UNIVERSITY OF CALIFORNIA SAN DIEGO

# Predictions of Chaotic Systems with Physical Models and Machine Learning

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

in

**Physics with Specialization in Computational Science**

by

Adrian S. Wong

Committee in Charge:

Professor Henry D. I. Abarbanel, Chair
Professor Daniel Arovas
Professor Kamalika Chaudhuri
Professor Alexander Cloninger
Professor Yi-Zhuang You

2022

The dissertation of Adrian S. Wong is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2022

DEDICATION

To my mom, who taught me so much while always believing in me.
To my stepdad, who has been an endless source of wisdom and confidence.
To my fiancée, who has been by my side, in the trenches, from the very beginning of this journey.

EPIGRAPH

## Curiosity

may have killed the cat; more likely
the cat was just unlucky, or else curious
to see what death was like, having no cause
to go on licking paws, or fathering
litter on litter of kittens, predictably.

Nevertheless, to be curious
is dangerous enough. To distrust
what is always said, what seems,
**to ask odd questions**, interfere in dreams,
leave home, smell rats, have hunches
do not endear cats to those doggy circles
where well-smelt baskets, suitable wives, good lunches
are the order of things, and where prevails
much wagging of incurious heads and tails.
Face it. Curiosity
will not cause us to die–
only lack of it will.
Never to want to see
the other side of the hill
or that improbable country
where living is an idyll
(although a probable hell)
would kill us all.
Only the curious
have, if they live, a tale
worth telling at all.

Dogs say cats love too much, are irresponsible,
are changeable, marry too many wives,
desert their children, chill all dinner tables
with tales of their nine lives.
Well, they are lucky. Let them be
nine-lived and contradictory,
curious enough to change, prepared to pay
the cat price, which is to die
and die again and again,
each time with no less pain.
A cat minority of one
is all that can be counted on
to tell the truth. And what cats have to tell
on each return from hell
is this: that dying is what the living do,
that dying is what the loving do,
and that dead dogs are those who do not know
that dying is what, to live, each has to do.

— Alastair Reid

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor Professor Henry Abarbanel for his support throughout my doctoral studies, and for his mentorship as I conducted my research. His encouragement to pursue my interests were invaluable to my younger, aimless self. In addition, his guidance on how to view the world from the 'perspective of a physicist' has shaped me not just as a researcher, but as a critical thinker.

I would also like to thank my dissertation committee: Professors Alexander Cloninger, Yi-Zhuang You, Daniel Arovas, and Kamalika Chaudhuri for both their time and willingness to accommodate my situation through the recent disruptions. They were incredibly patient and understanding as I figured my way through my research and the following dissertation process.

In addition to my committee, I would like to thank Professors Michael Holst and Melvin Leok for their teaching, support, and encouragement early in my graduate career. They were incredibly helpful and generous with their time as I transferred from the CSME program to Physics. They both bridged lot of the gaps between my understanding of mathematics and physics.

Additional thanks is in order to the folks at the Air Force Research Laboratory for providing employment during the final stretch of this dissertation. Their support and accommodations have had a direct impact on the quality and depth of the research leading up to this dissertation.

Last but certainly not least, I would also like to thank all the student affairs staff of the physics department. Their work, too often, goes unnoticed and is under-appreciated by many. Their willingness to help students, such as myself, successfully navigate their undergraduate and graduate studies is inspiring and deeply appreciated.

Chapters 2, 3, 4, 5, and 6, in part, use material and results that appear in Precision Annealing Monte Carlo Methods for Statistical Data Assimilation and Machine Learning, submitted in Physical Physical Review Research, 2(1), 013050. Fang, Zheng, Wong, Adrian S., Hao, Kangbo, Ty, Alexander J., and Abarbanel, Henry D. I. (2020). The dissertation author was one of the primary investigators and authors of this paper.

Chapters 7, 8, and 9, in part, use material and results that appear in Robust Forecasting Using Predictive Generalized Synchronization in Reservoir Computing, submitted in Chaos: An Interdisciplinary Journal of Nonlinear Science, 31(12), 123118. Platt, Jason A., Wong, Adrian S., Clark, Randall, Penny, Stephen G., and Abarbanel, Henry D. I. (2021). The dissertation author was one of the co-authors of this paper.

# VITA

University of California San Diego — 2014
*Bachelors in Science, Physics*

University of California San Diego — 2015–2022
*Teaching Assistant, Physics and Mathematics*

Lawrence Livermore National Laboratory — 2016
*Computation Intern, Summer*

San Diego Supercomputer Center (High Performance Geocomputing Laboratory) — 2017
*High Performance Computing Intern, Spring*

Lawrence Livermore National Laboratory — 2017
*High Energy Density Physics Intern, Summer*

University of California San Diego — 2017–2022
*Research and Teaching Assistant*

University of California San Diego — 2022
*Doctor of Philosophy, Physics with Speicialization in Computational Science*

Air Force Research Laboratory, In-Space Propulsion Branch — 2021–present
*Applied Mathematician*

# PUBLICATIONS

Platt, J. A., **Wong, A. S.**, Clark, R., Penny, S. G., and Abarbanel, H. D. I. (2021).
Robust Forecasting Using Predictive Generalized Synchronization in Reservoir Computing. Chaos: An Interdisciplinary Journal of Nonlinear Science, 31(12), 123118.

Fang, Z.*, **Wong, A. S.***, Hao, K.*, Ty, A. J., and Abarbanel, H. D. I. (2020).
Precision Annealing Monte Carlo Methods for Statistical Data Assimilation and Machine Learning. Physical Review Research, 2(1), 013050. (* co-first authors)

ABSTRACT OF THE DISSERTATION

**Predictions of Chaotic Systems with Physical Models and Machine Learning**

by

Adrian S. Wong

Doctor of Philosophy in Physics with Specialization in Computational Science

University of California San Diego, 2022

Professor Henry D. I. Abarbanel, Chair

Dynamical systems are an incredibly broad class of systems that pervades every field of science, as well as every aspect of daily life. Not only are they pervasive, but they often exhibit complex behavior resulting from microscopically simple interactions. Examples of such systems are the weather, our brains, animal population, and even home prices – to name but a few. As such, predictions of these systems pose a towering yet necessary challenge, and this work aims at making at dent in this effort. To the extent that models are available, they are useful in that constraints are automatically satisfied, and a mechanistic understanding of the system naturally follows. Part one of this work addresses this case by leveraging our knowledge of the physical model. However, it is often the case that the model is not known, so an effective surrogate model is desired. Part two proceeds in this vein, where the availability of large amounts of data is utilized in constructing surrogate models. Though the theory of dynamical systems still applies to the constructed surrogate model, this approach disregards the physics of the underlying system and has a machine learning flavor.

# 1

# Introduction

## 1.1 Motivations

Predicting the behavior of complex systems is surely one of the most important and ancient problems that humans have faced. Some examples of such complex systems are the weather, the economy, spread of disease, population ecology, social networks, to name but a few. If we can successfully predict the future, we will be able to reduce the infinite landscape of possibilities into a smaller, more manageable subset. Our ability to accurately predict the outcomes of events is certainly one of the reasons that civilization developed because it allows individuals and societies to effectively plan ahead and allocate resources accordingly, such that we can find and take full advantage of these momentary pockets of certainty. This exact paradigm exists till the present day – interest rates and disease modeling immediately come to mind – and all signs indicate that predicting the future is more desired now than ever before.

For the most general type of complex system, there can be many approaches to this prediction problem, some statistical and others more mechanistic [1]. In this work, we will focus almost entirely on physical systems where we have some concrete understanding of its behavior and causal structure. Predictions in non-physical systems (i.e. stock prices, artificial neural network outputs, etc.), while possible, deserved a separate treatment. The understanding of a particular (physical) system of interest comes from expert knowledge and consensus following years of careful study, which is codified in something called *the model*. Meanwhile, the scientific method dictates that we also collect *observa-*

*tions* from the real-world such that we have something to compare our carefully constructed models to. This contrast and tension between the models and the observations often generate strong and long-lasting disagreements, but is nevertheless the cornerstone of science and is absolutely necessary if we hope to make progress. If we have models with no data, we risk using models that are miss small but crucial details. Conversely, if we have data with no models, then the data itself cannot be utilized in a meaningful way, say for prediction. Once we get our hands on both a model and some data, we should ask: what now?

The systematic and quantitative discipline of combining models and data is called data assimilation [2, 3]. Another way of describing this combination is that we are transferring information from the observations to some interpreter of the model, generally a computer running code [4]. There might be multiple stated goals that data assimilation seeks to accomplish, but all these goals are ultimately serving one purpose – to predict the future[1]. If we have the ability to exert influence on the system of interest, we might then ask ourselves how we can shape the future to better suit our needs, wants, and desires.

## 1.2 Problem Statement

We are given a set of time-series measurements of a physical system, within a certain window of time. We then want to make predictions of the same system, that is to generate a time-series beyond the window of available measurements. The curator of the measurement data, which we shall refer to as the 'experimentalist', is privy to certain rudimentary information regarding the data. Examples of such are the sampling rate, understanding of the measurement devices, and the system that is being measured. For the most part, we will trust the experimentalist regarding the quality of the data set that was provided, though the distinction of quality is not always clear.

In the case where the experimentalist can propose or suggest a physical model that may describe the data measured, it makes sense leverage the information that the model can afford us, at least initially. We shall refer to this approach as data assimilation or model-based, and this will be the focus of the first half of this dissertation. There is also the case where the experimentalist

[1]This is a rather bold claim that requires some reflection and maybe even some justification. One stated goal of data assimilation is to retrieve the best approximation to the true state. Another stated goal would be to verify or validate certain physical model. In both cases, one should keep in mind that accumulating knowledge for the sake of knowledge is a rather romantic idea, but a purposeless one too. To the extent that knowledge is valuable, it should also be useful. The use case here, in the context of dynamical systems, is to correctly predict the (near) future and attempt to adapt to or change this future. Hence my claim.

cannot propose a plausible physical model, or if the previous approach failed for whatever reason. We shall refer to this approach as the data-driven or model-free. The phrase 'machine learning' comes up often, but it too often encompasses too many approaches [5]. The model-free approach, specifically Reservoir Computing, will be the focus of the second half of this dissertation. There are certain hybrid approaches that use both model-based and model-free approaches, but we do not cover such problem within this work [6].

The problem statement laid out here seems very basic, and in many ways, it is just that simple. However, it encompasses a large class of problems and therein lies the difficulty. Some of these methods, particularly the Reservoir Computing approach, is not yet well-understood and deserved further study. We understand that the formulation thus far has been imprecise, but the precision of the problem statement will be addressed in the relevant chapters.

## 1.3 Outline and Structure

Part I of the dissertation is on tackling the data assimilation problem with the aid of physics-based models. It begins with Chapter 2 introducing the theoretical foundations of our data assimilation approach by establishing the necessary integrals or cost functions to extremize. The following three chapters establishes the techniques and tactics that we employ in order to approximate and evaluate these integrals. We then follow up with one entire chapter on the heuristics that were used within (and in the lead up to) this work, as the heuristics prove to be a substantial boon to our productivity and understanding of the problems we hope to tackle. Lastly, we finish Part I with an entire chapter showing some of the results that come from our approach and techniques. The editorial choice of placing all the results in one chapter was done in order to avoid distractions from the theoretical foundations of the methods, and also for better side-by-side comparison.

The second part, also the last part, of the dissertation starts at Chapter 7 and focuses on model-free methods, specifically the reservoir computing approach. Reservoir Computing is often placed under the machine learning umbrella of methods, but when machine learning is applied to time-series data of physical systems, it is nearly identical in scope and in practice to data assimilation. The most prominent difference is that reservoir computing does not make use of physics-based models that describe the data, which has its own set of advantages and challenges that will be explored. We lay out the groundwork of Reservoir Computing and show some results. We then interpret the

method and its foundations from our own perspective of dynamical systems and synchronization of chaos, rather than the standard machine learning and applied mathematical framework.

The structure within each chapter is, in my opinion, the best way to arrive at the core ideas of said chapter assuming little familiarity with the material. My writing philosophy and style places more focus on brevity, clarity, readability, and the overall flow of ideas rather than rigor and completeness. This results in chapters that are relatively short and self-contained. My hope is that this dissertation is laid out in such a way that newcomers to the field find it as a helpful guide or introduction to how we approach these topics.

In the process of writing this work, I noticed many of the branching ideas that started in the main chapters began to feel misplaced within the chapter itself. These branches of thoughts were distracting and sometimes irrelevant to the chapter. My editorial choice was to gather all the short and off-shoot ideas and place them in the appendix. The topics in the appendix were included in the dissertation because they served as an intermediate but important step in shaping my thought process and intuitions, so I assume that these might be useful to some readers as well. It was incredibly helpful, for me, just to write and chronicle these ideas as they provided me invaluable nuggets of clarity and understanding. It seemed appropriate that these ideas that were unrelated to one another were left for the appendix, where there is no theme. A somewhat unfortunate consequence is that the appendix itself is comparable in length to the main parts of the work. Some of the ideas in the appendix warrant much further investigation, but I could not afford to do so if I hope to ever graduate. Still, I hope to have time to continue developing these ideas in the near future.

# Part I

# Model-based Methods

# 2

# Models, Data, and their Assimilation

## 2.1 Model and Observations

A model is the attempt at explaining the phenomena we perceive in the natural world[1]. Systems in nature are effectively infinite dimensional, but having an almost infinite degree of freedom renders computations extremely difficult or impossible. It is by reducing these infinite degrees of freedom into a much smaller subset that we are able to perform meaningful calculations in order to better understand and predict physical systems. By this definition, models have to be a finite-dimensional encoding of our knowledge and understanding. As an encoding, models will invariably omit some less relevant details of the problem in order to focus on explanatory power. Our focus in this work is on physical systems, so we will only really discuss models that explain the physical world, and these models almost exclusively exist in the form of differential equations. Models of physical systems arise from physical principles and should have all relevant physical laws (though often approximated) somehow being derivable or encoded into the equations. Designing a model often involves a good physical understanding of the system of interest from first principles [7].

Even a well-designed model is wrong at some scale, but being slightly wrong is the best anyone could have done given the practicality or the scope of the problem. We should recognize and accept that this is an inherent property of models. We shall assume that the models are nonlinear

[1] It is classified only as an attempt because no finite-dimensional model is able to capture the full complexity of reality. There are models that are extremely accurate and powerful, but every single model will break down at some point. Nevertheless, there is an agreed upon 'best' model by some metric, and whatever that model happens to be will be our choice.

and chaotic, since these are properties that may appear in the most general of cases. We shall also leave the specifics of the model to the expert that know them best. This allows us to focus on how to best make estimations and predictions (which is a difficult task by itself) without being overtly bogged down about the complexity of the models themselves.

Observations are on the other side of the divide, where they represent measurements of the physical system of interest. We will use the words *observations* and *data* interchangeably for the entirety of this work. In many a sense, the observations reflect the true nature of the system and exists independently of the simplified man-made models [8]. At the end of the day, however, these man-made models have to agree with the data, and the data always has the final say. If the data and the model do not agree, then it is likely the result of a faulty model. There are some corner case exceptions to this rule, but generally speaking the data is regarded as 'more correct'.

Invariably, the data will be exposed to some measurement noise that come from imperfect measurement devices. This type of noise is unavoidable realistically. Measurements are also collected by a separate set of experts. These experimental experts need to understand the nature of the system from a different point of view than the theorists in order to make good measurements. There are many questions here to address, one of which is how we measure certain state variable, and there is a fair amount of creativity involved in making good measurements. However, just like with the model, we will not concern ourselves too much with the details of the data, only that we trust the experts. Our focus will be on techniques that attempt to assimilate or merge good models with good data.

Even though the data and the model are technically unrelated to one another, they are both really describing the same natural phenomena. Hence, in order for a model to be deemed 'good', it should be able to reproduce the data in a meaningful and accurate way. Exactly how this is done varies from field to field. For a 'very good' model, the model not only need to reproduce the existing results, but also make prediction of yet-unseen phenomena. This is the true hallmark of a successful model.

## 2.2 Twin Experiments and Self-Imposed Handicaps

Real world systems and their measurements are infamously tricky to deal with [9]. A common situation is that one seldom has access to all state variables of a system, on top of the noise of those measurement. From the data assimilation point of view, such tasks are difficult, and we first require

some well-behaved testing grounds where we can test our techniques. Instead of real data, we use simulated data where one has full information of the system at all times, by definition. However, we purposely add noise to these measurements in order to replicate the noise that one would get from physically measuring the system. The true noiseless state is never presented to the data assimilation system and kept only for the sake of comparison. Only the noisy measurements are presented to the system for the purposes of estimation and prediction. Another common practice is to purposely omit certain variables of the twin experiment from the data assimilation program. This attempts to replicate the situations where the measurement device might be broken, or when the variables are unmeasured for practical reasons.

The ultimate purpose of these data assimilation methods would be applying them to real-world data, but running the twin experiments as mentioned here allows us to troubleshoot and develop the method, without dealing with the technical challenges of real-world data. After all, if any proposed method fails to work on simulated data in a twin experiment, then there is no hope of it working on real-world data. In other words, these twin experiments provide a training ground for new methods of data assimilation to be tested and evaluated.

Regardless of whether we are dealing with simulation or real-world data, another common practice is to withhold the latter segment (roughly half) of the time series for comparison. We start off with the full time series data or some system. The first segment of the data is fed into the computer for data assimilation, and it should not be surprising if the data assimilation method is able to reproduce this segment of the data. The true test is to see if the data assimilation method is able to make prediction beyond the data that it was given, which is the second segment of the data. The second segment of the data set is usually never given to the computer, but purposely withheld to see how well our data assimilation method has performed.

In the slightly related machine learning community, the way we use the first segment of the data is equivalent to the training set since it is presented to the computer explicitly. The second segment of the data is then equivalent to the validation set, where we validate the performance of our algorithm.

## 2.3 Dynamical Systems and State Estimates

The underlying dynamics of a deterministic and continuous time system can be described in the most general (hence nonlinear) sense in the following form.

$$\dot{x} = f(x, t) \tag{2.1}$$

The variable $x$ here is an $N$-dimensional vector that fully describes the state of system and the time-derivative of the state is given by $\dot{x}$, which is described as a function $f$ of the state $x$ and the time $t$. As such, the function $f(\,\cdot\,)$ is also called the model of the system. We shall assume that there are no parameters in the system for the moment. The scope and notation of the problem appears to handle the one-dimensional case, but the multi-dimensional extension is straightforward yet tedious, which is why it will be left for the appendix. The ultimate goal here is to assimilate (to be discussed in a later section) the model with measurements of the system, which are (if available) almost always as a set of discrete measurements in time. Hence, it would be prudent here to immediately consider the discrete time version of the dynamical system $x(t_m) = \{x_1, x_2, \cdots, x_M\}$ before proceeding further, and never looking back at the continuous counterpart.

$$\begin{aligned} x_{m+1} &= x_m + \int f(x_m, t)\, dt \\ &= F(x_m, t_m) \end{aligned} \tag{2.2}$$

The above integral can be evaluated with any numerical quadrature rule of choice, assuming that the time-interval between measurements is small enough. The choice of numerical quadrature depends on many things regarding the system, and should be evaluated on a cases-by-case basis. There is no requirement whatsoever that the time-intervals have to be uniform, but it is an assumption that is made here for the sake of simplicity in notation. The result is a dynamical system describes by a discrete map; it provides the rule to bring the state $x_m$ one step forward in time to $x_{m+1}$. The above map will also be called the model of the system since it is essentially a one-to-one correspondence to the continuous counterpart. This model, it turns out, only holds when the true state is evolving in time. Given that we will never have meaningful access to the true state $x$, we should assume that there might be uncertainties in the estimated state. We need to look at the best estimate of the true state $\hat{x}$ and what conclusions we may draw from it. The best estimate $\hat{x}$ is assumed to be somewhat

close to the true state $x$ such that $\hat{x} = x + \delta x$.

$$\begin{aligned} \hat{x}_{m+1} &= x_{m+1} + \delta x_{m+1} \\ \hat{F}(\hat{x}_m, t_m) &= F(x_m, t_m) + \boldsymbol{\nabla}_x F(x_m, t_m)\delta x_m + \mathcal{O}[\delta x_m^2] \end{aligned} \tag{2.3}$$

Following the claim that our best estimate is close to the true state, we write down the above equations by applying simple polynomial expansion. With the assumption that $\delta x$ is distributed according to the Gaussian distribution, the remainder of $\hat{x}_{m+1} - F(\hat{x}_m, t_m)$ should also be given by a slightly wider Gaussian distribution. This extra widening occurs due to the sum of two Gaussian distributed variables, and also due to the $\boldsymbol{\nabla}_x F(x_m, t_m)$ matrix. There are some $\delta x_m^2$ that are distributed by the beta distribution, but this is extremely small and negligible.

$$\hat{x}_{m+1} - F(\hat{x}_m, t_m) = \cancelto{0}{x_{m+1} - F(x_m, t_m)} + \underbrace{\delta x_{m+1} - \boldsymbol{\nabla}_x F(x_m, t_m)\delta x_m - \mathcal{O}[\delta x_m^2]}_{\simeq \xi} \tag{2.4}$$

The justification for this is the following – placing $\hat{x}_m$ into our forward map $F(\hat{x}_m, t_m)$ will not result in $\hat{x}_{m+1}$. Both $\hat{x}_m$ and $\hat{x}_{m+1}$ are to be estimated independently and the condition $\hat{x}_{m+1} = F(\hat{x}_m, t_m)$ holds if and only if $\hat{x}_m = x_m$ and $\hat{x}_{m+1} = x_{m+1}$ are both true. Since we expect that the true and estimated values will, even in the best case, deviate slightly from one another. All the unknown dynamics that come from using the forward map on the estimated state is conveniently encapsulated in the stochastic term $\xi$, which is distributed by the Gaussian $\mathcal{N}(0, \sigma^2)$ up to a good approximation.

$$\hat{x}_{m+1} - F(\hat{x}_m, t_m) \simeq \xi \tag{2.5}$$

We believe that this formulation is more general and more useful as well. We have taken the liberty to assume that the central limit theorem holds, such that the mismatch is that of white noise, i.e. $\xi \sim \mathcal{N}(0, \sigma^2)$, but none of this is strictly required for the formulation. The original equation for the evolution of the true state can be recovered from the evolution of the estimated state when $\sigma \to 0$. This formulation prescribes a probabilistic/stochastic approach to the problem and will allow us progressively refine our estimations. Readers familiar with stochastic processes will inevitably know the above equation as describing a drift-diffusion process, but this is not strictly true. Here, we are concerned really in the relation between two local-in-time states. The estimate is not actually going to evolve according to the above equation.

## 2.4 Probabilistic View of Dynamical Systems

From the previous section, we see that the estimate of the system, locally, evolve stochastically. In the spirit of this probabilistic view, one can write down (albeit trivially) the conditional probability of observing $x_{m+1}$ given $x_m$ for a deterministic system.

$$p(x_{m+1}|x_m) = \delta[x_{m+1} - F(x_m, t_m)] \tag{2.6}$$

Where $\delta[\,\cdot\,]$ is the Dirac delta function. The above formulation is almost completely tautological, but it gives a keen insight once certain relaxations are taken [4]. Specifically, the Dirac delta function is not a probability distribution in the strictest sense, but one can view it as the limiting case of a Gaussian distribution with near-zero standard deviation. The physicists' usual lack of formal mathematical rigor clearly shows in this approach, but hopefully the explanation is clearer and/or intuitive this way. Through this simple relaxation, we arrive at a more practical definition. The following equation describes the probability of observing some estimated state $x_{m+1}$ given $x_m$, where the deterministic limit $\sigma \to 0$ approaches a Dirac delta function.

$$p(\hat{x}_{m+1}|\hat{x}_m) = \frac{1}{Z} \exp\left[-\frac{1}{2}\left\|\frac{\hat{x}_{m+1} - F(\hat{x}_m, t_m)}{\sigma_{\text{model}}}\right\|^2\right] \tag{2.7}$$

The new variable $Z = \left(2\pi\sigma^2\right)^{-N/2}$ is the normalizing constant typical of statistical mechanics. From the above equation, one can easily see that given $\hat{x}_m$, the probability of $\hat{x}_{m+1}$ being valid is higher when $\hat{x}_{m+1} \simeq F(\hat{x}_m, t_m)$. This equation also implies that the process is Markovian, which is the assertion that the probability of an event is dependent only on the previous state. Finally, the above conditional probability is maximized when $\hat{x}_{m+1} = F(\hat{x}_m, t_m)$, which implies that the state estimate has converged to the true state, i.e. $\hat{x}_m \to x_m$. This formulation effectively states that the deterministic dynamical system $x_{m+1} = F(x_m, t_m)$ behaves much the same as a stochastic Markov process [10]. In other words, the dynamics in the state space of the system $x(t)$ may be nonlinear, but in probability space the transitions are linear. The tradeoff is that even though the transitions are linear, the probability space is infinite dimensional. Hence, there are no computational savings in this formulation, merely a different approach to use when thinking of such problems.

## 2.5 Markov, Chapman, and Kolmogorov

From here on, there is no need to talk about the true state $x$ in any meaningful way since there is no way to access that state without some measurement process, which induces noise. We will be replacing $\hat{x}$ with $x$ going forward. We shall consider the trajectory of a state as $x(t) = \{x_1, x_2, \cdots, x_M\}$ given as an ordered sequence of $M$ states $x_m$ that are each $N$-dimensional. In the usual treatment of dynamical systems, the Markov property is already tacitly assumed. Such an assumption is relatively harmless, yet it provides tremendous computational and analytical simplification. Treating a deterministic dynamical system from a probabilistic perspective outlined above, along with the Markov assumption, we can invoke the Chapman-Kolmogorov equation.

$$p(x_{m+1}) = \int dx_m \, p(x_{m+1}|x_m)p(x_m) \tag{2.8}$$

Starting from the final state $x_M$, we can iteratively apply the Chapman-Kolmogorov equation and the Markov property on the reversed-ordered states $x_m$ until we reach the initial state $x_1$ [11]. Once the entire expansion then simplification in done, we can collect almost all the terms into a product, as demonstrated below.

$$\begin{aligned}
p(x_M) &= \int dx_{M-1}dx_{M-2}\cdots dx_1 \, p(x_M|x_{M-1}, x_{M-2}, \cdots, x_1)p(x_{M-1}, x_{M-2}, \cdots, x_1) \\
&= \int dx_{M-1} \, p(x_M|x_{M-1})p(x_{M-1}, x_{M-2}, \cdots, x_1) \\
&= \int \underbrace{dx_{M-1}dx_{M-2}\cdots dx_1}_{(M-1)\text{-times}} \underbrace{p(x_M|x_{M-1})p(x_{M-1}|x_{M-2})\cdots p(x_2|x_1)}_{(M-1)\text{-times}} p(x_1) \\
&= \int \left[ dx_m \prod_{m=1}^{M-1} p(x_{m+1}|x_m) \right] p(x_1)
\end{aligned} \tag{2.9}$$

A very clear pattern emerges from the decomposition, and it can be written in the more compact and more insightful form. With a small change of perspective, it is simple to make a quick substitution below to recover the form resembling a path integral, which is familiar to physicists from its use in quantum mechanics. There is a key difference in the form below in that the exponent is entire real rather than imaginary. Fortunately, real exponents are handled in statistical physics and there are links between statistical and quantum physics can be drawn in order to build the necessary

intuition.

$$K[x(t)] = \prod_{m=1}^{M-1} p(x_{m+1}|x_m) \qquad ; \qquad \mathcal{D}x(t) = \prod_{m=1}^{M-1} \frac{dx_m}{Z} \tag{2.10}$$

$$p(x_M) = \int \mathcal{D}x(t) Z^M K[x(t)] p(x_1) \tag{2.11}$$

The extra factor of $Z^M$ is introduced with the purpose of having an all-encompassing normalizing constant. This normalizing constant $Z^M$ will not play any role in our eventual optimization procedure since it is, after all, a constant that scales the probability distribution function. It does not affect the relative shape of the function, nor the location of the extrema. It is also worth mentioning that with this notation, each $x_m$ is a $D$-dimensional vector at time $t_m$, so $dx_m$ itself should be treated as a product of differentials. Now we substitute our previous definition of the conditional probability function $p(x_{m+1}|x_m)$ , which is an approximation to the first order, into the propagator functional $K[x(t)]$. We then choose to write the resulting equation in an exponential form.

$$\begin{aligned} K[x(t)] &= \prod_{m=1}^{M-1} \frac{1}{Z} \exp\left[ -\frac{1}{2} \left\| \frac{x_{m+1} - F(x_m, t_m)}{\sigma_{\text{model}}} \right\|^2 \right] \\ &= \frac{1}{Z^M} \exp \underbrace{\left[ -\frac{1}{2} \sum_{m=1}^{M-1} \left\| \frac{x_{m+1} - F(x_m, t_m)}{\sigma_{\text{model}}} \right\|^2 \right]}_{-\mathcal{S}[x(t)]} \end{aligned} \tag{2.12}$$

The argument of the exponential function $\mathcal{S}[x(t)]$ is called the action due to its roots in physics. See the appendix on path integrals for further elaboration. The most striking point here is to notice that the action $\mathcal{S}[x(t)]$ is scalar function of a path $x(t)$ and that the full nonlinearity of the dynamics $F(x_m, t_m)$ is preserved. There is no linearizaton of any kind, merely the assumption that the proposed model works well in estimation the forward state $x_{m+1}$ from $x_m$.

$$\mathcal{S}[x(t)] = \frac{1}{2} \sum_{m=1}^{M-1} \left\| \frac{x_{m+1} - F(x_m, t_m)}{\sigma_{\text{model}}} \right\|^2 \tag{2.13}$$

We now have access to a statement regarding the probability of the dynamical system being at some state $x_M$ at time $t_M$. Not only that, but this probability is achieved by integrating over every possible path that can precede $x_M$. All the relevant information about the states transitioning across time is encoded in this action $\mathcal{S}[x(t)]$. It is this action that essentially 'picks out' the most likely path

among all possible ones.

$$p(x_M) = \int \mathcal{D}x(t)\, e^{-\mathcal{S}[x(t)]}\, p(x_1) \tag{2.14}$$

Rewriting the previous probability function with the action $\mathcal{S}[x(t)]$ in mind, we get the above function. This equation is a statement regarding the probability distribution and can be made even more general, such that any function or quantity (vector or scalar) defined along the path can be calculated by this integral evaluation [12]. We simply insert any function $G[x(t)]$ inside the integrand, and we get an expectation value for this function $\langle G[x(t)]\rangle$.

$$E(G[x(t)]) = \langle G[x(t)]\rangle = \int \mathcal{D}x(t)\, e^{-\mathcal{S}[x(t)]}\, G[x(t)] \tag{2.15}$$

At this point, only the assumptions made so far are threefold: 1) that the Markov property holds, 2) the system can be discretized in time smoothly, and 3) that $p(x_{m+1}|x_m)$ can be relaxed into a Gaussian-like distribution. All three of these assumptions are rather mild, and the last of which only determines the functional form of the action, not the path integral [13, 4]. Of particular interest is when $G[x(t)]$ is the identity function such that we are solving for the average trajectory $\langle x(t)\rangle$ of the system.

$$\langle x(t)\rangle = \int \mathcal{D}x(t)\, e^{-\mathcal{S}[x(t)]}\, x(t) \tag{2.16}$$

For now, this average trajectory $\langle x(t)\rangle$ is evaluated over an infinite number of possible solutions, or paths. This gives us nothing useful for now, as there are no other constraints to consider. However, if we can condition the action $\mathcal{S}[x(t)]$ sufficiently, say with priors in the form of measurement data, then we will be able to make some good use of this path integral formulation, which is the topic of the following section.

