampling rate was slower for the real neuron data. With this case done, the next question

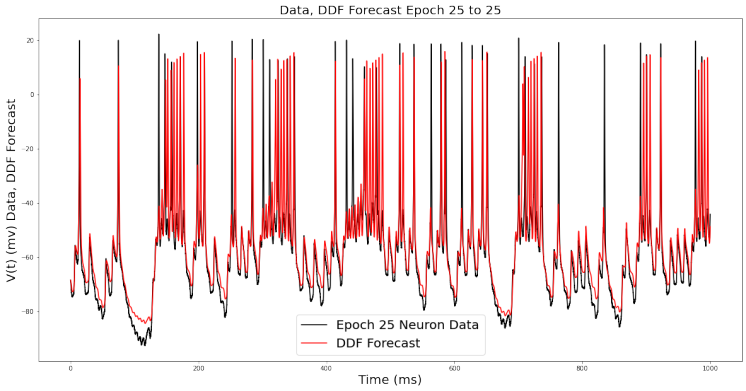

Figure 5.8. The DDF Neuron was trained for 1000 ms prior to the start of the forecasting window shown here. Only V(t) was observed and subsequently forecasted on. The forecast was for 1000 ms. h = 0.02 ms, $\tau = 2h$, $N_c = 5000$, $D_E = 4$, R = 10^{-3}, $\beta = 10^{-3}$.

to ask DDF is how flexible is it to changes in stimulus; we'll see how it performs in the next section when we take this DDF Neuron trained on epoch 25 data and forecast it with epoch 26 stimulus and compare it to the epoch 26 voltage.

5.2.3 Training a DDF Neuron on One Epoch and Forecasting it on Another Epoch

Our goal now is to show the flexibility of the DDF Neuron and its capacity to correctly react to stimuli that aren't the same as what it was trained upon. This is an important question to ask, since the final goal of this body of work in Neurodynamics is not to just model a single neuron, but to model whole networks of them. For a DDF Neuron to work in a network of neurons, it must be able to have some amount of flexibility in the stimulus, so it can work as a functioning network.

Here in Figures (5.9,5.10) we can see the epoch 26 data collected in the Margoliash Lab.

With the data collected by C.D. Meliza we can perform our test. The DDF Neuron is fitted to epoch 25 first, now using the same fitted parameters $\boldsymbol{\omega}$ and centers, $\boldsymbol{c}(q)$, we take this DDF Neuron and feed into it the stimulating current, $I_{stim}(t)$, that was used in

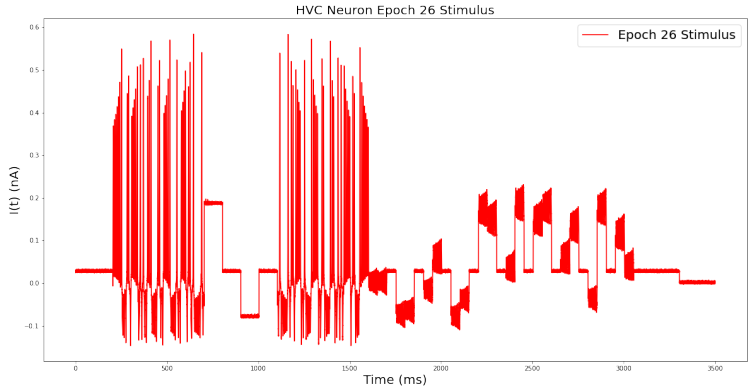

Figure 5.9. Stimulating current for a current clamp experiment at the Margoliash Lab at the University of Chicago. This stimulating current, $I_{stim}(t)$ was used during epoch 26.

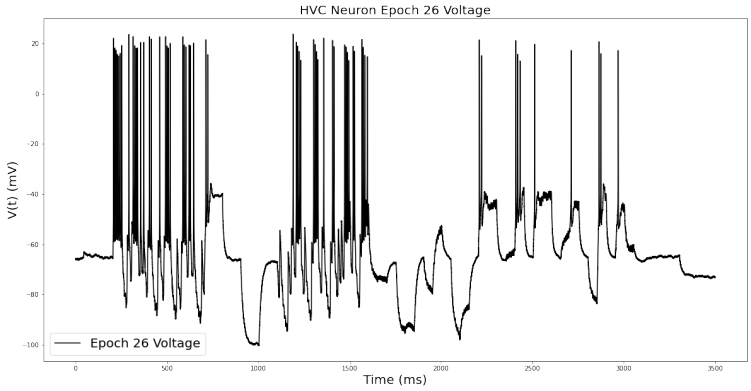

Figure 5.10. This is the membrane voltage response recorded at the Margoliash lab at the University of Chicago during epoch 26. C.D. Meliza was the scientist that collected this data.

epoch 26 shown in Figure(5.11). We now perform the forecast (no training is needed, as that was already performed during the epoch 25 test) and display the results in Figure (5.12).

We can see how successful the DDF Neuron is in adapting to a stimulus from another epoch. Granted, the stimulating current is still a waveform from the x dimension of the Lorenz 1963 system, we can still make the claim that the DDF Neuron has some amount of flexibility to its stimulating input and is capable of handling real noisy data

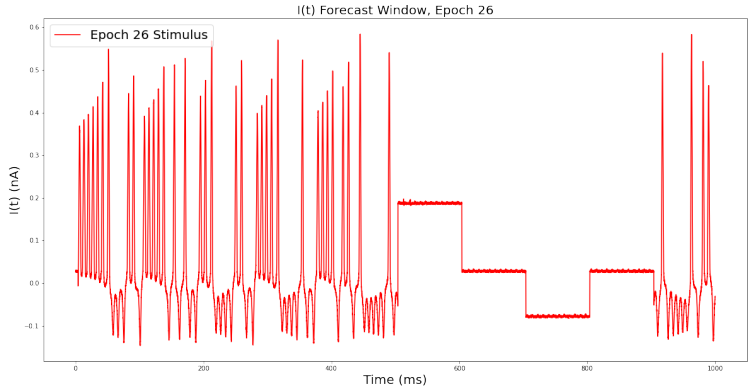

Figure 5.11. This is the stimulating current that was inputted into the DDF Neuron trained on epoch 25. This stimulating current, however, is from epoch 26 and is specifically displayed during the forecasting window.

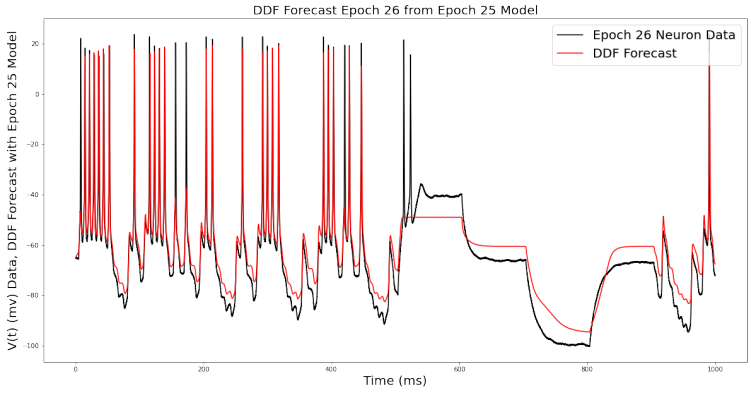

Figure 5.12. This is the analysis of epoch 26 and the DDF neuron's (trained on 1000 ms of epoch 25 voltage data) forecast of the voltage. The performance is worst for regions of $I_{stim}(t)$ consisting of square pulses; which is consistent with the observations of [41]. h = 0.02 ms, $\tau = 2h$, $N_c = 5000$, $D_E = 4$, R = 10^{-3}, $\beta = 10^{-3}$.

which is the greatest accomplishment of the Neurodynamics section in our view; this success paves the way for its use in networks of neurons. This will conclude our work with real biological data though as we wish to now transition to testing and developing a formulation for how to build DDF Neuron Networks and to do this, we'll need to compare with synthetic data generated from the HH model as well as delving into a discussion on synaptic connections that hold HH neurons together.

5.3 Networks of Neurons

One important goal of using neuron models trained by data alone (DDF Neurons), is to provide a reduced model based on biophysical observations to employ in building networks models. We demonstrate this in two ways, first in the most basic network comprised of just two neurons connected by gap junctions, and secondly, another two neuron network connected by a more realistic ligand gated synaptic connection. Note that it is only the presynaptic and postsynaptic voltages that convey information from any neuron in this circuit to others in the circuit. Both a pair of HH neurons and DDF neurons are used in this circuit, either connected by a gap junction or ligand gated synapse.

5.3.1 The Gap Junction Connection

The gap junction is a simple, straightforward method of connecting two neurons together in a network by relating their voltages to each other directly without any complex opening and closing gating variable at play. We assign a label number 1 to the first neuron in our circuit, this neuron receives an external stimulus as well as the stimulus from the gap junction. We assign the label number 2 to the second neuron in our two neuron circuit that doesn't receive an external stimulus, but is connected to the first neuron by a gap junction.

Gap Junctions are synaptic connects that are found in nature that are defined by channels that are permeable to ions and other small molecules that allow rapid but attenuated exchange of membrane voltage between neurons and are usually bidirectional. Sufficiently strong gap junctions can enable the action potential in one neuron to trigger an action potential in the paired neuron, which is something we'll tune our gap junction parameters to enable. While it is not the most complex synapse, it can lead to complex effects. [36][42]

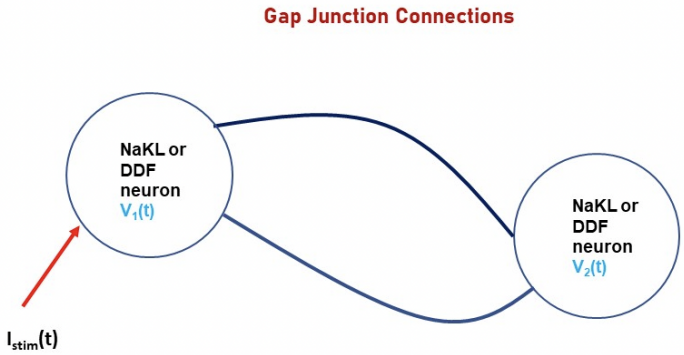

Figure 5.13. A two neuron circuit comprised of either two HH neurons or two DDF Neurons trained on HH voltage data. There are gap junction connections between the two neurons in the circuit. The circuit is driven by the stimulating current $I_{stim}(t)$ presented to neuron one. The computational task using DDF Neurons is simplified as only membrane voltage plays a role and no integration of HH differential equations is required in establishing the behavior of the neural circuit.

The voltage equations for the two neuron HH model becomes:

$$\frac{dV_1(t)}{dt} = F_V(V_1(t), \boldsymbol{A}_1(t)) + \frac{g_{12}}{C}(V_2(t) - V_1(t)) + \frac{I_{stim}(t)}{C} \tag{5.14}$$

$$\frac{dV_2(t)}{dt} = F_V(V_2(t), \boldsymbol{A}_2(t)) + \frac{g_{21}}{C}(V_1(t) - V_2(t)) \tag{5.15}$$

The F_V functions are the base HH dynamics covered earlier and have been condensed down to this form. To generate data from these equations, we can use our trusty ODEINT tool from scipy to produce our training and test data; however, we are interested in deriving a discrete form of these equations in which we can model our DDF update rule after. To derive a useful discrete form of the equations, we start by integrating over the interval [t,

t+h], and arriving at:

$$V_1(t+h) = V_1(t) + f_{V1}(\boldsymbol{S}_1(t)) + \frac{h}{2C}[I_{stim}(t+h) + I_{stim}(t)] + \frac{g_{12}h}{C}(V_2(t+h) + V_2(t) - V_1(t+h) - V_1(t)) \tag{5.16}$$

$$V_2(t+h) = V_2(t) + f_{V2}(\boldsymbol{S}_2(t)) + \frac{g_{21}h}{C}(V_1(t+h) + V_1(t) - V_2(t+h) - V_2(t)) \tag{5.17}$$

and then

$$g_{12+}V_1(t+h) = f_{V1}(\boldsymbol{S}_1(t)) + g_{12-}V_1(t) + \frac{h}{2C}[I_{stim}(t) + I_{stim}(t+h)] + \frac{g_{12}h}{2C}[V_2(t) + V_2(t+h)] \tag{5.18}$$

$$g_{21+}V_2(t+h) = f_{V2}(\boldsymbol{S}_2(t)) + \frac{g_{21-}}{C}V_2(t) + \frac{g_{21}h}{2C}[V_1(t) + V_1(t+h)] \tag{5.19}$$

$$g_{12\pm} = 1 \pm \frac{g_{12}h}{2C} \tag{5.20}$$

$$g_{21\pm} = 1 \pm \frac{g_{21}h}{2C} \tag{5.21}$$

By defining the two-dimensional vector $\boldsymbol{v}(t) = [V_1(t), V_2(t)]$, we may put the above equations into a more digestible matrix form:

$$\boldsymbol{M}_L\boldsymbol{v}(t+h) = \boldsymbol{M}_R\boldsymbol{v}(t) + \boldsymbol{J}(t) \tag{5.22}$$

in which

$$\boldsymbol{M}_L = \begin{pmatrix} g_{12+} & -\frac{g_{12}h}{2} \\ -\frac{g_{21}h}{2} & g_{21+} \end{pmatrix} \tag{5.23}$$

$$\boldsymbol{M}_R = \begin{pmatrix} g_{12-} & \frac{g_{12}h}{2} \\ \frac{g_{21}h}{2} & g_{21-} \end{pmatrix} \tag{5.24}$$

and $\boldsymbol{J}(t) = [f_{V1}(\boldsymbol{S}_1(t)) + \frac{h}{2C}[I_{stim}(t) + I_{stim}(t+h)], f_{V2}(\boldsymbol{S}_2(t))]$. Then the desired discrete map for gap junction coupling in a two neuron map is then:

$$\boldsymbol{v}(t) = \boldsymbol{M}_L^{-1}(\boldsymbol{M}_R\boldsymbol{v}(t) + \boldsymbol{J}(t)) \tag{5.25}$$

Now to rewrite this in the context of DDF, equation (5.25) is our DDF update rule, but $\boldsymbol{J}(t)$ changes from using the discrete form of the HH dynamics to the RBF expansion approximation. In addition to using RBF's the $\frac{h}{2C}$ term is dropped in exchange for a fitting coefficient ω_C. The parameters h, $gsyn_{12}$, and $gsyn_{21}$ that are used in the generation of the HH data set are also made known to the DDF update rule and are chosen specifically to make sure action potentials are induced in neuron 2. One keynote that is very important is that when a DDF Neuron is trained (in this case on a HH neuron), the trained RBF's will be the same in both Neuron 1 and Neuron 2; the $f_{V1}(\boldsymbol{S}_1(t))$ and $f_{V2}(\boldsymbol{S}_2(t))$ will be the same function, but with different inputs ($f_{V1} = f_{V2}$ in the DDF RBF representation).

5.3.2 A Two Neuron Network of the Gap Junction

We are now ready to use the discrete dynamical mapping derived in the previous section. To move forward with creating a two neuron network of DDF Neurons we'll need to first choose a stimulating current, $I_{stim}(t)$, which will like previous examples be taken from the x dimension of the Lorenz 1963 system. Then we'll use this external stimulus to generate a HH data set from a single lone neuron. This neuron will be duplicated and put into our two neuron network. We'll carefully choose our synaptic gap junction parameters to ensure the synaptic stimulus is sufficient to trigger action potentials in neuron 2. Finally, we'll compare the DDF Neuron 1 and 2 voltage to the HH Neuron 1 and 2 voltage to show how well DDF Neurons can operate in a simple network.

Our first step in building this two neuron network is to train a DDF Neuron on a single HH data set of voltage only data, which is done so in Figure(5.15).

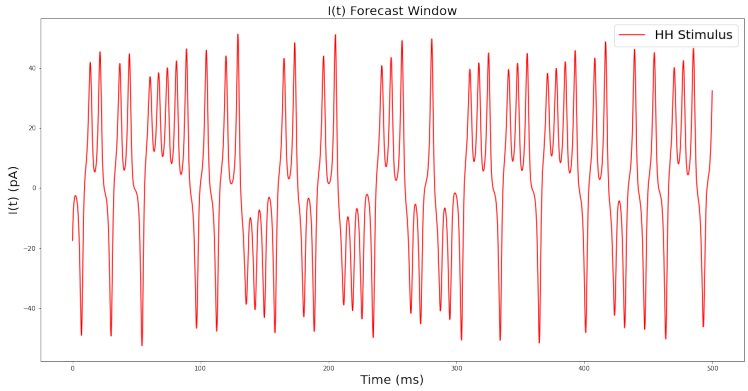

Figure 5.14. Stimulus for the training window of 500 ms. This stimulus was used as our external forcing for the DDF neuron and HH neuron during forecasting, which came directly after the training.

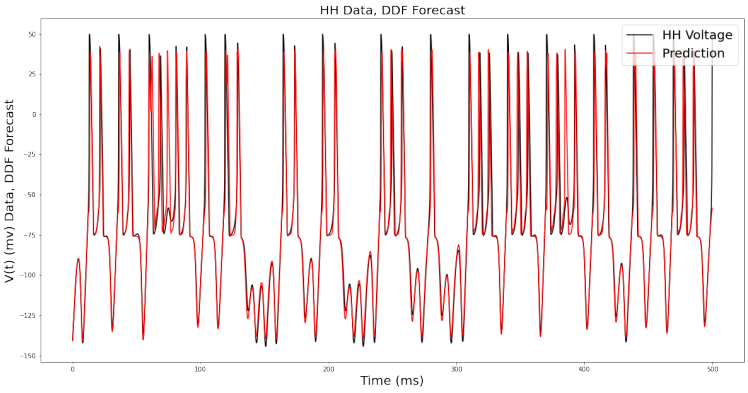

Figure 5.15. A HH model neuron and DDF Neuron were driven by the same 500 ms of stimulus, $I_{stim}(t)$. This DDF Neuron was trained specifically for the use of the two neuron network, its properties will be duplicated and placed into DDF Neuron 1 and DDF Neuron 2 in the next test. Also note that this is again a voltage only observation, so Time Delay Embedding was used on the voltage measurement to create a DDF Neuron that predicts voltage only. The DDF parameters are h = 0.02ms, $D_E = 3$, $\beta = 10$, R = 0.1, τ = 3h, $N_c = 5000$

Now we have our DDF Neuron, so our next goal is to generate a set of data for the two neuron circuit. Using ODEINT we generate the data for both neurons according to equations(5.14,5.15,5.8) and then according to the discrete dynamical update rule we create a forecast with DDF according to equation(5.25).

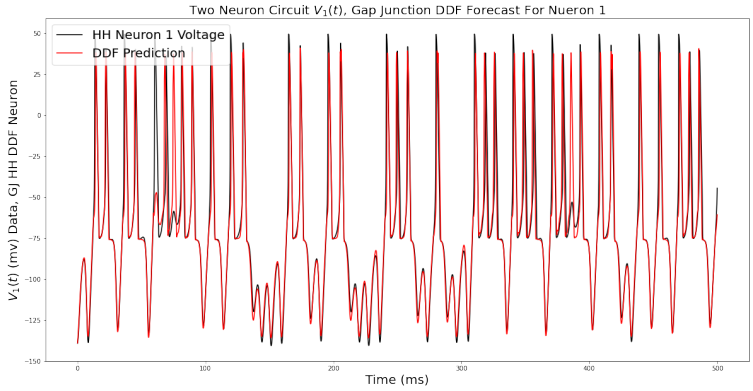

Figure 5.16. This is a comparison of the Neuron 1 behavior for the HH model in the gap junction and the DDF Neuron in the same gap junction. Both Neurons receive the same $I_{stim}(t)$ from Figure (5.14). The behavior of this neuron 1 is similar to the voltage response as the HH neuron in Figure(5.15) as the dominant stimulus is the external stimulus, but the behavior is slightly different.