## 2.6 Including Measurements

Thus far, we have converted a deterministic model into a probability distribution, roughly speaking. The question now is how time-series measurements of the system of interest $y(t_m) = \{y_1, y_2, \cdots, y_M\}$ should be incorporated into this probability distribution in order to arrive at useful and sensible state estimates $x(t_m) = \{x_1, x_2, \cdots, x_M\}$. We shall assume that we can measure state variables directly so that $H(x_m) = x_m$ is in the same space as $y_m$, where $H(x_m)$ is a nonlinear

measurement function. Also implicitly assumed, in the notation, is that the time-series data is available at the same time-scale as the model dynamics, which is seldom the case for weather systems but it is the case when dealing with neuronal data collected at the rate of several kilo-Hertz. In the case of a mismatch between measurement and model time-steps, we could introduce a Dirac delta function into the measurement term, but this will not be address in this work. The simplest and most naive approach of including measurement data would be to impose a standard Gaussian distribution at every point in time, which introduces the following term multiplicatively.

$$p(x_m|y_m) = \frac{1}{Z}\exp\left[-\frac{1}{2}\left\|\frac{H(x_m)-y_m}{\sigma_{\text{meas}}}\right\|^2\right] \equiv \frac{1}{Z}\exp\left[-\frac{1}{2}\left\|\frac{x_m-y_m}{\sigma_{\text{meas}}}\right\|^2\right] \tag{2.17}$$

This white-noise spectrum should be a good approximation of the noise spectrum of most well-tuned measurement devices. We can then make a claim regarding the estimate of the state at a particular time $x_m$, conditioned on the previous estimated state $x_{m-1}$ and a measurement at the current time $y_m$. This is effectively using the measurements to introduce an enveloping term on top of the existing probability density function, which previously only considered the model dynamics. This addresses the issue previously had, where there were infinite number of paths to sum over. In effect, we use the measurements to reduce the number of paths to sum over, and this makes estimations a lot more useful.

$$\begin{aligned} p(x_m|x_{m-1},y_m) &\equiv p(x_m|x_{m-1})p(x_m|y_m) \\ &= \frac{1}{Z}\exp\left[-\frac{1}{2}\left\|\frac{x_m-y_m}{\sigma_{\text{meas}}}\right\|^2 - \frac{1}{2}\left\|\frac{\hat{x}_m - F(\hat{x}_{m-1},t_{m-1})}{\sigma_{\text{model}}}\right\|^2\right] \end{aligned} \tag{2.18}$$

All technical issues regarding the normalization are entirely absorbed into the $Z$ term for simplicity. We do not spend time on issues regarding $Z$ since the normalizing constant will not play a role in our calculations. It is noted that the above conditional probability does not necessarily follow from the previous equations, but that they are merely consistent with one another. We have assumed that model estimates $x_m$ are independent from the measured data $y_m$, which is a relatively safe and conservative assumption. Of course, the estimate and the data must be intrinsically related given that they are both attempting to describe the same system, so this assumption results in a search over a space that is larger than necessary. This handicap, however, is partially overcome by the use of other

Markov chains that will be discussed in the next chapter.

$$p(x_M|y(t)) = \int \mathcal{D}x(t)\, e^{-\mathcal{S}[x(t)|y(t)]}\, p(x_1) \tag{2.19}$$

The exact treatment of the previous section is given to this conditional probability function, with the only notable difference being the messiness of the intermediate steps [4]. This results in a new form of the action $\mathcal{S}[x(t)|y(t)]$ that is now conditioned on data as well as the model.

$$\mathcal{S}[x(t)|y(t)] = \frac{1}{2}\sum_{m=1}^{M}\left\|\frac{x_m - y_m}{\sigma_{\text{meas}}}\right\|^2 + \frac{1}{2}\sum_{m=1}^{M-1}\left\|\frac{x_{m+1} - F(x_m, t_m)}{\sigma_{\text{model}}}\right\|^2 \tag{2.20}$$

Some additional care should be taken when dealing with this action, primarily due to the physical models as a constraint interacting with the inescapable presence of physical units in the measurements. This might be unclear and ambiguous, but consider the following example. Sea surface height is measured in meters and the crest-to-valley variation of ocean waves is on the order of fractions of meters. Wind velocity on the other hand is measured in meters per second, and say it is a windy day where the wind speed varies around tens of meters per second. If both these variables enter the model as it stands now, the variation in the wind speed will easily overshadow the variation in the sea surface height, biasing the probability distribution towards estimating the wind velocity variables well and forsaking the sea surface height estimation. More generally, without adjustments, this action will prioritize estimating variables with larger variation in their measurement. Assuming that neither variable is more important or reliable, it would make more sense to use a scale-free definition in the action such that the dynamics in the measurement window range from $-1$ to 1 approximately, effectively de-dimensionalizing the system and removing this bias. This is done by pushing the necessary scaling, as well as the inversion of the covariance of the system into two matrices $R_{\text{meas}}$ and $R_{\text{model}}$.

$$\mathcal{S}[x(t)|y(t)] = \frac{1}{2}\sum_{m=1}^{M}\|x_m - y_m\|^2_{R_{\text{meas}}} + \frac{1}{2}\sum_{m=1}^{M-1}\|x_{m+1} - F(x_m, t_m)\|^2_{R_{\text{model}}} \tag{2.21}$$

Both the $R_{\text{meas}}$ and $R_{\text{model}}$ are $D \times D$ symmetric positive definite matrices which modify the Euclidean norm by simply scaling the argument vectors[2]. Within the scope of this work, both these matrices are populated along the diagonal only and the resulting computations were good enough. There is room for more complicated covariant structures to be introduced within $R_{\text{meas}}$

and $R_{\text{model}}$, which should result is better convergence rates especially when applying this method to higher dimensional and more complex systems. The challenge of using the covariant structure is entirely technical and left for the interested reader to pursue.

## 2.7 Significance of the Action

Up till now, we have shown the emergence of a path integral from the probabilistic view of a dynamical system. This path integral leads to an action term $\mathcal{S}[x(t)]$, which we further augment by introducing measurements in a sensible manner, thus leading to the newer action $\mathcal{S}[x(t)|y(t)]$ which is now conditioned on the measurements $y(t)$. Having this action, we address how and why this entire path integral formulation is worth considering in that it gives two insights on how to proceed.

The first insight is that we might consider that each state $x_m$ can be treated as independent from the state right before it $x_{m-1}$ and right after it $x_{m+1}$. This is not an approximation, but a purposeful oversight. One is always free to assume, initially, that two objects are not related, at the risk of feigning too much ignorance. Due to the assumed lack of relationship between subsequent states, the space of possible solutions is drastically increased. Thankfully, a search for solutions in this larger space turns out to be better behaved as the landscape becomes smoother. The details of this claim, and the accompanying approach will be discussed in the following chapter.

$$p(x_M|y(t)) = \int \mathcal{D}x(t)\, e^{-\mathcal{S}[x(t)|y(t)]} \tag{2.22}$$

$$E(G[x(t)]|y(t)) = \int \mathcal{D}x(t)\, e^{-\mathcal{S}[x(t)|y(t)]}\, G[x(t)] \tag{2.23}$$

The second insight is that the action $\mathcal{S}[x(t)|y(t)]$ contains all the relevant information necessary to determine the probability distribution $p(x_M|y(t))$ exactly. This means that the states along the entire trajectory $x(t)$ is necessary for finding the most probable final state $x_M$, even though all the previous states are technically integrated out [14, 15, 16]. Evaluating this integral exactly, if even possible, is extremely difficult due to the spatial dimension $D$ and the number of time points $M$ of $x(t) \in \mathbb{R}^{D\times M}$. As such, we have to resolve to either using analytical approximations or numerical methods for evaluating the $p(x_M|y(t))$ integral. The chosen approximation is that of the Laplace

[2]To elaborate in a practical manner, some term written as $\|z\|_R$ is practically computed, in a linear algebra sense, as $z^\top Rz$. Here $z \in \mathbb{R}^D$ is a standard column vector and $R$ must be a matrix of dimension $D \times D$.

method. Due to the form of the resulting action $\mathcal{S}[x(t)|y(t)]$, the conditions for the chosen approximation to work well on $p(x_M|y(t))$ are easily met and the approximation itself also holds extremely well for $E(G[x(t)]|y(t))$ as well. The conditions of the Laplace method, applied to this case, are:

(i) there exists a stationary path $x^*(t)$ (hopefully a minimum) of the action $\mathcal{S}[x(t)|y(t)]$

(ii) the action is sharp enough at the stationary point given by $\frac{\partial^2 \mathcal{S}}{\partial x(t)^2}\Big|_{x^*(t)}$

(iii) the action evaluated at the stationary path $\mathcal{S}[x^*(t)|y(t)]$ must be large relative to the possible variation in $x(t)$

Under these conditions, it so happens that the integral of $E(G[x(t)]|y(t))$ can also be well approximated by the Laplace method, at no significant loss of accuracy. The usage of these approximations to solve for these stationary paths $x^*(t)$ will be called variational methods for the purposes of this work [17].

As for numerical methods, the Monte Carlo family of methods is the approach of choice due to the high dimensions of $x(t)$. The dimensions are high enough that any grid-based or quadrature methods are rendered useless, but the Monte Carlo methods are not affected as much. The challenges of high dimensional space are still present for Monte Carlo methods, but there are efficient ways to overcome the high dimensions that quadrature methods simply do not have access to. Monte Carlo method are much more expensive than the previous alternative of variational methods, in general, but offer additional forms of accuracy that are not addressed in the variational methods. Namely, the Monte Carlo methods will explore the shape of the distribution around the stationary points, considering all higher order statistics of the distribution (notably the third-moment or skew) if given a long enough exploration time. This is in contrast to the variational method, which assumes that the first- and second-moments (mean and standard deviation) are sufficient for estimating the integrals. All the details of these claims will be handled and discussed in the following chapter.

## 2.8 Summary and Interpretations

The path integral formulation/framework give us an action that we hope to extremize. This action pins the measured noisy state at the center of a ball, shown in the first term above. We impose a simple Euclidean penalty so that the states do not wander off too far from the measured state. Such is standard practice, and this measurement error term is sometimes also called the background error.

If measurements were performed properly, the true state should be relatively close by the measured state and $y(t)$ can be used as the first guess in our search for the best estimate $x(t)$.

$$\mathcal{S}[x(t)|y(t)] = \frac{1}{2}\sum_{m=1}^{M} \|x_m - y_m\|^2_{R_{\text{meas}}} + \frac{1}{2}\sum_{m=1}^{M-1} \|x_{m+1} - F(x_m, t_m)\|^2_{R_{\text{model}}} \tag{2.24}$$

In addition to the data itself, we also have a credible model which describes the approximate dynamics of the observed data. Not only should the best-estimated state be somewhat consistent with the data, but it should also be somewhat consistent with the model $F(x, t)$. This is the purpose of the second term shown above, which can be viewed as a constraint term [18]. The best-estimate should simultaneous be close to the measured state while satisfying the constraint imposed locally by the model dynamics. We can think of extrema of the action as the solutions to the most-probable path defined by the path integral.

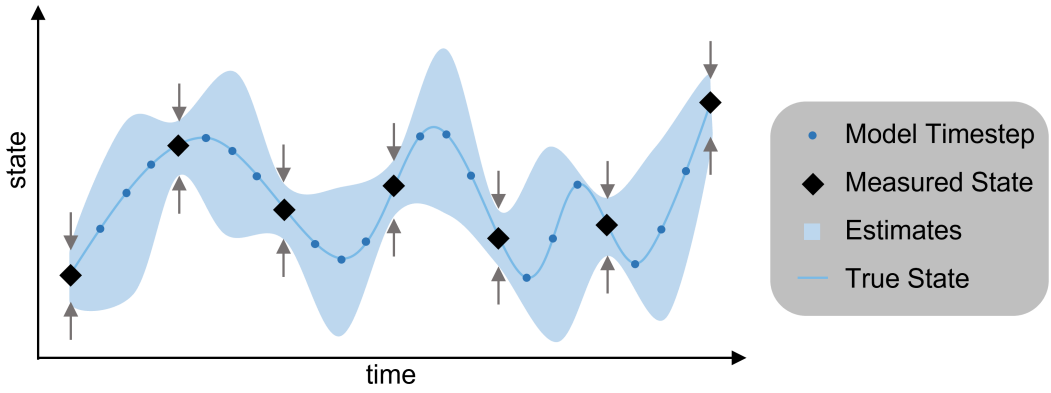


**Figure 2.1:** Rough illustration of the estimation process with the assumption of very low measurement noise for simplicity in the figure. The lightblue regions may be viewed as the continuous error bars of the estimate. Measurements provide a 'pinching' in the possible trajectories, shown by the grey arrows, as we iteratively refine the estimates. Where there are no measurements, we see that there is a larger spread in the possible values that the estimate can take on. The estimation process handles all points in time simultaneously, not sequentially, which is the distinguishing feature of the path integral approach compared to weak-constraint 4Dvar.

This path integral approach for dynamical systems is not something completely new. Our approach bears very strong resemblance to the Onsager-Machlup function for a most probably diffusion path. There is some slight difference in the conclusion and approach since their definition of diffusion occurs in the time dimension $t$. If anything, our approach is to 'undo the diffusion' of a state that results from the act of measurement. We will expound on this topic more in the heuristics chapter. This action is also very similar to the cost function of the 4Dvar family of methods, more specifically long-window

weak-constraint 4Dvar. These similarities subside rather quickly since the approach of assimilating long-window weak-constraint 4Dvar with the data is considerably different past the functional form of the cost function. We will not be further discussing 4Dvar or their differences/similarities to this work.

There is also a strong resemblance to (perhaps even an untraced inspiration from) the Path Integral Monte Carlo (PIMC) method and its premises. The biggest difference is that PIMC is applied to physical problems where the dynamics (say the Schrodinger equation) is extremely precise, so the physical quantities like the energy and the action are on much firmer ground[3]. In comparison, the action we use for data assimilation is qualitatively correct in the proper limit, but there are no explicit physical constraints or reasons from nature why it would be *de facto* true. PIMC almost exclusively uses the Metropolis-Hastings technique for evaluating the integral, while we use other techniques as well. These other techniques perform well on our action and do not manifest the issues that PIMC faces, perhaps due to the lack of physical realism in our formulation.

In summary, this action is the most significant result of the path integral formulation of data assimilation, as it lends us a lot of intuition on how to proceed with practical calculations. The goal is to get a best estimate of the dynamical system $x(t)$ from measurements $y(t)$ and the model $F(x,t)$. The interpretation and approach are provided by the path integral formulation outlined thus far, which lets us know that the focus should be placed on the finding the stationary points of the action. This action also implies that the Laplace method approximation can be used. Lastly, the missing piece of this puzzle are the tools with which we could use to find these stationary points. These tools have been teased many times in this chapter, but will be the focus of the following chapter.

## 2.9 Acknowledgments

Chapter 2, in part, uses material and results that appears in Precision Annealing Monte Carlo Methods for Statistical Data Assimilation and Machine Learning, submitted in Physical Physical Review Research, 2(1), 013050. Fang, Zheng, Wong, Adrian S., Hao, Kangbo, Ty, Alexander J., and Abarbanel, Henry D. I. (2020). The dissertation author was one of the primary investigators and authors of this paper.

[3]It is common practice in PIMC to use approximate or effective actions, rather than the full action, due to the interaction terms of the Hamiltonian or action being incredibly sophisticated and computationally expensive to evaluate.

# 3

# Evaluating Integrals

## 3.1 Overview

The previous chapter dealt with showing that estimating the state of a dynamical system can be treated as a statistical problem. We then further explore how this statistical treatment leads to a sensible probability density function, and that data can be used to further condition this function. This then allows for the calculation of expectation values by evaluating some high dimensional integral that closely resembled a path integral from physics. High dimensional integrals of arbitrary but bounded functions are not an easy task, and estimating or approximating the integral is a common challenge in many areas of engineering, statistics, and the sciences. The resemblance to a path integral informs our intuition about how one can approach such integrals. The goal of this chapter is to address the methods of choice when faced with integrals of the general form below. The specifics of the implementation require another chapter discussing additional heuristics for these method, and will be covered in that chapter.

$$E(G[x(t)]|y(t)) = \int \mathcal{D}x(t)\, e^{-\mathcal{S}[x(t)|y(t)]}\, G[x(t)] \tag{3.1}$$

Also in the previous chapter, we asserted that the action necessarily has the form of two quadratic terms, the first being linear and the second nonlinear. The quadratic form was arbitrarily prescribed in the previous chapter as a sensible and believable form for a cost function, but it is

important to address the assumptions that were made.

$$\begin{aligned} p(x_M|y(t)) &= \int \mathcal{D}x(t)\, e^{-\mathcal{S}[x(t)|y(t)]} \\ &= \int \underbrace{\left[ dx_m \prod_{m=1}^{M-1} p(x_{m+1}|x_m) \right]}_{\text{model-consistency}} \underbrace{\left[ \prod_{m=1}^{M} p(x_m|y_m) \right]}_{\text{conditioned on data}} p(x_1) \end{aligned} \tag{3.2}$$

The first assumption is that the dynamics of the system are Markovian, which has been tacitly assumed this entire work. The second assumption is that the measurement error $x_m - y_m$ is given by uncorrelated (in time) Gaussian noise. The third assumption is that the model and measurement error are independent. All three of these assumptions are very mild and commonplace assumptions. The third assumption is that the model error or model-consistency term $x_{m+1} - F(x_m, t_m)$ is also given by uncorrelated Gaussian noise, which is perhaps the assumption that is hardest to satisfy.

$$\begin{aligned} \mathcal{S}[x(t)|y(t)] &= -\sum_{m=1}^{M-1} \ln p(x_{m+1}|x_m) - \sum_{m=1}^{M} \ln p(x_m|y_m) + \cancelto{\text{constant}}{\ln p(x_1)} \\ &= \frac{1}{2} \sum_{m=1}^{M} \|x_m - y_m\|^2_{R_{\text{meas}}} + \frac{1}{2} \sum_{m=1}^{M-1} \|x_{m+1} - F(x_m, t_m)\|^2_{R_{\text{model}}} + \text{constants} \end{aligned} \tag{3.3}$$

There are two approaches to estimating the expectation values given by the integral. One such approach is to employ the Laplace method, which makes an asymptotic expansion around the mode (assuming for now that there is but one mode) of the probability function $e^{-\mathcal{S}[x(t)|y(t)]}$. The mode of this function is necessarily located at the same path $x^*(t)$ that satisfies the stationary condition of the action $\mathcal{S}[x(t)|y(t)]$.

$$\left. \frac{\partial \mathcal{S}}{\partial x(t)} \right|_{x^*(t)} = 0 \tag{3.4}$$

By finding the stationary path $x^*(t)$, the Laplace method allows us to approximate the integral under suitable conditions, the specifics of which will be covered in its own section. This is remarkable (yet commonplace) as the problem of evaluating of integrals is replaced with a minimization problem of a cost function, in our case, the action $\mathcal{S}[x(t)|y(t)]$. The Laplace method allows us to effectively avoid explicit evaluation of the integral by fitting the closest possible Gaussian curve. This entire process only uses functions evaluated at the minimum $x^*(t)$, namely $\mathcal{S}[x^*(t)|y(t)]$ and $\left. \frac{\partial^2 \mathcal{S}}{\partial x(t)^2} \right|_{x^*(t)}$, and does not require exploration around the minimum.

On the other hand, there is the Monte Carlo approach to evaluating integrals, which is used

alongside Markov chains. These family of methods explores the *typical set* of the probability distribution function, but it is still guided by the principle that the mode and its immediate neighborhood will dominate the integral. Generally speaking, it does not matter what the initial state or parameter is set to because it should always converge. This is not the case given limited computational resources, but having a theoretical guarantee is always nice. We visit the Metropolis-Hastings method and the Hamilton Monte Carlo method later in this work.

## 3.2 Laplace Method

### 3.2.1 Original Usage

The Laplace method an approximating technique for integrals of the following form. Once the integral is evaluated, we get a function $I(N)$ which is an integral function of some parameter $N$.

$$I(N) = \int dz \, \exp[-N \, f(z)] \tag{3.5}$$

$$J(N) = \int dz \, g(z) \exp[-N \, f(z)] \tag{3.6}$$

For the purposes of this work, we will limit the scope of the problem such that 1) the bounds of integration are set to be the entire space, 2) the stationary point is necessarily a minimum, and 3) that we are working in the limit for $N \to \infty$. We also assume that there is only one mode in the distribution which dominates the integral. A detailed treatment will be handled in the appendix handling the multiple-mode case as well, but a quick version is presented in this chapter.

$$I(N) = \sqrt{\frac{2\pi}{N \det[f''(z^*)]}} \exp[-N \, f(z^*)] + \mathcal{O}\left[\frac{1}{N}\right] \tag{3.7}$$

$$J(N) = \sqrt{\frac{2\pi}{N \det[f''(z^*)]}} g(z^*) \exp[-N \, f(z^*)] + \mathcal{O}\left[\frac{1}{N}\right] \tag{3.8}$$

We begin by 'searching' for the location $z^*$ of the minima, which is accomplished by solving $f'(z^*) = 0$ for $z^*$. This step can be done analytically in one dimension, even for nonlinear $f'(\cdot)$. In higher dimensions and nonlinear $f'(\cdot)$ we would have to rely on numerical methods. We can

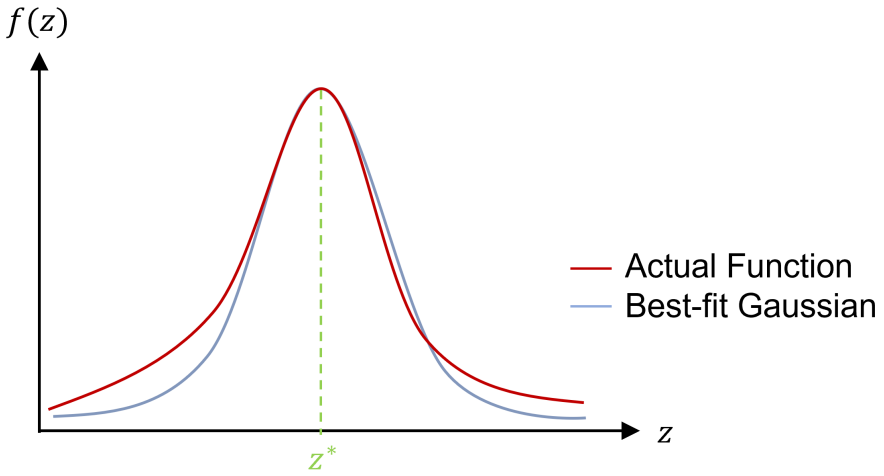


**Figure 3.1:** Rough idea of the Laplace method for an arbitrary but large $N$. The mode $z^*$ and height of the actual function of interest $\exp[-N\,f(z)]$ are inherited by the best-fit Gaussian. Additionally, the Gaussian requires some definition of a thickness or width, and this is provided by the second derivative of $\exp[-N\,f(z)]$ evaluated at $z^*$. The best fit Gaussian here underestimates the actual function due to it having fat tails.

then proceed to calculate the height and width of this best-fit Gaussian once we have found $z^*$ given by $\exp[-N\,f(z)]$ and $\det[f''(z^*)]$ respectively. Some care should be taken given that $f''(z^*)$ is a Hessian matrix in dimensions higher than 1. This asymptotic approximation holds very well when $N$ is sufficiently large due to the leading order error which scales as $1/N$, assuming no other pathological behavior of the function $f(\cdot)$.

Our problem of integral evaluation has been exchanged with a numerical optimization problem instead. There is a small trade-off in that we are using as asymptotic approximation by using the Laplace method. We use the Newton family of methods for the optimization procedure, which uses gradient information of the cost function $f(\cdot)$. As a practical matter, we use quasi-Newton methods for the procedure as it avoids calculating the inverse of the Hessian $f''(\cdot)$ within the iterative part of the method.

### 3.2.2 Applied to the Data Assimilation Action

The Laplace method allows us to estimate the value of integrals based on information local to the mode of the distribution. It gives us the following approximation where the leading order error

(not shown) scales at $\frac{1}{N}$.

$$J(N) = \int dx\, g(z) \exp[-N\, f(z)] \simeq \sqrt{\frac{2\pi}{N \det[f''(z^*)]}} g(z^*) \exp[-N\, f(z^*)] \tag{3.9}$$

This is the general case, and we want to apply the Laplace method to our particular probability distribution function. Saying that this estimation holds better when some parameter $N$ is high reduces our understanding to a mechanistic one, with no intuition to guide us any further. A more useful way of phrasing this is that the Laplace method works better when the distribution gets sharper, defined in our case as when the second derivative or Hessian $\mathcal{S}''[x(t)^*|y(t)]$ is large. However, this is all assuming that there are no serious pathologies in the distribution (e.g. very fat tails, extreme skew, multiple deep minima, etc.). We address these serious pathologies in a later section.

$$E(G[x(t)]|y(t)) = \int \mathcal{D}x(t)\, e^{-\mathcal{S}[x(t)|y(t)]}\, G[x(t)] \simeq \sqrt{\frac{2\pi}{\det\{\mathcal{S}''[x(t)^*|y(t)]\}}} G[x(t)^*] \exp\{-\mathcal{S}[x(t)^*|y(t)]\} \tag{3.10}$$

The above form is recovering when we use the notation for the data assimilation action and the stationary path $x(t)^*$. It can be further simplified by looking at the specific structure of the action $\mathcal{S}[x(t)|y(t)]$.

$$\mathcal{S}''[x(t)^*|y(t)] \equiv \left.\frac{\partial^2 \mathcal{S}}{\partial x_i \partial x_j}\right|_{x^*(t)} = R_{\text{meas}} + R_{\text{model}} Q[x(t)^*] \tag{3.11}$$

Some simplification is swept into $Q[x(t)^*]$, suffice to say that it approaches a large diagonal matrix (or just a constant) in the immediate neighborhood of the stationary path $x(t)^*$. This amounts to saying that our action can be well approximated by a quadratic function close to the stationary path $x(t)^*$, which is an alternative statement to having $Q[x(t)^*] = N \to \infty$. Since $Q[x(t)^*]$ approaches a large constant, this is tantamount to saying that the model error term of the action dominates the measurement error term. Admittedly, the notation in $\frac{\partial^2 \mathcal{S}}{\partial x_i \partial x_j}$ is extremely vague here, but we refer to subsection 4.3.1 for a more rigorous treatment. A detailed treatment of all these claims will be included in the appendix.

$$E(G[x(t)]|y(t)) \simeq \sqrt{\frac{2\pi}{\det R_{\text{model}}}} G[x(t)^*] \exp\{-\mathcal{S}[x(t)^*|y(t)]\} \tag{3.12}$$

So far, $R_{\text{model}}$ is a prescribed diagonal matrix, meaning that the user can decide on its magnitude and values. The matrix $Q[x(t)^*]$ can then be absorbed into $R_{\text{model}}$. The important

takeaway here is that we need to work with an action $\mathcal{S}[x(t)|y(t)]$ where $R_{\text{model}}$ applies a strong scaling structure to the action. The mathematical condition is given by $\det R_{\text{model}} \gg \det R_{\text{meas}}$, and conceptually this means that we should work in the limit of a strong-constraint model.

### 3.2.3 Approach of the Numerical Scheme

We have shown that the evaluation of an integral can be posed as an optimization problem in the first section of the chapter. The second section that resolves some of the notational issues and extend the Laplace method to the specific form of our action. The natural follow up question would be how one approaches this optimization problem. The optimization problem itself is nonlinear owing to the nonlinear terms in model error of the action.

$$x(t)^* = \underset{x(t)}{\operatorname{argmin}}\, \mathcal{S}[x(t)|y(t)] \tag{3.13}$$

Our method of choice for solving this optimization problem is the quasi-Newton method. The standard Newton method makes use of the second-derivative (or Hessian) $\mathcal{S}''[x(t)|y(t)]$ of the action by inverting it. More specifically, the quasi-Newton method of choice recommended and used within this work called Low-Memory Broyden-Fletcher-Goldfarb-Shanno(L-BFGS), which uses approximates of the Hessian $\mathcal{S}''[x(t)|y(t)]$ and its inverse. The result is a computational complexity of $\mathcal{O}[N^2]$ rather than $\mathcal{O}[N^3]$ for the standard Newton method, where $N = MD$ is the number of variables we are optimizing over. The low memory aspect of the method helps avoid costly memory fetches if everything can fit on the CPU cache, thus offering a potential speed upgrade. L-BFGS is a standard algorithm available in every major numerical toolkit, regardless of the programming language. This numerical method is left as an option in any optimization routine. The exact details of the method and implementation are far beyond the scope of this work.

At this point, there is no need to dive further into details. The complications of implementation in code are minimal. It is worth mentioning that we are looking for the stationary path $x(t)^* \in \mathbb{R}^{D\times M}$, which involves optimization over $(D \times M)$-dimensional space. Implementation in code requires a 'flattening' of $x(t)^*$ into a vector, which is handled in detail in subsection 4.3.1.

## 3.3 Energy Landscape and Search Space

The treatment of the optimization problem thus far is classified as a weak-constraint approach, which means that error in the model are assumed. A strong-constraint approach would assume that the model is near perfect, which is almost never the case. Even if we have access to the perfect model, this approach is unstable for the general nonlinear case particularly when the system is in a chaotic regime of its dynamics. As mentioned in the previous section, the Laplace approximation requires us to work in the limit where $\det R_{\text{model}} \gg \det R_{\text{meas}}$. This is us working in the limit of the strong-constraint, but not an absolutely-strong-constraint.

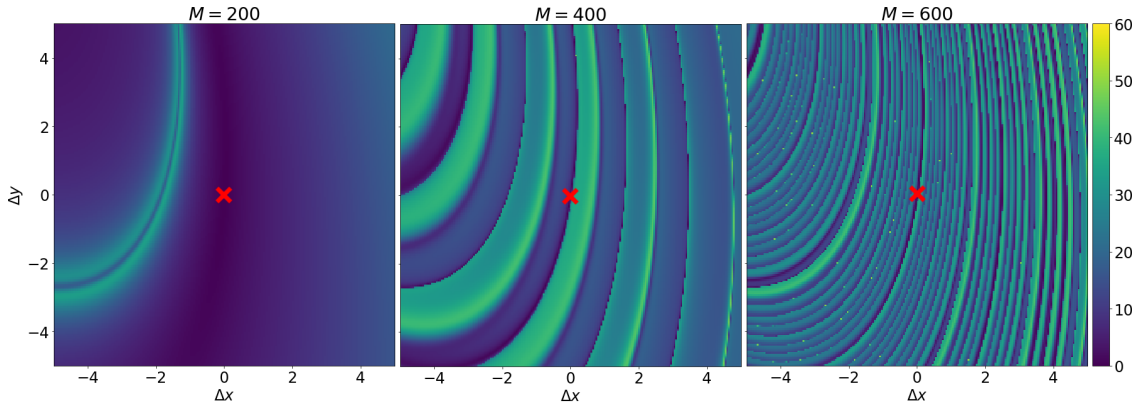


**Figure 3.2:** Energy surface of the Lorenz system using the hard-constraint cost function, i.e when $x_{m+1} = f(x_m)$ is obeyed absolutely. There plots were generated from using the respective $M$ values, which are the number of $\Delta t = 0.01$ time steps within the measurement window. The energy surface gets more complicated as $M$ increases, and the global minimum is located at the origin and marked by a red cross.

In the (absolutely) strong-constraint, the formulation of our problem breaks down substantially because it no longer makes sense to perform a search of path space $x(t) = \{x_1, x_2, \cdots, x_m\}$. Rather, the hand-constraint ties $x_{m+1} = f(x_m)$ unquestionably, so the search space is inevitably going to be over the first state $x_1$ only. Such an approach obviously fits within our formulation since the space of $x_1$ is a subspace of $x(t)$, but a lot of the built-up structure and notation becomes obsolete. The weak-constraint approach is over a much larger space comparatively, by a rather large factor of $M$. There are both advantages and drawbacks for doing so, at least within our formulation.

The advantage that weak-constraint has is that the search over path space $x(t)$ is much more stable than only searching over the initial condition space $x_1$, at least for nonlinear and even chaotic system. This is due to the extremely sharp and jagged surface of the cost function when varying

over $x_1$ alone, which comes from the nonlinearity of imposing $x_{m+1} = f(x_m)$ and is increasingly exaggerated for long windows of measurement. This is particularly true if the underlying system of interest passes through a chaotic region of its dynamics in the optimization routine. Finding the best local minima then becomes a rather tedious and challenges task, as one would very likely get stuck in a inferior local minima very easily. The surface of the cost function is much smoother in our formulation, as it generates extremely deep and well-separated local minima. This is not without consequence, as we need to make sure that we initialize the search close to the basin of attraction of a superior local (hopefully global) minimum. We explore a solution to this weak in the Appendix D. The search is over a much larger space, so our numerical methods of choice need to work well for extremely high dimensional ($10^5$ at least) space. We believe that our approach is well worth its trade-offs, and the L-BFGS method alleviates the computer memory issues of high dimensions fairly well.

There is also the question of the total run time of each approach. Weak-constraint offers optimization over a simple but higher-dimensional space, whereas strong-constraint offer optimization of a complicated but lower-dimensional space. We make the case that the simpler geometry that the weak-constraint offers allows for a much larger step-size and even less iterations. Strong-constraint methods use very small step-size to deal with all the issues outlined above, as well as some additional tricks to help it break-out of local minima. The rest of this work only uses the weak-constraint approach in the limit of a strong-constraint. The specifics of the claims here in the context of Deep Learning and Backpropagation are handled in Appendix C.

## 3.4 Summary and Further Directions

We would like to emphasize, in closing, that the strong-constraint problem is indeed the correct problem to solve. However, the cost function associated with the strong-constraint problem is difficult to handle due to its jagged surface. An alternative is the weak-constraint problem taken to the limit of the strong-constraint, which is an approach equivalent to applying the Laplace method to our action $\mathcal{S}[x(t)|y(t)]$. Doing so increases the dimension of the system tremendously and also hints at iteratively optimizing the problem. These are computationally far more intense than the simple optimization of the hard-constraint, but it avoids the problematic jaggedness associated with the strong-constraint. We believe that the net result of our weak-constraint approach is a modest yet acceptable increase in computation time, at the benefit of a more stable problem to solve. The jagged

surface requires a large number of small step sizes, whereas our approach may use a fewer number of larger step sizes.

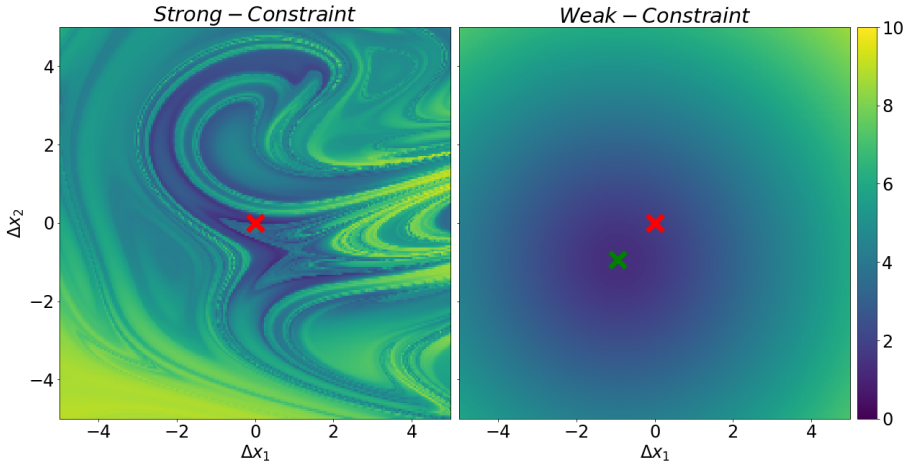


**Figure 3.3:** Comparison of the energy surface of the Lorenz 1996 system for $M = 600$ time steps. The hard-constraint energy surface is incredibly complicated and even displays fractal structures, whereas the weak-constraint energy surface is smoother but approximate. The red cross is the location of the true global minimum, and the green cross is for the estimated global minima using the weak-constraint. This green cross is close to the red cross and will converge to the red cross upon iteratively applying the algorithm.

In addition, the Laplace method is not the only method we use to evaluate the integrals of interest to this work. There are many other options, notably the Monte Carlo family of methods. These methods are incredibly rich in detail and go far beyond just integral evaluation. For these reasons, Monte Carlo methods have been given a chapter on their own rather than being included in this chapter. We briefly mentioned about the use of heuristics that aid in our challenging task of evaluating these integrals. The biggest clue of such is given by the condition $\det R_{\text{model}} \gg \det R_{\text{meas}}$ under which the Laplace method holds well. This limit, however, is challenging if taken too seriously so some strategy is employed in hopes of overcoming this issue. We use the fact that $R_{\text{model}}$ is actually a prescribed value of our confidence in the model, and we start by setting the magnitude of $R_{\text{model}}$ to be comparable to $R_{\text{meas}}$. We slowly increase the magnitude of $R_{\text{model}}$ during the optimization process, practically speaking we iteratively run the L-BFGS algorithm with a higher magnitude $R_{\text{model}}$ each time. Such practices are called heuristics, and this is similar to the annealing heuristic, or the penalty method used in certain area of optimization. An entire chapter is dedicated to these heuristics, which will also tackle some of the implementation aspects of this work.

## 3.5 Acknowledgments

Chapter 3, in part, uses material and results that appears in Precision Annealing Monte Carlo Methods for Statistical Data Assimilation and Machine Learning, submitted in Physical Physical Review Research, 2(1), 013050. Fang, Zheng, Wong, Adrian S., Hao, Kangbo, Ty, Alexander J., and Abarbanel, Henry D. I. (2020). The dissertation author was one of the primary investigators and authors of this paper.

# 4

# Monte Carlo Methods

*All human wisdom is contained in these two words - Wait and Hope*

- Alexander Dumas, *The Count of Monte Cristo*

## 4.1 Introduction

Monte Carlo methods are an extremely broad class of computational techniques which leverage random number generation to perform a variety of calculations. Monte Carlo methods are, loosely speaking, further divided into three paradigms: optimization, integration, and sample generation. Each of these paradigms have been further leveraged in countless ways for all sorts of applications, tools, and sub-methods which permeate various fields such as physics, chemistry, biology, mathematics, epidemiology, meteorology, artificial intelligence, and finance. Even within these fields, there are various sub-fields under which Monte Carlo methods are used in substantially different ways [19]. The full scope of possible applications is astonishing, frankly. It is no wonder that Monte Carlo methods have become a staple in a within science and engineering.

The fundamental idea of Monte Carlo methods is that a deterministic problem can be solved with a method in which the individual steps are random – these are called stochastic methods. Monte Carlo methods are generally used when the problem is incredibly complicated, rendering the available analytic approaches either impractical or intractable.

## 4.2 Metropolis, Hastings, and Rosenbluth

### 4.2.1 Description of the Method

The Metropolis-Hastings algorithm is a procedure for exploring state space through generating random samples. These samples are not completely random but are proposed as random perturbations to the most recent accepted state. We assume to have access to an energy or cost function that, in effect, tells the algorithm how to differentiate between bad and good states. This energy function also happens to be the logarithm of the probability density function.

**Algorithm 1** Original instantiation of Metropolis-Hastings algorithm. $E(\cdot)$ is a prescribed or assumed energy function and the distribution of $p(x) \propto \exp[-E(x)]$ is tacitly implied.

```
x randomly initiated
while not converged do
    Δx ~ 𝒩(0, σ²)                    ▷ Δx drawn from the Gaussian distribution.
    x' ← x + Δx                      ▷ Generate a new state from the old one.
    ΔE ← E(x') − E(x)                ▷ Calculate the change in energy.
    u ~ 𝒰(0, 1)                      ▷ Generate a random uniform number between 0 and 1.
    if exp[−ΔE] < u then             ▷ Accept the new state probabilistically.
        x ← x'
```

Lower energy states are better in general, guided by the principle that nature trends to the lowest energy states locally. By moving towards the better states on average, the algorithm not only explores the space to a large degree, but it will also spend more time in areas with higher probability. Under ideal behavior, the algorithm updates the state as it slowly approaches the lowest stable energy state. Upon reaching the local minimum, the state will deviate slightly around it and produce samples around the local minimum in line with the underlying distribution. A record of all previous accepted states $\left\{x^{(0)}, x^{(1)}, \cdots, x^{(N)}\right\}$ is kept, and from this history we can calculate the expectation values.

$$\hat{f}_N = \frac{1}{N} \sum_{i=1}^{N} f(x^{(i)}) \tag{4.1}$$

This method is simple to implement and understand, yet unbelievably effective under reasonable conditions. The entire algorithm constitutes of basic math operators, if statements, for loops, and function evaluations. These reasons combined meant that the algorithm would be implementable and useful as early as the 1950s, given the technology at the time. In other words, the method was exceptionally useful and practical given the time period and the available computational technology. As problems became increasingly complex and computational power comes by easier, the staggeringly

inefficiency of the original Metropolis-Hastings algorithm became more and more apparently. The method runs into issues of slow convergence when the dimension of state space becomes too large or if there are pathological smoothness issues in the probability function. This method is nevertheless a good starting point pedagogically, and there are parallelization techniques that can help overcome the potential inefficiency of the approach. Other methods, like Hamilton Monte Carlo, were also developed as an attempt to overcomes the inefficiencies of the Metropolis-Hastings algorithm. As a technical and historical note, we used the name *Metropolis-Hastings algorithm* for historical accuracy. A more appropriate name would be called the Random-Proposal Monte Carlo (RPMC), and the name *Metropolis-Hastings* refers to the acceptance-rejection step specifically, which is used in other methods. We tackle and consider Hamilton Monte Carlo in a later section, which uses the Metropolis-Hastings refers to the acceptance-rejection step, but not random proposals.

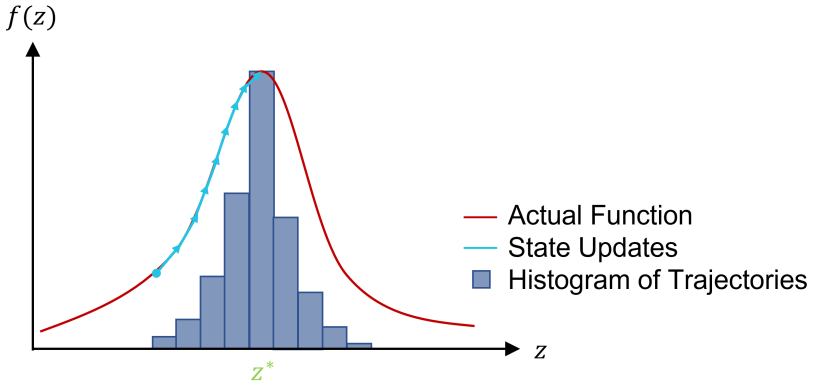


**Figure 4.1:** Rough idea of the Markov Chain Monte Carlo method. The latest state, on average, finds its way to the mode leaving a history of state. This history serves as samples from the typical set of the underlying probability distribution function. Note: not depicted (for visual clarity) are the potential state transitions to regions of lower probability, as well as the state sampling around the mode upon reaching there.

The RPMC algorithm came from research that was originally done at Lawrence Livermore National Laboratory, which means to say that it must have had some contribution to the development or maintenance of nuclear armaments [20]. A lot of the computation and theoretical work in the original work was done by the Rosenbluths, who are often not credited enough for their contributions. In 1970, Hastings published remarkably concise work that generalized and formalizes the mathematics that was utilized in the original work. We explore these generalizations in the next section [21].

### 4.2.2 Some Formalism

Algorithm 1 is a particular instantiation (the original instantiation) of the RPMC algorithm, chosen here for its simplicity and originality rather than its generality. In this section, we discuss some of the generalizations of the method and even include some mathematical formalism. The above instantiation gives a basic idea of what the algorithm does and what it is attempting to accomplish. Once we can appreciate the basic idea, we can dive into the details more and generalize the idea. There are two ways of generalizing the basic instantiation of Metropolis-Hasting.

The first generalization is to notice that the proposal mechanism from the old to the new state is somewhat arbitrary. There is no prescribed way of choosing the standard deviation $\sigma$ of the standard Gaussian $\mathcal{N}(0, \sigma^2)$ – even the choice of using the standard Gaussian is arbitrary. For the general case, we can say that $x'$ is generated according to some conditional transition function $\pi(x'|x)$. However, this still does not tell us how to choose or construct $\pi(x'|x)$ in general, merely that it is the most general formulation of the method. This does not bode well since a properly construction transition function $\pi(x'|x)$ is key to getting faster convergence, and one wants to save as much computational resources as possible. There are many heuristics to how to construct $\pi(x'|x)$ for faster convergences, but there is no obvious winner because it all depends on the use case. Various ways of generating proposals have been explored, and it is a fairly well studied field. We will not explore this in further detail in this work.

The second generalization is in the acceptance mechanism, given in Algorithm 1 roughly by $u \sim \mathcal{U}(0, 1)$ and $\exp(-\Delta E) < u$. This results in the following rule of how we choose to update the states:

> "If the new proposed state results in a lower energy, then we always accept that proposal. If it results in a higher energy state instead, accept the new state with some probability that is proportion to $\exp(-\Delta E)$. "

This rule comes from a slight modification to the original principle that nature trends to lower energy states. The modification is that nature trends to the lower energy states *on average*. From the molecular physics point of view, this means that particles are able to take on a higher energy state microscopically, perhaps by absorbing some thermal energy from the surroundings. The exact mechanism that excites the particle into a higher energy state is unimportant, but merely allowing for that possibility is important.

One consequence of this modified principle and update rule is that the RPMC method can now hop out of shallow local minima, where it might get prematurely stuck, thus exploring a larger space. The second consequence is that we can now satisfy detailed balance, which is a nice additional property to have. Detailed balance happens to be a sufficient (but not necessary) condition for the system to have when the algorithm converges.

$$\text{Detailed Balance:} \qquad \pi(x'|x)p(x) = \pi(x|x')p(x') \tag{4.2}$$

If the particle is never allowed to assume a higher energy state, then detailed balance cannot be satisfied as even if $\pi(x'|x)$ results in $p(x') > p(x)$, the reverse process $\pi(x|x')$ is necessarily zero given that the particle cannot increase in energy. Hence, detailed balance is not obeyed in the case where "only lower-energy updates are allowed".

It is important to note that a convergence criterion was not defined in the entire section, only because it is not clear what such a criterion should be. Though poorly defined, it is generally the case that convergence is defined as simply having the algorithm run 'long enough'. The conditional transition function $\pi(x'|x)$ is also left as an open option, and it is also not clear that this function should be for the general case.

### 4.2.3 Applied to the Data Assimilation Action

Within our data assimilation action, the space which we explore is the path space $x(t) \in \mathbb{R}^{D\times M}$. We can effectively threat this as a two-dimensional $D \times M$ grid – one of these dimensions is the original time dimension given by index $m$, the other is the spatial dimension (the elements of $x_m \in \mathbb{R}^D$) not shown explicitly in this notation. Each of the elements on this grid is perturbed in the step $x' \leftarrow x + \Delta x$ as a means of generating the Markov chain. Below is the action that we are attempting to minimize, which will act as the 'energy' function of this RPMC method.

$$\mathcal{S}[x(t)|y(t)] = \frac{1}{2}\sum_{m=1}^{M} \|x_m - y_m\|^2_{R_{\text{meas}}} + \frac{1}{2}\sum_{m=1}^{M-1} \|x_{m+1} - F(x_m, t_m)\|^2_{R_{\text{model}}} \tag{4.3}$$

However, recall that the RPMC algorithm has the following step $\Delta E \leftarrow E(x') - E(x)$. We are only concerned about the relative change in energy, so calculating the full action $\mathcal{S}[x(t)|y(t)]$ is redundant hence a waste of computational resources. To overcome this, we define a localized action $\mathcal{S}_{\text{local}}[x_m|y_m]$ so that when we perturb the states $x$ to be $x'$, only the change local to $t_m$ will count. The

interaction of an element of $x_m$ is every element of that vector, along with the two nearest temporal neighbors $x_{m+1}$ and $x_{m-1}$.

$$\begin{aligned}\mathcal{S}_{\text{local}}[x_m|y_m] = \frac{1}{2}\|x_m - y_m\|^2_{R_{\text{meas}}} + \frac{1}{2}\|x_{m+1} - F(x_m, t_m)\|^2_{R_{\text{model}}} \\ + \frac{1}{2}\|x_m - F(x_{m-1}, t_{m-1})\|^2_{R_{\text{model}}}\end{aligned} \tag{4.4}$$

The other practical issue is the multi-dimensional normal distribution that we are sampling from $\mathcal{N}(0, \sigma^2)$. As mentioned before, we want to avoid variables with large variations dominating the action $\mathcal{S}[x(t)|y(t)]$, so $\sigma^2$ should follow the same structure as $R^{-1}_{\text{meas}}$. Almost everything else about the implementation of the RPMC method is identical to the original. Graphical Processing Units (GPUs) are naturally suited for such an algorithm. Such a practice is commonplace and for the RPMC algorithm. GPUs are made up of a large number (nowadays 2000-10000) of independent cores which are individually weak but, thankfully, still powerful enough to handle the above algorithm reasonably. The speedup can be massive due to the large number of cores and, in the limit that the number of time steps and the number of cores approaches infinity, the speed up is given by $\text{speedUp} = \min(M/2, \text{numCores})$.

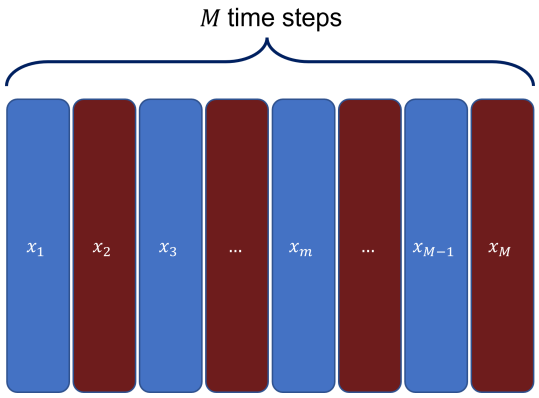


**Figure 4.2:** Trajectory $x(t) = \{x_1, x_2, \cdots, x_M\}$ laid out on a two-dimensional grid. All odd/blue states are simultaneously processed and updated, following but the even/red states. In between each update, the states need to be updated on their respective processors by a simultaneous memory transfer from a core to its immediate neighbors.

Of course, one cannot realistically achieve this speedup in practice due to slow process of memory transfer that occur at the end of each parallel block. There is also some overhead/start-up cost associated with using GPUs. Once everything is accounted for, the speed up is roughly ten to a hundred times slower than the theoretical maximum. This ratio depends entirely on the hardware that

is used, the efficiency and skill of the implementation, and the specifics of the problem. Nevertheless, a speed up is always welcome no matter its efficiency. This parallelization scheme overcomes a lot of the inefficiency of the crude and naive proposal method given by $x'_m \leftarrow x_m + \Delta x_m$, though in a brutish manner. This leads us to the next topic, in which a much more elegant proposal method is explored.

The RPMC also suffers from occasionally taking too small or too large of a step size in space $x$ [22]. There are methods that change a 'block' of states at one, which is equivalent to taking a 'medium sized' step. Within the PIMC literature, this technique is known as a Brownian Bridge and can help alleviate some of the slow convergence issues that RPMC suffers from [23]. One can even construct bridges such that the proposal is almost certainly going to be accepted, but doing so involve additional computation that may or may not be worth the cost. We do not explore the idea of a Brownian Bridge, as we decided to pursue other more promising methods. The use of a Brownian Bridge also decreases the potential speedup that one might get from parallelization.

## 4.3 Hamilton Monte Carlo

### 4.3.1 Notation Change

Before we dive into the Hamilton Monte Carlo (HMC) bit of this work, we should take a moment to adjust the notation that was previously used. The new notation is exceptionally useful for the purposes of HMC and de-clutters the notation at the price of additional complications. We could have made this simplification sooner, at the risk of confusing the reader and distracting from the essence of the other topics. For these reasons, we only use this change in notation within this chapter. The nature of these complications is detailed for the purposes of implementation and should not distract the reader beyond that scope. This change in notation effectively 'flattens' the original path $x(t) \in \mathbb{R}^{D\times M}$ into a column vector representation $\boldsymbol{X} \in \mathbb{R}^{DM}$. One motivating reason is to get rid of the notion of 'time' in the data. The path integral formulation treats the data set more like a two-dimensional mesh. Doing so absorbs the sum over the time index into the definition of the norm, which is no different from applying the Frobenius norm $\|\cdot\|_F$ on the slightly modified $x(t) - y(t)$ matrix.

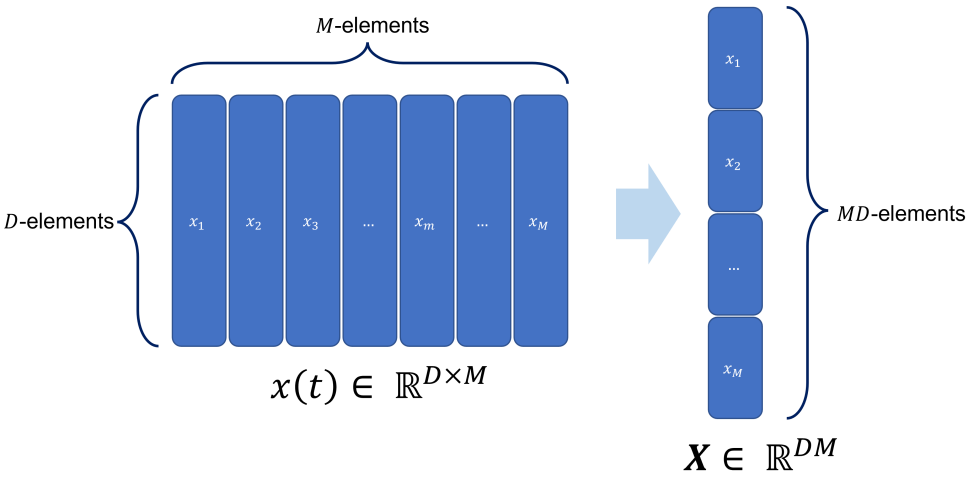


**Figure 4.3:** The left part shows the linear algebraic setup of the original path $x(t)$, which was stored as a matrix. The right part shows the setup of $\boldsymbol{X}$, defined as a 'flattened' column vector. This is no different from representing a two-dimensional grid by a column vector.

$$R = \underbrace{\begin{bmatrix} R_{1,1} & & & \\ & R_{2,2} & & \\ & & \ddots & \\ & & & R_{D,D} \end{bmatrix}}_{D\text{-times}} \quad \Longrightarrow \quad \boldsymbol{R} = \underbrace{\begin{bmatrix} R & & & \\ & R & & \\ & & \ddots & \\ & & & R \end{bmatrix}}_{M\text{-times}} \tag{4.5}$$

Even the two matrices $R \in \mathbb{R}^{D\times D}$ for the metric gets adjusted. This is just a trivial embedding performed by repeating the $D \times D$ matrix along the diagonal for a total of $M$ times, forming a new matrix $\boldsymbol{R} \in \mathbb{R}^{MD\times MD}$. Again, all these adjustments are only highlighted here for the purposes of implementation since we are adjusting the notation. The core method and approach is not any different from before.

$$\|\boldsymbol{X} - \boldsymbol{Y}\|^2_{\boldsymbol{R}_{\text{meas}}} \equiv \sum_{m=1}^{M} \|x_m - y_m\|^2_{R_{\text{meas}}} \equiv \left\| \sqrt{R_{\text{meas}}}[x(t) - y(t)] \right\|^2_F \tag{4.6}$$

The symmetric positive definitive matrix $R_{\text{meas}}$ that modifies the norm was described in a previous chapter. This norm and the sum are absorbed into a new norm $\|\cdot\|_{\boldsymbol{R}_{\text{meas}}}$. The presence of the square root is only for formalities, as the full matrix $R_{\text{meas}}$ is recovered in the process of calculating the Frobenius norm. However, the square root suggests that the $R_{\text{meas}}$ matrix resembles a metric tensor in many ways. Applying these changes in notation consistently to the action results in the

following form.

$$\mathcal{S}[\boldsymbol{X}|\boldsymbol{Y}] = \frac{1}{2}\|\boldsymbol{X}-\boldsymbol{Y}\|^2_{\boldsymbol{R}_{\text{meas}}} + \frac{1}{2}\left\|L^+\boldsymbol{X} - \boldsymbol{F}(L^-\boldsymbol{X})\right\|^2_{\boldsymbol{R}_{\text{model}}} \tag{4.7}$$

The shift operators $L^\pm$ here are $(M-1)D \times MD$ matrices introduced for accounting purposes, where all the diagonal elements are 1. The positive operator $L^+$ excludes the last $D$ elements of $\boldsymbol{X}$, and the negative operator $L^-$ excludes the first $D$ elements of $\boldsymbol{X}$. We defined a modified function $\boldsymbol{F}(\boldsymbol{X}) \equiv \{F(x_1,t_1), F(x_2,t_2), \cdots, F(x_M,t_M)\} \in \mathbb{R}^{MD}$. For the above case, the $L^-$ operator is acting on $\boldsymbol{X}$ to exclude the $F(x_1,t_1)$, so $\boldsymbol{F}(L^-\boldsymbol{X}) \equiv \{F(x_1,t_1), F(x_2,t_2), \cdots, F(x_{M-1},t_{M-1})\} \in \mathbb{R}^{(M-1)D}$. Consequently, $L^+\boldsymbol{X} = \{x_2, x_3, \cdots, x_M\}$. These shift operators $L^\pm$ are necessary for getting the calculations right, but are otherwise unimportant since they are just an artifact of the change in notation. Conceptually, the second term of the action should be viewed as the nonlinear constraint term just as before.

The adjustment in notation chronicled above may not appear as a simplification thus far, but it will soon be apparent when we dive into the HMC method in detail. The slight addition of notation clutter is of no pedagogical consequence, as most of the work detailed in this section is for the purposes of implementation. These are nonetheless important details to consider for pedagogical understanding, but it will suffice to sweep all these detail under the carpet that we call $\mathcal{S}[\boldsymbol{X}|\boldsymbol{Y}]$.

### 4.3.2 Motivations

Hamilton Monte Carlo (HMC) was originally called Hybrid Monte Carlo. It was developed originally to tackle the computational issues with Lattice Quantum Chromodynamics (Lattice QCD) in the 1980s, specifically the problem of inefficient sample generation in high dimensions [24]. Hybrid Monte Carlo used information from the gradient to generated efficient Markov chain such that $\pi(x'|x)$ will almost always generate samples that are accepted at order unity, all while preserving the stochastic acceptance mechanism of the original RPMC [25]. It is the combination of using gradient information while stochastically accepting states that inspired the 'hybrid' part of Hybrid Monte Carlo. The authors even considered usage of the leap-frog method (more specifically the Størmer-Verlet method) in order to preserve the phase space volume of their Hamiltonian. It was David MacKay who began calling it Hamilton Monte Carlo instead of Hybrid Monte Carlo due to the stark similarities in the underlying structure [26]. The new name stuck, perhaps due to the similarities but perhaps due to his leaving the acronym intact.