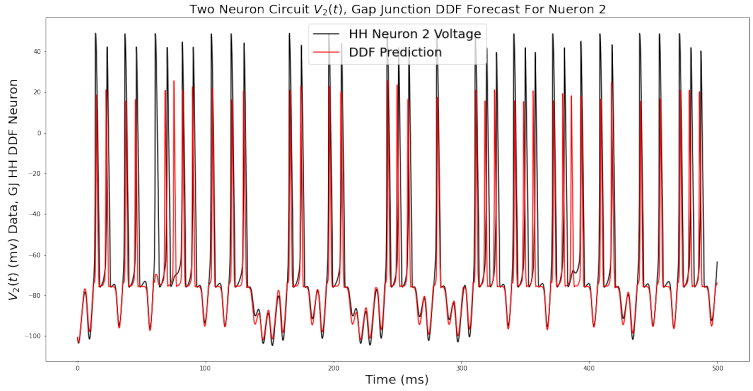

Figure 5.17. This is a comparison of the Neuron 2 behavior for the HH model in the gap junction and the DDF Neuron in the same gap junction. Both Neurons only receive an external forcing from the gap junction.

5.3.3 Ligand Gated Synaptic Connections

After the success of using the simpler Gap Junction (see Figure(5.16) and Figure(5.17)) and showing how DDF Neurons can be implemented in a network (albeit a very small one), we seek to expand the complexity of our network. We want to enrich our network by replacing the gap junction with a ligand gated synapse. We constructed

a network segment in which Neuron 1, with membrane voltage $V_1(t)$, is driven by a stimulating current, $I_{stim}(t)$, and this neuron drives a second neuron, with membrane voltage $V_2(t)$, via an excited ligand gated synapse. This network segment is shown in Figure (5.18). But first, we shall describe the nature of this ligand gated synaptic channel.

The synaptic current from the presynaptic neuron with voltage $V_1(t)$ that drives the postsynaptic neuron voltage, $V_2(t)$, is described by

$$I_{synaptic} = g_{syn} A(t, V_1(t))[E_{rev} - V_2(t)] \tag{5.26}$$

$$\frac{dA(t, V_1(t)}{dt} = \frac{A_0(V_1(t)) - A(t, V_1(t))}{\tau_A(A_1 - A_0(V_1(t)))} \tag{5.27}$$

where $A(t, V_1(t))$ is a synaptic gating variable (different from the gating variable we associate with the ion channels of a single neuron). It is opened when neurotransmitters bind onto a receptor on the postsynaptic cell, resulting in $A(t, V_1(t)) \approx 1$. It is closed when that neurotransmitter is released from the post synaptic receptor, resulting in $A(t, V_1(t)) \approx 0$. This behavior can be clearly seen in Figure(5.21) as it depicts the synaptic gating variable as a response to both the HH neuron 1 and DDF Neuron 1 voltages.

We represent the driver of this transition in the neighborhood of at a voltage V_0 from closed to opening by

$$A_0(V) = \frac{1}{2}[1 + \tanh \frac{V - V_0}{dV_0}] \tag{5.28}$$

This function moves from very near zero when $V << V_0$ to very near 1 when $V >> V_0$, as desired, and it does so over an interval of voltage dV_0.

For the excitatory synaptic connection, we select $g_{syn} = 0.5$, $E_{syn} = 50mV$, $\tau_A =$ 0.1 ms, $A_1 = 9/8$, V_0 = -50mV and $dV_0 = 10$ mV.

5.3.4 A Two Neuron Network of the Ligand Gated Synaptic Connection

Our two neuron network will be defined by the following equations and Figure(5.18)

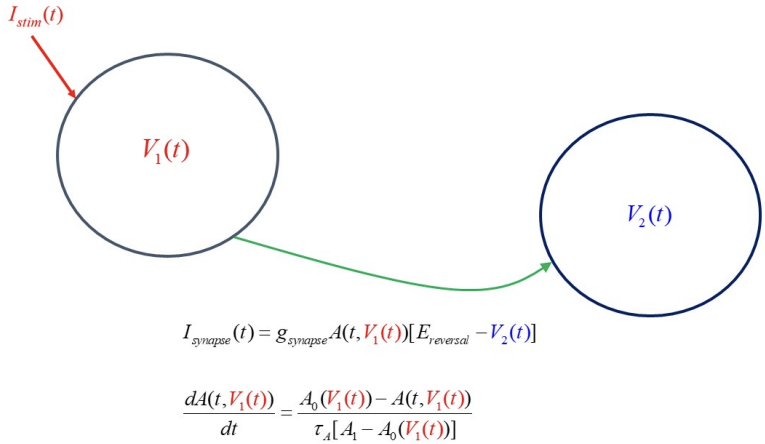

Figure 5.18. A network segment which has a presynaptic neuron with membrane voltage $V_1(t)$ driven by a stimulating current $I_{stim}(t)$ connected to a post synaptic neuron with membrane voltage $V_2(t)$ by a ligand gated connection.

These are the HH equations for this model:

$$\frac{dV_1(t)}{dt} = F_V(V_1(t), \boldsymbol{A}_1(t)) + \frac{I_{stim}(t)}{C} \tag{5.29}$$

$$\frac{dV_2(t)}{dt} = F_V(V_2(t), \boldsymbol{A}_2(t)) + \frac{g_syn}{C} A(t, V_1(t))[E_{rev} - V_2(t)] \tag{5.30}$$

Since Neuron 1 only receives the usual external stimulus, we can treat it as an independent neuron upon which we can train our DDF Neuron on (There's no reason why we couldn't use the DDF Neuron trained in the previous section other than it was easier and quicker to train a new one than it was to dig up the previous DDF Neuron). Then with this trained DDF Neuron, we'll duplicate it and put it in the Neuron 2 position. Here is how we will set up our discrete DDF update rule for the system.

$$V_1(n+1) = V_1(n) + f_V(V_1(n)) + \frac{\omega_C}{2}[I_{stim}(n+1) + I_{stim}(n)] \tag{5.31}$$

$$V_2(n+1) = V_2(n) f_V(V_1(n)) + \omega_C g_{syn} A(n, V_1(n))[E_{rev} - V_2(n)] \tag{5.32}$$

Again, we only observe the voltage for these experiments (the m, h, and n variables are discarded after the data set is generated through numerical integration) and time delay embedding is used. Note how the fitted parameter ω_C has been reused in Neuron 2; the reasoning behind this is that DDF Neuron 1 learned how to regulate its input stimulus according to this fitted parameter, which is effectively its capacitance (hence the C subscript). We argue that it makes the most sense to treat this fitted parameter as Neuron 2's reaction to all external stimulus be it from another neuron or injected current from a current source as it was learned from Neuron 1's fitting to external stimulus.

The manner in which we will perform our update rule will go as follows, at time n. First, take the stimulus, $I_{stim}(t)$, and voltage, $V_1(t)$, at time t=n and input it into the DDF update rule for Neuron 1, generating $V_1(n+1)$. Next, we must calculate the update to the synaptic gating variable $A(t, V_1(t))$; given $A(t, V_1(t)$ at time t=n+1 and $V_1(t)$ at times n and n+1, we integrate equation(5.27) using ODEINT to find $A(t, V_1(t))$ at time t=n+1. Finally, we plug the calculated value of $A(n+1, V_1(n+1))$ into equation (5.26) to calculate the synaptic current going into Neuron 2 to calculate the update rule to $V_2(t)$ and forecast its voltage. This recursive process can be repeated indefinitely. In Figures (5.21) and (5.22) we see the 500 ms window result of this process.

This network segment result indicates that, indeed, we may replace the more complex HH neuron voltage activity with the reduced dimension, biophysically trained, V(t) DDF Neuron in synaptic connections occurring in a network of neurons.

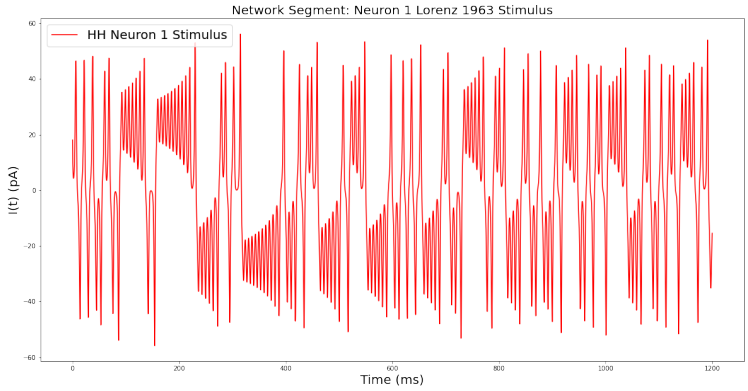

Figure 5.19. This is the stimulus input into Neuron 1 during the 500 ms forecasting window.

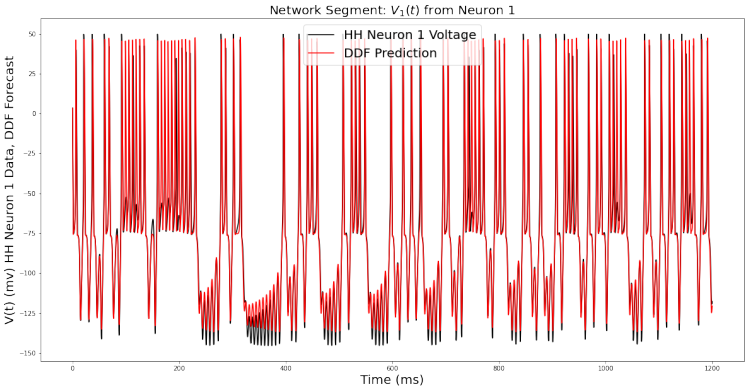

Figure 5.20. This is the HH and DDF Neuron 1 voltage response to the external stimulus. Our DDF Neuron was trained and validated with this data set; specifically, this DDF neuron was trained on the 500 ms of data that came prior to the forecasting window. This voltage activity will drive the synaptic current, which activates the action potential in neuron 2. Note that in keeping the two neuron circuit as accurate as possible, two separate gating variables are produced from each Neuron 1 voltage to stimulate their respective Neuron 2's. The parameters of this model are h = 0.02 ms, $D_E = 4$, $\tau = 2$h, $\beta = 10$, R = 10^{-2}, and $N_c = 500$.

5.4 Data Collection of Neuronal Systems

We want to have a discussion now on the different ways data is collected in neuroscience and how this will relate back to DDF. The entirety of this section has been data

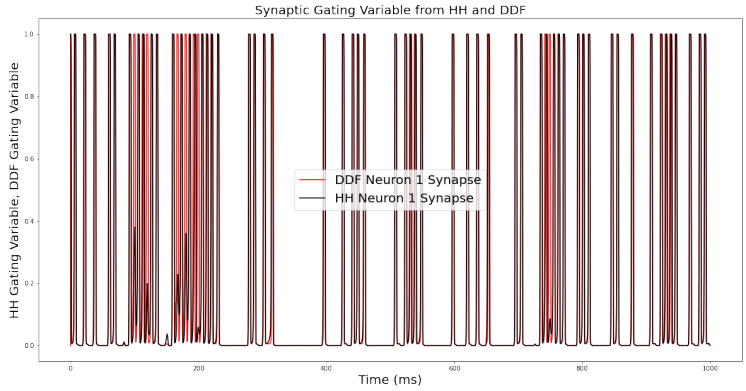

Figure 5.21. This is the gating variable $A(t, V_1(t))$. Note how it scales from 0 to 1 with the consistency of the action potential from the above graph depicting each plot's respective $V_1(t)$.

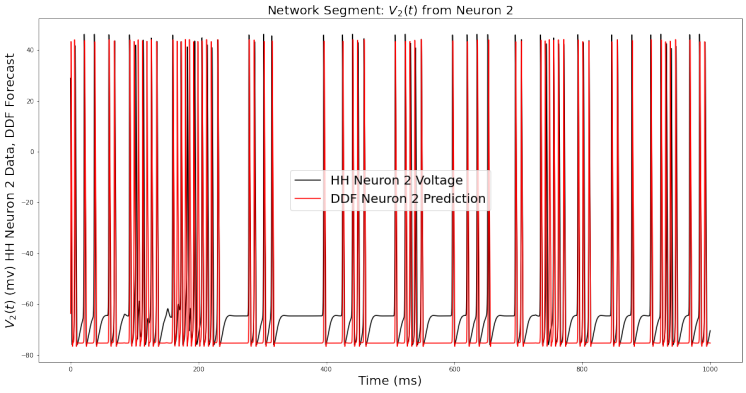

Figure 5.22. Here we see the driving effect from Neuron 1's presynaptic connection through a ligand gating synapse stimulating a voltage response in Neuron 2 for both the DDF and HH Neuron.

obtained from in vitro current clamp experiments, which is a technique that exists under the umbrella of electrophysiological approaches to studying the brain. Electrophysiological recordings are generally the only recordings available to us that offer enough information to properly reconstruct the model dynamics of a neuron, but they are very limited in the number of neurons that they can study, typically just a single neuron at a time. There exists many other methods of studying the brain that explore larger sections of the brain if

not the whole brain, but currently lack the detail needed to construct biophysical neuron models from data. Methods other than current clamp experiments include whole brain imaging (like MRI, cerebral angiography, computerized tomography, and diffusion MR imaging), extracellular recording, intracellular recording, and florescent techniques (like the calcium fluorescent technique)[43, 44].

We've taken an interest specifically in the calcium fluorescent technique described in [44]. Through the use of injections of a fluorescent binding agent that binds to calcium, neuroscientists can monitor large clusters of calcium ions across an individual neuron's cell membranes, providing us with an indirect observation of the action potential of the neuron. This is the type of technique that is desperately needed to advance our study of neurons from single neurons models to large neuronal networks. But as other papers have commented, [45], the accuracy of the method is limited despite offering a strong spatial resolution. Methods for inferring the spiking behavior and synaptic connections have been invented, [46] to try and circumvent these issues.

With the lack of concrete neuronal network data and rigorous successful study of inferring network connections, the step to working in large networks of neurons is a difficult one. This simply means that the study of new techniques is pivotal, both experimentally and numerically, from observed data. DDF has shown to be capable of replicating real neuron behavior even when exposed to signals it didn't train on. Future work in this field could involve the combined use of calcium fluorescent data, inferred network connections, and DDF neurons to generate depictions of network voltage responses.

5.5 Concluding Remarks on Neurodynamics

This section is a melding of ideas from nonlinear dynamics and applied mathematics to the goal of construction biophysically based models of observables in neurobiology. These data driven models encode the full information in experimental observations on the

complex system, the neuron, which is observed in current clamp experiments, i.e. one with a given $I_{stim}(t)$ with membrane voltage V(t) observed, and permit forecasting the voltage response of the observed neuron to the other stimuli.

The method Data Driven Forecasting (DDF) and the model construction produces discrete time nonlinear mappings $V(t) \rightarrow V(t+h)$ that may be used in neural networks with synaptic or gap junction connections, as the dynamical function of each is determined by the voltages of the presynaptic and postsynaptic cells. We have shown this in simple networks for gap junction neuronal connections and for a network where a presynaptic neuron, stimulated by $I_{stim}(t)$, drives a postsynaptic neuron via an excitatory synapse.

The DDF based network permits a computationally efficient network model where the trained DDF discrete time map, trained on biophysical data, replaces complex Hodgkin Huxley models typically used in contemporary research. The computational advantage is achieved in two ways. The first way by circumventing the need for a differential equation, and thusly, having to solve a differential equation. The second is that the models are substantially reduced in complexity, as the only observable V(t) is forecast. Timing comparisons we performed on the HH model neuron solved by fourth order Runge-Kutta ODE solver compared to the forecasting efficiency of DDF forecasting V(t) alone showed that DDF had an improvement factor of about 3.7 (keep in mind this was done with code not professionally optimized and with a MacBook Air M1 chip with 8 GB of memory).

One must recognize that while much is gained by using DDF Neuron constructions, something is set aside, and that is the knowledge of the biophysics in detailed HH models of individual neurons including the operation of gating variables for the ion channels selected for the model, parameters determining the dynamics and strength of those ion channels, and other yet unobserved biophysics at play in the neurons.

Chapter 5 has been adapted from Randall Clark, Lawson Fuller, Jason A Platt, and Henry DI Abarbanel. "Reduced-Dimension, Biophysical Neuron Models Constructed From Observed Data". Neural Computation, 34(7):1545–1587, 2022. The dissertation author

was the primary investigator and author of this paper.

Chapter 6

DDF in Fluid Dynamics

Using data alone, without knowledge of underling physical models, nonlinear discrete time regional forecasting dynamical rules are constructed employing well tested methods from applied mathematics and nonlinear dynamics. Observations of environmental variables such as wind velocity, temperature, pressure, etc. allow the development of forecasting rules that predict the future of these variables only. A regional set of observations with appropriate sensors allows one to forgo standard considerations of spatial resolution and uncertainties in the properties of detailed physical models. Present global or regional models require specification of details of physical processes globally or regionally, and the ensuing, often heavy, computational requirements provide information of the time variation of many quantities not of interest locally. In this section, we formulate the construction of DDF models of geophysical processes and demonstrate how this works with the familiar example of a 'global' model of shallow water flow on a β plane. A sub-region, where observations are made, of the global flow is selected. A discrete time dynamical forecasting system is constructed from these observations. DDF forecasting accurately predicts the future of observed variables. [3]

6.1 Modern Uses of Machine Learning in Geo Physics

Currently, there is a large amount of interest in applying machine learning techniques to geophysical problems. Efforts are being made to replace modern numerical weather forecasting with data trained networks [47][48][49][50]. However, while there is success being found in this field, there is a hesitancy in the climate and weather forecasting community to openly accept these methods as they are viewed as black boxes that have no relation to the physics that they model other than the data that is provided to them [51]. It is for reasons like this that there have been efforts in the machine learning community to publish papers with a large focus on the efficacy and trustworthiness of their machine learning tools [52][53]. Members of the machine learning community in weather forecasting have gone an additional step further by implementing hybrid models that are machine learning devices that include physical constraints in some form [54][55][56]. There have been findings showing that these weather forecasting tools that account for physical constraints (typically in their loss functions) show improvement over machine learning tools that utilize no knowledge of the underlying physics [57]. What this all amounts to is that there is both an interest and benefit to the hybridization of physics and machine learning models, and we argue that the DDF model we put forward embodies the hybridization of physics and machine learning in an atypical way and is performed to reap the forecasting benefits while maintaining a high degree of transparency in what it's doing.

DDF in Fluid Dynamics, as will be shown in explicit detail later, incorporates both the physics of the underlying model and machine learning tools to create an update rule for forecasting. Unlike many hybridization machine learning models, it doesn't enforce a physics constraint in its training function, it's training proceeds as standard regularized ridge regression. The update rule for DDF is a sum of radial basis functions and physically inspired terms which include, but are not limited to, forcing terms and polynomial variables

in the original model. We are not creating a pure forecasting tool, rather, we are creating an approximate form of the differential equations of the system to reconstruct the dynamics; it is this perspective that enables us to use tools like time delay embedding that aid us in reconstructing the state space of a reduced dimensional observation. The physics inspired terms in the update rule are naturally thought to capture the behavior of the model they represent while the sum of radial basis functions account for the parts of the model that are unaccounted for (in our example this would be the non forcing and non-polynomial terms) and would capture the "left over" dynamics. It is in this sense that we can construct a forecasting tool purely from data and physical inspiration without assuming too much of the entire model.