As mentioned in the section regarding RPMC, perhaps the most vexing issue in the original implementation is the extremely naive proposal mechanism $\pi(x'|x)p(x)$. This naive mechanism becomes terribly inefficiency as the dimensionality of the system grows, because the volume of space required to explore grows exponentially. This is one of the facets of the *curse of dimensionality*, and HMC aims to tackle this specific inefficiency. Another great issue that RPMC faces is the limited search range of each proposal. Too large of a search range will result in many rejections of the samples, and too small a search range results in borderline-trivial updates. HMC uses information regarding the gradient of the cost function $\boldsymbol{\nabla}_{\boldsymbol{X}}\mathcal{S}[\boldsymbol{X}|\boldsymbol{Y}]$ in order to make proposals that will be accepted with high probability. The efficiency of the method does not end there, because using a phase-space preserving integrator like the Størmer-Verlet method, means that we can take as many steps as we would like when integrating the Hamiltonian dynamics. As a result, we are able to explore huge swaths of the space $\boldsymbol{X}$ very efficiently.

### 4.3.3 Using Hamiltonian Dynamics

The goal of HMC is to introduce a better way of sampling the underlying space $\boldsymbol{X}$ which does not involve naive and random generation of samples. Using information of the gradient $\boldsymbol{\nabla}_{\boldsymbol{X}}\mathcal{S}[\boldsymbol{X}|\boldsymbol{Y}]$ during the conception of the method was not a new idea by any stretch [27]. Gradient descent methods were already around, but built into that method inherently is a slow downhill crawl towards the minimum, then it idles once it is there. The lack of exploration makes gradient descent particularly susceptible to even the shallowest local minima [28]. Instead, we want an algorithm that can explore the typical set around the mode for additional information, rather than just seeking and then staying at the mode. This allows us to break away from shallow local minima.

$$\mathcal{H}[\boldsymbol{X},\boldsymbol{P}|\boldsymbol{Y}] = \mathcal{S}[\boldsymbol{X}|\boldsymbol{Y}] + \frac{1}{2}\boldsymbol{P}^{\top}\boldsymbol{M}^{-1}\boldsymbol{P} \tag{4.8}$$

The original proposal of HMC imposed on an additive kinetic energy term $\frac{1}{2}\boldsymbol{P}^{\top}\boldsymbol{M}^{-1}\boldsymbol{P}$ to the function we wish to optimize over $\mathcal{S}[\boldsymbol{X}|\boldsymbol{Y}]$, as Duane[24] wrote, 'by fiat' to form a new function called the Hamiltonian $\mathcal{H}[\boldsymbol{X},\boldsymbol{P}|\boldsymbol{Y}]$. This Hamiltonian function is defined to be constant in 'time' $\tau$, but not the time defined in the measurement data, rather 'computer time' or 'optimization epoch time'. This structure naturally imposes a momentum $\boldsymbol{P}$ as a proposal mechanism, which tells the algorithm where/what the next proposal should be. The dynamics induced by the HMC method is no different

from a particle moving around in a phase space $(\boldsymbol{X}, \boldsymbol{P})$ with the energy or Hamiltonian conserved in time $\tau$ [29, 30].

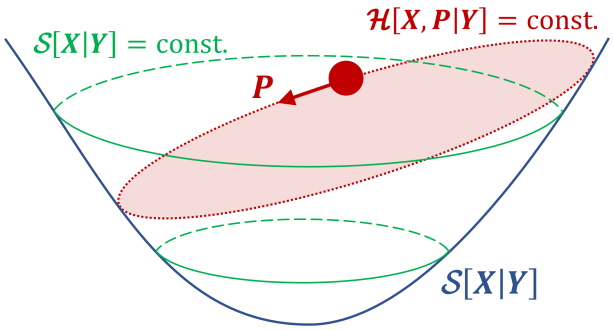


**Figure 4.4:** A physical system equivalent to the HMC method. The particle is exploring $\boldsymbol{X}$ around the minimum of $\mathcal{S}[\boldsymbol{X}|\boldsymbol{Y}]$ using a random level set of the Hamiltonian $\mathcal{H}[\boldsymbol{X}, \boldsymbol{P}|\boldsymbol{Y}]$. If the particle is very far from the minimum, it will tend towards it on average. Once it is close to the minimum, it could begin to show periodicity of orbits. This natural mix of downhill-seeking and exploration is the hallmark of the HMC method.

Another noteworthy consequence in the introduction of 'computer time' $\tau$. In fact, one of the primary reasons we switched notations was to get rid of the variable $t$ which might prove to be confusing for some. By simply imposing that the Hamiltonian function $\mathcal{H}[\boldsymbol{X}, \boldsymbol{P}|\boldsymbol{Y}]$ be conserved in computer time $\tau$, we get the following rules for generating new proposals.

$$\underbrace{\frac{\mathrm{d}\mathcal{H}}{\mathrm{d}\tau}}_{=0} = \frac{\partial \mathcal{H}}{\partial \boldsymbol{P}} \frac{\mathrm{d}\boldsymbol{P}}{\mathrm{d}\tau} + \frac{\partial \mathcal{H}}{\partial \boldsymbol{X}} \frac{\mathrm{d}\boldsymbol{X}}{\mathrm{d}\tau} + \underbrace{\frac{\partial \mathcal{H}}{\partial \tau}}_{=0} = 0 \tag{4.9}$$

$$\frac{\partial \mathcal{H}}{\partial \boldsymbol{P}} \underbrace{\frac{\mathrm{d}\boldsymbol{P}}{\mathrm{d}\tau}}_{\equiv \dot{\boldsymbol{P}}} = - \underbrace{\frac{\mathrm{d}\boldsymbol{X}}{\mathrm{d}\tau}}_{\equiv \dot{\boldsymbol{X}}} \tag{4.10}$$

$$\dot{\boldsymbol{X}} = \frac{\mathrm{d}\mathcal{H}}{\mathrm{d}\boldsymbol{P}} = \boldsymbol{M}^{-1}\boldsymbol{P} \qquad \dot{\boldsymbol{P}} = -\frac{\partial \mathcal{H}}{\partial \boldsymbol{X}} = \boldsymbol{\nabla}_{\boldsymbol{X}} \mathcal{S}[\boldsymbol{X}|\boldsymbol{Y}] \tag{4.11}$$

Keep in mind here that the superscript dots are for the derivatives of the computer time or update time $\tau$, which are how new proposals are generated. Since $\boldsymbol{X}$ and $\boldsymbol{P}$ are independent variables, generating or even prescribing $\boldsymbol{P}$ will leave the underlying distribution of $\boldsymbol{X}$ unchanged, as long as we follow these rules. We are then free, within reason, to prescribe $\boldsymbol{P}$ from the correct distribution in order to explore the level sets of $\mathcal{H}[\boldsymbol{X}, \boldsymbol{P}|\boldsymbol{Y}]$. Given that $\boldsymbol{P}$ shows up as a quadratic term in the Hamiltonian, it necessarily has to be distributed according to the Gaussian distribution

and the geometry of is defined by $\boldsymbol{M}^{-1}$, i.e. $\boldsymbol{P} \sim \mathcal{N}(0, \boldsymbol{M}^{-1})$. In order to be efficient with our proposal, $\boldsymbol{M}^{-1}$ should inherit some rough geometry from the original objective function $\mathcal{S}[\boldsymbol{X}|\boldsymbol{Y}]$. If we then sequentially re-prescribe $\boldsymbol{P}$ from the same Gaussian distribution, we are able to sample $\boldsymbol{X}$ substantially by jumping across different level sets of the Hamiltonian. The inherent randomness of generating $\boldsymbol{P}$ is the reason that this method still belongs in the Monte Carlo family. That said, we still have to obey the underlying deterministic Hamiltonian structure after $\boldsymbol{P}$ is generated [30, 25]. This combination of a random sample followed by deterministic dynamics earned HMC its initial name of Hybrid Monte Carlo.

$$\mathcal{H}[\boldsymbol{X}, \boldsymbol{P}|\boldsymbol{Y}] = -\log \pi(\boldsymbol{X}, \boldsymbol{P}|\boldsymbol{Y}) = \underbrace{-\log \pi(\boldsymbol{P}|\boldsymbol{X}, \boldsymbol{Y})}_{=\mathcal{K}[\boldsymbol{X}, \boldsymbol{P}|\boldsymbol{Y}]} \underbrace{-\log \pi(\boldsymbol{X}|\boldsymbol{Y})}_{=\mathcal{S}[\boldsymbol{X}|\boldsymbol{Y}]} \tag{4.12}$$

This method of generating the momenta $\boldsymbol{P}$ is of a particular preference or choice. More general, we can write the Hamiltonian in the form below, where $\pi(\boldsymbol{X}, \boldsymbol{P}|\boldsymbol{Y})$ is the proposal mechanism under which we generate samples that obey our Hamiltonian structure. Our choice of kinetic energy $\mathcal{K}[\boldsymbol{X}, \boldsymbol{P}|\boldsymbol{Y}]$ is independent of $\boldsymbol{X}$.

$$\mathcal{K}[\boldsymbol{X}, \boldsymbol{P}|\boldsymbol{Y}] \equiv \frac{1}{2}\boldsymbol{P}^\top \boldsymbol{M}^{-1}\boldsymbol{P} \quad \Longrightarrow \quad \pi(\boldsymbol{X}, \boldsymbol{P}|\boldsymbol{Y}) = \pi(\boldsymbol{X}|\boldsymbol{Y})\pi(\boldsymbol{P}|\boldsymbol{Y}) \tag{4.13}$$

Even when we are considering distribution that are independent of $\boldsymbol{X}$, we have settled specifically on the standard Gaussian distribution to sample the momenta, i.e. $\boldsymbol{P} \sim \mathcal{N}(0, \boldsymbol{M}^{-1})$. The generation of standard Gaussian samples is very well studied and readily available in all programming languages – as a result, it is very efficiently implemented, so the resulting code will be very efficient. The quadratic momentum term in $\mathcal{K}[\boldsymbol{X}, \boldsymbol{P}|\boldsymbol{Y}]$ also results in the update rule $\dot{\boldsymbol{X}} = \boldsymbol{M}^{-1}\boldsymbol{P}$, which is simple and straightforward conceptually. Computationally, this involves a standard matrix-vector multiple that is also readily available and extremely efficient. For these reasons, other distribution of $\boldsymbol{P}$ can compare[1]. The alternative is to consider a kinetic energy term $\mathcal{K}[\boldsymbol{X}, \boldsymbol{P}|\boldsymbol{Y}]$ that takes $\dot{\boldsymbol{X}}$ into account, but unless there is a good *a priori* reason for doing so, this task is much slower computationally and generally not worth the effort.

[1]There have been interesting proposals to use quasi-periodic Hamiltonians (such as a quartic $\boldsymbol{P}$) to generate the samples in order to avoid re-sampling points along the same orbit. These attempts have better mixing theoretically, but the implementation might be slower. This feature likely displaces the main proposal of imposing additional nonlinear dynamics to induce better mixing [31]. The use case seems to be limited only to low-dimensions, where periodicity may readily show up. The quadratic momentum $\boldsymbol{P}$ should still be preferred, especially as dimensions get higher.

### 4.3.4 Resulting Algorithm

For an initial guess $\boldsymbol{X}_0$, we prescribe an initial $\boldsymbol{P}_0$ in a random direction and a magnitude that is sensible according to $\boldsymbol{M}^{-1}$. We then evolve the system in computer time $\tau$ for some number of steps $J$ to give us a proposal $\boldsymbol{X}_1'$ and $\boldsymbol{P}_1'$. The new proposed momentum $\boldsymbol{P}_1'$ is of little importance and served more as a technical accounting tool because of the prescribed Hamiltonian structure. This new proposal $\boldsymbol{X}_1'$ has yet to be accepted, but likely will be accepted due to the preservation of the Hamiltonian structure. The proposal $\boldsymbol{X}_1'$ is accepted or rejected according to the following criteria, where $a$ is a randomly generated number from the uniform distribution in the range of 0 and 1 given by $\mathcal{U}(0,1)$.

$$\begin{aligned} \Delta\mathcal{H} &= \mathcal{H}[\boldsymbol{X}_1', \boldsymbol{P}_1'] - \mathcal{H}[\boldsymbol{X}_0, \boldsymbol{P}_0] \\ \exp(-\Delta\mathcal{H}) < u \qquad &; \qquad u \sim \mathcal{U}(0,1) \end{aligned} \tag{4.14}$$

If $\boldsymbol{X}_1'$ is rejected, we revisit the first step of generating new momenta $\boldsymbol{P}_0$ and the above steps are repeated. If $\boldsymbol{X}_1'$ is accepted, we save this new proposal $\boldsymbol{X}_1 \leftarrow \boldsymbol{X}_1'$ and prescribe a new momentum $\boldsymbol{P}_1$ and repeat the above steps for the next time step. This whole process is repeated until some convergence criteria, whether *a priori* or *a posteriori*, is met.

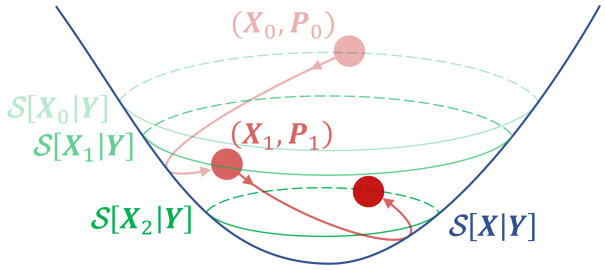


**Figure 4.5:** Graphic showing HMC making two consecutive proposals that result in a lower action $\mathcal{S}[\boldsymbol{X}_j|\boldsymbol{Y}]$ each time. At the end of each proposal, the momentum is re-sampled. This causes the particle to explore different sets of the Hamiltonian $\mathcal{H}[\boldsymbol{X}_j, \boldsymbol{P}_j|\boldsymbol{Y}]$ given by the red trajectories. This graphic only shows the transition to the lower energy sets, which is generally the case if we are far from the minimum. It is possible, likely even, that the transitions result in a higher $\mathcal{S}[\boldsymbol{X}_j|\boldsymbol{Y}]$. Keep in mind the level sets of $\mathcal{S}[\boldsymbol{X}_j|\boldsymbol{Y}]$ (green) are distinct but related to the level sets of $\mathcal{H}[\boldsymbol{X}_j, \boldsymbol{P}_j|\boldsymbol{Y}]$ (red).

The structure from the Hamiltonian, whether we are talking about the physical system of the sampling technique, relies very strongly on the preservation of phase space volume. This structure is called *symplectic* and is a property of the system or method itself. When we are generating new trajectories, the numerical integration scheme must also respect this symplectic structure, otherwise the phase space volume will not be preserved in the integration steps. A lack of phase space volume

preservation is especially poignant when the number of integration steps $K$ is large, and will result in spurious acceptance or rejection of newer proposals simply by virtue of $K$ being large. In other words, our symplectic structure will break down without the proper numerical integration techniques, and we rely on this structure immensely when generating distant proposals.

Thankfully, such numerical integration techniques are available, well-studied, and easily implementable. These schemes belong in the symplectic integrator family of methods, unsurprisingly, and we will focus entirely on the Størmer-Verlet method. Higher-order integrators can be used, but to no noticeable affect for this application [32]. As a quick remark, keep in mind that these integrator are only used in between $(\boldsymbol{X}_j, \boldsymbol{P}_j)$ and $(\boldsymbol{X}'_{j+1}, \boldsymbol{P}'_{j+1})$, which are the level sets of the Hamiltonian $\mathcal{H}[\boldsymbol{X}_j, \boldsymbol{P}_j|\boldsymbol{Y}]$ given by the red trajectory vectors in Figure 4.5.

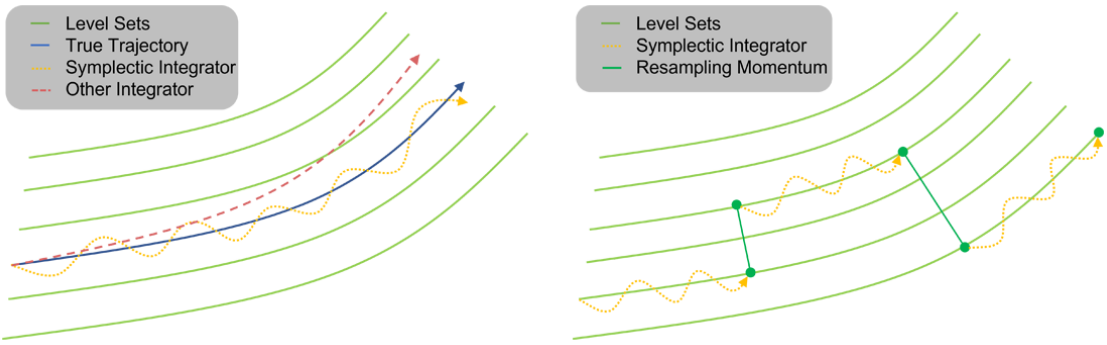


**Figure 4.6: Right:** Graphic comparing the trajectories of using a symplectic vs non-symplectic integrator. The true trajectory stays on a prescribed level set, whereas the symplectic integrator stays close by to the true trajectory indefinitely. Any non-symplectic integrator (red) will result in deviation from the level set, thus destroying the necessary symplectic structure required for the method to work efficiently. **Left:** Graphic showing two jumps in the level sets as a result of resampling the momentum $\boldsymbol{P}_j$ at the points marked in dark green.

Symplectic mechanics is its own (rather deep) field of study that belong under the differential geometry umbrella. It is interesting in that a manifold is defined by demanding that trajectories be generated such that $\mathcal{H}[\boldsymbol{X}_j, \boldsymbol{P}_j|\boldsymbol{Y}]$ is unchanged, and that such a simple demand leads to an entire field of research and study. Much like how the position $\boldsymbol{X}$ and momentum $\boldsymbol{P}$ are conjugate variable that form the vector basis for the symplectic/phase space, the Hamiltonian $\mathcal{H}[\boldsymbol{X}_j, \boldsymbol{P}_j|\boldsymbol{Y}]$ (in units of energy) is the scalar conjugate to time $\tau$.

$$\text{If} \quad (\boldsymbol{X}_j, \boldsymbol{P}_j) \xrightarrow{\text{proposes}} (\boldsymbol{X}'_{j+1}, \boldsymbol{P}'_{j+1}) \quad \text{then} \quad (\boldsymbol{X}'_{j+1}, -\boldsymbol{P}'_{j+1}) \xrightarrow{\text{proposes}} (\boldsymbol{X}_j, -\boldsymbol{P}_j). \tag{4.15}$$

Conservation of the Hamiltonian then implies reversibility of time shown by the statement above. Reversibility is a property of the symplectic structure itself, but it also has to be present in the integrator of choice. The basic Størmer-Verlet or leap-frog method is one of the most well-known and common symplectic integrator, detailed by the steps below. We have added some extra notation here and only here such that $\boldsymbol{P}(0) \equiv \boldsymbol{P}_0$ and $\boldsymbol{P}(K\Delta\tau) \equiv \boldsymbol{P}_1'$, but this notation is not used elsewhere in order to avoid confusion and an overload of notation/indexing.

$$\begin{aligned} \boldsymbol{P}(k/2) &\leftarrow \boldsymbol{P}(0) - \frac{\Delta\tau}{2}\boldsymbol{\nabla}_{\boldsymbol{X}}\mathcal{S}[\boldsymbol{X}(0)|\boldsymbol{Y}] \\ \boldsymbol{X}(k) &\leftarrow \boldsymbol{X}(0) + \Delta\tau\boldsymbol{M}^{-1}\boldsymbol{P}(k/2) \\ \boldsymbol{P}(k) &\leftarrow \boldsymbol{P}(k/2) - \frac{\Delta\tau}{2}\boldsymbol{\nabla}_{\boldsymbol{X}}\mathcal{S}[\boldsymbol{X}(k)|\boldsymbol{Y}] \end{aligned} \tag{4.16}$$

Notice that there is an inherent time-reversibility in the Størmer-Verlet scheme, which was alluded to in the previous paragraph on symplectic structure. This integration scheme is second order accurate in $\Delta\tau$. Størmer-Verlet is carried out for a total of $K$ uniform steps of step size $\Delta\tau$. Between each initial condition $(\boldsymbol{X}_j, \boldsymbol{P}_j)$ and its proposal $(\boldsymbol{X}'_{j+1}, \boldsymbol{P}'_{j+1})$, we will perform this integration step $K$-times before evaluating whether $(\boldsymbol{X}'_{j+1}, \boldsymbol{P}'_{j+1})$ is viable. Once $(\boldsymbol{X}'_{j+1}, \boldsymbol{P}'_{j+1})$ is accepted or rejected, the momentum is resampled and the above integration is performed again, $K$-times. This entire process is repeated iteratively until the number of accepted samples is $J$. To recap, total number of times we resample the momentum will be at least $J$ times, if and only if every proposal is accepted. For each of these resampling events, there will be $K$-steps of the Størmer-Verlet integration on the corresponding level set.

**Table 4.1:** Variables in the HMC method.

| Variable | Description |
|---|---|
| $\Delta\tau$ | Integration step size |
| $K$ | Number of integration steps |
| $J$ | Number of accepted proposals |
| $(\boldsymbol{X}'_{j+1}, \boldsymbol{P}'_{j+1})$ | Most recent proposal, yet to be accepted or rejected |
| $(\boldsymbol{X}_{j+1}, \boldsymbol{P}_{j+1})$ | Most recent **accepted** proposal |

Once the HMC algorithm is completed, we should have $J$ samples of $\boldsymbol{X}$ that form a history of accepted state. These accepted states give a good representation of the typical set of our target distribution $\pi(\boldsymbol{X}|\boldsymbol{Y}) = \exp(-\mathcal{S}[\boldsymbol{X}|\boldsymbol{Y}])$. As mentioned before, the history of accepted values of $\boldsymbol{P}$ are of no consequence, therefore were discards in the resampling process. Lastly, there are three

hyperparameters associated with the HMC method – $\Delta\tau$, $J$, and $K$.

The integration step size $\Delta\tau$ is not terribly important since the Størmer-Verlet method ensures that our method generates trajectories that leave the Hamiltonian invariant, up to some deviation proportional to $\Delta\tau^2$. It is good practice, nevertheless, to use sensible values, because the acceptance probability does scale with $\Delta\tau$. As a rule of thumb, we tune $\tau$ for an acceptance rate of roughly 80% so as to have a $\Delta\tau$ that is large enough to explore quickly while not wasting too much time generating samples that will be rejected. We could also tune $\Delta\tau$ on-the-fly so as to adapt to the geometry of the problem and overcome any *a priori* assumptions.

$$\hat{f}_N = \frac{1}{N}\sum_{i=1}^{N} f(x^{(i)}) \tag{4.17}$$

Lastly, we have $J$ and $K$. $J$ corresponds to the number of proposals that are accepted. $K$ corresponds to the number of integration steps performed for each proposal. Assuming a 100% acceptance rate, this means that there will be a total of $JK$ integration steps taken. In our usage of HMC, $J$ is fixed and are kept on the order of $10^2$ so the law of large numbers applies to our statistics. $K$ is also fixed to be on the order of $10^2$ as well. Once we have a large number $J$ of samples around the mode, we can use calculate expectation values but taking the average value of the sample, as shown in the equation above. The values that these hyperparameters take on is problem dependent. There are more advanced and sophisticated techniques that can be added to HMC, like the No-U-Turn Sampler, such that these hyperparameters are optically chosen or tuned. These are interesting and useful topics, but we will not discuss this in our work.

## 4.4 Comparison and Remarks

The Metropolis-Hasting method used random proposal that avoided the use of a gradient. Fundamentally, the RPMC proposal is driven by diffusion that is symmetrically distributed from its latest accepted state. As a direct result, the Metropolis-Hasting is terribly inefficient when the latest accepted state is in the following situations:

(i) Flat regions of the Hamiltonian with a gradual slope

(ii) Far away from local minima

(iii) Within a sufficiently deep local minima

(iv) Regions of the Hamiltonian with 'deep valleys', mathematically given by the condition number of the Hessian[2]

In cases (i), (ii), and (iii), we would likely require a large number of proposals in order to drift significantly from the initial starting point. Case (iv) would likely see a steep drop in acceptance rates due to there being 'wrong directions' to propose. One can alleviate these problem by tuning the step sizes accordingly, but doing so with *a priori* information requires additional considerations in the algorithm – doing so with *a priori* information requires direct calculation of the Hessian, which is costly and requires derivatives. This almost immediately puts us in the wheelhouse of HMC and defeats one of the original reasons for using RPMC, which was to avoid derivative calculations. The use of parallelization, particularly when leveraging GPUs, is a partial and brutish solution to the above stated issues since it does not directly address the flaws of RPMC.

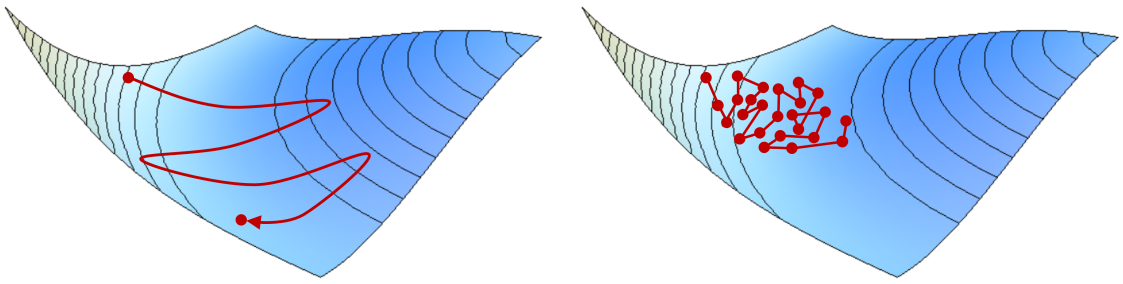

**Figure 4.7: Right:** Illustration showing the HMC method generating and accepting a proposal that is far from the initial point. **Left:** Illustration showing RPMC generating and accepting many proposals, each very close to one another. Roughly speaking, these should have comparable run times on a computer.

Needless to say, HMC addresses these issues well at the cost of including gradient information, making it the state-of-the-art and preferred method of exploring these high-dimensional problems. The RPMC assumes that there is but one mode. There can be situations where the mode is given by a set. We can compare gradient descent or even stochastic gradient descent to HMC, but SGD searches initial condition space which is really jagged. Whereas HMC is much more suited for smoother surfaces. So concludes the description of RPMC and HMC, however, there are further steps that we can take to ensure that our integral evaluations yield better results. These additional steps are the focus of the next chapter, and will simply be called heuristics.

## 4.5 Acknowledgments

Chapter 4, in part, uses material and results that appears in Precision Annealing Monte Carlo Methods for Statistical Data Assimilation and Machine Learning, submitted in Physical Physical Review Research, 2(1), 013050. Fang, Zheng, Wong, Adrian S., Hao, Kangbo, Ty, Alexander J., and Abarbanel, Henry D. I. (2020). The dissertation author was one of the primary investigators and authors of this paper.

[2]The condition number of a square matrix is given by the ratio of the largest to the smallest eigenvalue magnitude of a matrix, i.e. $|\lambda_{\max}/\lambda_{\min}|$. If this square matrix is the Hessian of a Hamiltonian function, then this is equivalent to saying that the variations of the Hamiltonian in certain directions is much larger than others. Keep in mind that the Hamiltonian is assumed to be twice differentiable everywhere in order for the corresponding probability function to be properly defined.

# 5

# Heuristics and Their Implications

In the two previous chapters, we laid out three methods for evaluating integrals. These integrals are high-dimension, easily capable of $10^4$ (Lorenz System) to $10^9$ (Shallow Water Equations) within the scope of problems considered in this work. These three methods for evaluating integrals are the quasi-Newton solver, RPMC, and Hamilton Monte Carlo (HMC). These method, though excellent in their own right, can be made even more effective at robustly and accurately evaluating the integral with the use of additional steps. We shall call these additional steps as *heuristics* from here on. These heuristics are additional steps that we can take to improve existing methods, but are largely unrelated to the original method. Heuristics were given a chapter of their own since they can apply to each of the three integral evaluation methods, hence they do not belong in either of the previous chapters.

We cover two heuristics in this chapter, namely the simulated annealing method and (a basic) multiple-start method. These two heuristics affect the original optimization problem in different ways, hence they can both be used at the same time [33]. These methods, though costly, makes the optimization routine more robust [34]. We are much more likely to get better solutions when using these heuristics, so we believe they are well worth the cost. In fact, we believe that the using them both at once actually complement one another, because some of the weaknesses in simulated annealing is address with the multiple-start method, as we will address when discussing the methods themselves.

## 5.1 Annealing

### 5.1.1 Introduction

The first of these heuristics is the simulated annealing method, also known in some circles as the penalty method. The term Simulated Annealing is more prevalently used in the physics and even the computer science community. Simulated Annealing gets its name from a practice of *actual annealing* used by glass-smiths and blacksmiths till this day. These professionals noticed that their crafts were much less likely to chip, crack, or fracture when they allowed it to cool gradually from their initially high temperatures. This process was unlikely well-understood within the practice of metal or glass smithing, but we have come to understand the reasons why annealing works. Annealing allows for the system to access more thermodynamic states due to it cooling slowly. This effectively results in the system 'exploring' a larger number of states and freeing it shallow local minima.

The fundamental idea of simulated annealing is exactly the same. By introducing a concept of an artificial temperature to the system, we have some say of the general behavior of the optimization routine. We begin at high temperature, where the optimization routine is allowed to make move into regions of lower probability more often. This allows the optimization routine to have access to a wider array of states, corresponding to the exploratory phase. As the temperature lowers, the likelihood of states transitioning to region of lower probability becomes less likely. At a sufficiently low temperature, exploratory behavior diminishes and the optimization routine focuses more on strict optimization.

As a result, the practice of simulated annealing results in, empirically speaking, superior local minima on average. However, it does not guarantee it or even provide any form of assurance strictly speaking. Nevertheless, we believe that a lack of assurances or guarantees to be discouraging, as we have found this practice to be extremely beneficial for our purposes. From here on we will be referring to this as simply *annealing* for simplicity.