Our work with the regional DDF model is not the first of its kind. Regional models of different types have been researched before [58–61] and there exists a historical interest in them. There also have been other dynamical reconstructors in the past [62–67] that perform a similar task of reconstructing the dynamics of their respective system. In the context of the SWE, we seek to compare the models already applied to the SWE [68, 69] and the results we show later with DDF; while it is not an apples to apples comparison as the specific SWE data sets differ, the results show DDF's resiliency to noise, its accuracy over time, and adaptability to study under reduced dimensional observations.

6.2 The Geophysical Problem

In geophysical models for global and regional weather or climate forecasting, solutions of the Navier-Stokes equations, expressing conservation of mass and momentum, are accompanied by a thermodynamic equation describing energy conservation. An equation of state relating the thermodynamic variables to each other is required, as are further parameterizations to represent unresolved Physics below the model grid scale. Cloud moisture dynamics is a critical example of the latter.

Numerical solutions to these partial differential equations (PDEs) using, for example, a finite difference method [9][70] lead to formulations with a very large number of ordinary differential equations (ODEs) at a global set of spatial grid points. In global models, the number of these degrees of freedom (ODEs) may range from 10^8 to 10^{10} while for regional models this may still be large, say 10^6 to 10^7. Even then, the resolution of the largest operational General Circulation Models (GCMs) today is about 9 km in the horizontal [71]. Models with scales down to 1.4 km are being developed and tested [72], and the computational challenges grow as these are being realized.

If one's interest is in forecasting the weather or climate only in a selected region, another point of view may be employed. Instead of studying the entire global system, with DDF we can build the relevant dynamics using only observations of the state variables (pressure, velocity, temperature,...) one wishes to forecast within a desired region. Knowledge of the forcing of the system at the location of the observations is required, but this is already estimated in the formulation of the global models [73][74].

Working from data alone avoids uncertainties in initial conditions for the ODEs, uncertain physical features of the models and their boundary conditions, and the like. It also circumvents the growing computational complexity as the spatial resolution of big models is increased.

Not surprisingly, one loses something in a formulation that bypasses knowledge of the fundamental physical dynamical equations of the problem. At the same time, the computations for forecasting observables only is enormously simplified compared to calculating the full set of physical properties in a region or globally. In this section, we will combine the mathematical tools of DDF with the essential physics of geophysical models to construct regional models through observations of geophysical variables.

6.3 Geophysics Background

Our goal now is to more rigorously define the models of which we are interested in, our DDF experiments will focus on generated data from PDE solvers, and we must now discuss the PDEs in question. We will begin our discussion with the most fundamental equations of Computational Fluid Dynamics (CFD), the Navier Stokes equations. Upon describing these equations and their significance, we will aim to solve a simpler problem, one that makes the PDE integration more tractable in the form of the Shallow Water Equations (SWEs).

While we are interested in constructing approximate models that don't need or rely on models and only require data, we can learn a lot from the physical models and use this knowledge to better help us construct our DDF approximate model. In addition to using the physical intuition gleaned from models in the application of DDF, we seek to perform twin experiments with this data (i.e. tests that aim to use computer simulated data of fluids rather than direct measurements of observed data) which means we'll require the models and PDE solvers to enact our experiment.

6.3.1 Navier-Stokes Equations

The Navier-Stokes (NS) Equations are a collection of seemingly complicated equations but are derived from basic physics principles and calculus; they were originally found independently by M. Navier and G. Stokes independently in the first half of the nineteenth century. These fundamental equations are arrived at by the continuity equation (also known as mass conservation), momentum conservation (which relies on Newton's Second Law), and energy conservation. All of CFD is based on these governing principles, even the simplified Shallow Water model we'll discuss later is just a reduced version of the NS model.

6.3.2 The Continuity Equation

The continuity equation, also known as the mass conservation equation, shows up repeatedly in physics in different forms but holds the same fundamental principle that we cannot destroy or create matter. If there is more or less matter in a system after a set period of time, the flux of in and out of matter must reflect that. There are multiple ways to derive the continuity equation, but let's take the view of an infinitesimally small fluid with fixed mass m. This fixed element of fixed mass will have an arbitrary shape and volume that can change with the flow through the larger body of fluid. We can then define the relationship between the mass and volume as:

$$\delta m = \rho \delta V \tag{6.1}$$

Where ρ is the density, which is a function of space and time. Since mass is conserved in this infinitesimal fluid element we can make the mathematical statement:

$$\frac{D(\delta m)}{Dt} = 0 \tag{6.2}$$

Where D/Dt is the substantial derivative

$$\frac{D}{Dt} = \frac{\partial}{\partial t} + u\frac{\partial}{\partial x} + v\frac{\partial}{\partial y} + w\frac{\partial}{\partial z} \tag{6.3}$$

Note that u, v, and w are the velocity of the infinitesimal fluid in the x, y, and z direction, respectively.

Now let's combine equations (6.1) and (6.3) to get:

$$\frac{D(\rho \delta V)}{Dt} = \frac{D\rho}{Dt}\delta V + \rho\frac{D(\delta V)}{Dt} = 0 \tag{6.4}$$

$$\frac{D\rho}{Dt} + \rho[\frac{1}{\delta V}\frac{D(\delta V)}{Dt}] = 0 \tag{6.5}$$

As shown by Anderson in [75] this term on the right side, the substantial derivative of the volume, can be simplified by recognizing that the substantial derivative (or total time derivative) is equal to the flux of the velocity across the surface of the infinitesimal fluid.

$$\frac{D(\delta V)}{Dt} = \iint_S \boldsymbol{V} \cdot \boldsymbol{dS} \tag{6.6}$$

Then by applying the divergence theorem:

$$\frac{D(\delta V)}{Dt} = \iiint_{\delta V} (\nabla \cdot \boldsymbol{V}) dV = \delta V (\nabla \cdot \boldsymbol{V}) \tag{6.7}$$

The last step is achieved by recognizing that for an infinitesimal volume of fluid, the divergence of the velocity is essentially the same value throughout and therefore our continuity equation is:

$$\frac{D\rho}{Dt} + \rho(\nabla \cdot \boldsymbol{V}) = 0 \tag{6.8}$$

6.3.3 The Momentum Equation

The momentum equations are derived from Newton's Second Law. By expressing all the forces applied to an infinitesimal fluid element in Cartesian coordinates (a cube shaped infinitesimal fluid element), we can construct the momentum conservation equation. We start by recognizing all the types of forces that can be acted upon our fluid element; these are body forces (gravity, electric, and magnetic forces) and surface forces, τ, (pressure applied by external sources and the shear and normal stresses from the influence and friction of outside fluid).

Let's consider the sum of surface forces along the x direction. There are four shear forces from friction from the two y and two z faces, there is the pressure force on both x

faces, and two normal forces along the x faces resulting in:

$$\begin{aligned}\text{Net Surface Force} = [p - (p + \frac{\partial p}{\partial x}dx)]dydz + [(\tau_{xx} + \frac{\partial \tau_{xx}}{\partial x}dx) - \tau_{xx}]dydz \\ + [(\tau_{yx} + \frac{\partial \tau_{yx}}{\partial y}dy) - \tau_{yx}]dxdz + [(\tau_{zx} + \frac{\partial \tau_{zx}]}{\partial z}dz) - \tau_{zx}]dxdy\end{aligned} \tag{6.9}$$

The total force along the x direction then is:

$$F_x = [-\frac{\partial p}{\partial x} + \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z}]dxdydz + \rho f_x dxdydz \tag{6.10}$$

where f_x is the sum of body forces acting on the fluid element in the x direction. This formula can be cleaned up by utilizing the simple formula $m = \rho dxdydz$. The momentum equation in x, y, and z are the following:

$$\rho\frac{Du}{Dt} = -\frac{\partial p}{\partial x} + \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} + \rho f_x \tag{6.11}$$

$$\rho\frac{Dv}{Dt} = -\frac{\partial p}{\partial y} + \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \tau_{yy}}{\partial y} + \frac{\partial \tau_{zy}}{\partial z} + \rho f_y \tag{6.12}$$

$$\rho\frac{Dw}{Dt} = -\frac{\partial p}{\partial w} + \frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \tau_{zz}}{\partial z} + \rho f_x \tag{6.13}$$

This set of equations and the continuity equation derived in the previous section are known as the non-conservative form of the NS equations. These equations are known as the non-conservative form because they are derived from a moving fluid element, whereas the conservative form of the PDEs are derived from a fluid element fixed in space. Through manipulation one from can be achieved from the other, for the sake of succinctness we chose to only show one as they are very similar. Both sets can be found in [75].

6.3.4 The Energy Equation

The third and final equation in the NS equations is the energy conservation equation. Since the energy equation doesn't play a role in the Shallow Water Equations, we choose

not to go into as much detail in its derivation, as this body of work does not make use of it. For the sake of completeness we list the non-conservative form of the energy equation here:

$$\begin{aligned}\rho\frac{D}{Dt}(e+\frac{V^2}{2}) = \rho\dot{q} + \frac{\partial}{\partial x}(k\frac{\partial T}{\partial x}) + \frac{\partial}{\partial y}(k\frac{\partial T}{\partial y}) + \frac{\partial}{\partial z}(k\frac{\partial T}{\partial z}) - \frac{\partial up}{\partial x} - \frac{\partial vp}{\partial y} - \frac{\partial wp}{\partial z} \\ + \frac{\partial u\tau_{xx}}{\partial x} + \frac{\partial u\tau_{yx}}{\partial y} + \frac{\partial u\tau_{zx}}{\partial z} + \frac{\partial v\tau_{xy}}{\partial x} + \frac{\partial v\tau_{yy}}{\partial y} + \frac{\partial v\tau_{zy}}{\partial z} + \frac{\partial w\tau_{zx}}{\partial x} \\ + \frac{\partial w\tau_{zy}}{\partial y} + \frac{\partial w\tau_{zz}}{\partial z} + \rho\boldsymbol{f}\cdot\boldsymbol{V}\end{aligned} \tag{6.14}$$

e is the internal energy due to random molecular motion, T is temperature, q is heat transfer, and k is thermal conductivity.

Future work with geophysical models will necessitate the inclusion of the energy equation (or some simplification of it) to grasp the full nature of the geophysical medium we seek to study. For the purposes of this initial step into solving geophysical problems, the SWEs are a necessary first step to both test the capability of DDF and to set up the groundwork for future work in a geophysical context.

6.3.5 The Shallow Water Equations

The Navier Stokes equations we just discussed are fundamental to all of fluid dynamics, but are too difficult and complex to work with for our introductory problem, so we turn to Shallow Water Equations for an inviscid flow. The Shallow Water Equations (SWE) are a set of equations modelled after the NS equations, but make a couple of helpful assumptions and approximations to reduce the complexity and difficulty of the problem in both the generation of a data set and applying DDF to it. Applying DDF to the SWE will be the first important step in developing the utilization of DDF for regional weather forecasting.

We make three key approximations in using the shallow water model. The first is that we treat water as though it has constant density throughout, something that is not

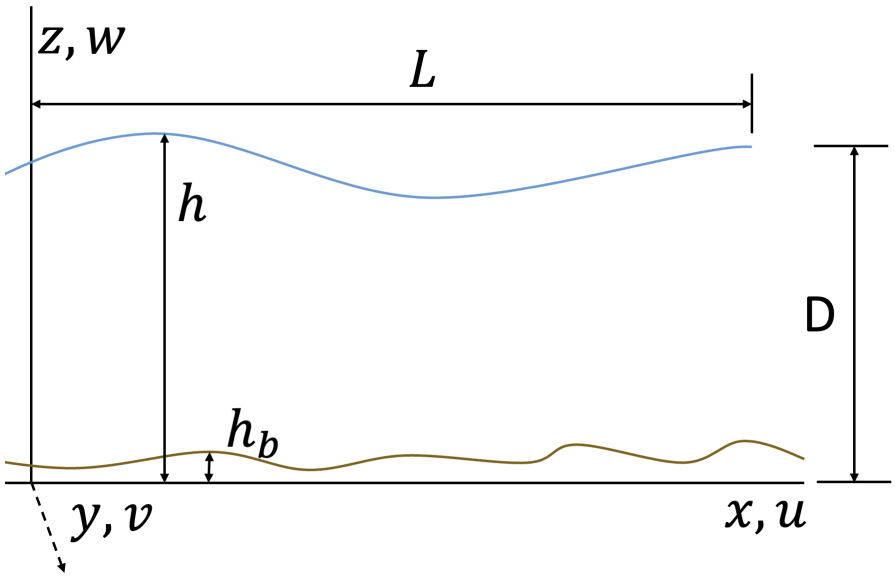

Figure 6.1. This image depicts our Shallow Water and how we describe fluid flow in the x, y, and z directions with velocities u, v, and w, respectively. As this is a Shallow water, L is much greater than D. It is also common in the literature to describe a non-constant floor, h_b, but for our purposes we usually set this to zero and work with a flat ocean floor.

true at all depths or even horizontally depending on impurities in the water. The second is that we are modelling an inviscid flow, meaning we are not accounting for viscosity in our equations to make them more tractable. The third approximation is, as the name describes, that the water we are modelling is shallow, it's width is far greater than its depth. For an average fluid depth, D, and length of the water, L, we assume for the SWE model:

$$\delta = \frac{D}{L} << 1 \tag{6.15}$$

With these tools in mind we now look to the continuity equation, or mass conservation equation, from NS and assert our assumptions of constant density:

$$\frac{D\rho}{Dt} + \rho(\nabla \cdot \boldsymbol{V}) = 0 \tag{6.16}$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \tag{6.17}$$

Since the density is constant, the substantial derivative is zero everywhere and drops out, which then allows us to cancel out the density and expand out the divergence of the velocities. Now this form may be sufficient for some purposes, but not ours, we wish to mold this mass conservation equality a bit more. For starters, we will want to describe a term as a function of time. With the foresight of knowing that the two momentum equations as derived later on will give us time derivative equations of the u and v velocities, we seek to use this equation to model the height of the fluid as a function of time (this quantity h is the height of the fluid, not to be confused with the location in space z). Additionally, we will want the equation to fit a specific form for later when we go to integrate through finite differencing means; we'll be using Robert Sadourny's integration scheme [76], so we want to derive our SWE to match the model he uses (the functionality of SWE will still be the same, but it will look different).

Before we can manipulate the mass conservation equation further, we must address the relationship between the z coordinate, which we wish to integrate over for both sides of the mass conservation equation, and horizontal velocities u and v. Pedlosky argues in [77] that through an analysis of the pressure gradients that u and v are independent of z. He starts by deriving a formula for the pressure. Using the shallow water approximation he comes to the following formula for the pressure:

$$p = \rho g(h - z) + p_0 \tag{6.18}$$

Where z is the distance from the ocean floor (naturally this equation only applies for $z \leq h$) and p_0 is the pressure at the surface of the water which we take to be zero.

$$\frac{\partial p}{\partial x} = \rho g \frac{\partial h}{\partial x} \tag{6.19}$$

$$\frac{\partial p}{\partial y} = \rho g \frac{\partial h}{\partial y} \tag{6.20}$$

With the horizontal pressure gradient calculated, we can see that there is no z dependence in it. Therefore, the accelerations will be independent of z (again, z is not the height of the fluid, rather a coordinate along the vertical z axis) and the velocities will be independent of z too so long as they are independent in their initiation. We can then integrate our mass conservation equation to get:

$$w = -z(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}) \tag{6.21}$$

We mentioned previously that we want to mold the equation to match the form that Sadourny used in his paper, so we can easily integrate the model using his integration scheme; therefore we will do a little bit of manipulation now (also note that Sadourny chooses to make pressure, not fluid height, his third state variable, with u and v being the other two).

$$w = \frac{\partial z}{\partial t} = \frac{1}{\rho g}\frac{\partial p}{\partial t} \tag{6.22}$$

$$\frac{\partial p}{\partial t} = -\rho g z(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}) = -p(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}) \tag{6.23}$$

Next we turn to the momentum equations, recall that we are solving the problem for an inviscid flow, meaning that the dissipative transport of viscosity and mass diffusion are neglected. For the momentum equation, this means we drop the stress tensor. Later on, we will discuss ways to bring back viscosity and friction through additional terms in the momentum equations, as well as how to account for the Coriolis force on a beta plane. The momentum equation is now (in the x direction):

$$\rho\frac{Du}{dt} = -\frac{\partial p}{\partial x} + \rho f_x \tag{6.24}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + f_x \tag{6.25}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + f_x \tag{6.26}$$

Where we used the result from before that the u and v velocities are independent of z, so the derivative drops out of the equation. From here, we will perform some final manipulations before coming to our desired output:

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} - v\frac{\partial u}{\partial y} - \frac{1}{\rho}\frac{\partial p}{\partial x} + f_x \tag{6.27}$$

$$\frac{\partial u}{\partial t} = v(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}) - \frac{\partial}{\partial x}(P/\rho + \frac{1}{2}(u^2 + v^2)) + f_x \tag{6.28}$$

$$\frac{\partial u}{\partial t} = vp\eta - \frac{\partial}{\partial x}(H) + f_x \tag{6.29}$$

$$\eta = \frac{1}{p}(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}) \qquad H = p/\rho + \frac{1}{2}(u^2 + v^2) \tag{6.30}$$

Using the same process, the momentum equation in the y direction is found to be:

$$\frac{\partial v}{\partial t} = up\eta - \frac{\partial}{\partial y}(H) + f_y \tag{6.31}$$

With equations (6.23), (6.29), and (6.31) we have our three fundamental equations of the SWE model using conservation of momentum and mass principles. Our next objective that is covered in the next section is to integrate them, or numerically solve them.

6.3.6 Solving the Shallow Water Equations

In our effort to show DDF as an applicable model to fluid dynamical systems, we must provide it with the data it needs to learn and forecast the behavior of a fluid system. To obtain this data, we'll need to carefully choose an integration scheme for the time evolving initial value PDE. Our principal concerns with choosing a proper integration scheme is the stability of the algorithm we choose as well as the ability for this algorithm

to conserve physical quantities like vorticity and energy (for conservative systems; we'll later discuss forcing terms and frictional terms that we'll add into the equations that will make the system non-conservative).

As described in Numerical Recipes [9], many reasonable looking algorithms can fail is due to being unstable. It is for this reason that many tests, like the Von Neumann stability analysis, are used to provide insight into whether an integration scheme will fail or not and under what circumstances it will fail (typically some relationship between the length of the time step and the distance between points on the grid). Recall that we are going to be solving a finite difference scheme on a square grid, as in figure(1.3). The partial derivatives with respect to space of the state variables at each location will be calculated, and must be done so in a formulation that is stable.

It is to our benefit that the meteorological community has already gone to great lengths of inventing many different solutions to the SWEs, we simply must choose one that fits our needs, and this would the formulation invented by Robert Sadourny in 1974 [76]. His is a finite differencing method, while these methods are known to be simple and computationally light, they can poorly represent the properties of the original equations unlike the more complex and computationally intensive spectral method for solving PDE's. Sadourny defends the finite difference method and argues that not all properties of original equations are equally important, and chooses to construct finite differencing schemes that focus on preserving the more important properties. He makes a comparison between conserving energy and conserving enstrophy and through a series of tests and references to literature, argues that the conservation of enstrophy is a very important property to achieve exact conservation with as it plays a central role in the model's ability to correctly exchange energy between modes and allow for the proper internal mixing process. Going forward, we will use and describe this integration scheme in all of our future experiments (note that enstrophy won't be conserved when we add forcing terms to our model).

The absolute potential enstrophy is defined as:

$$Z = \int_S \frac{1}{2}\eta^2 p dS \tag{6.32}$$

where η is the same as it was previously defined in equation(6.30), p is the pressure, and S is the surface. The simple expression for the absolute potential enstrophy in discrete form is:

$$Z = \frac{1}{2}\sum(\eta^2 < p >_x) \tag{6.33}$$

Where the pressure, p, is averaged across the x direction. This integration scheme uses averages across the x and y direction to achieve exact conservation of enstrophy. Some useful relations are listed below:

$$\delta_x q(x,y) = \frac{1}{2d}[q(x+d,y)] - q(x-d,y)] \tag{6.34}$$

$$\delta_y q(x,y) = \frac{1}{2d}[q(x,y+d)] - q(x,y-d)] \tag{6.35}$$

$$< q >_x (x,y) = \frac{1}{2}[q(x+d,y)] + q(x-d,y)] \tag{6.36}$$

$$< q >_y (x,y) = \frac{1}{2}[q(x,y+d)] + q(x,y-d)] \tag{6.37}$$

Where d is the distance between nodes on our square grid. Now using these terms we define what Sadourny calls a simple enstrophy conserving model [76]:

$$\frac{\partial u}{\partial t} =< \eta >_y < pv >_{xy} -\delta_x H \tag{6.38}$$

$$\frac{\partial v}{\partial t} =< \eta >_x < pu >_{xy} -\delta_y H \tag{6.39}$$

$$\frac{\partial p}{\partial t} = -\delta_x(pu) - \delta_y(pv) \tag{6.40}$$

With this formulation of the SWE PDEs, we can safely use a standard Runge-Kutta

method (typically the ODEINT program from the scipy library) to integrate these equations forward in time while assuming periodic boundary conditions. With these equations at our disposal, we can depart from the discussion of integrating PDE's and move towards one about how we will add terms to the SWE PDE. Specifically, we are interested in adding the Coriolis force for a rotating ocean, dissipative forces like viscous and frictional forces, and forcing terms like wind forcing; since our goal is to use this simpler model for testing DDF, we want to add these terms to showcase the resiliency of DDF to complex wave motions not seen in a conservative (and repetitious) SWE system without these forces.

The viscosity of a fluid comes from an analysis of the Reynold Stress Tensor, or shear forces, on the fluid's momentum equations. Through a crude and simple assumption that the stresses of the Reynold tensor depend linearly on the spatial derivatives of the large scale flow velocity (namely u and v for the SWE). This discussion goes much deeper in [77]. The viscous term in the SWE will take the following form then (Where A is an effective kinematic viscosity):

$$\boldsymbol{F}_{viscous} = A(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2})(\boldsymbol{u} + \boldsymbol{v}) \tag{6.41}$$

The drag force is the frictional force acting in the opposite direction of the motion of the fluid, unlike the viscous force, which acts on a fluid from all directions due to shear forces. The drag force can be simplified down to a fluid that is sufficiently slow as a force linear in velocity with a fluid dependent parameter ϵ, the Rayleigh friction coefficient:

$$\boldsymbol{F}_{drag} = -\epsilon(\boldsymbol{u} + \boldsymbol{v}) \tag{6.42}$$

Now we seek to implement the Coriolis force for our square and flat surface (as opposed to the spherical earth), but Rossby in (1939) characterized the dynamics in a simple manner for a sheet of fluid, called the β plane, on a sphere. His results state that

the only effect of the sphericity is the variation of the Coriolis parameter with latitude. We now list out these results:

$$f \approx f_0 + \beta_0 y \tag{6.43}$$

$$f_0 = 2\Omega \sin(\theta_0) \tag{6.44}$$

$$\beta_0 = \frac{2\Omega}{r_0} \cos(\theta_0) \tag{6.45}$$

Where θ_0 is the angle above (or below) the equator. The Coriolis Force we use is then:

$$\boldsymbol{F}_{Coriolis} = (f_0 + \beta_0 y)(v\hat{\boldsymbol{u}} + u\hat{\boldsymbol{v}}) \tag{6.46}$$

Now we move on to the final addition to our SWE, and that is the external forcing term. This term can contain a number of different sources, but for the sake of our tests we restrict ourselves to wind forcing. We choose to define our wind force as an x directional wind that varies along the y direction:

$$\boldsymbol{F}_{wind} = F_0 \cos(2\pi Y/L)\hat{\boldsymbol{u}} \tag{6.47}$$

Now we are able to put together the full set of equations that we will use for generating SWE data for the remainder of this section. Combining the prior equations derived for exact conservation of enstrophy with the added forces, we come to the following equations:

$$\frac{\partial u}{\partial t} = < \eta >_y < pv >_{xy} - \delta_x H + A\nabla^2 u - \epsilon u + (f_0 + \beta_0 y)v + F_0 \cos(2\pi Y/L) \tag{6.48}$$

$$\frac{\partial v}{\partial t} = < \eta >_x < pu >_{xy} - \delta_y H + A\nabla^2 v - \epsilon v + (f_0 + \beta_0 y)u \tag{6.49}$$

$$\frac{\partial p}{\partial t} = -\delta_x(pu) - \delta_y(pv) \tag{6.50}$$

For a depiction of how this shallow water evolves over time, see figures (6.2) and (6.3).

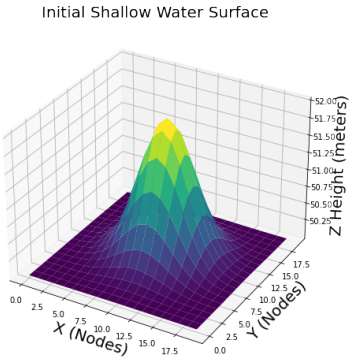

Figure 6.2. This is the initial condition for a 20 by 20 SWE grid described in equations(6.48,6.49,6.50). The initial condition is taken to be a Gaussian in the height, and the velocities are initialized to be a constant times the slope of height in the respective direction. The SWE parameters are chosen to be $f_0 = 10^{-5}$, $\beta = 10^{13}$, A $= 500$, $\epsilon = 8 * 10^{-7}$, $F_0 = 10^{-5}$

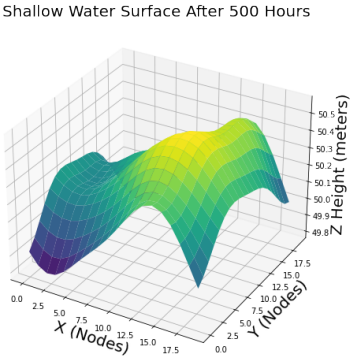

Figure 6.3. This is the same SWE from above, Figure(6.2), but has been evolved 500 hours ahead in time to show the influence of the forces on the water and how the initial condition has been transient as a result of the forces.

6.3.7 Regional Weather Forecasting

We now make some final remarks on the construction of regional data for our shallow water. Recall, our goal is not to study a General Circulation Model (GCM) for an entire globe, but rather to study a regional model without access to the knowledge of what is going on outside our region. In practice this would mean studying a section of ocean, or a port of economic importance, or even a section of coastline for environmental study; this enables us to use a high density of measuring devices giving us a high spatial resolution not seen in global models.

In the context of the SWE, we perform the data generation for the whole water on say some 10 by 10 grid with measurements of velocity in the horizontal direction and pressure, giving us three state variables at each grid point. Instead of supplying all 300 state variable data sets to DDF for forecast, we instead provide only a subset of the grid points to DDF and discard the rest. This will be our mock trial for conducting regional weather forecasting and as we don't have the full set of observations, we will need to perform time delay embedding to reconstruct the state space of the whole ocean to properly forecast the few parts of the ocean we do observe.

6.4 Shallow Water Flow on a β Plane in a DDF Context

We reiterate the main goal of the Fluid Dynamics section is to address how we can use data from regional measurements alone, without knowledge of the underlying dynamical equations or knowledge of the states of the global system outside the subgrid, to allow us to forecast the regional state variables $[u(t), v(t), p(t)]$. In applying the DDF method developed here, one only needs measurements of the state variables we wish to forecast at spatial locations in a sub-region. There is no 'grid' where we must place the sensors for the desired observation. While we do not require information outside the

subgrid, we do require knowledge of the forcing, which is already known in the global formulation of the problem [73][74].

In this section, we'll take our existing formulation of DDF and mold it to one that matches the regional weather forecasting model. We'll adapt our function representation to match the dynamics of the SWE including polynomial terms and forcing terms, we'll discuss time delay embedding in a SWE context, and we'll discuss how the inclusion of forcing terms changes the cost function. This will all culminate together to show DDF's forecasting power in the next section when we go through a few examples of regional weather forecasting with DDF.

6.4.1 Dynamics on the Subgrid; The Regional Model

We initiate our considerations of using DDF in SWE by establishing the global SWE model in a compact form. There are $D_G = 3(n_x * n_y)$ ODEs for the states $\boldsymbol{X}(i,j,t) = [u(i,j,t), v(i,j,t), p(i,j,t)]$:

$$\frac{d\boldsymbol{X}(i,j,t)}{dt} = \boldsymbol{F}_{i,j}(\boldsymbol{X}(i,j,t),\theta) + [F(i,j,t),0] \tag{6.51}$$

Where F is the sum of all body forces acting on the fluid motion, and θ are the fixed parameters of the global system. $\boldsymbol{F}_{i,j}(\boldsymbol{X}(i,j,t),\theta)$ is the vector field of D_G (global) nonlinear differential equations for the shallow water flow.

Next, select a subgrid, our region where we make measurements. The locations in the observation region are denoted by $\boldsymbol{R} = R_0 + [R_{10} + I\delta x, R_{20} + J\delta y; I = 0,1,2,...,N_x - 1, J = 0,1,2,...,N_y - 1]; N_x \leq n_x, N_y \leq n_y$. It is in this region that we collect observations at a subset of the full complement of state variables $\boldsymbol{X}(I,J,t)$.

The observations are $\boldsymbol{O}(\boldsymbol{R},t) = \boldsymbol{O}(I,J,t) = [u(I,J,t), v(I,J,t), p(I,J,t)]$ which satisfy:

$$\frac{d\boldsymbol{O}(\boldsymbol{R},t)}{dt} = \boldsymbol{F}_{\boldsymbol{R}}(\boldsymbol{X}(\boldsymbol{r},t),\theta) + [F(\boldsymbol{R},t),0] \tag{6.52}$$

$$\frac{d\boldsymbol{O}(I,J,t)}{dt} = \boldsymbol{F}_{I,J}(\boldsymbol{X}(i,j,t),\theta) + [F(I,J,t),0] \tag{6.53}$$

The number of regional variables is $D_R = 3(N_x * N_y) \leq D_G$. $\boldsymbol{F_R}(\boldsymbol{X}(\boldsymbol{r},t))$ is the global vector field restricted to the region $\boldsymbol{R}$. It is a function of the states of the global dynamics $\boldsymbol{X}(i,j,t)$. Take careful note that the observed states still depend on the unobserved states in their differential equation term $\boldsymbol{F}_{I,J}$; this dependency is resolved through the use of time delay embedding in the DDF framework.

Using data from observations on the $\boldsymbol{O}(\boldsymbol{R},t)$, without knowledge of the vector field $\boldsymbol{F}_{I,J}(\boldsymbol{X}(i,j,t),\theta)$, we want to construct a discrete time dynamical rule which takes $\boldsymbol{O}(\boldsymbol{R},t)$ forward in time; $\boldsymbol{O}(\boldsymbol{R},t) \rightarrow \boldsymbol{O}(\boldsymbol{R},t+h)$. This discrete time dynamical map is our forecasting system for the region.

Observations are made at N_O times, $t_n = t_0 + nh; n = 0,1,...,N_O - 1$ giving us $\boldsymbol{O}(\boldsymbol{R},n)$ at all those times. These observations form a trajectory in $D_R \leq D_G$ dimensional space.

Now we integrate the regional dynamical equation over the interval $[t_n, t_n + h] = [t_n, t_n + 1]$. This gives us the flow of the D_R dimensional dynamical system, which we find to be:

$$\boldsymbol{O}(\boldsymbol{R},n+1) = \boldsymbol{O}(\boldsymbol{R},n) + \int_{t_n}^{t_{n+1}} dt'(F_{\boldsymbol{R}}(\boldsymbol{X}(\boldsymbol{r},t') + [F(\boldsymbol{R},t'),0])) \tag{6.54}$$

As we do not know the vector field $F_{\boldsymbol{R}}(\boldsymbol{X}(\boldsymbol{r},t))$, we must represent this integral over it using our DDF function representation tools. The integral over the external forces in the fluid in the subregion, $[F(\boldsymbol{R},t'),0]$, can be approximated as the values of the external forcing are known to us. The forcing of the fluid is an aspect of the flow that is not intrinsic to the fluid properties themselves, and it is important to observe that this external forcing is additive. Essentially, these are just Newton's equations of motion.

Utilizing both pieces of information, we rewrite our update rule, first by using the trapezoidal rule to solve the integral over the known forcing, and secondly, by representing

the integral over the unknown intrinsic fluid properties function as a fitted DDF function:

$$\boldsymbol{O}(\boldsymbol{R}, n+1) = \boldsymbol{O}(\boldsymbol{R}, n) + \boldsymbol{f}_{I,J}(\boldsymbol{O}(\boldsymbol{R}, n), \omega) + \frac{h}{2}[[F(I, J, n), 0] + [F(I, J, n+1), 0]] \quad (6.55)$$

The trapezoidal rule was used to approximate the integral over the known forces which should be quite adequate as the measurements are only known at intervals of size h. The ω represents all the fitted coefficients in the DDF function representation.

Now we must think of a way to appropriately model our function representation $\boldsymbol{f}_{I,J}(\boldsymbol{O}(\boldsymbol{R}, n), \omega)$. We select a method of representing this function using radial basis functions as described earlier in this dissertation. In using this method, we make the distinct choice to model the function representation as a sum of Gaussian radial basis function plus the first order polynomials of the observed state variables.

$$\boldsymbol{f}_{I,J}(\boldsymbol{O}(\boldsymbol{R}, n), \omega) = \sum_{q=1}^{N_c} \omega_{IJq} \phi(||\boldsymbol{O}(\boldsymbol{R}, n) - \boldsymbol{c}(q)||) + \sum_{l=1}^{\widehat{m}} \omega'_{IJl} p_l(\boldsymbol{O}(\boldsymbol{R}, n)) \quad (6.56)$$

$$\boldsymbol{f}_{I,J}(\boldsymbol{O}(\boldsymbol{R}, n), \omega) = \sum_{q=1}^{N_c} \omega_{IJq} e^{(-R||\boldsymbol{O}(\boldsymbol{R}, n) - \boldsymbol{c}(q)||^2)} + \sum_{l=1}^{\widehat{m}} \omega'_{IJl} p_l(\boldsymbol{O}(\boldsymbol{R}, n)) \quad (6.57)$$

The ω'_{IJl} and ω_{IJq} values are fitted during the training to learn the intrinsic behavior of the water. R is a hyperparameter that is chosen by the user of DDF to get the best performance out of DDF. The centers, $\boldsymbol{c}$ are chosen from the training data set using a K-means clustering algorithm to choose the N_c centers. The polynomial order only goes to one, so there will only be D_R terms in the polynomial summation. The reason we choose the polynomial order to be one and the reason we include a polynomial term at all is to match the behavior in the actual SWE. Since there exist single order polynomial terms in the SWE, we choose to include them in our function representation (albeit we include all state variables, not just the select few in the equations); this choice has rewarded us as testing has shown a noticeable increase to the predictive power of DDF (note that always

including polynomial terms won't always improve predictive power, it depends on the original equations that define the behavior of the dynamical system).

Now before we can plug our function representation into equation(6.55) and happily go about forecasting the weather, we must address the very important fact that we are studying a reduced dimensional observation, and as such, are working with incomplete dynamics. Before we can go any further with developing DDF for fluid dynamics, we must adapt our current method to utilize time delay embedding techniques to reconstruct the state space of our regional model.

6.4.2 Utilizing Time Delay Embedding

Addressing the question of developing a dynamical map for the regional subset $\boldsymbol{O}(\boldsymbol{R}, n)$ of our dynamical variables, we see that the observations are a projection from dimension $D_G \to D_R < D_G$. To proceed we require a space of state variables equivalent to the full state space, $\boldsymbol{X}(\boldsymbol{r}, t)$, so we must "unproject" $\boldsymbol{O}(\boldsymbol{R}, t)$ to a space equivalent to $\boldsymbol{X}(\boldsymbol{r}, t)$.

As discussed in previous sections, there exists a dynamical method for accomplishing this in the nonlinear dynamics literature. It rests on the fact that as the observed quantities move from some time $t - \tau$ to time t, they depend on all the state variables $\boldsymbol{X}(\boldsymbol{r}, t)$. Using time delays of the observed regional variables provides us the desired information on the unobserved state variables.

This suggests creating a D dimensional time delay embedding space with D_O observed dimensions and D_E sets of time delays to create the $D = D_O * D_E$ dimensional vector space:

$$\boldsymbol{TD}(\boldsymbol{R}, t) = [\boldsymbol{O}(\boldsymbol{R}, t), \boldsymbol{O}(\boldsymbol{R}, t - \tau), ..., \boldsymbol{O}(\boldsymbol{R}, t - (D_E - 1)\tau)] \tag{6.58}$$

This vector of time delays depends only on observed quantities in the region labeled

by $\boldsymbol{R}$ and their time delays. It is through those time delays that $\boldsymbol{TD}$ inherits information about the dynamics outside $\boldsymbol{R}$.

Using this time delay vector in the observed dynamics gives us:

$$\boldsymbol{O}(\boldsymbol{R}, n+1) = \boldsymbol{O}(\boldsymbol{R}, n) + \boldsymbol{f}_{I,J}(\boldsymbol{TD}(\boldsymbol{R}, n), \omega) + \frac{h}{2}[[F(\boldsymbol{R}, n), 0] + [F(\boldsymbol{R}, n+1), 0]] \quad (6.59)$$

It is useful to rewrite this in component form, where we will also unpack the function representation $\boldsymbol{f}$. The $\boldsymbol{O}(\boldsymbol{R}, t)$ are D_R dimensional, such that $\boldsymbol{O}(\boldsymbol{R}, t) = [O_\alpha(t); \alpha = 1, 2, ..., D_R]$. The dynamical map then becomes:

$$O_\alpha(\boldsymbol{R}, n+1) = O_\alpha(\boldsymbol{R}, n) + f_\alpha(\boldsymbol{TD}(\boldsymbol{R}, n), \omega) + \frac{h}{2}[[F(\boldsymbol{R}, n), 0] + [F(\boldsymbol{R}, n+1), 0]]_\alpha \quad (6.60)$$

$$f_\alpha(\boldsymbol{TD}(\boldsymbol{R}, n), \omega) = \sum_{q=1}^{N_c} \omega_{\alpha q} e^{(-R||\boldsymbol{TD}(\boldsymbol{R},n) - \boldsymbol{c}(q)||^2)} + \sum_{l=1}^{\widehat{m}} \omega'_{\alpha l} p_l(\boldsymbol{O}(\boldsymbol{R}, n)) \quad (6.61)$$

DDF will create D_R of these discrete time update rules; this update rule in equation(6.59) is the actual update rule that is used in obtaining all the results of the Fluid Dynamics section. Note that while we use the time delay embedding vector of $D_E * D_R$ dimensions in the radial basis functions, we do not use it for the polynomials; this was an arbitrary choice of the user to not include extra polynomial terms and this choice will stay constant throughout the entirety of the fluid dynamics section of this dissertation. Also note that the centers, $\boldsymbol{c}$ will also need to be $D_E * D_R$ as opposed to their previous D_R dimension to accommodate all the time delay vectors in $\boldsymbol{TD}$. The centers will be chosen again using a K-means clustering algorithm, but not on the original observed data, but the time delayed observed data.