### 5.1.2 Applied to the Data Assimilation Action

Recall that the original problem that we would like to solve is to find a path $x^*(t) = \{x_1^*, x_2^*, \cdots, x_M^*\}$ that agrees with some measurements $y(t) = \{y_1^*, y_2^*, \cdots, y_M^*\}$ of the same system, while simultaneously satisfying the dynamics of the system $x_{m+1}^* = F(x_m^*, t_m)$ for all $m$ exactly, which we call the hard-constraint. We discussed the fact that since the dynamics must be obeyed exactly,

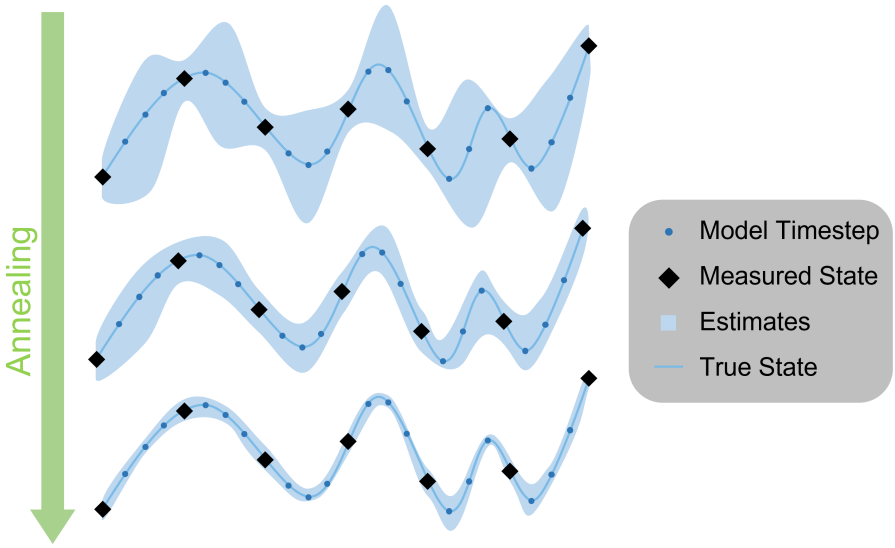


**Figure 5.1:** Illustration on how the annealing procedure affects the state estimate. If we can think of the spread of the estimate as being described by some distribution, the annealing procedure forces the estimate to obey the model more strictly and tightens the width of the distribution. In other words, we can slowly enforcing the constraint that the model properly describes the observed dynamics.

the optimization is actually performed over the initial state $x(t_1)$ rather than the entire path. This constrained optimization problem is then given by the equation below.

$$\begin{aligned} &\min_{x(t_1)} \frac{1}{2} \sum_{m=1}^{M} \|x_m - y_m\|^2_{R_{\text{meas}}} \\ &\text{s.t.} \quad \dot{x} = f(x,t) \end{aligned} \tag{5.1}$$

The landscape of this search is difficult for any optimization routine to traverse. In certain areas, the step size must remain small due to the evaluation of the gradient being local, but this can also lead to slow convergence. Significant effort is put into a balance of stability and convergence due to the troublesome and jagged landscape. Our approach is to assume that there will always be some mismatch between subsequent state $x(t_i)$ and $x(t_{i+1})$, which effectively means that we are softening the hard-constraint.

$$\mathcal{S}[x(t)|y(t)] = \frac{1}{2} \sum_{m=1}^{M} \|x_m - y_m\|^2_{R_{\text{meas}}} + \frac{1}{2} \sum_{m=1}^{M-1} \|x_{m+1} - F(x_m, t_m)\|^2_{R_{\text{model}}} \tag{5.2}$$

We constructed the above cost function, which we call the action, as a consequence of approaching the problem from a soft constraint. The original constrained minimization problem can

then be reformulated as an unconstrained minimization problem.

$$\min_{x(t)} \mathcal{S}[x(t)|y(t)] \tag{5.3}$$

These two problems, to constrained and unconstrained, are not disparate by any means. In fact, the limit of the unconstrained optimization problem returns us to the original constrained optimization problem.

$$\begin{aligned} &\min_{x(t_1)} \frac{1}{2} \sum_{m=1}^{M} \|x_m - y_m\|^2_{R_{\text{meas}}} = \min_{x(t)} \left\{ \lim_{R_{\text{model}} \to \infty} \mathcal{S}[x(t)|y(t)] \right\} \\ &\text{s.t.} \quad \dot{x} = f(x,t) \end{aligned} \tag{5.4}$$

By virtue of this relationship, we work in the limit of $\det R_{\text{model}} \to \infty$. The heuristic of annealing very naturally fits into our soft-constraint approach, almost suggesting that it be used. Notice that $R_{\text{model}}$ enters the equation much like an inverse temperature of, say, a Boltzmann distribution $p(x) \propto \exp\left(-\frac{E}{k_B T}\right)$. Working in the limit of $\det R_{\text{model}} \to \infty$ is akin to working in the low temperature regime from the annealing point of view. In annealing, allowing the system to cool slowly will generate solutions of the optimization problem that lie within deeper and more stable minima. Following this logic, we can approach $\det R_{\text{model}} \to \infty$ slowly by varying the individual components of the matrix $R_{\text{model}}$. We give a new index $\beta$ to the this matrix such that $R^{(\beta)}_{\text{model}}$ is the $\beta$-th iterate. The same adjustment is made for the action where $\mathcal{S}^{(\beta)}[x(t)|y(t)]$ corresponds to the old action $\mathcal{S}[x(t)|y(t)]$ with the substitution $R_{\text{model}} \leftarrow R^{(\beta)}_{\text{model}}$.

$$\lim_{\beta \to \infty} \left\{ \min_{x(t)} \mathcal{S}^{(\beta)}[x(t)|y(t)] \right\} \tag{5.5}$$

Practically speaking, the implementation is simple. We iteratively perform the optimization routine, using the minimizer/solution $x^*(t)$ of the previous routine as the initial condition of the following call of the optimization routine. Between each optimization call, we have to also slightly increase the magnitude of $R^{(\beta+1)}_{\text{model}}$. We use a simple exponential update to illustrate the change, but more complicated and advanced update rules can always be implemented instead. There should be no change at all in the core optimization routine aside from these changes.

$$R^{(\beta+1)}_{\text{model}} \leftarrow \alpha\, R^{(\beta)}_{\text{model}} \quad \text{s.t.} \quad \alpha > 1 \tag{5.6}$$

Once this entire optimization routine is complete, we say that we have solve the constrained optimization problem. We keep the value of $R^{(0)}_{\text{model}} < 1$, but not too much smaller. Practically speaking, we have found that the stopping condition of $\det R^{(\text{final})}_{\text{model}} \simeq 10^5 \det R^{(0)}_{\text{model}}$ is a fair place to stop. This corresponds to roughly 20 to 150 iterations of the method, depending on the chosen value of $\alpha$. The scaling constant $c$ is a hyperparameter that needs to be adjusted depending on the problem. We usually choose $\alpha$ to be as close to 1 as possible, since the exponential nature of the scaling is quite dominant as $\beta$ gets larger. We noticed that the necessity of using smaller values of $\alpha$ is tied to high complexity of the dynamics $x_{m+1} = F(x_m, t)$.

$$R^{(\beta)}_{\text{model}} = R^{(0)}_{\text{model}} \alpha^{\beta} \tag{5.7}$$

We can also think of the value of $R^{(\beta)}_{\text{model}}$ as an indicator of the more of the behavior. Values of $R^{(\beta)}_{\text{model}} < 1$ can be thought of as the 'exploratory phase' of the Simulated Annealing addition to HMC. since the system will not consider the nonlinear effects coming $x_{m+1} - F(x_m, t_m)$ very much. Hence, the choice of $R^{(0)}_{\text{model}} < 1$ is so that the system may should more of its 'artificial' time on exploration. When $R^{(0)}_{\text{model}} \sim 1$ is roughly of order unity, there is no clear phase that it is in. But as $R^{(0)}_{\text{model}} \gg 1$, we can think about the system as 'forgetting' the data set, at least explicitly speaking. The contribution of the data set can be thought of as

At this point, the astute reader would recognize at this point that everything has been framed from the perspective of minimization, whereas the Monte Carlo methods are about generating samples. Optimization and generating samples with Markov chains are equivalent problems, and the same routine employed about can be used for the Monte Carlo methods. Everything discusses in about annealing so far will apply once we replace the minimization routine with a Monte Carlo sample generation routine.

### 5.1.3 Plateauing of the Action

For a moment, let us rewrite the action by extracting the $R$ diagonal matrices and treat them as constants temporarily. This is done for simplicity can even be implemented in code following this scheme, provided that the ratios of the diagonal elements of $R$ are handled carefully.

$$\mathcal{S}[x(t)|y(t)] = \frac{1}{2}\sum_{m=1}^{M} R_{\text{meas}} \|x_m - y_m\|^2 + \frac{1}{2}\sum_{m=1}^{M-1} R_{\text{model}} \|x_{m+1} - F(x_m, t_m)\|^2 \tag{5.8}$$

Since we have allowed $R_{\text{model}}$ to be updated according to the rule $R_{\text{model}}^{(k+1)} \leftarrow c\, R_{\text{model}}^{(k)}$, it can effectively be treated as a variable that changes in artificial time $\tau$. We will not be handling the explicit relationship between $R_{\text{model}}$ and $\tau$ here, as it adds nothing but clutter for now. We will impose that $R_{\text{model}}$ increases as $\tau$ decreases, monotonically, so $R_{\text{model}} \to \infty$ implies $\tau \to 0$. We then take a look at how the action varies in $R_{\text{model}}$.

$$\begin{aligned}\frac{\mathrm{d}\mathcal{S}}{\mathrm{d}R_{\text{model}}} = \sum_{m=1}^{M} R_{\text{meas}} \frac{\mathrm{d}x_m}{\mathrm{d}R_{\text{model}}} \cdot (x_m - y_m) \\ + \sum_{m=1}^{M-1} R_{\text{model}} \left( \frac{\mathrm{d}x_{m+1}}{\mathrm{d}R_{\text{model}}} - \frac{\partial F}{\partial x_m} \cdot \frac{\mathrm{d}x_m}{\mathrm{d}R_{\text{model}}} \right) \cdot \Big( x_{m+1} - F(x_m, t_m) \Big) \\ + \frac{1}{2} \sum_{m=1}^{M-1} \| x_{m+1} - F(x_m, t_m) \|^2\end{aligned} \tag{5.9}$$

Notice that the first two terms can be simplified by factoring out $\frac{\mathrm{d}x_m}{\mathrm{d}R_{\text{model}}}$ from both terms. This common factor $\frac{\mathrm{d}x_m}{\mathrm{d}R_{\text{model}}}$ vanishes as $R_{\text{model}} \to \infty$, which also pushes the first term to zero. The second term is a little trickier to handle, as the large term $R_{\text{model}}$ is present. However, we argue that both $\big(x_{m+1} - F(x_m, t_m)\big)$ and $\frac{\mathrm{d}x_m}{\mathrm{d}R_{\text{model}}}$ approach zero, and these two terms out-compete the growing $R_{\text{model}}$ term. The arguments thus far have argued for the disappearance of the first two terms are from a numerical point of view. Analytically speaking, these first two terms are in fact the Euler-Lagrange equations of the action if we define $\mathcal{S}[x(t)|y(t)] \equiv \sum_m \mathcal{L}[x_m, x_{m+1}]$, where the sum is over the system time $t$ (not optimization time $\tau$).

$$\left\{ R_{\text{meas}}(x_m - y_m) + \frac{1}{2} R_{\text{model}} \frac{\partial}{\partial x_m} \Big[ x_{m+1} - F(x_m, t_m) \Big]^2 \right\} \cdot \frac{\mathrm{d}x_m}{\mathrm{d}R_{\text{model}}} \tag{5.10}$$

By virtue of the first two terms satisfying the Euler-Lagrange equations of the original system, they necessarily must equal to zero for the stationary path $x^*(t)$ that satisfies the dynamics. The claim that the first two terms are zero stands – only the last term remains. The full derivative of the action, with respect to the time proxy $R_{\text{model}}$ is then given solely by the same partial derivative, which is a positive definite quantity.

$$\frac{\mathrm{d}\mathcal{S}}{\mathrm{d}R_{\text{model}}} = \frac{\partial \mathcal{S}}{\partial R_{\text{model}}} = \frac{1}{2} \sum_{m=1}^{M-1} \| x_{m+1} - F(x_m, t_m) \|^2 > 0 \tag{5.11}$$

If and when our estimate of the state $x^*(t)$ converges, which should ultimately occur in the limit of $R_{\text{model}} \to \infty$, its derivative with respect to $R_{\text{model}}$ will be zero. This corresponds to our estimate converging to the best state estimate $x^*(t)$.

$$\lim_{R_{\text{model}} \to \infty} \frac{\mathrm{d}x(t)}{\mathrm{d}R_{\text{model}}}\bigg|_{x^*(t)} = 0 \tag{5.12}$$

The telltale sign of our approach is effective is a combination of the two previous equations. We will notice, as the method runs, that the action $\mathcal{S}[x(t)|y(t)]$ will steadily grow. Any behavior that is not a gradual and steady growth in actual should be met with suspicion. Upon reaching the best estimate or solution $x^*(t)$, the action will cease to increase, effectively forming a plateau, i.e. it has a zero slope with respect to the time proxy $R_{\text{model}} \to \infty$. Of course, the slope will never hit exactly zero, as there will be inevitable errors in the term $\|x_{m+1} - F(x_m, t_m)\|$ whether from an imperfect model or from simple numerical discretization [35]. A more general statement is that we will see a positive slope quickly approach a zero slope. We do not believe that such issues pose a serious problem to our method, as model-mismatch error should be small for a good model and numerical error scale with $\Delta t^2$. This may be useful for identifying seriously flawed models that do not describe the data, as our method will not display a plateauing of the action.

The fact that in the limit of $R_{\text{model}} \to \infty$, we see the action plateau is not tied to the act of annealing. Rather, annealing is a technique we can leverage to approach this limit while giving the optimization routine the added benefits of generally producing superior minima of the action. It also allows for the above analysis, which gives us some insight on and appreciation of the method. Our method makes no guarantee on the iterations of annealing steps required or the magnitude of the relevant hyperparameters. Lastly, keep in mind that $R_{\text{model}} \to \infty$ should be taken too literally, otherwise one would inevitably run into numerical precision issues.

We stress that anytime a plateau is observed, the solution will satisfy the equations of motion. That is not to say that the solution corresponds to the global minima, just any local minima. We have the possibility of multiple local minima since the dynamics of the system $x_{m+1} = F(x_m, t_m)$ are nonlinear. As usual with such problems, determining whether our local minima is indeed the global minima is a NP-hard (non-polynomial hard) problem. In other words, any such algorithm requires an unrealistic run-time that scales exponentially with problem size. We can offer, however, the next best thing – we can run many copies of this method in parallel to search for multiple solutions. The

solution that corresponds to the lowest action $\mathcal{S}[x(t)|y(t)]$ is the best that we could do under those circumstances. The following section will discuss why and how we use multiple initial conditions to give better solutions.

### 5.1.4 Artificial Time

The parameter that we are annealing $R_{\text{model}}$ is very similar to the inverse temperature of the system, generally called $\beta$ in those cases. Sending $R_{\text{model}} \to \infty$ is akin to sending temperature to zero. A rather interesting result occurs when we treat $R_{\text{model}}$ as the inverse of the artificial time, specifically $R_{\text{model}} = \frac{1}{\tau}$. Recall the probability density function with the following notation for HMC, detailed in subsection 4.3.1. We substitute the relation between $R_{\text{model}}$ and $\tau$.

$$\begin{aligned} p(\boldsymbol{X}|\boldsymbol{Y}) &= \int \mathcal{D}\boldsymbol{X}\ \exp\left\{-\mathcal{S}[\boldsymbol{X}|\boldsymbol{Y}]\right\} \\ &= \int \mathcal{D}\boldsymbol{X}\ \exp\left\{ -\frac{1}{2}R_{\text{meas}}\|\boldsymbol{X}-\boldsymbol{Y}\|^2 - \frac{1}{2\tau}\underbrace{\left\|L^+\boldsymbol{X} - \boldsymbol{F}(L^-\boldsymbol{X})\right\|^2}_{\equiv\|\boldsymbol{G}(\boldsymbol{X})\|^2} \right\} \end{aligned} \tag{5.13}$$

We look at the derivative of this probability function with respect to artificial time $\tau$, and it gives the following form.

$$\frac{\partial p}{\partial \tau} = p(\boldsymbol{X}|\boldsymbol{Y})\left[\frac{1}{2\tau^2}\|\boldsymbol{G}(\boldsymbol{X})\|^2\right] \tag{5.14}$$

We then use look at the second derivative with respect to the position $\boldsymbol{X}$, however, keeping only the highest order term. There have been many other terms that are neglected due to the $\frac{1}{\tau^2}$ dominating the behavior of the system at low values of $\tau$.

$$\frac{\partial^2 p}{\partial \boldsymbol{X}^2} \simeq p(\boldsymbol{X}|\boldsymbol{Y})\left[\frac{1}{\tau^2}\|\boldsymbol{G}(\boldsymbol{X})\|^2 \cdot \boldsymbol{G}'(\boldsymbol{X})^2\right] \quad \text{as} \quad \tau \to 0 \tag{5.15}$$

A particularly interesting relationship emerge as we recover the linearized diffusion equation, only valid for low values of $\tau$. Since we are prescribing the values of $R_{\text{model}}$ which is inversely proportional to time $R_{\text{model}} = \frac{1}{\tau}$, sending $R_{\text{model}} \to \infty$ is functionally the same as $\tau \to 0$.

$$\frac{\partial p}{\partial \tau} \simeq \frac{1}{2\boldsymbol{G}'(\boldsymbol{X})^2}\frac{\partial^2 p}{\partial \boldsymbol{X}^2} \quad \text{as} \quad \tau \to 0 \tag{5.16}$$

This implies that the act of annealing is attempting to 'undo diffusion' or decrease entropy,

which is a rather amusing implication of the algorithm. This allows further interpretation of the measurement process. Imagine that the probability density function exists for an unmeasured state, if it can be formulated that way, then it must be that of a delta function or an extremely sharp Gaussian. We can think of the act of measuring a state as taking a sample or a snapshot of a diffusion process unfolding in artificial time. As a result, we get a diffused state when measuring a system, where the width of the probability function is the magnitude of noise. From this point of view, the act of annealing is akin to reconstructing the original probability function at $\tau = 0$, which is well-approximated by a sharp Gaussian.

## 5.2 Multiple-Start and Parallelization

In the previous section, we discussed that finding a solution to nonlinear optimization is a difficult problem, requiring method that scale exponentially with the size of the problem. As for minimization, guaranteeing the global minima is not feasible for the high-dimensional problems that we are tackling. The best we can hope for is that we are not in a shallow minimum when there exists a much deeper minima close by. Again, we cannot guarantee this, but we have taken steps to attempt to break out of local minima, for example, the random sampling of momentum in HMC. Then again, we can still do more. We could run multiple copies of the same method, one copy on each processor (typically a CPU node). Each of these copies are differentiated from one another in some way or fashion. The two simplest ways are the following:

(i) Run the same method on each processor/core, but for a different initial condition

(ii) Run the same method on each processor/core, but using a different set of samples (for Monte Carlo only)

Approach (i) starts each method with a different initial condition, and is as simple as that. Assuming a completely deterministic algorithm, different initial conditions may place us in different basins of attraction, potentially exploring a wider space and getting deeper minima[1] This approach is a little crude and wasteful, since there will undoubtedly be overlap in the form of multiple initial conditions being in the same basin of attraction. A simple illustration of this is the case when there is only one minima of the system, then all initial conditions will be attracted towards the same point,

[1]Even though we use random sample, we can make the sampling deterministic by fixing the random seed that is used to generate random numbers, effectively assigning the same set of random numbers for instance of the algorithm.

making the entire practice redundant. Nevertheless, this practice has been fruitful when tackling more difficult problems at the cost of being inefficient at simpler tasks. We always have the option to not use this practice if we have *a priori* knowledge that the problem is simple.

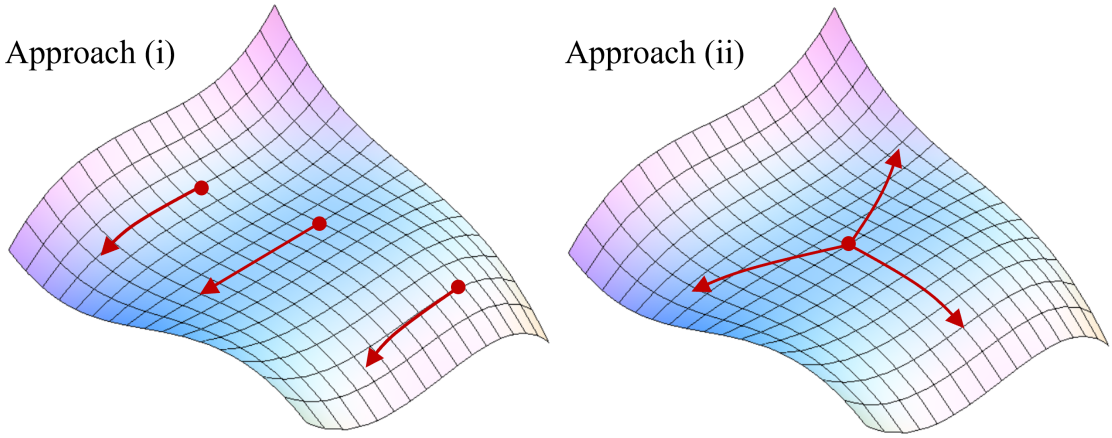


**Figure 5.2:** Illustration, from the HMC perspective, of approaches (i) and (ii) for handling multiple starts on the manifold defined by the action $\mathcal{S}[x(t)|y(t)]$. Approach (i) has multiple initial conditions, each with the same initial momentum vector $\boldsymbol{P}_1$. Approach (ii) has the same initial condition, but now each with a different initial momentum vector.

Approach (ii) only works when we use some kind of stochasticity in our method, so it is primarily applicable to the Monte Carlo methods (both RPMC and HMC). Strictly speaking, we can always introduce stochasticity into the quasi-Newton or gradient descent methods, but that is definitely a special case of their application[2]. In the case of HMC, a different set of momentum samples given to each instance of the algorithm means that each instance will explore the solution space in a different manner. A simple but illustrative example is that a different momentum will make each instance explore a different direction. One of these directions may or may not lead to a better minimum, but at the very least they will not be exploring the same space.

It turns out that we can even use both these approaches at once as long as there is no correlation between the initial position $\boldsymbol{X}_1$ and the initial momentum $\boldsymbol{P}_1$, using the terminology and notation of HMC. Each instance will explore the manifold from whatever initial position and momentum they were given, all in parallel. This parallelization is done on top of the annealing that was discussed previously. For the instances that do converge, we simply compare the final values of

[2]There are a whole communities that study nonlinear optimization method, with considerable success achieved by adding stochastic elements to otherwise deterministic methods. This research came to prominence with the introduction and success of the Stochastic Gradient Descent (SGD) algorithm to tackle the optimization problem associated with deep neural network and backpropagation. There are now entire families of methods that are extensions or variants of the SGD algorithm.

their actions and choose the solutions that corresponds to the lowest level action.

## 5.3 Remarks

Overall, these heuristics add an extra amount if robustness to our methods. We find that the added computational costs were not prohibitive, as parallel computing capabilities are becoming increasingly commonplace. The problems that we worked on to develop these methods are relatively small compared to the real world problems (weather systems, plasma, fluid dynamics, etc). We understand that computational limits and resources may be prohibitive for such problems, but we hope that they might and further research/development in these techniques might uncover additional methods or structures that allow for faster computations.

It is worth stressing that these additional and potentially costly heuristics are optional. They are not required in order for the algorithm to run. It might be best to think about them as extra steps that you can take if you have the means or the resources to do so. In our case, we did have these additional resources and we were also interested in the limits of these methods when we have done our utmost to get good results.

## 5.4 Acknowledgments

Chapter 5, in part, uses material and results that appears in Precision Annealing Monte Carlo Methods for Statistical Data Assimilation and Machine Learning, submitted in Physical Physical Review Research, 2(1), 013050. Fang, Zheng, Wong, Adrian S., Hao, Kangbo, Ty, Alexander J., and Abarbanel, Henry D. I. (2020). The dissertation author was one of the primary investigators and authors of this paper.

# 6

# Data Assimilation Results

## 6.1 Method of Choice

In this chapter, we present the results accumulated in the process of writing this work and explore their implications. We have covered three methods so far for evaluating the data assimilation integral, which are (i) the quasi-Newton root solver, (ii) Random-Proposal Monte Carlo, and (iii) Hamilton Monte Carlo. Functionally speaking, these three methods perform more or less the same task of evaluating the integral, but the strategies that they invoke are substantially different [36].

The quasi-Newton solver differs the most from the two Monte Carlo methods in that the solver attempts to find the location of the minima, whereas the Monte Carlo methods locates the approximate area of the minima and generates samples around it. The difference in the outcome is that the minimum seeking quasi-Newton method will accumulate more error if there is significant skew (third moment) in the distribution, or any higher odd-numbered moments, which diminishes the quality of the estimated states. The Monte Carlo methods are technically not affected by any of the moments, but the solution that they converge to depends on the number of samples that are taken around the minimum. The differences in these methods are fairly fundamental, so comparing them to one another is not a straightforward task.

The quasi-Newton solver, executed specifically with the L-BFGS algorithm, converges well when the size of the problem is not too large, such that the gradient information can be store on the CPU/processors cache (on-board memory), where memory fetches and storage is quick. If the gradient

information no longer fits on the cache, the code may quick become many orders of magnitude slower. The factors that may affect the emergence of this problem is the size of the cache relative to the size of the problem. It is difficult to address these issues without the focus of this work switching to that of high-performance computing. The RPMC method excels when GPU parallelization is readily available since the proposal mechanism is very inefficient. This method is perhaps the most agnostic to complexities of the model, but is beholden to the curse of dimensionality more so than the other methods due to the inefficient sampling. Lastly, the HMC method is very versatile in the sense that it inherently has an exploration and sampling stage built into it. HMC is also incredibly efficient in generating sample compared with the RPMC method. The tradeoff is that there are a number of extra hyperparameters that are introduced in this method compared to RPMC. There are sensible yet simple ways to pick these hyperparameters, which avoid spending considerable time tuning/finding parameters that work.

The ideal use cases of each method may vary significantly, depending primarily on the complexity of the equations and the size of the problem. As such, we avoid comparing these methods directly to one another. Our goal is to highlight the advantages of using the path integral approach to data assimilation. The individual methods are different instances of the same approach with some variety in computational approaches. Instead of examining the differences between the methods, which is not the intent of the work, we focus on using the HMC method to showcase the strengths of the path integral approach, which allows us to avoid duplicating results.

## 6.2 Details of the Setup

The data set that we focus on is the $D = 10$ dimensional chaotic Lorenz 1996 (L96) model. This L96 system is autonomous in that the dynamics do not have an explicit time dependence, but there is a uniform forcing parameter $\mu = 8.17$ applied to the vector field of every element. This puts the system into the chaotic dynamics regime. The spatial indices of the L96 system lay on a periodic grid where, for example, such that the vector field $\dot{x}_d$ at spatial grid point $d = 9$ sees data points from the $d = 7$, $d = 8$, and $d = 0$ spatial grid points.

$$\text{Lorenz 1996 model:} \qquad \dot{x}_d = (x_{i+1} - x_{i-2})x_{i-1} - x_i + \mu \tag{6.1}$$

These dynamics are simulated with a time step of $\Delta t = 0.025$ for a total of $M = 400$ steps, which gives a time-window of 10 arbitrary time units. This data is the measurement data that will be presented to the algorithm. We then continue the simulation for another $M = 400$ steps, but this segment of data is kept aside for later comparisons and not presented, in any fashion, to the algorithm. We use the explicit RK4 integrator to generate the subsequent state given the current state and its vector field. This data is considered synthetic data, in the sense that data is generated from a prescribed model where we have perfect information of the states. To make sure that the problem is not trivially easy, artificial additive noise is introduced to the data. The spectrum of the noise is given by the zero-mean Gaussian distribution and standard deviation of 2.5% of the total range of dynamics, in this case that value is roughly 0.6.

**Table 6.1:** Parameters for generating the Lorenz 1996 data.

| Variable | Description | Value |
|---|---|---|
| $\Delta t$ | Time step | 0.025 |
| $D$ | Number of dimensions | 10 |
| $M$ | Number of time steps for data and predictions | 400 |
| $\mu$ | Uniform forcing parameter | 8.17 |

Once the data set is generated, it is given to the HMC algorithm (detailed above) to run as prescribed. Upon completing one entire cycle of the HMC process, we increase $\beta$ and repeat the HMC process again. The HMC process with the new $\beta$ value is given the best guess from the previous $\beta$ value, and these steps are repeated $\beta_{\max} = 30$ times. This constitutes the **annealing method** that we laid out in Chapter 5. We run $N = 20$ copies of the HMC algorithm, each using a different set of random number generators. The initial condition that each copy gets is $\boldsymbol{Y}$ for the observed variables, whereas the unobserved variables are given by the dynamic initialization procedure laid out in Appendix D. The initial guess for the forcing parameter $\mu$ is randomly chosen from a sensible range of values, in our case we choose the uniform random distribution from 7 to 8. As a result, the sampling done by each copy of the algorithm is slightly different. Once all instances of the algorithms converge, we can choose to use any of the $N = 20$ solutions. In particular, we can use the final state within that time-window, and integrate that forward in time $M = 400$ steps to generate predictions. The resulting trajectory is compared with the $M = 400$ points of data that we previously generated, not presented to the algorithm, and kept aside for this explicit purpose.

$$\begin{aligned} &\text{Measurement Error:} \quad \frac{1}{2}\sum_{m=1}^{M} \|x_m - y_m\|^2 \\ &\text{Model Error:} \quad \frac{1}{2}\sum_{m=1}^{M-1} \|x_{m+1} - F(x_m, t_m)\|^2 \end{aligned} \tag{6.2}$$

Each of the $N = 20$ solutions should have different values associated with the action, measurement error, and model error. We can compare these $N = 20$ solutions based on these three metrics. The measurement error should reach a plateau and the annealing step progresses, signaling that the solution has converged. The model error should also decrease until it approaches a constant value, signaling that the solution obeyed the dynamics imposed by the model. Ideally, we want to observe that the measurement error increases until some limit, at which we say that it has converged. We also want to observe model error decreasing to indicate that we have found a solution that is consistent with the model.