In the next section, we'll have a brief discussion on training the ω values to fit to a data set.

6.4.3 Changes to the Cost Function

The use of Ridge Regression to minimize our cost function has already been discussed at length earlier in this dissertation, we do however want to make some quick remarks about slight changes to the cost function to accommodate a forcing term that doesn't have an associated fitting parameter in front of it (unlike in the Nuerodynamics section which does include a fitting parameter to its external forcing i.e. stimulus). In the typical fashion of using the update rule to inspire the cost function, we now write out our cost function in component form. The cost function for the α dimension of the observed dimensions:

$$\begin{aligned} C_\alpha(\omega) = \sum_{n=1}^{N} [(O_\alpha(\boldsymbol{R}, n+1) - O_\alpha(\boldsymbol{R}, n)) - f_\alpha(\boldsymbol{TD}(\boldsymbol{R}, n), \omega) \\ - \frac{h}{2}[[F(\boldsymbol{R}, n), 0] + [F(\boldsymbol{R}, n+1), 0]]_\alpha]^2 \end{aligned} \tag{6.62}$$

See here that we now have this additional term on the right, our forcing function, and that this term doesn't quite fit in with the function representation as it doesn't have any fitted parameter out in front of it. The way we handle this term is just to simply add it into the other terms without coefficients, the observed "flow" from state O_α at time n to n plus 1.

$$\begin{aligned} C_\alpha(\omega) = \sum_{n=1}^{N} [(O_\alpha(\boldsymbol{R}, n+1) - O_\alpha(\boldsymbol{R}, n) - \frac{h}{2}[[F(\boldsymbol{R}, n), 0] + [F(\boldsymbol{R}, n+1), 0]]_\alpha) \\ - f_\alpha(\boldsymbol{TD}(\boldsymbol{R}, n), \omega)]^2 \end{aligned} \tag{6.63}$$

$$Y(n) = (O_\alpha(\boldsymbol{R}, n+1) - O_\alpha(\boldsymbol{R}, n) - \frac{h}{2}[[F(\boldsymbol{R}, n), 0] + [F(\boldsymbol{R}, n+1), 0]]_\alpha) \tag{6.64}$$

$$C_\alpha(\omega) = \sum_{n=1}^{N} [Y(n) - f_\alpha(\boldsymbol{TD}(\boldsymbol{R}, n), \omega)]^2 \tag{6.65}$$

Now we have our equation in a familiar form as we have dealt with that out of

place forcing term. The rest of the minimization proceeds normally as outlined in the DDF section.

6.5 Results from the Example of the SWE on a β Plane

Now that we have addressed all the necessary changes DDF must make to accommodate the format of Regional Weather Forecasting, all the pieces are in place to begin performing predictions with DDF.

6.5.1 Clustered Sensor Region; 3x3 Corner

The shallow water equations were solved using Sadourny's method [76] with periodic boundary conditions on a global grid consisting of $n_x = 10$ and $n_y = 10$. We solved 300 PDE's to generate the time series data for the state variables, $\boldsymbol{X}(\boldsymbol{r}, t)$, on this grid. A regional subgrid with $N_x = N_y = 3$ was selected and on this subgrid we "measured" $[u(\boldsymbol{R}, t), v(\boldsymbol{R}, t), p(\boldsymbol{R}, t)]$ to form the $D_R = 27$ dimensional observation vectors, $\boldsymbol{O}(\boldsymbol{R}, t)$. The regional sensor locations are shown in Figure(6.4). We generated N = 15,000 time steps of data with h = 0.1 hours. $N_c = 1000$ centers were selected from these data. These were used in a Polynomial plus Gaussian RBF representation. 1000 hours of this data was used to train the RBF representation of the discrete time flow vector field, then, 500 hours of forecasts were made for the 9 regional state variables.

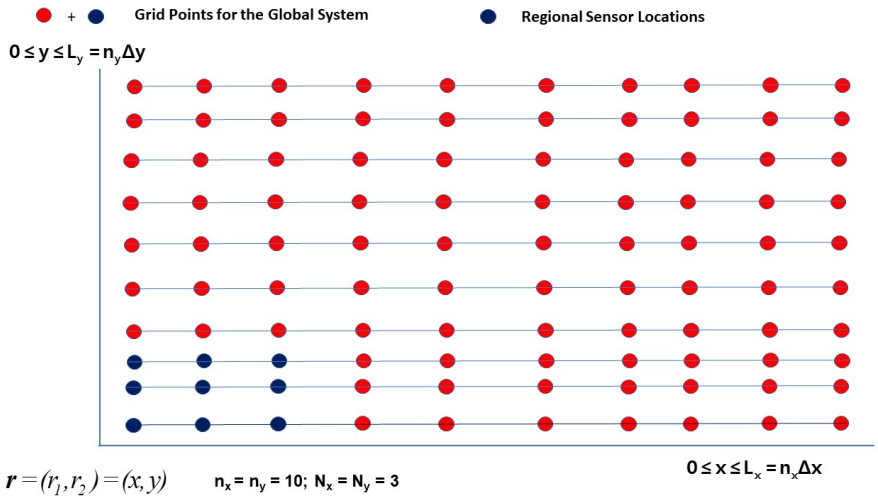

Figure 6.4. Shallow Water Equations on a $n_x = 10$ by $n_y = 10$ grid, $D_G = 3(n_x n_y)$). Red dots are grid points of the overall global dynamical system. Blue dots are the location of regional sensors, $\boldsymbol{R}$, where fluid pressure, $p(\boldsymbol{R}, t)$, and fluid velocity, $\boldsymbol{v}(\boldsymbol{R}, t)$, are recorded. There are $(N_x = 3) * (N_y = 3) = 9$ sensors total and $D_R = 3(N_x N_y) = 27$ measured time series. We have observed 27 regional time series out of the 300 global time series.

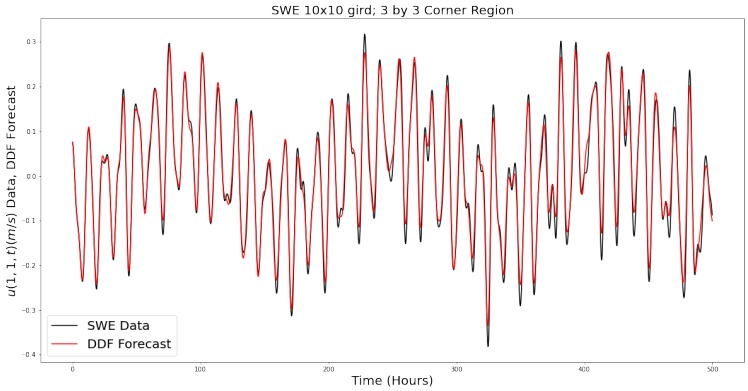

Figure 6.5. SWE on a 10 by 10 grid. The sensor region consists of the 9 blue dots in Figure(6.4), in a 3 by 3 corner region. We display the data and DDF forecast for the x-velocity, u(1,1,t). The time delay parameters are $D_E = 20$, $\tau = 20h$, h = 0.1 hour and the DDF parameters are $\beta = 10^{-9}$, $R = 10^{-6}$, $N_C = 1000$, and train time = 1000 hours.

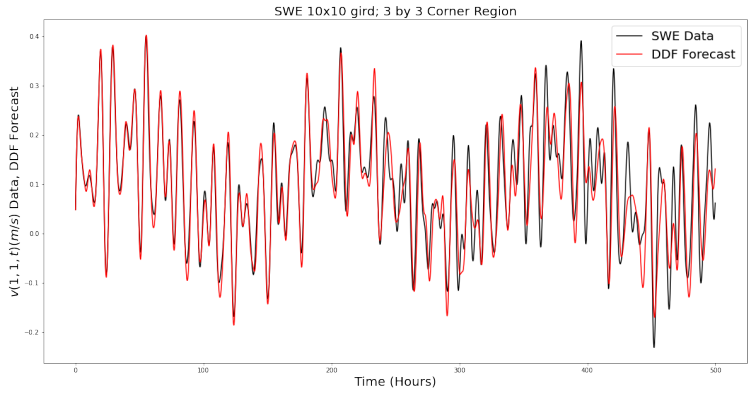

Figure 6.6. SWE on a 10 by 10 grid. The sensor region consists of the 9 blue dots in Figure(6.4), in a 3 by 3 corner region. We display the data and DDF forecast for the y-velocity, v(1,1,t). The time delay parameters are $D_E = 20$, $\tau = 20h$, h = 0.1 hour and the DDF parameters are $\beta = 10^{-9}$, $R = 10^{-6}$, $N_C = 1000$, and train time = 1000 hours.

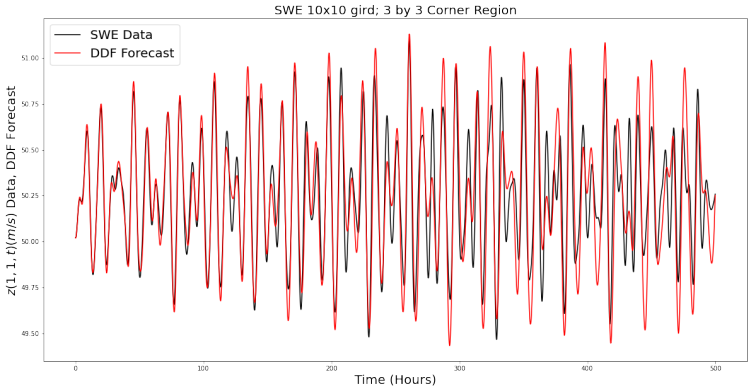

Figure 6.7. SWE on a 10 by 10 grid. The sensor region consists of the 9 blue dots in Figure(6.4), in a 3 by 3 corner region. We display the data and DDF forecast for the z height, z(1,1,t) (converted from the pressure equation $p = \rho g z$; also note that the base height is set to $H_0 = 50$ meters). The time delay parameters are $D_E = 20$, $\tau = 20h$, h = 0.1 hour and the DDF parameters are $\beta = 10^{-9}$, $R = 10^{-6}$, $N_C = 1000$, and train time = 1000 hours.

6.5.2 Off Center; 2x2 Region

This next test is to show off a different perspective of an even smaller region (2 by 2 as opposed to 3 by 3) and this time the data is just off center as opposed to in a corner. Displaying the predictability of all 12 dimensions is quite difficult to do in a compact and neat form, so we opt to show off one of the sensor locations forecasts for u, v, and z (the height).

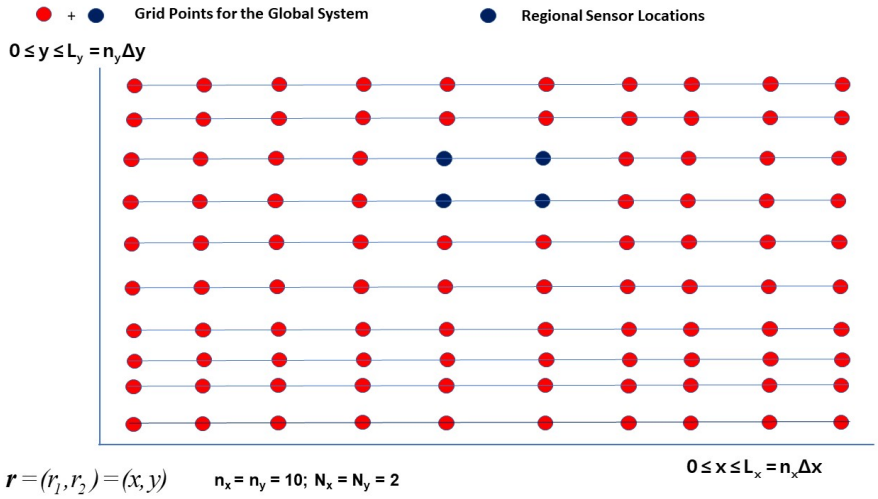

Figure 6.8. Shallow Water Equations on a $n_x = 10$ by $n_y = 10$ grid, $D_G = 2(n_x n_y)$). Red dots are grid points of the overall global dynamical system. Blue dots are the location of regional sensors, $\boldsymbol{R}$, where fluid pressure, $p(\boldsymbol{R}, t)$, and fluid velocity, $\boldsymbol{v}(\boldsymbol{R}, t)$, are recorded. There are $(N_x = 2) * (N_y = 2) = 4$ sensors total and $D_R = 3(N_x N_y) = 12$ measured time series. We have observed 12 regional time series out of the 300 global time series. Regional grid is 2 by 2 and is located off center within the global grid.

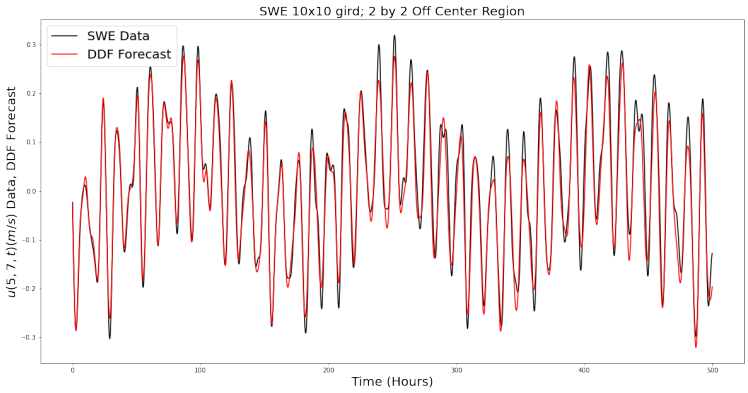

Figure 6.9. SWE on a 10 by 10 grid. The sensor region consists of the 4 blue dots in Figure(6.8), in a 2 by 2 off center region. We display the data and DDF forecast for the x-velocity, u(5,7,t). The time delay parameters are $D_E = 20$, $\tau = 18h$, h = 0.1 hour and the DDF parameters are $\beta = 10^{-7}$, $R = 10^{-5}$, $N_C = 1000$, and train time = 1000 hours.

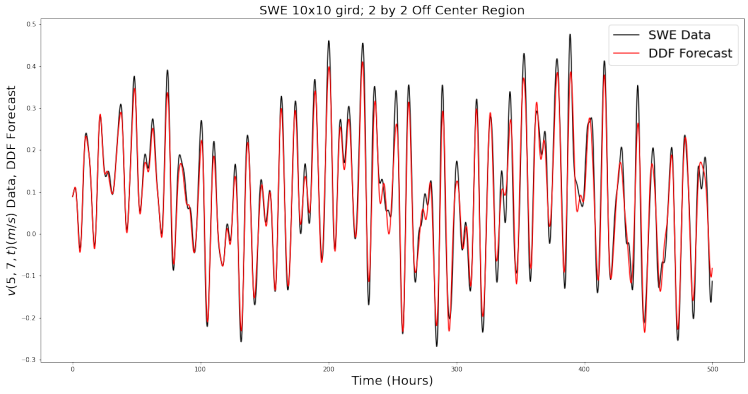

Figure 6.10. SWE on a 10 by 10 grid. The sensor region consists of the 4 blue dots in Figure(6.8), in a 2 by 2 off center region. We display the data and DDF forecast for the y-velocity, v(5,7,t). The time delay parameters are $D_E = 20$, $\tau = 18h$, h = 0.1 hour and the DDF parameters are $\beta = 10^{-7}$, $R = 10^{-5}$, $N_C = 1000$, and train time = 1000 hours.

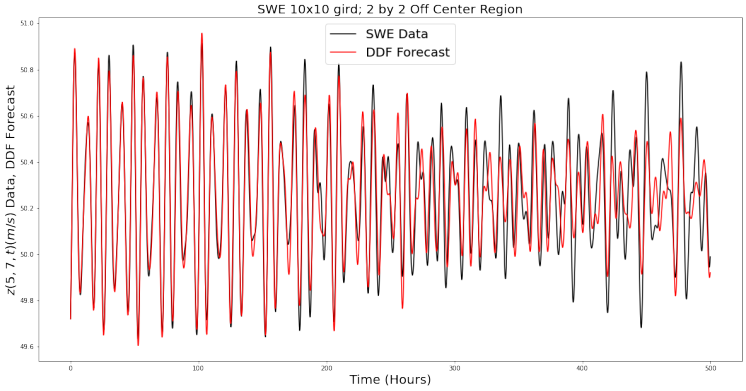

Figure 6.11. SWE on a 10 by 10 grid. The sensor region consists of the 4 blue dots in Figure(6.8), in a 2 by 2 off center region. We display the data and DDF forecast for the z-height, z(5,7,t). The time delay parameters are $D_E = 20$, $\tau = 18h$, h = 0.1 hour and the DDF parameters are $\beta = 10^{-7}$, $R = 10^{-5}$, $N_C = 1000$, and train time = 1000 hours.

6.5.3 Region With Sparse, Distributed Sensors

In the next set of outcomes for DDF regional forecasting, we now select our sensors to be dispersed over the 'global' region. Figure(6.12) shows ten sensors dispersed along the 100 grid locations of the global dynamical regime. The sensor locations were selected at random among the 100 possible sites available on the 'global' grid. The main purpose of this example is to demonstrate that the sensor sites where measurements are made, in the SWE example, need not all be contiguous.

The success of DDF forecasting using sensors in a broadly dispersed sensor region $\boldsymbol{R}$ suggests one could use the strategy to forecast in a quite broad geographical sub-region of a global dynamical system. We show the results for three different sensor points to display the forecasting strength at a variety of the sparse location, and that DDF doesn't become overtrained on any one dimension in particular in this sparse set up.

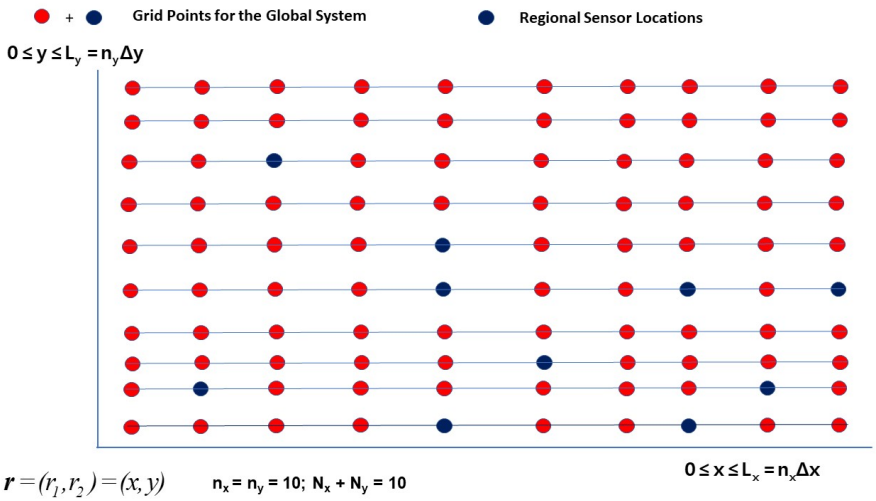

Figure 6.12. Sparse Regional Sensor Locations. Shallow Water Equations on a $n_x = 10$ by $n_y = 10$ grid, $D_G = 3(n_x n_y)$). Red dots are grid points of the overall global dynamical system. Blue dots are the location of regional sensors, $\boldsymbol{R}$, where fluid pressure, $p(\boldsymbol{R}, t)$, and fluid velocity, $\boldsymbol{v}(\boldsymbol{R}, t)$, are recorded. We have observed 30 regional time series out of the 300 global time series. The sensor region, indicated by blue dots, consists of 10 locations selected at random among 100 possible sensor locations.

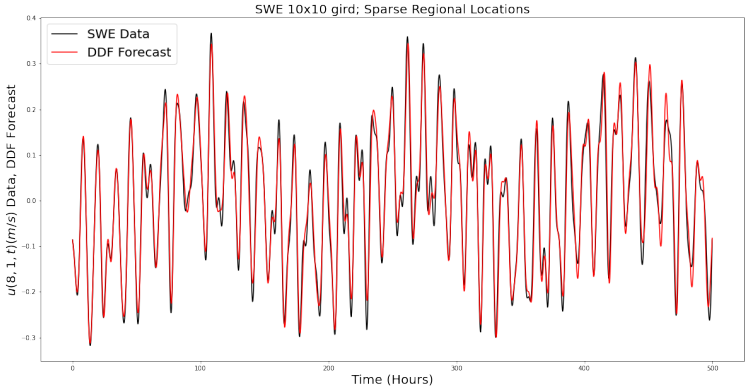

Figure 6.13. SWE on a 10 by 10 grid. The sensor region consists of 10 locations, blue dots in Figure(6.12), selected at random. We display the data and DDF forecast for the x-velocity, u(8,1,t). The time delay parameters are $D_E = 10$, $\tau = 20h$, h = 0.1 hour and the DDF parameters are $\beta = 10^{-9}$, $R = 10^{-6}$, $N_C = 1000$, and train time = 1000 hours.

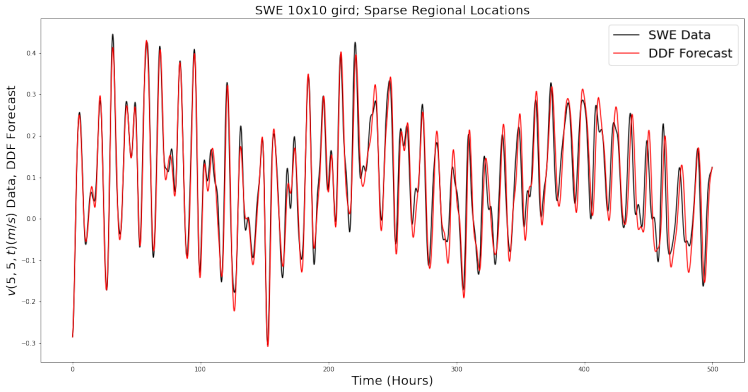

Figure 6.14. SWE on a 10 by 10 grid. The sensor region consists of 10 locations, blue dots in Figure(6.12), selected at random. We display the data and DDF forecast for the y-velocity, v(5,5,t). The time delay parameters are $D_E = 10$, $\tau = 20h$, h = 0.1 hour and the DDF parameters are $\beta = 10^{-9}$, $R = 10^{-6}$, $N_C = 1000$, and train time = 1000 hours.

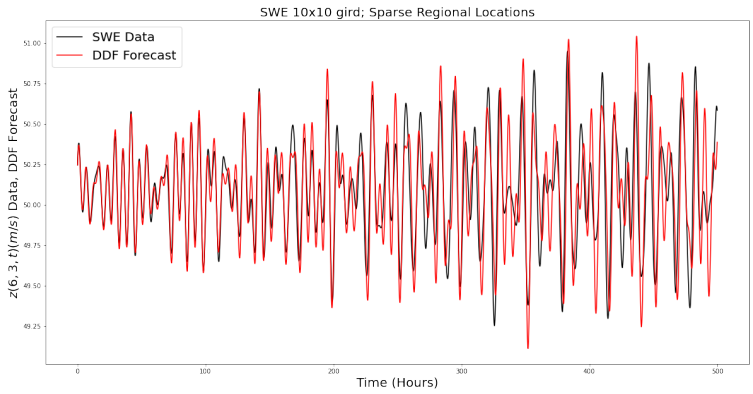

Figure 6.15. SWE on a 10 by 10 grid. The sensor region consists of 10 locations, blue dots in Figure(6.12), selected at random. We display the data and DDF forecast for the z-height, z(6,3,t). The time delay parameters are $D_E = 10$, $\tau = 20h$, h = 0.1 hour and the DDF parameters are $\beta = 10^{-9}$, $R = 10^{-6}$, $N_C = 1000$, and train time = 1000 hours.

6.6 Average Regional Error of Fluid Heights

We now discuss our forecasting results in a context that benchmarks their accuracy for future comparison with future machine learning models of a similar nature. We will take the three existing SWE forecasting examples we have examined in the text above and quantify their accuracy through the use of an average percent error equation. This equation is Equation(6.66) which averages the height z over a region $\boldsymbol{R}$ at each forecast time in the validation window. We define the regional average percent error as:

$$< z >_{\boldsymbol{R}} (t) = \frac{100}{\dim \boldsymbol{R}} \sum_{\boldsymbol{R}} \frac{|z(\boldsymbol{R}, t)_{\text{data}} - z(\boldsymbol{R}, t)_{\text{DDF Forecast}}|}{|\text{Max}(z_{\text{data}}) - \text{Min}(z_{\text{data}})|} \tag{6.66}$$

dim $\boldsymbol{R}$ is the number of observed z across the region $\boldsymbol{R}$. $z(\boldsymbol{R}, t)_{\text{DDF Forecast}}$ is achieved using the time delay embedding method previously described and $z(\boldsymbol{R}, t)_{\text{data}}$ is the observed data, but only the z dimension of observed data.

The maximum and minimum observed z values are taken across the entire region $\boldsymbol{R}$, and are not location specific. The $< z >_{\boldsymbol{R}} (t)$ results we show in Figures (6.16,6.17,6.18) show quite accurate results across a long time in the forecasting window. From data alone, DDF can build a model of the shallow water equation output at the regional sensor locations that forecast up to 500 hours with an average percent error in z that is less than 10% as shown in the 3 by 3 and sparse examples.

The motivation for choosing the denominator as the difference between the largest wave height and lowest wave height comes from the fact that we view the error DDF produces as originating from trying to follow the attractor in a set volume of phase space. The choice of largest minus smallest height sets the scale of DDFs operating range and thusly allows for the most relevant scaling of error that something like the magnitude of the height would fail to achieve.

This forecasting metric of forecasting performance is nearly the same as [68], and

both provide an informative look at the performance of a ML construction. The work of [68] analyzes an instantiation of the SWE of the SWE with neither forcing nor dissipation.

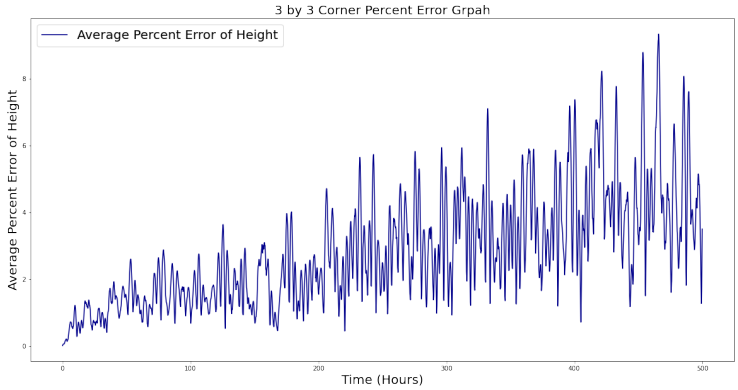

Figure 6.16. Using Equation(6.66) we evaluated $< z >_{\boldsymbol{R}} (t)$ for the data provided in Figure(6.7) for the 3 by 3 system.

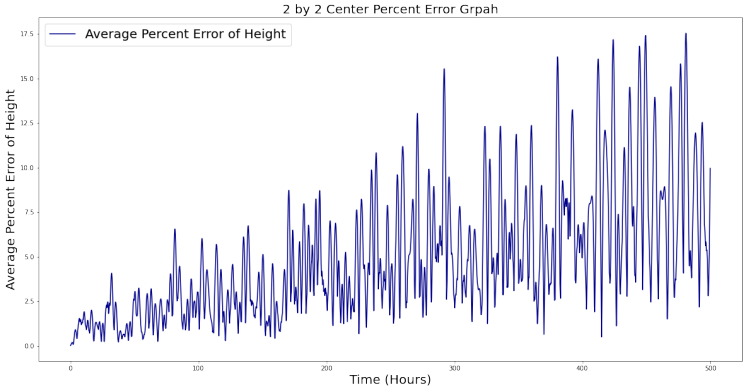

Figure 6.17. Using Equation(6.66) we evaluated $< z >_{\boldsymbol{R}} (t)$ for the data provided in Figure(6.11) for the 2 by 2 system.

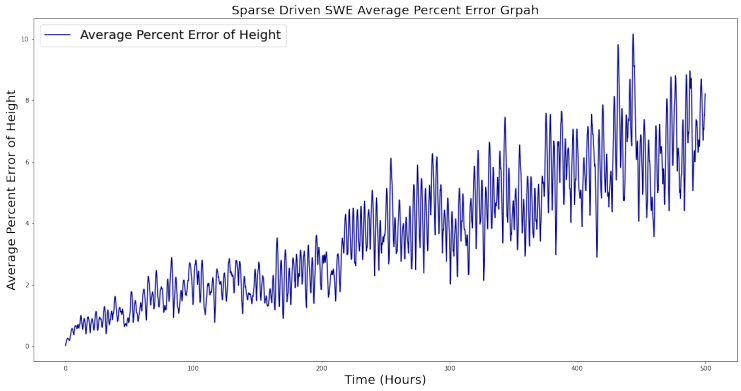

Figure 6.18. Using Equation(6.66) we evaluated $< z >_{\boldsymbol{R}} (t)$ for the data provided in Figure(6.15) for the sparse system.

6.7 Addressing Noisy Data

Our twin experiment examples using 'global data' from a shallow water flow have implicitly assumed we had data with a very high signal to noise ratio. Unlike the Neurodynamics section, which worked with real noisy data, our study of Fluid Dynamics has not dealt with noise yet. In this section, we take these very clean data and add Gaussian noise before constructing a DDF model of the regional dynamics.

We use 10 by 10 SWE data and focused on the 3 by 3 corner region. For each of the 27 time series in this region, we added Gaussian noise with zero mean and variance $S\sigma^2$ where σ is calculated independently for each of the 27 dimensions (this relative noise keeps dimensions with large standard deviations from having an unbalanced effect on dimensions with small deviations). $S\sigma^2$ is the variance of the signal $\boldsymbol{O}(\boldsymbol{R}, t)$ in the sensor region. In this configuration, the signal to noise ratio Signal/Noise = 1/S. For small S, the data is essentially noise free, as S approaches and exceeds unity, the noise level slowly overcomes the signal.

Three graphs have been chosen to be displayed as testaments to DDF's robustness to ever-increasing noise. We summarize the robustness of DDF to added noise in Figure(6.22) where the RMS error in the x-velocity is shown for the noise range level S we considered.

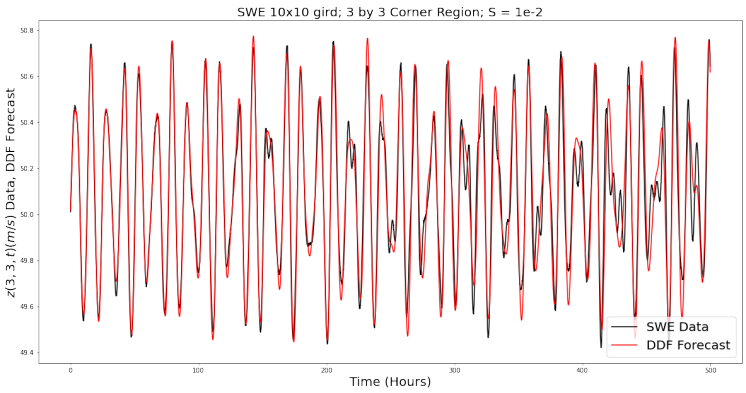

Figure 6.19. SWE on a 10 by 10 grid. The sensor region consists of 9 blue dots in a corner region. We display the z(3,3,t) noisy data and DDF forecast. S = 0.01

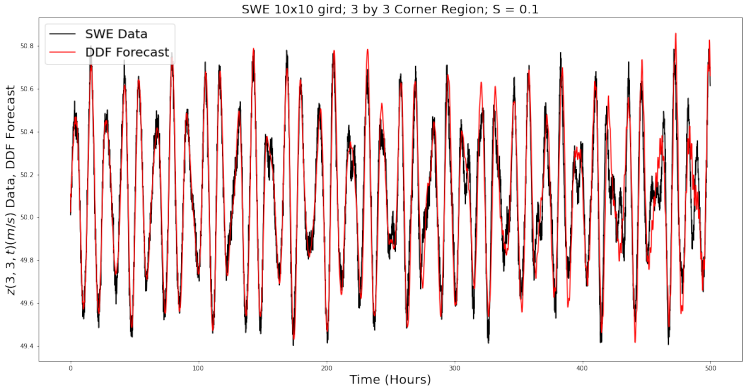

Figure 6.20. SWE on a 10 by 10 grid. The sensor region consists of 9 blue dots in a corner region. We display the z(3,3,t) noisy data and DDF forecast. S = 0.1

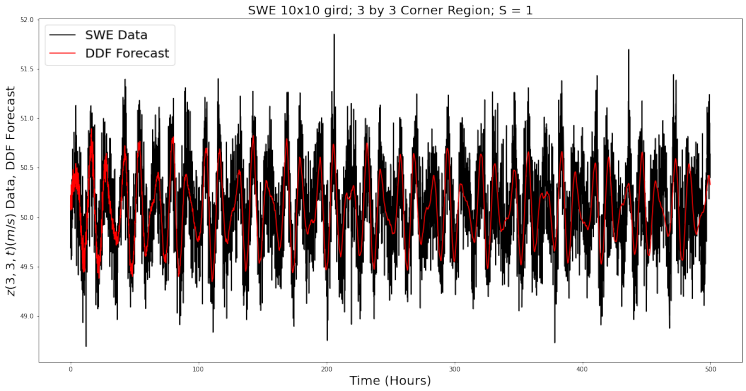

Figure 6.21. SWE on a 10 by 10 grid. The sensor region consists of 9 blue dots in a corner region. We display the z(3,3,t) noisy data and DDF forecast. S = 1

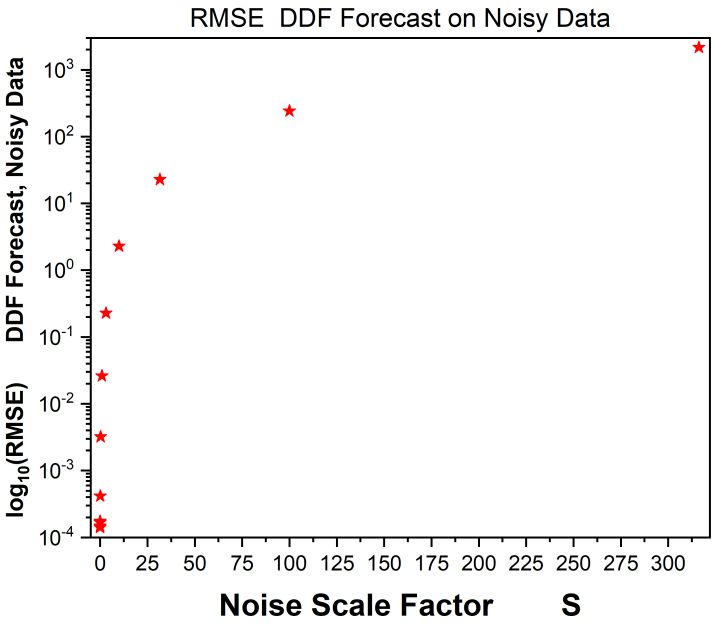

Figure 6.22. SWE on a 10 by 10 grid. The sensor region consists of 9 blue dots in a corner region. We display the RMS error in the x-velocity as a function of S over the range we considered.

6.8 Data Collection of Fluid Dynamical Systems

We had a discussion previously in the Neurodynamics section on the difficulties in making accurate observations of neuronal network behavior, fortunately, the observations of fluid dynamical systems are much easier and available to us. There are many different types of tools and measurements that are made in studying oceanic behavior beyond what we've done in the SWE, like wind speed (anemometers), wind direction (wind vanes), pressure (barometers), temperature (thermometers), salinity, conductivity, oxygen concentration, carbon dioxide concentration, pH (acidity), seawater density, and more. These measurements can be made by people on boats, drifters (buoyant objects that record currents as well as some of the observable quantities mentioned previously), moorings (a collection of devices anchored to the sea floor to recording measurements at various levels of height but always in the same location), weather balloon, and satellite [78].

As our interests in applying DDF are for regional purposes and with that comes some reliefs and some challenges. With DDF we can construct model dynamics to make forecasts on a grid in the SWE, we built into the assumptions that the state variables we are forecasting are at the same location in space at all observations; perhaps this seems obvious, but if we rely on drifters, we'd be dealing with the challenge of having a different observation at different locations in space at every data point meaning that each observation is of a different state variable as state variables vary with time but are rigid in space. This could mean drifters aren't reliable tools for dynamical model construction (neither would mobile weather balloons). Some organizations use data collected by boat cyclically throughout the year, but this comes with some challenges too; this data may be in the same spot (with some error which we can attribute to noise), but may not be often enough to fully flesh out the model dynamics as a function of time unlike the drifters that can record data continuously.

The most reliable form of data collection will come from mooring as they can

provide observations all the way from the air to the sea floor, support a wide variety of instrumentation, and unlike satellites are much easier to modify as needed; they also don't suffer from the issue of making data recordings in different locations as they are anchored (unlike drifters). The benefit of using systems like these are both economical and scientific; the ability to perform regional weather forecasting would aid in maritime coastal operations for economic purposes. Regional DDF would also be a useful tool in forecasting other important markers in the ocean that are being affected by both climate change and localized discharge of anthropologically enhanced nutrients[79]; cases like this show the importance of focusing on regional weather impacts in addition to the global climate. As of 2019 there are 370 moorings located around US coasts and lakes, this would mean that regional weather forecasting would likely be done with a single digit number of moorings for a coastal region, which as we've shown isn't a problem as long as time delay embedding is utilized [80].

6.9 Concluding Remarks on Fluid Dynamics

We have introduced a method for building a nonlinear discrete time forecasting system for observed state variables in geophysical dynamical settings where the underlying model of the dynamics is not known. The method relies on observed data to train model parameters in a representation of the unknown dynamical rules. Time series of the data are considered to be samples of the distribution in state space visited by the trajectories of the selected physical variables, and the representation of the vector field of the dynamical flow uses well tested interpolating basis functions to give information among the observed samples.

The method for representing the unknown dynamics of the physical flow adopted in our work uses weighted linear combinations of radial basis functions, and the estimation of the weights is a linear algebra problem. This linear algebra operations require a Tikhonov

regularization, also known as ridge regression [81][82]. These methods and algorithms implementing them are well tested and widely discussed in the literature.

Selection of observations in a region means that the full state space of the dynamics is not sampled. This entails a projection from the full state space to the subspace of selected observables. Construction of an equivalent state space to the unknown physical state space is, in effect, an unprojection from the observed subspace. This is accomplished using time delay embedding, something that is widely analyzed in the nonlinear dynamics literature.