**Table 6.2:** Parameters for the annealing process and parallelization instances.

| Variable | Description | Value |
|---|---|---|
| $\beta_{\max}$ | Total number of annealing steps | 30 |
| $N$ | Number of parallel instances | 20 |

For most cases, these 20 solutions will have different measurement and model errors. We should pick the solutions with the lowest model errors and lowest measurement errors. It is not clear, *a priori* exactly which of these two is more important. The action is one of the better candidates for determining the quality of our solution, with a lower action being indicative of a better solution. Once we have generated the measurement error and model error, we are able to observe the errors associated with each solution and exclude or include them in the calculation of an average trajectory that will be used for predictions.

$$\overline{\boldsymbol{X}} = \frac{1}{N}\sum_{n\in\{N\}} \boldsymbol{X}^{(n)} \tag{6.3}$$

Say that there are a set of solutions that pass muster such that they are deemed 'good solutions' – which we detail in the partial measurement case of 40% below. These solutions form the set $\{N\}$, of which there are $N$ such elements. The average solution that is used for both estimations, then prediction is given by the unweighted average of these 'good solutions'. We estimate the parameters

of the system the same way. After the averaging, we use the state at the end of the measurement window and the estimated parameters to generate predictions. We simply integrate the system forward in time, using the model dynamics previous specified, in order to get predictions.

## 6.3 Full State Measurements

The case of full state measurements refers to when all state variables are available as time-series data. Effectively, we are simply attempting to de-noise the data set. Trajectories that are inconsistent with the model dynamics will be discarded in the process. Such inconsistencies, which cannot be explained by the model dynamics, usually arise from the presence of noise. As we iterate the HMC procedure while slowly increasing the value of $\beta$ (keeping in mind that $R^{(\beta)}_{\text{model}} = R^{(0)}_{\text{model}}$), the algorithm refines the solution iteratively.

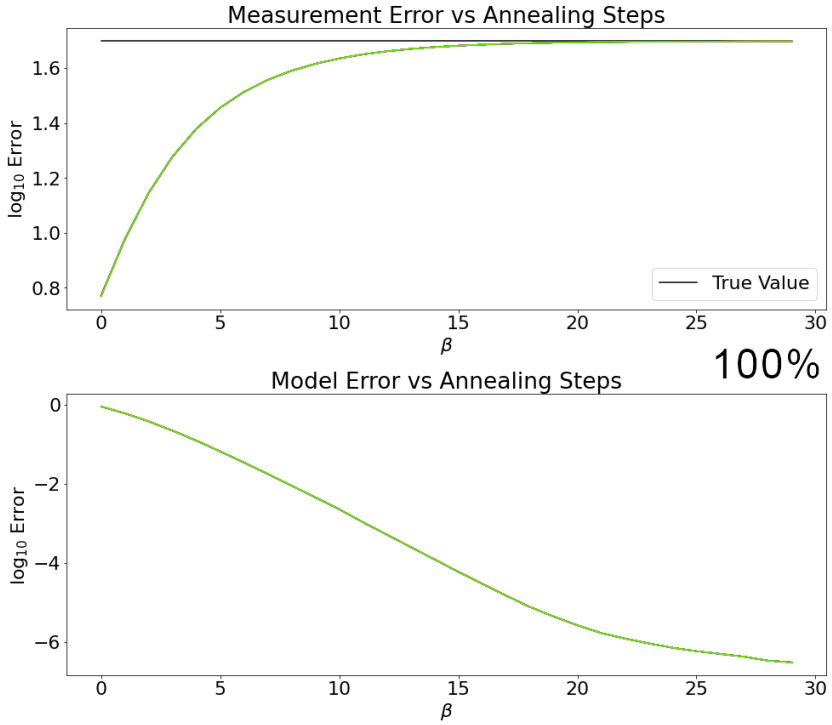


**Figure 6.1:** Measurement and model errors for the full measurement case as a function of annealing steps. All 20 instances are plotted here in different colors, with no particular order. The measurement error starts off small at $\beta = 0$ since we are using $\boldsymbol{X}_0 \leftarrow \boldsymbol{Y}$ to initialize the algorithm. It eventually plateaus, which indicates that our method has converged. Upon comparison to the true state (which we have kept aside earlier), our converged solution has the same error. The model error, on the other hand, starts off large at $\beta = 0$, since noisy data should not be consistent with the model. It eventually decreases to $10^{-6}$ as the solution state begins to agree more with the true state. All 20 solutions overlap one another completely for both graphs.

Only the solutions that are consistent with the model survive as the algorithm iterates. As the process goes on, the measurement error should increase smoothly while the model error decreases, also smoothly, as shown in Figure 6.1. The figure shows an ideal case of the state estimation, though this problem is almost trivially easy. Both the measurement and model error for the full measurement case is given by the equation in (6.4). Once the estimation part of the algorithm is complete, we may now run the predictions based on the last estimated state of the trajectory, namely $x_M$ in Figure 6.2. The case of full state measurements is a rudimentary but informative first step to establish, before delving into the partial measurement case.

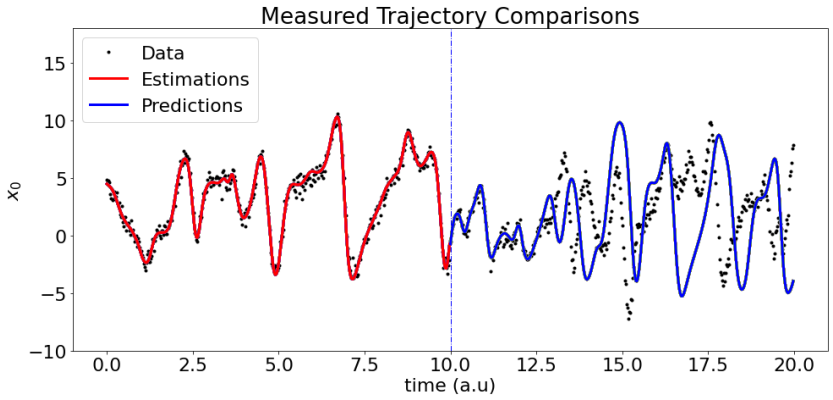


**Figure 6.2:** Estimations and predictions of the Lorenz 1996 model for the full measurement case. In this case, all 20 instances of the HMC algorithm converge to the exactly same trajectory (red) within numerical precision. The predictions (blue) agree well with the data (black) at first, only to diverge slightly later.

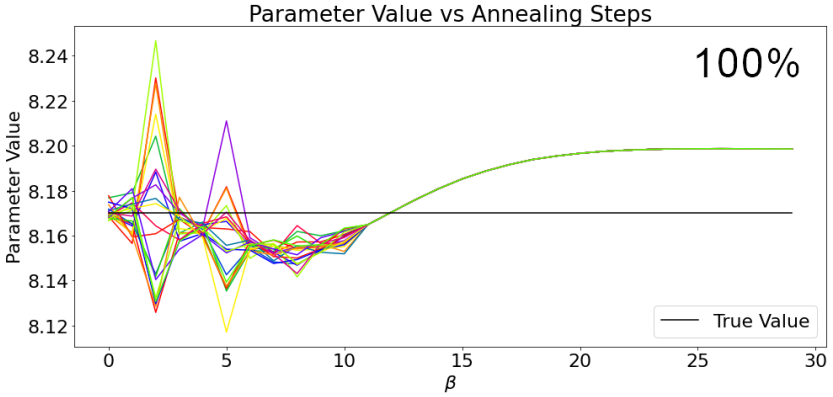


**Figure 6.3:** Convergence of the parameter values for the full measurement case. All 20 difference instances are plotted here, each in a different color with no particular order. The parameters have some variance at the lower value of $\beta$ only to converge around step 11. For some reason, all instances the algorithm converges to the value of roughly 8.20 when the true value is 8.17.

## 6.4 Partial State Measurements

The partial measurement case refers to the situation where the time-series for an entire state variable is missing from the measured data set. Situation like these may arise when there are broken sensors, or if certain variables of the system cannot be reasonably measured for practical/experimental reasons. The case of a broken sensor is easy to imagine, a simple case is when a car's speedometer only displays zero, or when the barometer of a weather station is not functional while the thermometer and wind gauge work just fine.

**Table 6.3:** List of measured variables for the cases demonstrated in this work. We attempts to space the measured variables out uniformly to avoid huge consecutive patches of missing data, which will immediately deteriorate the quality of the estimations and predictions.

| Case | Measured Variables |
|---|---|
| 100% | [0,1,2,3,4,5,6,7,8,9] |
| 80% | [0,1,3,4,5,6,8,9] |
| 60% | [0,1,3,5,6,8] |
| 50% | [0,2,4,6,8] |
| 40% | [0,3,5,8] |

There is some slight modification necessary for the measurement error term that was previously used, so that we may take into account the fact that only a subset of the state variables are measured. Let the indices of the measured variables be given by the set $\{L\}$ where $L$ itself is the number of measured variables, the measurement error will only need to take into account these variables, because only these variables are presented to the algorithm for optimization.

$$\begin{aligned} &\text{Measurement Error:} \quad \frac{1}{2}\sum_{\ell\in\{L\}} \|x_m - y_m\|^2 \\ &\text{Model Error:} \qquad\quad \frac{1}{2}\sum_{m=1}^{M-1} \|x_{m+1} - F(x_m, t_m)\|^2 \end{aligned} \tag{6.4}$$

As mentioned early, the measured variables are initialized with whatever values are present from $\boldsymbol{Y}$. We would like the unmeasured values to be 'close by' the true solutions, so the Dynamic Initialization trick of Appendix D is used as a 'better than random' initial guess. When choosing which variables to omit, we purposefully try to keep the omitted variables uniformly separated so that we not leave huge gaps between measured variables, as shown in Table 6.3.

The difficulty of the task increases as the number of measured variables decreases, as one would expect [37]. There is nothing particularly interesting between the 100% to 60% measurement

case, so figures will be shown only for the 80% and 60% cases with no discussion on them otherwise. To summarize the observation, the results almost identical for these cases. We do see some interesting behavior, however, in the 50% and 40% measurement cases. We shall discuss the observation in their respective subsection. Values below the 40% mark give results that do not display any reliable ability for predictions, so we omit these cases. To be clear, any of the conclusions that we arrive at within this entire section is conditioned on the Lorenz 1996 system, for the parameters prescribed to it specifically within this work.

### 6.4.1 80% Measurements

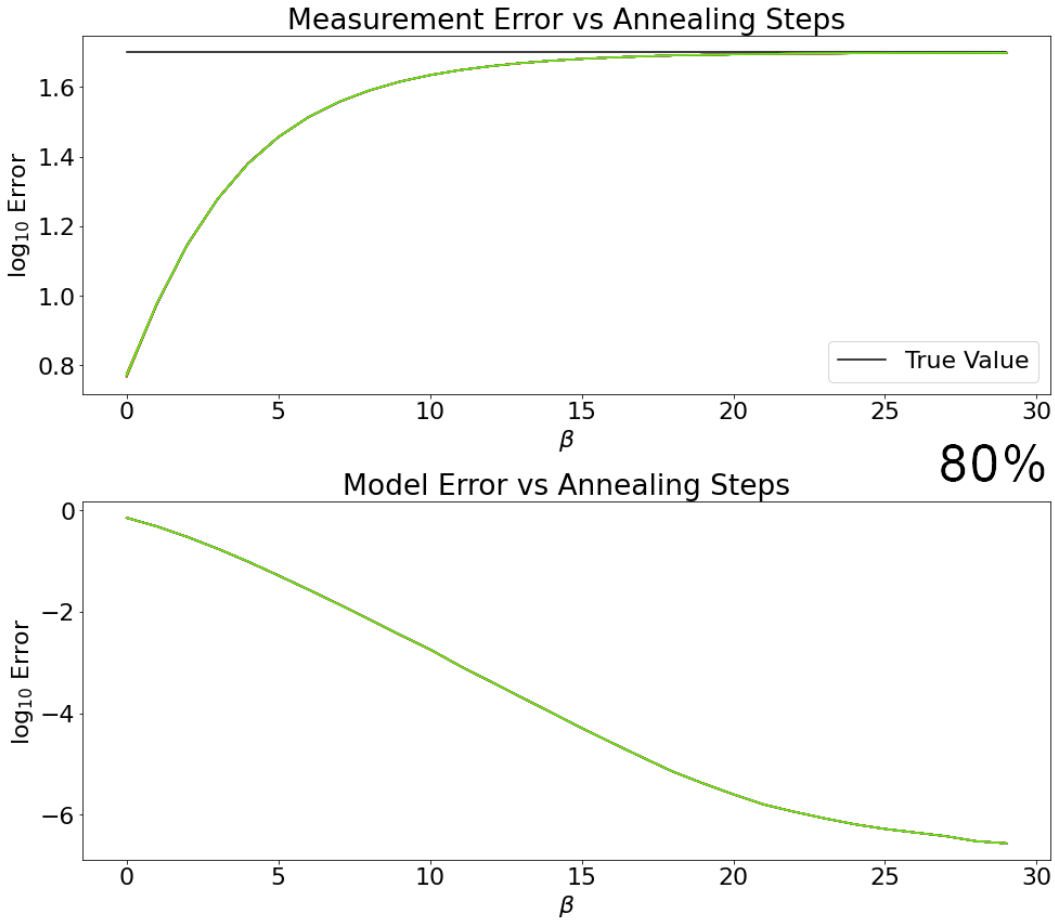


**Figure 6.4:** Measurement and model errors of the 80% measurement case. These values are not noticeably different from the full/100% measurement case. Side-by-side, they look almost identical, indicating that manifold of the 80% case is qualitatively similar to the 100% case.

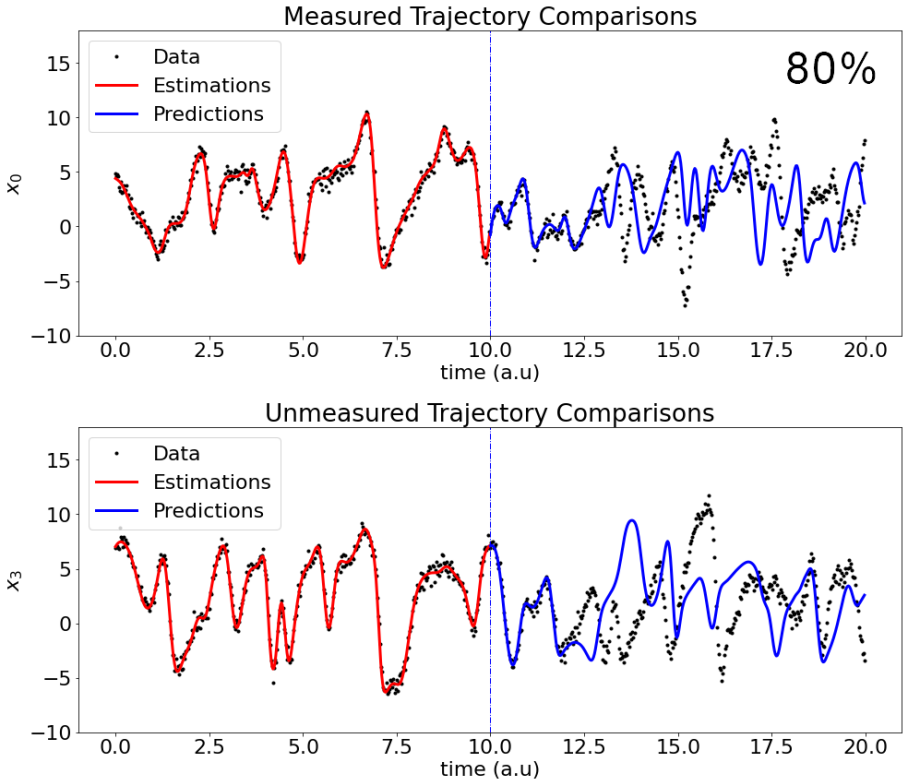


**Figure 6.5:** State estimation and prediction for the 80% case. All 20 instances of the algorithm have converged to within numerical error. In fact, the 80% and 100% case have exactly the same solution. This convergence may indicate the smoothness of the manifold defined by our data assimilation action. The prediction, as expected, is the same as the 100% case for both the measured and unmeasured cases. The bottom figure shows an unmeasured variable, where the data was available but artificially held back from the code, i.e. it was not presented to the code.

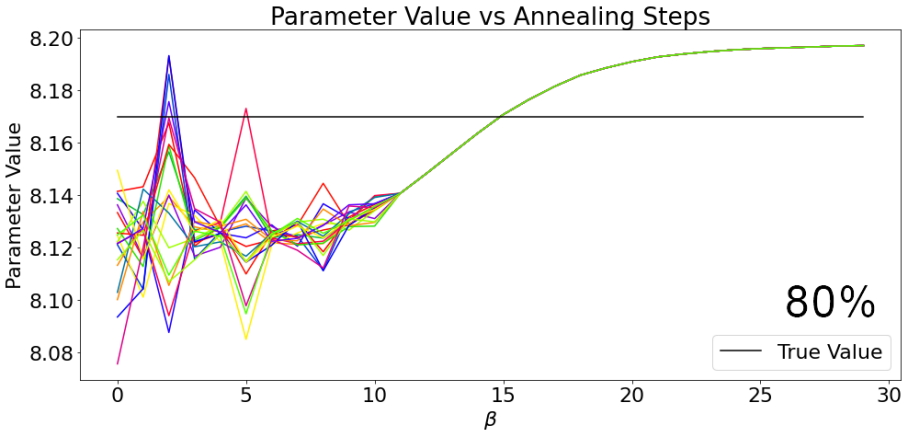


**Figure 6.6:** The parameter values for the 80% case start out different, but converge to the same value within numerical error. This is unsurprising given the numerical convergence of the estimated trajectories.

### 6.4.2 60% Measurements

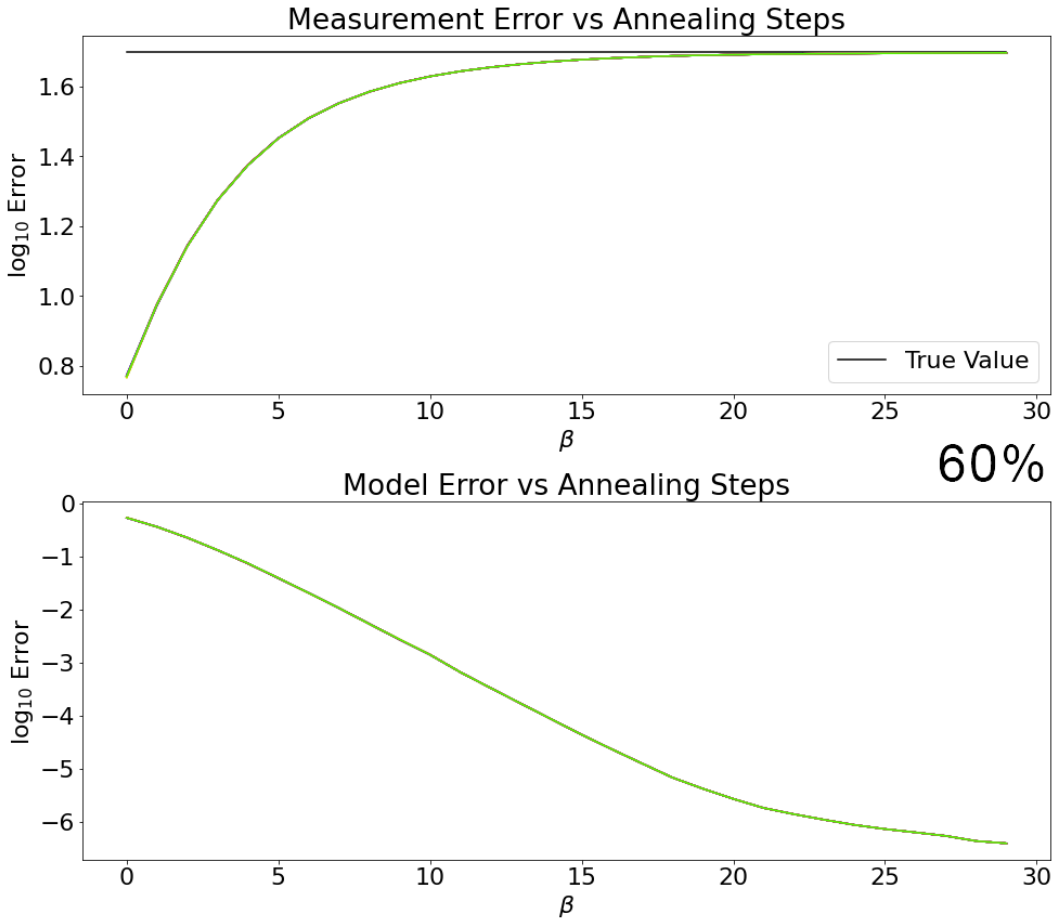


**Figure 6.7:** Measurement and model errors for the 60% measurement case. Both these graphs are nearly identical to both the 80% and 100% cases. This indicates the simplicity of the landscape all the way down to the 60% measurement case.

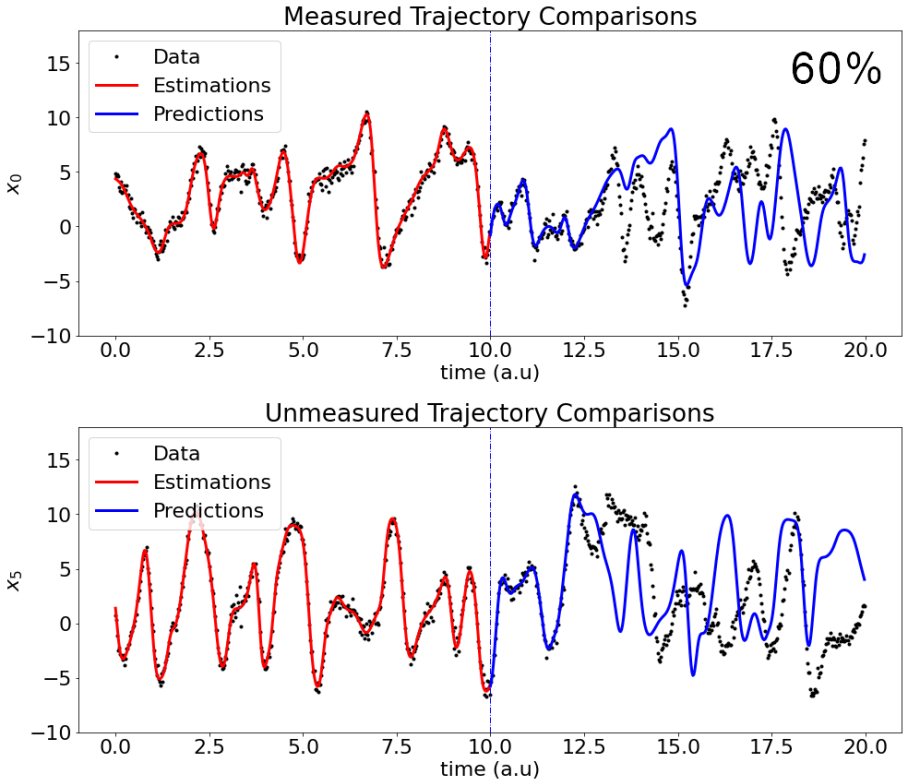


**Figure 6.8:** The state estimation and prediction of the 60% measurement case for both a measured (top) and unmeasured (bottom) variables. The unmeasured variable for this case is different from the 80% case. All 20 instances of the algorithm converge to the same numerical solution, which is different from the solutions in the 80% and 100% cases (which were both the same to one another). The duration of an accurate prediction is roughly the same.

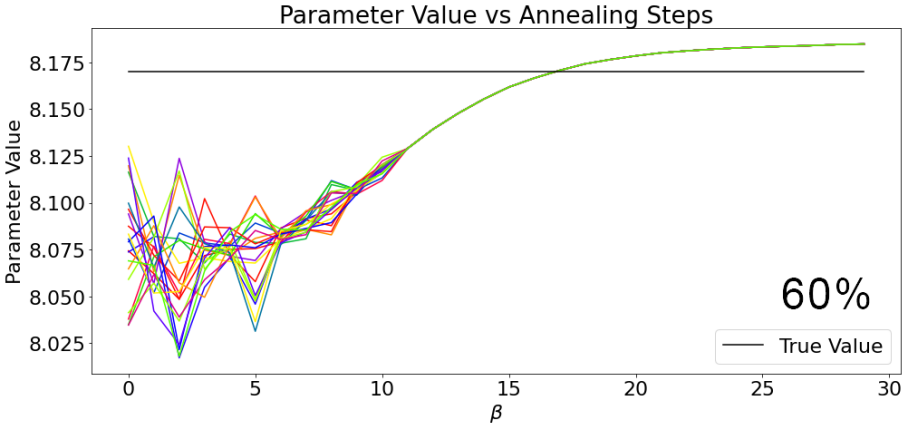


**Figure 6.9:** The parameter values for the 60% case converge in the same fashion as the 80% and 100% cases. However, the converged value at high values of $\beta$ is different from the 80% and 100% cases.

### 6.4.3 50% measurements

Thus far, the 100%, 80% and 60% cases had nothing substantial differentiating them from one another, but were nevertheless included because their similarities were themselves telling of some interesting phenomenon happening. Once we hit the 50% measurement case, however, we begin to see new behavior that was not observed in the cases that came before this. Specifically, 1 of the 20 parallel instances of the algorithm strays from the rest as the annealing step progresses. The algorithm solves for a state that very noticeably misses the global minima, and appears to be approaching a local minimum. This is the first time in the progression of this work that there is deviation in results among the 20 parallel instances.

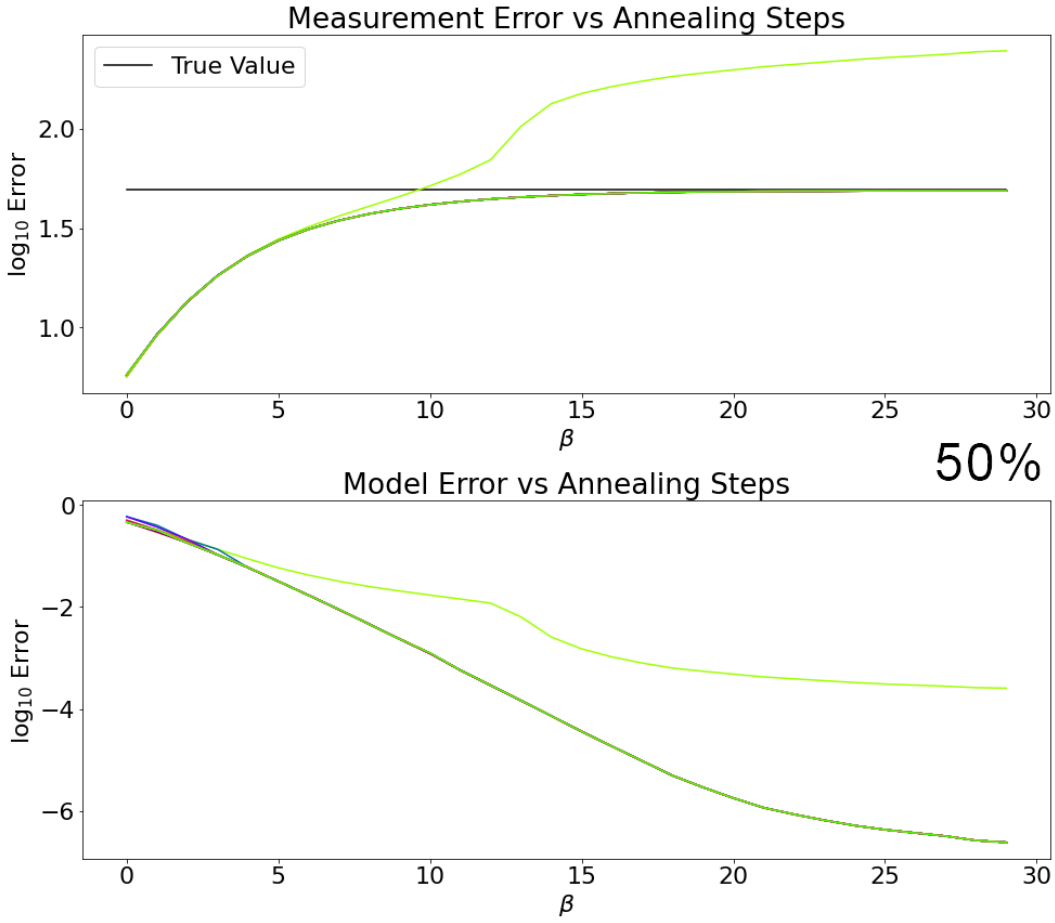


**Figure 6.10: Top:** As the annealing step progresses, the measurement error of 1 out of the 20 parallel instances (lime green) strays from the rest by growing significantly faster, noting that the $y$-axis is on a logarithmic scale. The state corresponding to the lime green appears to have missed the global minimum and instead settles in a local minimum that is close by. **Bottom:** The model error of the stray state decays much slower as the annealing steps proceed.

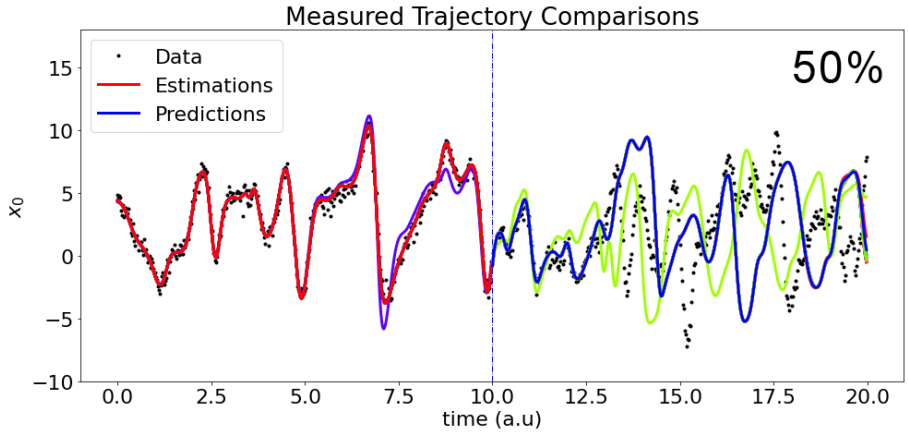


**Figure 6.11:** In the estimation window, one state (purple) generates a trajectory that is different from the rest of the (more accurate) estimated states. The stray state, in the estimation window, generates predictions (lime green) that deviates much quicker from the withheld data (black), as compared to the state that correspond to the lower measurement error (blue).