In the example of DDF regional forecasting we investigated in this section, we took as our global dynamics the PDEs of shallow water flow realized on a rectangular mid-latitude plane. Clearly this is a useful, but major reduction in complexity compared to the collection and set of data from real field observations. We have found it instructive to begin in this manner, and the results illustrate the issues to be encountered in a practical application of the formulation.

In this framework we demonstrated in a number of scenarios that observations of a subset of 'global' shallow water flows can be used to build a discrete time flow $\boldsymbol{O}(\boldsymbol{R}, t) \rightarrow \boldsymbol{O}(\boldsymbol{R}, t+h)$ allowing for accurate forecasting beyond a temporal domain where data has been previously collected.

In investigating the examples we presented, we were required to generate our own data, performing what is often called a twin experiment [83] at a choice of grid points used to approximate solutions to the SWE PDFs. As one proceeds to using our results to guide DDF weather forecasting, at no stage do we approximately solve some physical PDEs on a grid of some spatial resolution, we have no restrictions on where the regional measurements may be performed. In working with observed data, there are no grid points. These are only introduced to aid in the solution of the PDEs of a physical model.

The gains of the DDF models must be recognized as having been only achieved by relinquishing many of the details about earth systems processes that are captured by

detailed physically based PDEs. In DDF only the observed state variables can be forecast; however, when the forecast is all one needs to know, DDF provides a way to forecast based on sampled data and forcing terms alone.

Chapter 6, has been adapted from a paper that is currently submitted for publication in the journal Physical Review Fluids. The material may appear as Randall Clark, Luke C. Fairbanks, Ramon E. Sanchez, Pacharadech Wacharanan, and Henry D.I. Abarbanel. "Data Driven Regional Water Forecasting: Example Using the Shallow Water Equations". Submitted to Physical Review Fluids 2023. Arxiv: https://arxiv.org/abs/2303.16363. The dissertation author was the primary investigator and author of this paper.

Chapter 7

Conclusion

The goal of studying and creating methods that help us analyze dynamical physical systems is to enable physicists to both glean insight into physical systems in ways that may not be experimentally available and to forecast the future behavior of the system, which is a goal at the heart of understanding the system in the first place. In our introduction to DDF we have shown the build up to and construction of DDF, we have shown how this hybrid model captures elements of the physics while not constrained by complex equations; it is not entirely a black box as it makes a clear distinction between the parts of the interpolation that are physics based and that which are machine learning based. DDF is accurate, flexible to different interpolation schemes, adaptable to external forces it wasn't trained on within limitation, robust to noise, computationally fast, parallelizable, it can preserve fractal dimension, it is capable of exhibiting chaos, and it is easy to implement. DDF's reconstruction of the state space through data alone, even with reduced dimensions, is capable, as depicted in its applications to Neurodynamics and Fluid Dynamics, of being a useful tool for physicists interested in studying and forecasting dynamical systems with the use of observed data alone.

The Neurodynamics section of this dissertation made calculations displaying DDF's robustness to real noisy data and resiliency to external forces not previously trained upon. The creation of DDF neurons gave us the tools to take what is now a long-lost neuron

and ask it how it would respond to new stimuli (within limitation) not previously shown to it during its life. In the field of Neurodynamics, the neuron networks are a hot area of research and DDF Neurons have shown to be capable of being duplicated and put in networks. This all comes together to give physicists the tools to seek greater understanding of their models and systems through exploitation of DDF and its model approximations.

If the NeuroDynamics section represents DDF's value to scientific research and deepening understanding, then the Fluid Dynamics section represents the practical application of DDF to forecasting problems. In the Fluid Dynamics section, we've seen how DDF excels in making accurate forecasts in a regional model with reduced dimensional observations. The application of Time Delay Embedding enables DDF to reconstruct the state space information that is lost to us by the projection of the state space to a lower dimension when fewer than maximal dimensions are observed. As described in the Time Delay Embedding section, the use of time delays enables us to reconstruct the statistical properties of the dynamics, like the attractor. As opposed to numerical weather prediction of the whole world, which is a computational task of tremendous proportion, we present regional weather forecasting with DDF as a substitute for those interested in local forecasting (even outside a fluid dynamical context). The ease of implementation of DDF makes it a viable candidate for this use.

Finally, we conclude our thoughts on DDF by looking to the future and what is next for DDF. While DDF is a completed project and ready for service, the search for improvement goes on. Different interpolators, more careful selection of centers, different radial basis functions, choosing the R hyperparameter in the Gaussian RBF to be nearest neighbor dependent, or even more fundamental adaptations to how the step from time n to n+1 is implemented in the update rule, these could all be areas of focus for future DDF researchers to explore as new and challenging data sets present themselves. In the area of finding data sets, research in NeuroDynamics continues as an active field of study and weather forecasting remains an area of both research and commercial interest. It is

ultimately the hope and intention that this work gains traction in the scientific community as a viable and easy to pick up method for scientific research.

Appendix A

A.1 Choosing Hyper Parameters

Upon obtaining a data set one wishes to perform DDF on, we come to the important question of choosing the best hyperparameters that will give us the best fit to the data. The hyperparameters in question that we have come across in this dissertation are beta, the ridge regression parameter that prevents over fitting, R, the radial basis function parameter, τ, the length of the time delay, and D_E, the number of time delay dimensions. We've discussed at length how to choose the time delay parameters, but haven't discussed how we came about choosing R and beta. We don't have much intuition for choosing beta and since it is data set dependent, it will always require some guesswork. A simple method for choosing R as mentioned by Wu [23] is to let R be determined by nearest neighbors; the R for each center will be individual and determined by the center's distance to the next nearest center (or some combination of n^{th} nearest neighbors). In our experience, this method requires more testing for DDF, as initial exploration didn't yield noticeably better forecasts. For the purposes of this paper, we relied on two methods for choosing R and Beta.

The first and easiest method is with basic grid sweeping. Even with τ and D_E we found that a little bit of grid sweeping could optimize DDF performance. This method is very brute force, we would choose R and beta scale by magnitude, usually starting

around 1e-12 and cycling through to 1e-1 netting us 144 tests when done for both beta and R (There is no rule of thumb where beta and R should succeed in, these parameters are highly data set dependent). This amount of testing grows ever larger when we start varying τ and D_E. Fortunately, DDF is a quick process, and depending on the number of dimensions (and time delay dimensions) and centers, the testing for hundreds or thousands of trials can range from minutes to hours. This method is easily parallelizable and jobs can be doled out to a super computer with ease. We have had much success with this simple brute force method, many of the results in this dissertation were found from this method as the R and beta values chosen (particularly from the NeuroDynamics and Fluid Dynamics section) are just orders of magnitudes.

The second method that we have employed is the Differential Evolution (DE) method [84][85]. When we were very serious about choosing the most optimized possible hyperparameter, we turn to DE. DE is an evolutionary algorithm, also known as a genetic algorithm, that creates a parent set of hyperparameters that it combines in random ways to create test sets of hyperparameters to compare with. If the newly created sets perform better than the parent, they become the new parent for the next round of tests, otherwise, they were tossed out. This evolutionary algorithm repeats this cyclical process repeatedly and ultimately finds a very precise set of hyperparameters that maximize forecasting potential for a data set. The two reasons we didn't employ it all the time is that it doesn't search across orders of magnitudes as well as grid searching does (this is why DE is best performed after grid searching as a refinement process), the second reason we didn't always use it is that it can cause DDF to overfit to the data set it is training on, which is bad if we are trying to create DDF neurons or DDF oceans that can extrapolate to new external forces. It is still a very useful tool for improving forecasting, and is something handy to keep in our bag of tricks for training difficult models.

A.1.1 Differential Evolution

DE is a powerful global optimization technique that that behaves similarly to an evolutionary algorithm and has gained much attention from many researchers and, as such, has had many alterations and variations made in an effort to improve it [84][85]. DE was built with a handful of useful properties; It was built to handle nonlinear, multi-modal, non-differentiable cost function, it's easily parallelizable, it's easy to program, and has good convergence properties. In this section, we'll briefly go over the basic DE method that we use.

As mentioned, DE is a search method that will utilize NP D-Dimensional parameter vectors:

$$\boldsymbol{x}_{i,G}, i = 1, 2, ..., NP \tag{A.1}$$

where G represents that this vector comes from the G^{th} generation and will be used to make the trial vectors for the G+1 generation. To make the trial vectors, we put the previous generation G vectors through three phases, mutation, crossover, and selection. The mutation phase is simple, we create NP mutant vectors from the G generation vectors:

$$\boldsymbol{v}_{i,G+1} = \boldsymbol{x}_{r_1,G} + F(\boldsymbol{x}_{r_2,G} - \boldsymbol{x}_{r_3,G}) \tag{A.2}$$

F is a constant typically chosen to be 0.5, r_1, r_2, and r_3 are all randomly chosen different vectors and not the i^{th} vector. It is for this reason that NP must be at least 4. The next step is the cross-over step, this is done to increase the diversity of the perturbed vector. With this step we make the trial vector $\boldsymbol{u}_{i,G+1}$ by going through each dimension of the i^{th} mutant vector, $\boldsymbol{v}_{i,G+1}$, and i^{th} parent vector, $\boldsymbol{x}_{i,G}$ and randomly choosing between which elements of each vector to take. The method of choosing goes as follows: for a D dimensional vector first choose RN to be an integer between 1 and D and choose CR to be a value between 0 and 1 (typically 0.9), then the j^{th} dimension of vector $\boldsymbol{v}_{i,G+1}$ will be a

component of the trial vector $\boldsymbol{u}_{i,G+1}$ if j = RN or a random number generated between 0 and 1 is less than or equal to CR. If these conditions aren't met, then the j^{th} dimension of the trial vector will be taken from parent $\boldsymbol{x}_{i,G}$.

With the crossover finished, we will have the i^{th} trial vector comprised of the i^{th} mutant and parent vector. The trial vector will then be compared against its parent vector using the greedy criterion, whichever vector provides a lower value for the cost function will become the next parent for the next generation $\boldsymbol{x}_{i,G+1}$.

In our work with DDF, we initially used a standard cost function, the error squared across averaged dimensions. This method tends to let us down however because of the large amounts of error that are gained from all time past the divergence between the forecast and test data. This doesn't frequently result in DE trying to increase the length of the prediction, rather it tries to change the shape of the prediction to minimize error, not taking into account the forecast.

There is an easy and simple modification we make to the standard DE model, instead of minimizing a cost function, we maximize the forecast length with a point system. We use a metric called the valid prediction time (VPT) to test how long the forecast is valid for and at the time it becomes no longer valid, that time is recorded and is labelled as the score for that trial. The score, or time length, is listed as a number of time steps taken before the forecasts becomes invalid. We will use DE to select vectors that maximize the VPT. The VPT in practice takes the following form [17]:

$$VPT(t) = \sqrt{\frac{1}{D}\sum_{d=1}^{D}[\frac{x_i^{DDF} - x_i^{\text{true}}}{\sigma_i}]^2} > \epsilon \tag{A.3}$$

D is the dimension of the system, σ_i is the standard deviation of the i^{th} dimension to normalize each dimension's time series data, and epsilon is an arbitrary threshold usually taken to be 0.3-0.5.

DE helps us choose our hyperparameters through an iteration of easily parallelizable

generations of trial vectors to maximize the VPT. DE's success in this through our small modification has earned it a discussion in this appendix.

A.2 Calculating Lyapunov Exponents

As described by Oseledec in his multiplicative ergodic theorem [12], the lyapunov exponents can be calculated from the log of the eigenvalues from the Oseledec matrix, however, the task of calculating the eigenvalues from the Oseledec matrix is not numerically trivial and is ill posed. This difficulty comes from the fact that the eigenvalues of the Oseledec matrix, $\boldsymbol{OSL} \sim exp(2\lambda t)$, making the max eigenvalue significantly greater than the rest as t goes to infinity (which is a requirement for the lyapunov exponents to converge to their true value). The solution is to make use of recursive QR decomposition to calculate the lyapunov exponents for large time scales [13, 86]. Let's look back at the Oseledec matrix and break it down into usable terms:

$$\boldsymbol{OSL}(\boldsymbol{x}, N) = ([\boldsymbol{T}^N(\boldsymbol{x})]^T \cdot \boldsymbol{T}^N(\boldsymbol{x})) \tag{A.4}$$

We know from linear algebra that the square of a matrix has that same matrix's eigenvalues but also squared, therefore we only need to calculate the eigenvalues of the whole Oseledec matrix, we only need to calculate them from $\boldsymbol{T}^N(\boldsymbol{x})$ and square root it (effectively this just means that instead of dividing the final result by 2N, where N is the total time, we will divide by N). Let's break down this form:

$$\boldsymbol{T}^N(\boldsymbol{x}) = \boldsymbol{T}(\boldsymbol{x}(n+N-1)) \cdot \boldsymbol{T}(\boldsymbol{x}(n+N-2)) \cdots \boldsymbol{T}(\boldsymbol{x}(n)) \tag{A.5}$$

This $\boldsymbol{x}(n)$ will be data taken from either our generated data set from numerical integration of model equations, or from DDF's generation of data trained upon generated data. Now recall that $\boldsymbol{T}$ is the Jacobian of the update rule, not of the dynamical system equations, this

means that if we take a simple update rule, say Euler's method $x_i(n+1) = x_i(n)+f_i(\boldsymbol{x}, n)*h$ where i is the i^{th} dimension of a dynamical system and h is the time step chosen, we can put this in place of the $\boldsymbol{F}$ that $\boldsymbol{T}$ uses in its Jacobian calculations. So say we have our dynamical system with D dimensions and D equations, $f_i(\boldsymbol{x}, t)$, then we can also describe the Jacobian of this system as $J_{ab}(\boldsymbol{x}) = \frac{\partial f_a(\boldsymbol{x})}{\partial x_b}$. Now let's put this all together to describe our $\boldsymbol{T}$ matrix:

$$\boldsymbol{T}(\boldsymbol{x}) = \boldsymbol{I}_D + h * \boldsymbol{J}(\boldsymbol{x}) \tag{A.6}$$

$$\begin{aligned}\boldsymbol{T}^N(\boldsymbol{x}(n)) = (\boldsymbol{I}_D + h * \boldsymbol{J}(\boldsymbol{x}(n+N-1))) \cdot (\boldsymbol{I}_D + h * \boldsymbol{J}(\boldsymbol{x}(n+N-2))) \\ \cdots (\boldsymbol{I}_D + h * \boldsymbol{J}(\boldsymbol{x}(n)))\end{aligned} \tag{A.7}$$

Now we have our series of matrices in known terms (which we assume the Jacobian is known or at least able to be estimated) but need a way to calculate the eigenvalues. To do this calculation, we'll recursively break up the right most matrix into it's QR decomposition, recombine the Q term with the next $\boldsymbol{T}$ matrix to be QR decomposed on the next round and collect the upper right-hand matrix for calculating the eigenvalues. In mathematical terms, the recursion is as follows:

$$\boldsymbol{T}(j) \cdot \boldsymbol{Q}(j-1) = \boldsymbol{Q}(j) \cdot \boldsymbol{R}(j) \tag{A.8}$$

We begin this recursion by setting $\boldsymbol{Q}(0) = \boldsymbol{I}_D$, the identity matrix. Then we perform the recursion as follows:

$$\boldsymbol{T}(0) = \boldsymbol{Q}(1) \cdot \boldsymbol{R}(1) \tag{A.9}$$

$$\boldsymbol{T}(1) \cdot \boldsymbol{Q}(1) = \boldsymbol{Q}(2) \cdot \boldsymbol{R}(2) \tag{A.10}$$

Now taking this to completion:

$$\boldsymbol{T}^N(\boldsymbol{x}) = \boldsymbol{Q}(N) \cdot \boldsymbol{R}(N) \cdot \boldsymbol{R}(N-1) \cdots \boldsymbol{R}(1) \tag{A.11}$$

The lyapunov exponents can then be read off as:

$$\lambda_a = \lim_{N\to\infty} \frac{1}{N \cdot h} \sum_{n=1}^{N} \log(R_{aa}(n)) \tag{A.12}$$

A.3 Calculating The Fractal Dimension

The method we use for calculating the fractal dimension is the Correlation Algorithm, a popular method commonly used for its ease of implementation. A special note is made for the reader to make them aware of the box counting algorithm that is also commonly used but will not be used in this dissertation. The Correlation Algorithm is best explained in parts, the first part of the algorithm we need is our density function. We will define a density function as a ration of the number of data points in a sphere of user chosen radius r divided by the total number of data points in our data set. This function is modeled as:

$$\eta(\boldsymbol{x}, r) = \frac{1}{N} \sum_{k=1}^{N} \theta(r - |\boldsymbol{y}(k) - \boldsymbol{x}|) \tag{A.13}$$

Where N is the number of data points in the data set, θ is the Heaviside function, and $\boldsymbol{y}$ is the k^{th} data point in the data set. Now this function alone is not going to be useful to us because it is not an invariant of the motion, as time evolves, and as we move through phase space, this density value will change. The solution is to average it over the phase space resulting in a function known as the "correlation function" specifically defined as:

$$C(q, r) = \int d^D x \rho(\boldsymbol{x}) \eta(\boldsymbol{x}, r)^{q-1} \tag{A.14}$$

$$C(q, r) = \frac{1}{N} \sum_{k=1}^{N} [\frac{1}{N} \sum_{n=1}^{N} \theta(r - |\boldsymbol{y}(k) - \boldsymbol{y}(n)|)]^{q-1} \tag{A.15}$$

Where, again, $\boldsymbol{y}$ is taken from the data set. This method is quite easy to implement, but requires some finesse in choosing a proper r value. Also note the q term is chosen to

be 2 for our purposes (this is a pretty standard and easy choice for q). In the limit that r is small but not too small that it ignore the finite nature of our data, it is assumed that:

$$C(q,r) \approx r^{(q-1)D_q} \tag{A.16}$$

From this equation we can arrive at a formula for the fractal dimension. The fractal dimension is now defined by the limit:

$$D_q = \lim_{\text{r small}} \frac{\log(C(q,r))}{(q-1)\log(r)} \tag{A.17}$$

In practice, the way we will calculate this term is by calculating multiple values of C(q, r) at different r values and plotting out the numerator and denominator of equation(A.17) to calculate the slope (granted this will fail at large r and very small r, so a data specific range of r must be found). [13, 14]

A.4 Code for the implementation

We did all our testing and programming in Python. We provide links to the code published in GitHub. In the links is a DDF program written specifically for Shallow Water Equations, Bio Physical Neuron Networks, and a more general overview of the basics of DDF. The General overview listed as "Thesis DDF Code" includes python scripts and example Jupyter notebooks detailing how the scripts are to be used for Radial Basis Functions, Taylor Series Expansion Methods, and Time Delay Embedding Dimensions. The SWE code includes the python script and Jupyter notebook used to get the results in the dissertation (and original paper) as well as code to solve the shallow water equations. Here we list links to the GitHub repositories which store these codes:

- https://github.com/RandarserousRex/Thesis-DDF-Code
- https://github.com/RandarserousRex/DDF-Applications-to-Neurons

- https://github.com/RandarserousRex/SWE-DDF

A.5 Memory Management

One important note to consider is the size of the matrices that are involved in the training process, for they can grow to the size of gigabytes. If one is caught unaware, they could rapidly run into memory issues. The matrix that grows most prodigious is the training matrix that collects all the values of all the RBF's at all times; say we are training on a data set with 25,000 time series data points in it and are using 5,000 centers to accurately forecast the data set, then if we're using float64 values (which we normally do), the total data consumption from this matrix is 25,000 x 5000 x 8 bytes = 1 Gigabyte. There will also need to be enough memory to take transposes and inverses of this matrix. This could be the limiting factor if one is running multiple tests in parallel on a CPU or on a cluster with limited storage.