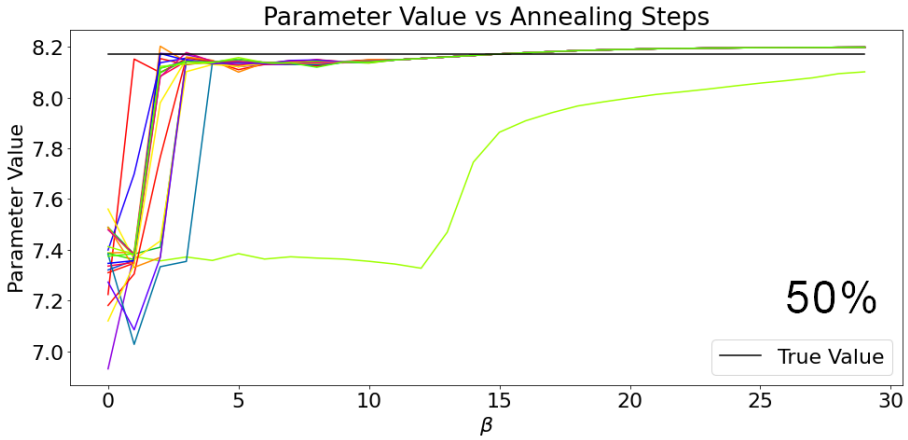


**Figure 6.12:** The estimated parameter values for the 50% case given the data. There is one instance whose parameter is poorly estimated and, unsurprisingly, this is the same instances that would generate poor estimations and predictions. For this reason, the solution that corresponds to this outlier is excluded from the analysis and usage in predictions.

The lone stray solution will noticeably have a higher measurement error and, in this case, also displays a higher model error. There is no reason to expect that stray solutions to always correspond to a higher model error. The model error is low when a trajectory obeys the dynamical equations of motion, used as a constraint here, well. Since it is always possible to find trajectories that satisfy the equations of motion, say by simply integration any initial condition on the attractor, a low model error is insufficient as a metric to determine a good solution.

As we are lowering the number of measured variables, this is the first time that we see a

'splitting ' of action levels as the annealing progresses. This suggests that the splitting of action leveling may be used an indicator of the prediction quality. Specially, it appears that higher values of the action, in general, corresponds to poorer quality of predictions and lower values of the action correspond to better quality predictions. Trajectories with a higher action either do not agree with the model dynamics or the data, as compared to the trajectories with lower action level. This may seem as slightly unsurprising due to the definitions of the measurement and model errors. However, it is encouraging that we could use these errors as a proxy for prediction quality, even though the errors are only defined in the measurement window. We could think of these errors as the amount of information that has been extracted from the data set.

### 6.4.4 40% measurements

The 40% measurement case is interesting because of the severe degradation in the average quality of predictions, as compared to the 50% case. Even though the difference is just one less variable being measured, it seems to be a significant challenge to make predictions in the 40% case [38]. This is demonstrated by having only 6 out of the 20 parallel instances of the algorithm converging to the expected level of measurement error. Indeed, this is a difficult problem to handle in general, not just for our algorithm to handle. This is attributed to a significant portion of the measurement data being missing. It happens that, at least for the $D = 10$ case, that this is the lower limit of partial measurements – going to 30% measurement results in not plateauing of the action whatsoever, regardless of how many parallel instances are used. As such, the results of cases with 30% measurements and lower are omitted.

When attempting to make predictions, we exclude the trajectories associated with higher errors and only use the 6 lowest-error trajectories. These 6 lowest trajectories are chosen *a priori* – that is, without knowing the expected errors beforehand. This can be done by looking for a cluster of low-error trajectories. Once we have found the cluster of low-error trajectories, we take their unweighted average and generate the predictions as usual. Though the average quality of the 40% measurement case is much lower than the rest, selecting the cluster of low-error trajectories yields predictions that are just accurate as the higher-measurement cases.

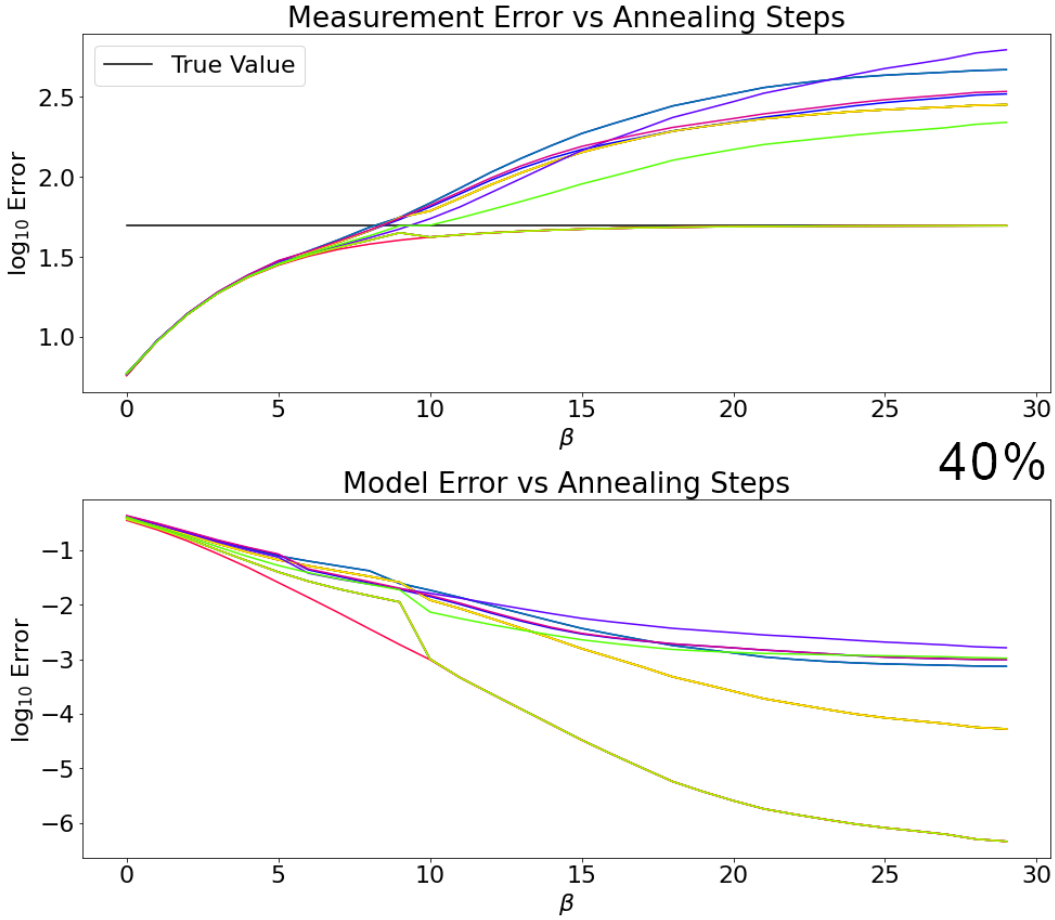


**Figure 6.13:** Measurement and model error of the 40% measurement case. Here, only 7 of the 20 parallel instances reach the theoretical minimum amount of error while the remaining 13 instances do not. Several of the 14 instances congregate at another error level, and we expect this to correspond to the existence of a nearby local minimum.

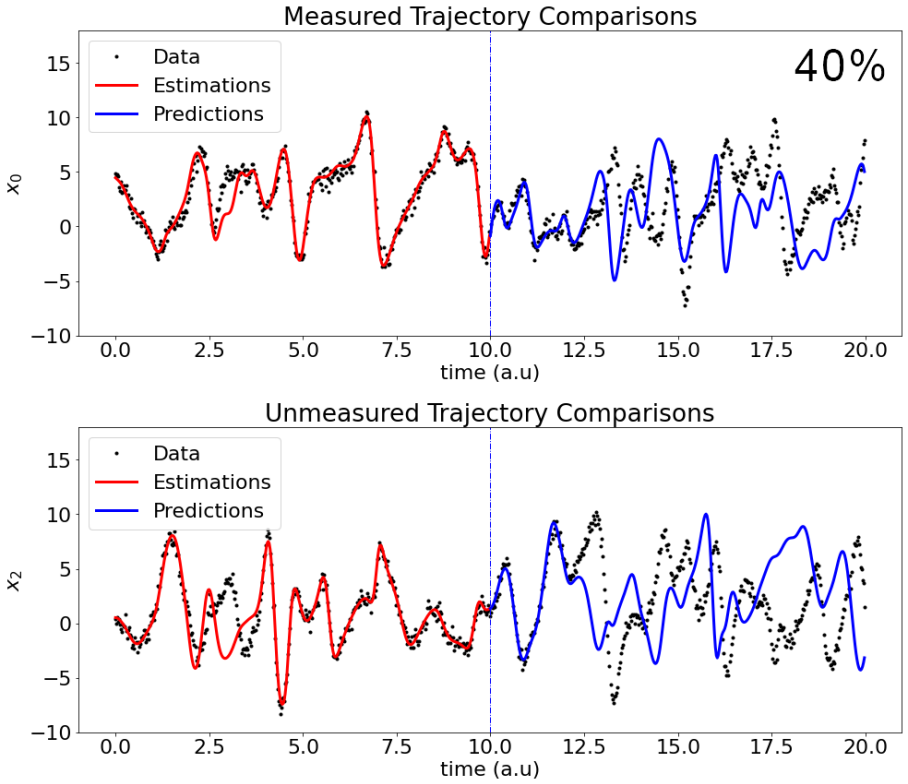


**Figure 6.14:** Using the solutions of the 7 lowest error values, we reconstruct the estimated states (red) for the measured and unmeasured variables of the 40% case. We use the final state of the measurement window to run predictions (blue), and they agree will with the data. The quality of the predictions is comparable to the 100%, 80%, 60% and 50% cases. As such, the dominant error in the prediction window is attributed to the limit of predictability of chaotic systems, not the algorithm.

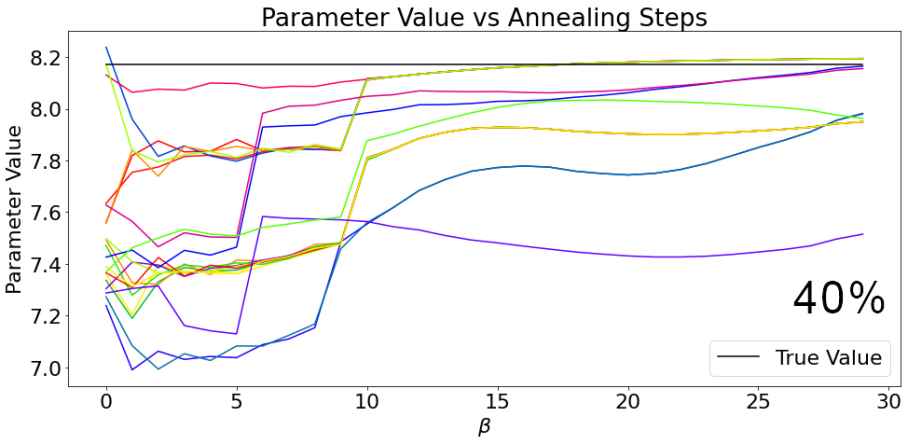


**Figure 6.15:** Parameter estimation history of the 40% measurement case. In this graphic, only 7 out of the 20 instances resulted in an estimated parameter close to our expected value. A good estimation of the parameter has direct effects on both the measurement and model errors.

## 6.5 Conclusions

Our method of data assimilation which iteratively enforces a model-consistency term is able to estimate trajectories of a given system well. It can even reconstruct variables that were not explicitly given to the system – what we call unmeasured state variables – due the consistency term of our data assimilation action. We started with 100% measurements, so the algorithm was effectively just a denoiser that recovered the true state of the system without any the noise. In our case, the added noise was artificial but in general this corresponds to noise that comes from the measurement apparatus.

The number of measured variables is gradually lowered, but it generated the same results down to the 60% measurement case. At the 50% measurement case, we begin to see that not every instances of the algorithm converges to the global minimum. The 50% case shows one instances that has strayed from the others into another local minimum. The 40% case shows even fewer trajectories converging to the global minimum, but choosing the trajectories that correspond to the lower cluster of errors still yielded predictions in line with the other cases.

Across the board, we are able to generate prediction that agree well with the (artificially withheld) prediction data up to 2.5 arbitrary time units, which is roughly 4 Lyapunov times[1]. Given the fact that the algorithm is able to perform well with more than half of the state variables being missing is very encouraging.

## 6.6 Acknowledgments

Chapter 6, in part, uses material and results that appears in Precision Annealing Monte Carlo Methods for Statistical Data Assimilation and Machine Learning, submitted in Physical Physical Review Research, 2(1), 013050. Fang, Zheng, Wong, Adrian S., Hao, Kangbo, Ty, Alexander J., and Abarbanel, Henry D. I. (2020). The dissertation author was one of the primary investigators and authors of this paper.

[1] A single Lyapunov time is the reciprocal of the largest Lyapunov exponent of the system. In our case of the Lorenz 1996 system with a uniform time-independent forcing parameter of 8.17, the largest Lyapunov exponent is roughly 1.6. The Lyapunov time of Lyapunov timescale is then roughly 0.6 arbitrary time units.

# Part II

# Model-free Methods

# 7

# Reservoir Computing

*Any sufficiently advanced technology is indistinguishable from magic.*

- Arthur C. Clarke, *Profiles of the Future*

## 7.1 Disclaimer

First and foremost, the terminology landscape of this field is difficult to navigate cleanly. The names and definitions vary from author to author, from subfield to subfield, and even from decade to decade. Some early authors refer to this method/approach exclusively as 'reservoir computing', and the structure doing the computation as 'the reservoir'. Other authors refer to it solely as 'reservoir computers', without distinguishing the method/approach from the structure that performs the computation. Both camps are slightly awkward grammatically, and I find myself somewhere in between them both. Fortunately, it does not seem that differing terminology impedes the understanding of the core concepts. Most readers should be able to parse this out with little effort if they even notice the difference at all. In this work, I will use semantics that are most natural to myself, which are some weighted average of what many established and contemporary authors would use [39].

## 7.2 Introduction

A reservoir computer (the method) takes input data $\boldsymbol{u}(t)$ **sequentially** from an external system of interest. The dynamics of the system that generates $\boldsymbol{u}(t)$ are one of the following: a) not

**Table 7.1:** Some of the variables and terminology in reservoir computing.

| Variable | Description |
|---|---|
| $\boldsymbol{u}(t)$ | data presented to the reservoir |
| $\boldsymbol{r}(t)$ | reservoir state |
| $W_{in}$ | input weights |
| $\boldsymbol{g}(\boldsymbol{r}, \boldsymbol{u})$ | dynamics of listening/estimating reservoir |
| $\boldsymbol{\Psi}[\cdot]$ | projection function |
| $\hat{\boldsymbol{u}}(t)$ | reservoir reconstruction of $\boldsymbol{u}(t)$ |

generally known, b) known but certain parameters are not measured or not well estimated, or c) known but the full model is too computationally expensive to warrant its use. All systems are nonlinear to some extent even though linear assumptions are often made, but in this work, we will further focus on chaotic systems. The reservoir computer itself also has an internal state $\boldsymbol{r}(t)$, sometimes called 'the reservoir', that is generated on-the-fly from its own previous state and some subset of the input data $\boldsymbol{u}(t)$. The reservoir states are generated **sequentially** from somewhat **arbitrary** nonlinear equations $\boldsymbol{g}(\boldsymbol{r}, \boldsymbol{u})$ as it takes in the data, so the reservoir can be viewed as a dynamical system in its own right. This process is called 'listening' in the ESN literature. The arbitrariness of these nonlinear equations does not exclude the use of other physical systems as the reservoir itself, but it merely suggests that the physical system that is used for the reservoir need not have any relation to the physical system of interest. If the equations $\boldsymbol{g}(\boldsymbol{r}, \boldsymbol{u})$ chosen to represent the reservoir has no physical basis, then the reservoir is best classified as an artificial neural network (ANN), specifically an echo state network (ESN) [40]. There is also another camp where biologically inspired equations of spiking neurons are used, called liquid state machines (LSM). Though the differences between ESN and LSM are mostly historic in nature, the treatment in this work is general enough to encapsulate both schools. The function $\boldsymbol{g}(\boldsymbol{r}, \boldsymbol{u})$ describing the dynamics of the system is called the 'activation function' and, in ESN, sigmoidal functions like the hyperbolic tangent or the logistic function is chosen. But other arbitrary functions like a sine or cosine function still results in a well-performing reservoir, no better on average than a sigmoidal function. Any sufficiently smooth function should work.

Due to the reservoir having to update its own state, it is classified more specifically as a recurrent neural network (RNN). Once the reservoir has 'listened' to the input data sufficiently, only the final/output/readout layer is subject to training, as compared to other networks where the internal or hidden layers receive some training. Only training the final layer greatly reduces the training time for reservoir computers, as compared to its RNN cousins, which often use the rather expensive and unstable Back-Propagation-Through-Time method. The output layer is chosen to be

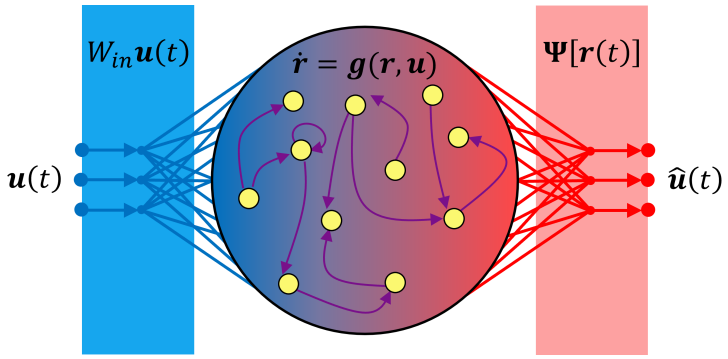


**Figure 7.1:** Structure of the Reservoir Computer in the listening and training mode.

linear with respect to the reservoir state, which introduces some uniqueness to the solution and is also the source of the quick training times. This linear readout layer of reservoir computers bypasses any vanishing/exploding gradient problems that many RNNs face completely. The result is a RNN that has the potential to be a universal function approximator [41]. However, there are still tradeoffs. Reservoir computers use random matrices, and there are only loose heuristics on how to properly construct these random matrices. Moreover, the random nature of generating these matrices leaves a lot of variability in performance unexplained. Despite the randomness and the use of heuristics, these properties make reservoir computing uniquely tailored toward dealing with sequential or time-series data.

The realm of predicting the future states of physical systems given some time-series data is generally and traditionally classified under the large umbrella of data assimilation. The data assimilation community mostly excludes the machine learning revolution, including reservoir computing, because such methods do not make use of physical model or physical constraints. This is justified by the lack of physical realism in most machine learning systems, such as not obeying the conservation of mass. Yet, there is a huge effort by many to bridge this apparent divine between machine learning and (traditional) data assimilation, to some success in hopes of leveraging the best of both worlds. My own interests in reservoir computing comes from the fact that the reservoir itself is a dynamical system, specifically a dynamical system that is being 'driven' by the input data. As such, many of the techniques, intuitions, and insights that are common in dynamical systems theory should directly apply to reservoir computing. Also, it seems like the effectiveness of reservoir computing can be attributed to **synchronization**, and helps peel back the black-box nature of reservoir computers by, potentially, providing some mathematical guarantees for the method. Synchronization of the reservoir and the

data is an interesting phenomenon in its own right, but successfully leveraging synchronization might facilitate the widespread use of reservoir computing.

## 7.3 Structure

Let us start off with an overview of the requirement of the data before diving into the specifics of the reservoir. To begin, we should have access to some time-series data set with some noise $\boldsymbol{u}(t)$, given to us by a trusted expert of the field of interest. This time-series is $D$-dimensional and consists of $M$ evenly spaced measurements. Any qualms regarding the quality, measurability, sampling resolutions, and anything of practical consequence will be tabled for subsequent sections – we shall assume merely that the data set $\boldsymbol{u}(t)$ is 'viable'. The only thing worth addressing about $\boldsymbol{u}(t)$ now is that measurements are rarely made on the actual state of the system, but rather on some downstream and more practical variables. We shall assume, in the setup here, that we have access to the state variables. The specifics will be discussed in a later section. As mentioned previously, we will focus here on times series data coming from chaotic systems. There are two versions of the reservoir, continuous and discrete, and it seems that both versions work much the same [42]. Our focus in this work will be on the continuous version. The structure of the reservoir can be cleanly broken down into three parts: input layer, reservoir dynamics, and output layer.

$$\begin{aligned} \text{Continuous: } \dot{\boldsymbol{r}}(t) &= \gamma(-\boldsymbol{r}(t) + \text{activ}[A\boldsymbol{r}(t) + \sigma W_{in}\boldsymbol{u}(t)]) \equiv \boldsymbol{g}(\boldsymbol{r}, \boldsymbol{u}) \\ \text{Discrete: } \boldsymbol{r}(t_{m+1}) &= \text{activ}[A\boldsymbol{r}(t_m) + \sigma W_{in}\boldsymbol{u}(t_m)] \end{aligned} \tag{7.1}$$

### 7.3.1 Input Layer

At each point in time, $\boldsymbol{u}(t)$ is embedded into a much higher $N$-dimensional ($N \gg D$) space through the use of an input weight matrix $W_{in}$. This input matrix $W_{in}$ is a relatively sparse and skinny $N \times D$ matrix that acts on $\boldsymbol{u}(t)$ as a preliminary step before this input is subject to use by the reservoir $\boldsymbol{r}(t) \in \mathbb{R}^N$. The sparsity of this matrix arises from the fact that each of the $N$ nodes inside the reservoir receives input from exactly one of the input states of $\boldsymbol{u}(t)$ at random, leaving the fraction of non-zero matrix entries at exactly $1/D$.

These relations do not vary as time progresses in the data set. The non-zeros entries of the $W_{in}$ matrix are stochastically generated using the Uniform distribution between $(-1, 1)$, though a

Normal distribution of standard deviation of 1 appears to work equally well. Since the row space of $W_{in}$ are of order unity on average, the hyperparameter $\sigma$ can be seen as a scaling parameter for the resulting $W_{in}\boldsymbol{u}(t)$ term.

The $\sigma$ term is perhaps the single most important hyperparameter in the reservoir. It is akin to the coupling parameter in the synchronization literature and shares the Goldilocks property – if $\sigma$ is either too high or too low, synchronization will not occur. It is also responsible for generating the correct Lyapunov spectrum in the reservoir dynamics. As usual, even for such an important hyperparameter, there is no *a priori* way of choosing $\sigma$ such that the reservoir has the desired behavior. A combination of heuristics and trial-and-error is necessary for finding the appropriate $\sigma$, but it just needs to be found once for a particular system. Luckily, there is a fairly large margin of error for getting the right value for $\sigma$, such that a small discrepancy in $\sigma$ does no generate wildly differing reservoir dynamics.

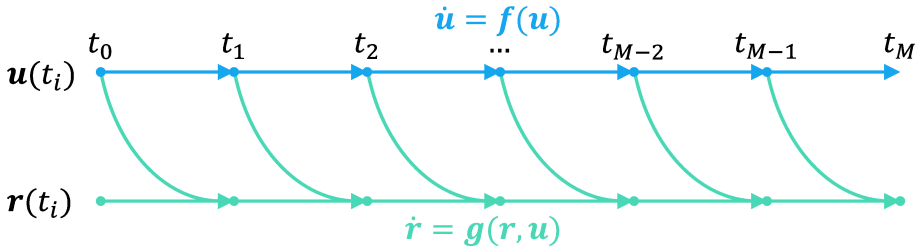


**Figure 7.2:** Flow diagram of the reservoir evolution. The reservoir uses both its previous internal state and the previous data point to update its state forward in time. This figure is for the continuous reservoir, but it holds equivalently for the discrete reservoir as well.

### 7.3.2 Reservoir Dynamics and Listening

There is also a matrix $A$ that determines the nature of the connectivity between the nodes $\boldsymbol{r}(t)$ within the reservoir, unsurprisingly called the connectivity matrix. Just like the $W_{in}$ matrix, the $A$ matrix is also sparse, however the sparse of the matrix is set explicitly such that only 2% of the matrix elements are non-zero, primarily so that the computation is much faster. Empirical evidence suggests that the sparsity may aid in the reservoir's performance, but this hyperparameter has worked for a variety of values and, of course, it is specific to the data set and the system of interest. Again, just like the $W_{in}$ matrix, the non-zero elements of $A$ are also generated stochastically, and the use of either the Uniform $[-1, 1)$ or the unit Normal distribution work equally well. Unlike the $W_{in}$ matrix, the spectral radius (largest eigenvalue) of $A$ is adjusted on a case-by-case basis, but this is no different

from the role of $\sigma$ for the input. The act of generating trajectories for reservoir states $\boldsymbol{r}(t)$ while taking in the data $\boldsymbol{u}(t)$ is also known as 'listening'.

There are two definitions of reservoir dynamics, which are the discrete or continuous reservoirs. Both are explored in the confines of this work, and there seems to be only minor differences their use. The discrete reservoir has less hyperparameters to tune, but the trajectories of $\boldsymbol{r}(t)$ generated are little bit more noisy/jagged than the continuous counterparts, but for the most part, parameters that work for the discrete and continuous reservoir are very similar. The evaluation of the advance state $\boldsymbol{r}(t_{m+1})$ is straightforward and quick. The continuous reservoir seems to be better at 'denoising' in general, since zero-mean noise in the input $\boldsymbol{u}(t)$ is fed into the $\dot{\boldsymbol{r}}$ equation, which shows up as a short-lived diffusion of trajectories. It also requires some integrator (typically RK2 or RK4) as an additional step, resulting in run times that are a multiple times slower than the discrete counterparts. This is no different from the relation between continuous and discrete dynamical systems.

A more general reservoir structure can sometimes be used for the continuous reservoir, where the magnitude and signs of the $\boldsymbol{r}(t)$ and activation functions are tuned, but this omitted in this work as the additional hyperparameter introduces another dimension of tuning when the degree of tuning required of this system is already very involved. The run times for the continuous reservoir is directly proportional to the order of time-integrator used and is much longer than the subsequent training phase. The listening phase is the bulk of the computation time by a huge margin and much effort should be spent in making sure that this phase is quick and efficient.

**Table 7.2:** Values for the reservoir parameters when using full state measurements. Partial measurement use a slightly different set of parameters.

| Variable | Description | Value |
|---|---|---|
| $N$ | number of dimensions/nodes | 1000 |
| $\gamma$ | time scaling constant | 23 |
| $\sigma$ | coupling constant | 0.03 |
| $\rho$ | density of non-zero elements of connectivity matrix $A$ | 2% |
| $\lambda$ | spectral radius of connectivity matrix $A$ | 0.9 |
| activ$[\cdot]$ | activation function | tanh$[\cdot]$ |

### 7.3.3 Training Output Layer and Estimation

Regardless of whether the discrete or continuous versions of the reservoir architecture is used, the resulting trajectory of $\boldsymbol{r}(t)$ has to be fit onto the input data $\boldsymbol{u}(t)$. Specifically, we have generated a time series $\boldsymbol{r}(t)$ from its own internal dynamics and $\boldsymbol{u}(t)$, but the goal now is to find a function that

best projects $\boldsymbol{r}(t_i)$ onto $\boldsymbol{u}(t_i)$ at every point in time. The simplest and most common practice is to use a Linear Least Squares (LLS) fit to find some projection $W_{out}$ such that $\boldsymbol{u}(t_i)$ and $W_{out}\boldsymbol{r}(t_i)$ are, on average, as close to one another as possible. All the data was used to train the reservoir including the transient data, which was very short lived. Empirically, it did not matter whether or not the transient data was used, as the on-attractor data points outnumber the transients by two or three orders of magnitude generally.

$$\underset{W_{out}}{\operatorname{argmin}} \sum_i \left\| \boldsymbol{u}(t_i) - W_{out}\boldsymbol{r}(t_i) \right\|_2^2 \tag{7.2}$$

This is the preferred method by many because, most importantly, solving for this relation is linear in $W_{out}$. Hence, it is quick and the solution $W_{out}$ is unique to $\boldsymbol{u}(t)$ and $\boldsymbol{r}(t)$. The tool of choice is the usual (Moore-Penrose) pseudoinverse, powered by the Singular Value Decomposition (SVD). Other methods involve constructing an extended basis $\boldsymbol{q}(t)$ and fitting this extended reservoir state, rather than the original state $\boldsymbol{r}(t)$, to $\boldsymbol{u}(t)$. Here, $\boldsymbol{q}(t)$ is constructed from nonlinear combinations of the reservoir state $\boldsymbol{r}(t)$, e.g. $\boldsymbol{q}(t) = [\boldsymbol{r}(t), \boldsymbol{r}^2(t)]$ where $\boldsymbol{r}^2(t)$ is the squared value of $\boldsymbol{r}(t)$ at each temporal and spatial point.

$$\underset{W_{out}}{\operatorname{argmin}} \sum_i \left\| \boldsymbol{u}(t_i) - \widetilde{W}_{out}\boldsymbol{q}(t_i) \right\|_2^2 \tag{7.3}$$

Of course, one could add more and more polynomial terms to $\boldsymbol{q}(t)$ *ad infinitum*, but the computational costs of doing so is hardly justifiable given that stopping at first order already gives marvelous results. In fact, the most general treatment has it that $\boldsymbol{u}(t) = \boldsymbol{\Psi}[\boldsymbol{r}(t)]$, but the space of such possible functions is effectively infinite, rendering the search for $\boldsymbol{\Psi}[\,\cdot\,]$ nigh impossible. For most applications, it seems that the first or second order (first order being the standard LLS) basis is more than enough to produce good predictions. The following sections will proceed assuming the use of the LLS fit as it performs very well, and it keeps the computational time low.

DO NOT do the following: $\boldsymbol{u}(t_i) = W_{out}\boldsymbol{r}(t_i) \quad \Rightarrow \quad W_{out}\cancel{\boldsymbol{r}(t_i)\boldsymbol{r}(t_i)^\dagger}^{\mathbb{1}} = \boldsymbol{u}(t_i)\boldsymbol{r}(t_i)^\dagger$ (7.4)

Next, we should talk about the practical matter of solving for $W_{out}$. If no additional structure

is imposed on $W_{out}$, then the method of choice would be the Moore-Penrose pseudo-inverse (denoted with †) shown above. Keeping in mind that the time series $\boldsymbol{r}(t)$ is treated as a $N \times M$ matrix, where $N$ is the number of reservoir nodes and $M$ is the number of measurements across time[1]. The resulting object $\boldsymbol{r}(t_i)\boldsymbol{r}(t_i)^\dagger$ is the $N \times N$ identity matrix and $W_{out}$ seems to solved with $\boldsymbol{u}(t_i)\boldsymbol{r}(t_i)^\dagger$. However, due to the large values of $M$ and $N$, the condition number of $\boldsymbol{r}(t_i)^\dagger$ is also large, passing on these large condition numbers to $W_{out}$ as well[2]. Repeated application of $W_{out}$ in the algorithm generated numerical instabilities which can be avoided through regularization shown below.

$$\boldsymbol{u}(t_i) = W_{out}\boldsymbol{r}(t_i) \quad \Rightarrow \quad \boldsymbol{u}(t_i)\boldsymbol{r}(t_i)^\top = W_{out}\boldsymbol{r}(t_i)\boldsymbol{r}(t_i)^\top \tag{7.5}$$

$$W_{out} = \boldsymbol{u}(t_i)\boldsymbol{r}(t_i)^\top \left[\boldsymbol{r}(t_i)\boldsymbol{r}(t_i)^\top + \alpha^2 \mathbb{1}\right]^{-1} \tag{7.6}$$

This regularization procedure may seem unnecessarily complicated, but it is absolutely necessary in order for predictions to be stable. It should be noted that this additional regularization is due to the inherent numerical stability of the LLS algorithm, not the reservoir computing framework per se. Practically speaking, this regularization reduces the strictness of the fit. Throughout this work, the value of the regularization parameter $\alpha = 0.01$ was used.