A.6 Parallelizing the DDF Calculations

This section serves to comment on the parallelizability of DDF and its associated programs for future users of DDF may tackle bigger and more challenging problems that may consume excessive computer resources and time. For these larger tasks (or even some of the tasks tackled in this paper), the using of parallel processes can make running DDF much faster.

The first and most obvious case for parallelizability comes from the hyperparameter searching discussed earlier in the appendix. Both the grid sweeping method and differential evolution method run evaluations of DDF with different sets of hyperparameters that have no reliance or relation to the tests that come before or after. For this reason, both method of hyperparameter searching are extremely parallelizable, and it's easy to do so as long as one has the computers to split up jobs on.

The second form speed increase from parallelizing processes comes from the large matrix operations that DDF undergoes. The bulk of the training time in my tests comes from the large matrix multiplications that come from the Ridge Regression process; the prediction phase is just a loop of matrix multiplications (or at least it can be coded this way as matrix multiplication is one of the things Python can do quickly), and while the loop isn't parallelizable because each iteration depends on the iteration coming before it, the matrix multiplication in each step is parallelizable. Using GPU's, matrix multiplication can be sped up dramatically depending on the size of the matrix (with greater speed-ups for larger matrices)[87].

Bibliography

[1] Jason A Platt, Adrian Wong, Randall Clark, Stephen G Penny, and Henry DI Abarbanel. "Robust forecasting using predictive generalized synchronization in reservoir computing". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 31.12 (2021), p. 123118.

[2] Randall Clark, Lawson Fuller, Jason A Platt, and Henry DI Abarbanel. "Reduced-Dimension, Biophysical Neuron Models Constructed From Observed Data". In: *Neural Computation* 34.7 (2022), pp. 1545–1587.

[3] Randall Clark, Henry Abarbanel, Luke C Fairbanks, Ramon E Sanchez, and Pacharadech Wacharanan. "Data Driven Regional Weather Forecasting: Example using the Shallow Water Equations". In: *arXiv preprint arXiv:2303.16363* (2023).

[4] Philip Holmes. "A short history of dynamical systems theory: 1885". In: *History of Mathe-matics* (2010), p. 115.

[5] Steven H Strogatz. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. CRC press, 2018.

[6] Henri Poincaré. "Sur les équations aux dérivées partielles de la physique mathématique". In: *American Journal of Mathematics* (1890), pp. 211–294.

[7] Daniel R Creveling. *Parameter and state estimation in nonlinear dynamical systems*. University of California, San Diego, 2008.

[8] Leonhard Euler. "Institutionum calculi integralis volumen tertium". In: (1770).

[9] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[10] J-P Eckmann and David Ruelle. "Ergodic theory of chaos and strange attractors". In: *The theory of chaotic attractors* (1985), pp. 273–312.

[11] Edward N Lorenz. "Deterministic nonperiodic flow". In: *Journal of atmospheric sciences* 20.2 (1963), pp. 130–141.

[12] Valery Iustinovich Oseledec. "A multiplicative ergodic theorem, Lyapunov characteristic numbers for dynamical systems". In: *Transactions of the Moscow Mathematical Society* 19 (1968), pp. 197–231.

[13] Henry Abarbanel. *Analysis of observed chaotic data*. Springer Science & Business Media, 2012.

[14] James Theiler. "Estimating fractal dimension". In: *JOSA A* 7.6 (1990), pp. 1055–1073.

[15] C Daniel Meliza, Mark Kostuk, Hao Huang, Alain Nogaret, Daniel Margoliash, and Henry DI Abarbanel. "Estimating parameters and predicting membrane voltages with conductance-based neuron models". In: *Biological cybernetics* 108.4 (2014), pp. 495–516.

[16] Zheng Fang, Adrian S Wong, Kangbo Hao, Alexander JA Ty, and Henry DI Abarbanel. "Precision annealing Monte Carlo methods for statistical data assimilation and machine learning". In: *Physical Review Research* 2.1 (2020), p. 013050.

[17] J. A. Platt, Stephen G. Penny, Timothy A. Smith, Tse-Chun Chen, and Henry D. I. Abarbanel. "A Systematic Exploration of Reservoir Computing for Forecasting Complex Spatio-temporal Dynamics". In: *Neural Networks* 153 (2022), pp. 530–552.

[18] Mantas Lukoševičius. "A practical guide to applying echo state networks". In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 659–686.

[19] Andrea Ceni, Peter Ashwin, and Lorenzo Livi. "Interpreting recurrent neural networks behaviour via excitable network attractors". In: *Cognitive Computation* 12.2 (2020), pp. 330–356.

[20] Jiwen Li, Tim Waegeman, Benjamin Schrauwen, Herbert Jaeger, et al. "Frequency modulation of large oscillatory neural networks". In: *Biological cybernetics* 108.2 (2014), pp. 145–157.

[21] MJD Powell. "Radial basis function methods for interpolation to functions of many variables". In: *HERCMA*. Citeseer. 2001, pp. 2–24.

[22] D Broomhead and D Lowe. *Multivariable functional interpolation and adaptive networks, complex systems, vol. 2*. 1988.

[23] Yue Wu, Hui Wang, Biaobiao Zhang, and K-L Du. "Using radial basis function networks for function approximation and classification". In: *International Scholarly Research Notices* 2012 (2012).

[24] Maggie E Chenoweth and Scott A Sarra. "A numerical study of generalized multiquadric radial basis function interpolation". In: *SIAM Undergraduate Research Online* 2.2 (2009), pp. 58–70.

[25] Rolland L Hardy. "Multiquadric equations of topography and other irregular surfaces". In: *Journal of geophysical research* 76.8 (1971), pp. 1905–1915.

[26] Ke-Lin Du and Madisetti NS Swamy. *Neural networks in a softcomputing framework*. Vol. 501. Springer, 2006.

[27] V David Sánchez A. "Second derivative dependent placement of RBF centers". In: *Neurocomputing* 7.3 (1995), pp. 311–317.

[28] Floris Takens. "Detecting strange attractors in turbulence". In: *Dynamical systems and turbulence, Warwick 1980*. Springer, 1981, pp. 366–381.

[29] I Vlachos and D Kugiumtzis. "State space reconstruction from multiple time series". In: *Topics on Chaotic Systems: Selected Papers from Chaos 2008 International Conference*. World Scientific. 2009, pp. 378–387.

[30] Ricardo Mañé. "On the dimension of the compact invariant sets of certain nonlinear maps". In: *Dynamical Systems and Turbulence, Warwick 1980*. Springer, 1981, pp. 230–242.

[31] Andrew M Fraser and Harry L Swinney. "Independent coordinates for strange attractors from mutual information". In: *Physical review A* 33.2 (1986), p. 1134.

[32] Matthew B Kennel, Reggie Brown, and Henry DI Abarbanel. "Determining embedding dimension for phase-space reconstruction using a geometrical construction". In: *Physical review A* 45.6 (1992), p. 3403.

[33] Peter Grassberger and Itamar Procaccia. "Measuring the strangeness of strange attractors". In: *Physica D: nonlinear phenomena* 9.1-2 (1983), pp. 189–208.

[34] Elisabeth Larsson, Erik Lehto, Alfa Heryudono, and Bengt Fornberg. "Stable computation of differentiation matrices and scattered node stencils based on Gaussian radial basis functions". In: *SIAM Journal on Scientific Computing* 35.4 (2013), A2096–A2119.

[35] Alan L Hodgkin and Andrew F Huxley. "A quantitative description of membrane current and its application to conduction and excitation in nerve". In: *The Journal of physiology* 117.4 (1952), p. 500.

[36] David Sterratt, Bruce Graham, Andrew Gillies, and David Willshaw. *Principles of computational modelling in neuroscience*. Cambridge University Press, 2011.

[37] Peter J Olver. "Nonlinear ordinary differential equations". In: *Nonlinear Ordinary Differential Equations. https://doi. org/10.1201/9780203745489* (2017).

[38] Bryan A Toth, Mark Kostuk, C Daniel Meliza, Daniel Margoliash, and Henry DI Abarbanel. "Dynamical estimation of neuron and network properties I: variational methods". In: *Biological cybernetics* 105.3 (2011), pp. 217–237.

[39] Mark Kostuk, Bryan A Toth, C Daniel Meliza, Daniel Margoliash, and Henry DI Abarbanel. "Dynamical estimation of neuron and network properties II: path integral Monte Carlo methods". In: *Biological cybernetics* 106.3 (2012), pp. 155–167.

[40] Alain Nogaret, C Daniel Meliza, Daniel Margoliash, and Henry DI Abarbanel. "Automatic construction of predictive neuron models through large scale assimilation of electrophysiological data". In: *Scientific reports* 6.1 (2016), pp. 1–14.

[41] Zachary F Mainen and Terrence J Sejnowski. "Reliability of spike timing in neocortical neurons". In: *Science* 268.5216 (1995), pp. 1503–1506.

[42] Daniel Johnston and Samuel Miao-Sin Wu. *Foundations of cellular neurophysiology*. MIT press, 1994.

[43] Matt Carter and Jennifer C Shieh. *Guide to research techniques in neuroscience*. Academic Press, 2015.

[44] Jason N MacLean and Rafael Yuste. "Imaging action potentials with calcium indicators". In: *Cold Spring Harbor Protocols* 2009.11 (2009), pdb–prot5316.

[45] Jean-René Martin. "In vivo brain imaging: fluorescence or bioluminescence, which to choose?" In: *Journal of Neurogenetics* 22.3 (2008), pp. 285–307.

[46] Yuriy Mishchencko, Joshua T Vogelstein, and Liam Paninski. "A Bayesian approach for inferring neuronal connectivity from calcium fluorescent imaging data". In: *The Annals of Applied Statistics* (2011), pp. 1229–1261.

[47] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. "Convolutional LSTM network: A machine learning approach for precipitation nowcasting". In: *Advances in neural information processing systems* 28 (2015).

[48] Andrew C Comrie. "Comparing neural networks and regression models for ozone forecasting". In: *Journal of the Air & Waste Management Association* 47.6 (1997), pp. 653–663.

[49] Vladimir M Krasnopolsky, Dmitry V Chalikov, and Hendrik L Tolman. "A neural network technique to improve computational efficiency of numerical oceanic models". In: *Ocean Modelling* 4.3-4 (2002), pp. 363–383.

[50] Peter D Dueben and Peter Bauer. "Challenges and design choices for global weather and climate models based on machine learning". In: *Geoscientific Model Development* 11.10 (2018), pp. 3999–4009.

[51] Martin G Schultz, Clara Betancourt, Bing Gong, Felix Kleinert, Michael Langguth, Lukas Hubert Leufen, Amirpasha Mozaffari, and Scarlet Stadtler. "Can deep learning beat numerical weather prediction?" In: *Philosophical Transactions of the Royal Society A* 379.2194 (2021), p. 20200097.

[52] Radek Mackowiak, Lynton Ardizzone, Ullrich Kothe, and Carsten Rother. "Generative classifiers as a basis for trustworthy image classification". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 2971–2981.

[53] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. "A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability". In: *Computer Science Review* 37 (2020), p. 100270.

[54] Nils Wandel, Michael Weinmann, and Reinhard Klein. "Learning Incompressible Fluid Dynamics from Scratch–Towards Fast, Differentiable Fluid Models that Generalize". In: *arXiv preprint arXiv:2006.08762* (2020).

[55] Aditya Grover, Ashish Kapoor, and Eric Horvitz. "A deep hybrid model for weather forecasting". In: *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 2015, pp. 379–386.

[56] Arka Daw, Anuj Karpatne, William D Watkins, Jordan S Read, and Vipin Kumar. "Physics-guided neural networks (pgnn): An application in lake temperature modeling". In: *Knowledge-Guided Machine Learning*. Chapman and Hall/CRC, 2017, pp. 353–372.

[57] K Kashinath, M Mustafa, A Albert, JL Wu, C Jiang, S Esmaeilzadeh, K Azizzadenesheli, R Wang, A Chattopadhyay, A Singh, et al. "Physics-informed machine learning: case studies for weather and climate modelling". In: *Philosophical Transactions of the Royal Society A* 379.2194 (2021), p. 20200093.

[58] James Knighton, Geoff Pleiss, Elizabeth Carter, Steven Lyon, M Todd Walter, and Scott Steinschneider. "Potential predictability of regional precipitation and discharge extremes using synoptic-scale climate information via machine learning: An evaluation for the eastern continental United States". In: *Journal of Hydrometeorology* 20.5 (2019), pp. 883–900.

[59] Veronica Nieves, Cristina Radin, and Gustau Camps-Valls. "Predicting regional coastal sea level changes with machine learning". In: *Scientific Reports* 11.1 (2021), pp. 1–6.

[60] Changjiang Xiao, Nengcheng Chen, Chuli Hu, Ke Wang, Zewei Xu, Yaping Cai, Lei Xu, Zeqiang Chen, and Jianya Gong. "A spatiotemporal deep learning model for sea surface temperature field prediction using time-series satellite data". In: *Environmental Modelling & Software* 120 (2019), p. 104502.

[61] Serkan Kartal. "Assessment of the spatiotemporal prediction capabilities of machine learning algorithms on Sea Surface Temperature data: A comprehensive study". In: *Engineering Applications of Artificial Intelligence* 118 (2023), p. 105675.

[62] Said Ouala, Duong Nguyen, Lucas Drumetz, Bertrand Chapron, Ananda Pascual, Fabrice Collard, Lucile Gaultier, and Ronan Fablet. "Learning latent dynamics for partially observed chaotic systems". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 30.10 (2020), p. 103121.

[63] Georg A Gottwald and Sebastian Reich. "Combining machine learning and data assimilation to forecast dynamical systems from noisy partial observations<? A3B2 show [editpick]?>" In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 31.10 (2021), p. 101103.

[64] Charles D Young and Michael D Graham. "Deep learning delay coordinate dynamics for chaotic attractors from partial observable data". In: *arXiv preprint arXiv:2211.11061* (2022).

[65] Henry DI Abarbanel and Upmanu Lall. "Nonlinear dynamics of the Great Salt Lake: system identification and prediction". In: *Climate Dynamics* 12 (1996), pp. 287–297.

[66] Ahmad Kazem, Ebrahim Sharifi, Farookh Khadeer Hussain, Morteza Saberi, and Omar Khadeer Hussain. "Support vector regression with chaos-based firefly algorithm

for stock market price forecasting". In: *Applied soft computing* 13.2 (2013), pp. 947–958.

[67] Jordan Frank, Shie Mannor, and Doina Precup. "Activity and gait recognition with time-delay embeddings". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 24. 1. 2010, pp. 1581–1586.

[68] Süleyman Yıldız, Pawan Goyal, Peter Benner, and Bülent Karasözen. "Learning reduced-order dynamics for parametrized shallow water equations from data". In: *International Journal for Numerical Methods in Fluids* 93.8 (2021), pp. 2803–2821.

[69] Xiaoqian Chen, Balasubramanya T Nadiga, and Ilya Timofeyev. "Predicting shallow water dynamics using echo-state networks with transfer learning". In: *GEM-International Journal on Geomathematics* 13.1 (2022), p. 20.

[70] Peter J Olver. *Introduction to partial differential equations*. Springer, 2014.

[71] Bjorn Stevens, Masaki Satoh, Ludovic Auger, Joachim Biercamp, Christopher S Bretherton, Xi Chen, Peter Düben, Falko Judt, Marat Khairoutdinov, Daniel Klocke, et al. "DYAMOND: the DYnamics of the Atmospheric general circulation Modeled On Non-hydrostatic Domains". In: *Progress in Earth and Planetary Science* 6.1 (2019), pp. 1–17.

[72] Nils P Wedi, Inna Polichtchouk, Peter Dueben, Valentine G Anantharaj, Peter Bauer, Souhail Boussetta, Philip Browne, Willem Deconinck, Wayne Gaudin, Ioan Hadade, et al. "A baseline for global weather and climate simulations at 1 km resolution". In: *Journal of Advances in Modeling Earth Systems* 12.11 (2020), e2020MS002192.

[73] Erich Roeckner, G Bäuml, Luca Bonaventura, Renate Brokopf, Monika Esch, Marco Giorgetta, Stefan Hagemann, Ingo Kirchner, Luis Kornblueh, Elisa Manzini, et al. "The atmospheric general circulation model ECHAM 5. PART I: Model description". In: (2003).

[74] staff. *Ifs documentation-cy47r3; part iii: Dynamics and numerical procedures. Technical report, European Center for Medium Range Weather Forecasting, 2021.* URL: https://www.ecmwf.int/en/publications/ifs-documentation.

[75] John David Anderson and John Wendt. *Computational fluid dynamics*. Vol. 206. Springer, 1995.

[76] Robert Sadourny. "The dynamics of finite-difference models of the shallow-water equations". In: *Journal of Atmospheric Sciences* 32.4 (1975), pp. 680–689.

[77] Joseph Pedlosky et al. *Geophysical fluid dynamics*. Vol. 710. Springer, 1987.

[78] Francisco P Chavez, J Timothy Pennington, Robert Herlien, Hans Jannasch, Gary Thurmond, and Gernot E Friederich. "Moorings and drifters for real-time interdisciplinary oceanography". In: *Journal of Atmospheric and Oceanic Technology* 14.5 (1997), pp. 1199–1211.

[79] Faycal Kessouri, James C McWilliams, Daniele Bianchi, Martha Sutula, Lionel Renault, Curtis Deutsch, Richard A Feely, Karen McLaughlin, Minna Ho, Evan M Howard, et al. "Coastal eutrophication drives acidification, oxygen loss, and ecosystem change in a major oceanic upwelling system". In: *Proceedings of the National Academy of Sciences* 118.21 (2021), e2018856118.

[80] Kathleen Bailey, Craig Steinberg, Claire Davies, Guillaume Galibert, Marton Hidas, Margaret A McManus, Teresa Murphy, Jan Newton, Moninya Roughan, and Amandine Schaeffer. "Coastal mooring observing networks and their data products: recommendations for the next decade". In: *Frontiers in Marine Science* 6 (2019), p. 180.

[81] Marvin HJ Gruber. *Improving efficiency by shrinkage: the James-Stein and ridge regression estimators*. Routledge, 2017.

[82] Wessel N van Wieringen. "Lecture notes on ridge regression". In: *arXiv preprint arXiv:1509.09169* (2015).

[83] Henry DI Abarbanel. *The statistical physics of data assimilation and machine learning*. Cambridge University Press, 2022.

[84] Swagatam Das and Ponnuthurai Nagaratnam Suganthan. "Differential evolution: A survey of the state-of-the-art". In: *IEEE transactions on evolutionary computation* 15.1 (2010), pp. 4–31.

[85] Rainer Storn and Kenneth Price. “Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces”. In: *Journal of global optimization* 11.4 (1997), pp. 341–359.

[86] J-P Eckmann, S Oliffson Kamphorst, David Ruelle, and S Ciliberto. “Liapunov exponents from time series”. In: *Physical Review A* 34.6 (1986), p. 4971.

[87] Zhibin Huang, Ning Ma, Shaojun Wang, and Yu Peng. “GPU computing performance analysis on matrix multiplication”. In: *The Journal of Engineering* 2019.23 (2019), pp. 9043–9048.