### 7.3.4 Echoing and Prediction

After this minimization or fitting task, we get access to the best-fit of $\boldsymbol{u}(t)$ which is given by $\hat{\boldsymbol{u}}(t) \equiv W_{out}\boldsymbol{r}(t)$. It is important to distinguish between the actual data $\boldsymbol{u}(t)$ and the reconstructed or best fit $\boldsymbol{u}(t)$. We can then loop the outputs back into the reservoir, by feeding in the output $\hat{\boldsymbol{u}}(t)$ rather than the original input $\boldsymbol{u}(t)$. This opens up the idea of being able to predict the state of the system past the given measurements within the time window.

The reservoir is said to be in the prediction, echoing, or autonomous mode when doing so. The term echoing comes from the audio-inspired naming scheme of classic literature. The description of the reservoir being autonomous comes from the lack of a driving/input stimulus $\boldsymbol{u}(t)$. Rather than

[1]The notation here has $M$ and $N$ flipped from the conventional notations. It is by complete coincidence an unfortunate downstream consequence of previous notation.

[2]The condition number is the ratio of the smallest to the largest eigenvalue or singular value, depending on the problem formulation. These condition numbers make more sense when discussing the SVD and eigendecomposition, but it directly affects the pseudo-inverse.

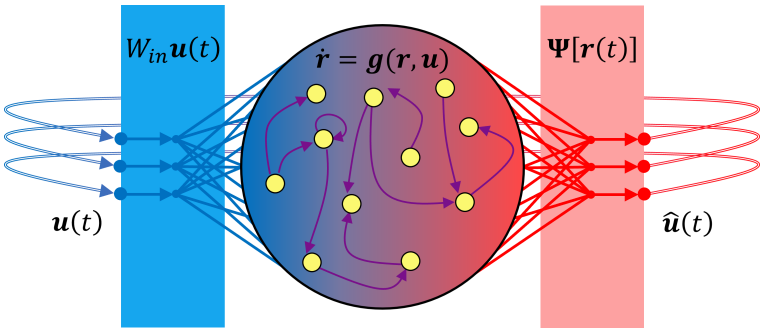


**Figure 7.3:** Structure of the Reservoir Computer in the predicting mode. This figure resembles the previous one except that output-to-input feedback is introduced, effectively defining the prediction mode.

following (7.1), we use the following equations instead, noting that $\boldsymbol{u}(t)$ is no longer used as $\hat{\boldsymbol{u}}(t)$ is purely a function of $\boldsymbol{r}(t)$.

$$\begin{aligned} \text{Continuous: } & \dot{\boldsymbol{r}}(t) = \gamma(-\boldsymbol{r}(t) + \text{activ}[A\boldsymbol{r}(t) + \sigma W_{in} W_{out} \boldsymbol{r}(t)]) \equiv \boldsymbol{g}(\boldsymbol{r}, \boldsymbol{u}) \\ \text{Discrete: } & \boldsymbol{r}(t_{m+1}) = \text{activ}[A\boldsymbol{r}(t_m) + \sigma W_{in} W_{out} \boldsymbol{r}(t)] \end{aligned} \tag{7.7}$$

Hopefully, at this point, the reservoir state $\boldsymbol{r}(t)$ and the weights $W_{in}$ has learned or captured something about the incoming steam of data, and is capable of producing accurate predictions. Once the hyperparameters have values that make the reservoir 'receptive' to the stimulus $\boldsymbol{u}(t)$, these predictions accurately predict the data, which is to say that it can forecast states of the system that it has never encountered before. In a colloquial sense, during the prediction phase, we are fooling the reservoir into believing that it is still being driven by the input data $\boldsymbol{u}(t_i)$, whereas in reality it is being driven by $\hat{\boldsymbol{u}}(t_i) = W_{out}\boldsymbol{r}(t_i)$.

## 7.4 Acknowledgments

Chapter 7, in part, uses material and results that appears in Robust Forecasting Using Predictive Generalized Synchronization in Reservoir Computing, submitted in Chaos: An Interdisciplinary Journal of Nonlinear Science , 31(12), 123118. Platt, Jason A., Wong, Adrian S., Clark, Randall, Penny, Stephen G., and Abarbanel, Henry D. I. (2021). The dissertation author was one of the co-authors of this paper.

# 8

# Results from Reservoir Computing

## 8.1 Introduction

Before even considering the idea of acquiring data, there are a few things consider. We have to first assume that there are a set of variables that can uniquely describe the system at all times. We also add in the extra constraint that we want a minimal set of variables such that we describe the system is a concise fashion. This is mainly to avoid scenarios where one can always introduce superfluous variables that may be linear combinations of another variable or, worse, variables that do not contribute any descriptive power at all. This minimal set makes up the state variables, and this state is usually (and hopefully) a unique description of the system. Once we have an idea of what physical quantities constitute the state, we can start thinking about measurements.

We also have to consider that the state variables might not be directly measurable due to practical limitations. For example, the temperature field over a large region of the ocean surface is the desired measurement for weather prediction, in whatever grid scheme or resolution necessary. It would be silly to even imagine placing thousands of millions of thermometers at every grid point. Instead, the one measures the intensity of radiation on those grid points, at the wavelengths that correspond to water's absorption bands[1]. The measured data $\boldsymbol{y}$ is a nonlinear function of the actual state variables $\boldsymbol{x}$ such that $\boldsymbol{y} = \boldsymbol{h}(\boldsymbol{x})$, and one must trust that $\boldsymbol{h}(\,\cdot\,)$ and $\boldsymbol{y}$ are properly vetted by the experts of the respective field. On top of all this, there is noise in the system such that the data $y$ that we are given must be assumed to contain (usually Gaussian distributed) noise that $\boldsymbol{y} = \boldsymbol{y}_{true} + \boldsymbol{\eta}$.

In this work, we are concerned about the prediction process rather than the measurement process, so we happily assume that the measurement function is simply the identity $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{x}$. These methods should hold, in theory, when there is a measurement function present, but there will be some added steps and potentially even added complications along the way that will not be addressed here.

**Table 8.1:** Parameter values for the data generation. There is some slight but otherwise inconsequential overlap in notation with reservoir parameters.

| Variable | Description | Value |
|---|---|---|
| $\Delta t$ | uniform time-step | 0.01 |
| $T$ | time window | 50 |
| $M$ | number of time steps | 5000 |
| $D$ | dimension of system | 3 |
| $\sigma$ | Prandtl number | 10 |
| $\rho$ | Rayleigh number | 28 |
| $\beta$ | convection constant | 8/3 |
| $\eta$ | variance of uniform noise | 1.5 |

Having access to the full state is a true luxury when it comes measurements, especially observational experiments – experiments that are not under strict laboratory controls. One every-day example would be weather systems, which are extremely high dimensional system. It is impossible to measure the wind speed and pressure at every point in a small town. But even if we are in a controlled laboratory setting, we still might not have access to certain variables. Such is the case in many biological systems, like neurons for instance, where the membrane potential can be measured, whereas the gating variables (quantifying the degree that certain ion channels are able to let ion through the gates) cannot be realistically measured at all. This is not so much a technological limit as it is the fact that measurements fundamentally change the system that they are measuring. In the case of biological systems, the act of measuring certain variables could cause irreparable damage to the organism or the cell, therefore changing the function of it entirely[2]. Another example is in fluid dynamics, where it is extremely difficult to measure the flow field at every spatial point without blocking the flow and fundamentally changing the system. Thankfully, the lack of full measurements does not put a halt to our ability to make predictions. One has to 'fill in' the data somehow, and the standard approach is to use the delay-embedding theorem of Takens [43].

[1]These are some examples of handy tidbits to know about the system but, for the most part, we will be relying on the experts in the field that specialize in the physics and engineering aspects of acquiring these measurements. The reality of the situation is that these measurements are very difficult to make. There are other factors such as cloud cover or the overlap of the absorption bands with other molecules that might confound the measurement process, but knowing just a little bit about the system of study is always a plus.

[2]The phenomena of death makes organisms truly nonlinear, among other things.

As such, this is the justification of separating the results section into two parts, handling the full- and partial-state measurement cases. Procedurally, these two cases are handled in much the same way. Having access to the full state is a natural starting point, though wishfully ignoring some of the more practical aspects of real-world measurements. Its main contribution is that such systems are used as an instructional and informative situation for study and proofs of concept. We establish certain baseline expectations when full state measurements are available, we can then move on to the partial-state case. There is an additional step of using delay-coordinate embedding as a 'post-processing' of the partial data. Some familiarity with standard delay-embedding practice is exercise in this work; there will be no in-depth discussion of delay-coordinates, attractor dimensions, or the like. The reader is encouraged to read some of the references. Most of the numerical experiments are focused on the Lorenz 1963 system, but the results do hold more generally as well if one is willing to do the hyperparameter search. Some experiments were also conducted on the Lorenz 1996 system as well and those results will be presented later without much exposition on the specifics.

Before moving on to the actual results, it worth mentioning that results will vary depending on the random number generation (RNG) of the computer, even if the same hyperparameters were used. Because of the inherent luck of the draw, there is going to be some fluctuations of the results from seed to seed. **None of the reported results in this work were cherry picked such that they performed particularly well or badly.** As far as we are aware, they are run-of-the-mill results.

## 8.2 Full State Measurements

As mentioned, the use of full state measurements is a natural starting point for experiments with time series prediction, simply because it is simple and requires less knowledge and exposure. The model of choice that generates the data is the Lorenz (1963) model, which was a simplification of a weather system and the progenitor of the concept of chaos. We will skip the subtleties of the Lorenz model except for the fact that our choice of parameters, as shown in Table 8.1, puts the system in the chaotic regime [44]. Once the entire data set of generated, some random uniform noise was added to the data[3]. Both the true trajectory $\boldsymbol{u}_{true}(t)$ and the noisy trajectory $\boldsymbol{u}(t)$ were kept for later use. It should be stated explicitly that **only the noisy trajectory is ever presented to the reservoir system**, whereas the true trajectory $\boldsymbol{u}_{true}(t)$ is **only** used for error calculation purposes.

[3]It makes little difference to the reservoir computer whether or not the noise was Gaussian or uniform.

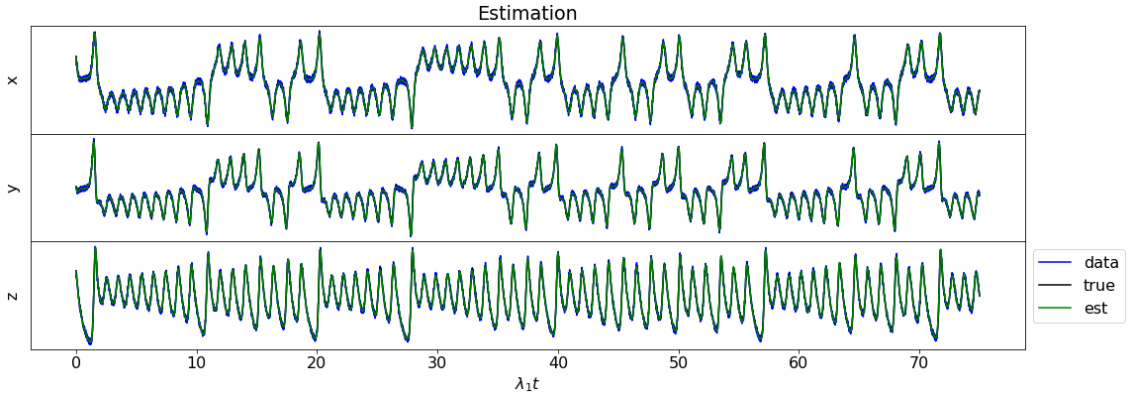


**Figure 8.1:** Estimation phase of the reservoir. 'True' refers to the noiseless simulation trajectory; 'data' refers to the data that one would get from measurements, which the 'true' state plus artificial noise; 'est' refers to the output of the reservoir after training. The true and estimated state are indistinguishable at this macroscopic scale. Microscopically, the difference resembles white noise.

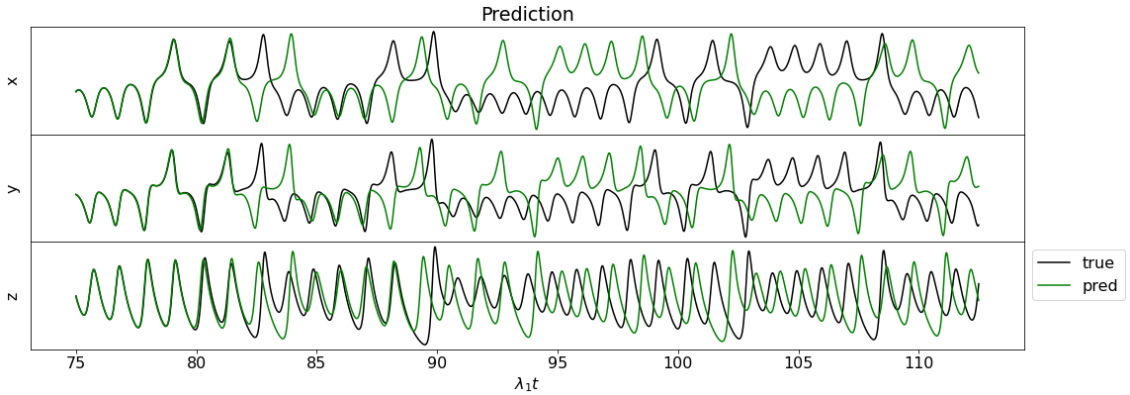


**Figure 8.2:** Comparison between the reservoir prediction and the actual/noiseless trajectory of the system. The reservoir is able to predict roughly seven Lyapunov times before prematurely making a lobe switch.

As shown in Figure 8.1, the reservoir is able to fit its internal states $\boldsymbol{r}(t)$ to the data $\boldsymbol{u}(t)$ very well. This is what one gets after the training phase is complete, and the combination of the listening and training phase is called the estimation. All observed cases have that the estimated state is always close to the true trajectory that the data itself. In other words, the estimated state is always less noisy than the data. This result is absolutely a necessary condition for the reservoir to predict beyond the window of available measurements. A reservoir that does a poor job estimating has not been able to generate good prediction, as one would naively expect.

Figure 8.2 show that the reservoir is able to make good predictions after a successful estimation phase. The overall attractor dynamics, sometimes called the 'climate', also seems to be replicated by

the reservoir. This level of prediction rivals the standard and traditional methods of data assimilation, but is made more impressive given that absolutely no model is given to the reservoir. Needless to say, this is particularly useful for real world scenarios where the physical system has parameters that also need to be estimated. The reservoir makes no distinction of bias towards the lack of parameters, though only sensible ranges of the parameters have been tested.

As mentioned earlier, these results will vary when attempting to reproduce them even though the exact hyperparameters are used. It is advised to use the same RNG seed for repeatability. The Lorenz system has certain regions of the attractor that vary in stability or instability. For example, the region between the two lobes is particularly unstable, leading to larger local Lyapunov exponents. In contrast, being within the lobes is generally more stable and the local Lyapunov exponents are smaller. One needs to be mindful when choosing the measurement window, such that it does not end at a particularly stable or unstable region of the attractor. Doing so may lead one to be mistakenly over- or under-confident in the reservoir's ability.

## 8.3 Partial State Measurements

This subsection will be very similar to the previous section in structure. The major difference being that this subsection assumes incomplete or partial measurements, and a time-delay embedding is used as a means of overcoming this obstacle. Such a method is standard in the field of dynamical system. There have also been other methods using RC that attempt to deal with incomplete state measurement. For example, there have been attempts to present one state variable to the RC and have it fit another state variable [45]. This will not be address in this work as we chose to focus on time-delayed formalism. We will briefly cover time-delay embedding, but a deeper dive will be included in the appendix. The exact same data and set up from the previous subsection is used here except that the $y$ and $z$ variables are purposely discarded for this partial measurement case. When doing time delay, we have to specify some new and important variables – one of which is the time-delay embedding (integer) dimension $D_y$.

This time-delay embedding state is given by $\boldsymbol{y}(t) = [x(t), x(t-\tau), x(t-2\tau), ..., x(t-\ell\tau)]$, and $\boldsymbol{y}(t)$ is fed into the reservoir instead of the full state $\boldsymbol{x}(t) = [x(t), y(t), z(t)]$. This example uses $\tau = 10\Delta t$ with $\Delta t = 0.01$ and $\ell = D_y - 1 = 2$. The embedding dimension is chosen to be $D_y = 3$ simply because it has been found to be the least number of dimensions necessary to capture the attractor

**Table 8.2:** Values for the reservoir parameters when using partial state measurements. Full measurement use a slightly different set of parameters.

| Variable | Description | Value |
|---|---|---|
| $N$ | number of dimensions/nodes | 1000 |
| $\gamma$ | time scaling constant | 23 |
| $\sigma$ | coupling constant | 0.12 |
| $\rho$ | density of non-zero elements of connectivity matrix $A$ | 2% |
| $\lambda$ | spectral radius of connectivity matrix $A$ | 0.9 |
| activ | activation function | tanh |

topology [46]. It just so happens that $D = D_y$ for the Lorenz system. In general, Takens' embedding theorem guarantees us that $D_y < 2D_{box} + 1$, where $D_{box}$ is the (fractional) box-counting dimension of the attractor. This is a rather fortunate and astounding result since it gives an upper bound of the time-delay embedding dimension when attempting to reconstruct a system in this manner. For this example, we are limiting ourselves to only measuring the $x$ variable, effectively a scalar time series, and then constructing the time-delay embedding from that. Since we are only measuring $x$, $D_y$ will also be the number of previous measurements of $x$, each delayed by some fixed interval $\tau$.

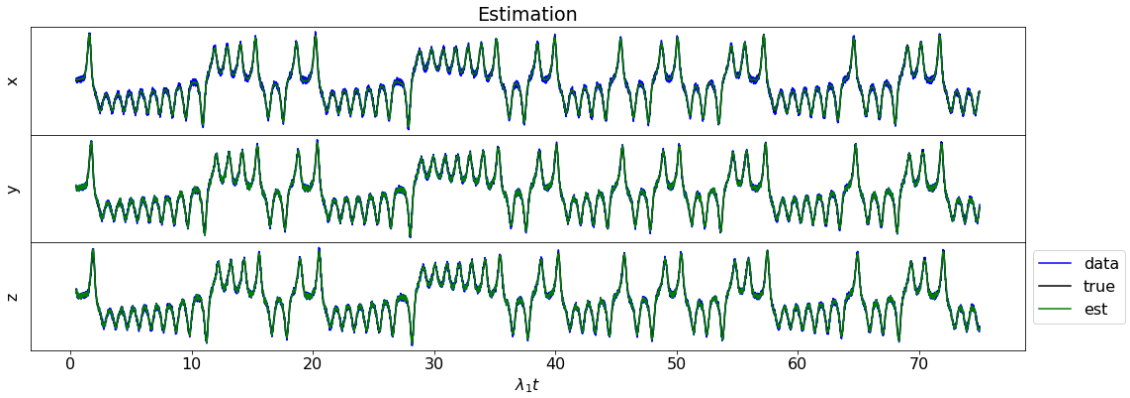


**Figure 8.3:** Similar to the full state case, the time-delayed or partial measurement case generated very good estimation.

The delay $\tau$ is chosen such that it is the first local minimum of the mutual information of $x(t)$ and $x(t-\tau)$. Having low mutual information, means that the states at these two times are the least informative of one another. Consequently, this means that using there two states $x(t)$ and $x(t-\tau)$ at once to begin constructing $\boldsymbol{y}(t)$ provides the most amount of information about the system[4]. This relation between should hold for any pair of time-delayed states since the calculation for mutual information is ergodic. Lastly, we are using exactly the same data (with the exact same noise) as the full-measurement case to generate $\boldsymbol{y}(t)$.

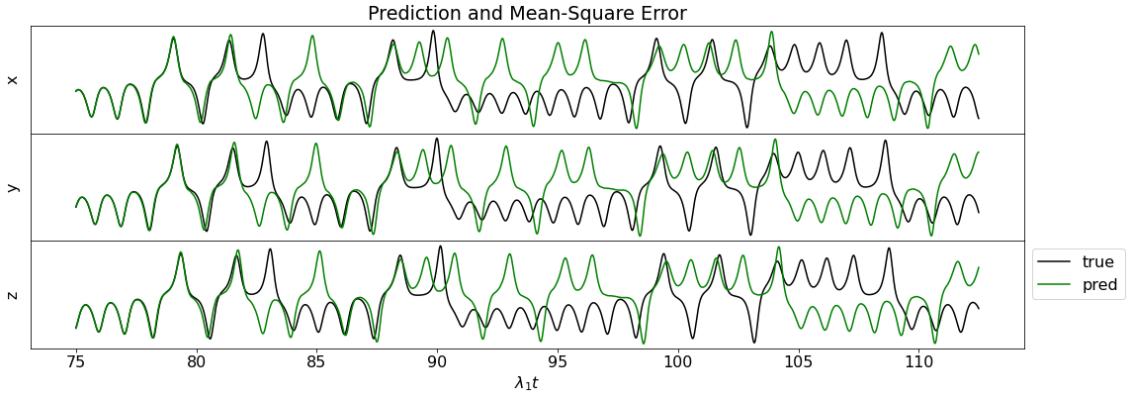


**Figure 8.4:** Comparison between the reservoir prediction and the actual/noiseless trajectory of the system, in the case of time-delayed data. The reservoir is able to predict roughly five Lyapunov times before slight deviation and an eventual lobe switch. It is very similar in performance to the full state case, falling only a little bit short arguably.

As for the reservoir, both the input weight matrix $W_{in}$ is identical as far as the individual elements. The connectivity matrix $A$ is almost the same, except that the coupling constant $\sigma$ was quadrupled. See Table 8.2 for detailed values. This was a purely empirical and practical adjustment made in order for the reservoir to give better prediction, but the change in coupling constant seems to be a fairly dramatic change . The similarities in $\boldsymbol{u}(t)$, $W_{in}$, and $A$ are only done for consistency and ease of comparison. It does not appear that any of these steps are or should be necessary practice for producing good results.

In this exercise, the reservoir acting on time-delayed data performed comparably well against the full state data. This was found to be the case, generally, when different RNG seeds were used, resulting in different data sets, input weights $W_{in}$, and connectivity matrices $A$. Again, keep in mind that these results will vary based on the RNG seed and the order of operations in the setup of weights $W_{in}$ and connectivity $A$.

In concluding this subsection on time-delayed data, we should look at whether the reservoir itself respects the presence of time-delay. Another way of putting this is that we hope that the reservoir has learned, somehow, that the incoming data $\boldsymbol{u}(t)$ is time-delayed. This might be an almost trivial question in the estimation phase, since we are fitting the reservoir states to the data explicitly, but it is an important constraint to obey when the reservoir is autonomous in the prediction phase. The

[4]This concept is a little counter-intuitive to newcomers of the field. It is somewhat akin to measuring the position of a point on the two-dimensional plane – measuring the $x$-coordinate is completely uninformative of the $y$-coordinate. Hence, the $x$ and $y$ coordinates on the two-dimensional plane gives us the most 'information' of the position of a point. Of course, this example is not statistical in any way cannot directly apply to information theory, but we feel that it is illustrative enough for pedagogical purposes.

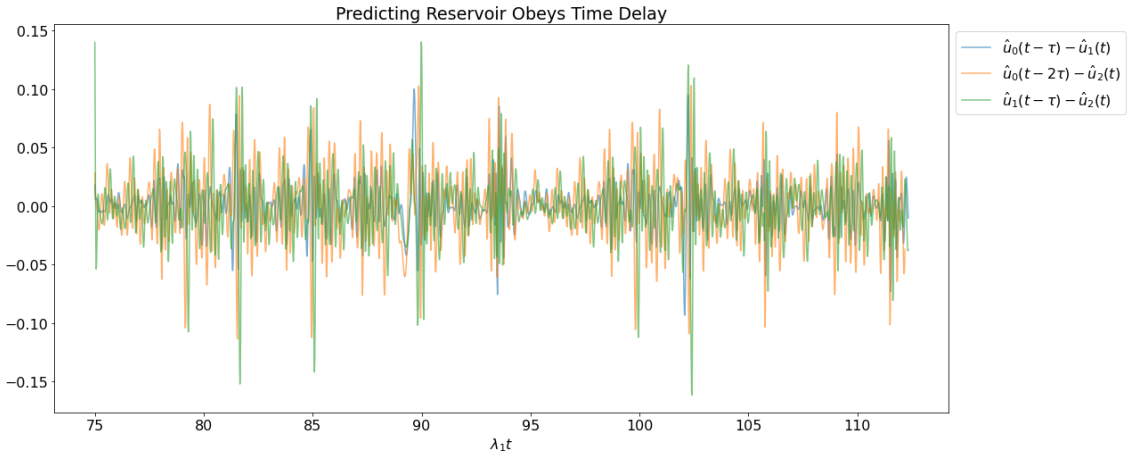


**Figure 8.5:** The autonomous/predicting reservoir produces trajectories that naturally obeys the time-constraint, albeit with some slight mismatch. A perfectly constrained reservoir should generate trajectories with zero mismatch, and this plot would be uniquely zero for all plotted variables. For example, in the incoming data set we have defined that $u_1(t) \equiv u_0(t-\tau)$ explicitly, but this relation is not necessarily true for the reservoir's reconstruction of these states $\hat{u}_0(t-\tau)$ and $\hat{u}_1(t)$.

remarkable result here is that the predicting reservoir does obey the time-delay constraint naturally and seems to obey it indefinitely. The predicting reservoir respects the time-delay constraint even better than the estimating reservoir. This constraint was never presented to the reservoir explicitly, but learned from the incoming data $\boldsymbol{u}(t)$ alone.

## 8.4 Different Versions of the Reservoir

When it comes to building a reservoir, there are so many ways of doing so, so many knobs to turn and buttons to push. There is no clear way of doing so that is transparent and widely accepted – only heuristics and hearsay – and this work does not provide additional clarity except that it is an additional data point. The reason for reservoir computing being in such as a state is, as far as we are aware, due to the flexibility of the reservoir computing framework. Flexibility here is a double-edged sword. It provides freedom for the user to design and mold their reservoir such that two completely different types of reservoirs can produce similarly good results. However, this freedom can be overwhelming sometimes because there is no clear starting point. There have been attempts using evolutionary optimization strategies to find the appropriate hyperparameters to some success. Still, such an approach is slightly antithetical to one of the reasons we chose to use reservoir computing, which is to bypass the need for an expensive training step of many neural network structures.

In this section, we will discuss the various options that the user would have when building their own reservoir. The purpose of this section is to summarize our past attempts of building reservoirs and their consequences. It is, by no means, an exhaustive list nor a steadfast set of rules – merely anecdotes and suggestions gathered through experience. It remains unfortunate that this process is still opaque and relies on heuristics. Hopefully, this will quickly change as the field matures, but how long this will continue, or the rate of improvement is anyone's guess.

### 8.4.1 Reservoir Dynamics

Perhaps one of the most pressing issues when designing reservoirs is which type of reservoir dynamics to choose. One has the choice between the continuous and discrete reservoirs, both of which have similarities and differences that are worth discussing. Some details about this were teased in an earlier section, but we will delve into more details here. One of the more noticeable differences is the trajectories generated from the two types of reservoirs. The discrete reservoir generates trajectories that inherit the noisiness from the input data. The continuous reservoir, on the other hand, is able to fend off the noise. We attribute this to the differential reservoir receiving zero-mean noise, which generates a slight diffusion of trajectories from the true one.

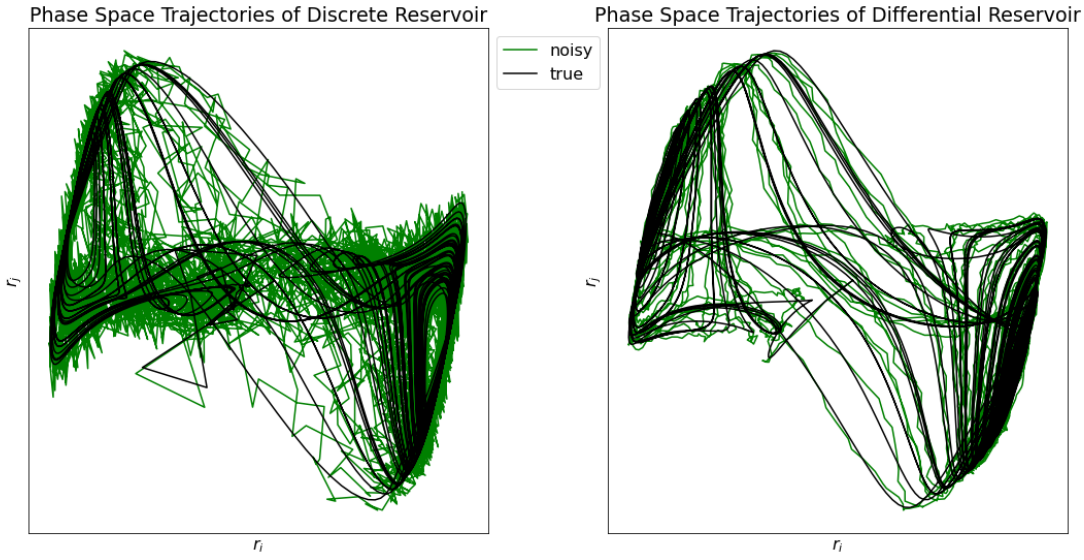


**Figure 8.6:** Using the exact same reservoir and data, changing nothing but the reservoir dynamics, we can see that the differential reservoir is better able to generate less noisy trajectories. We compare the phase space projection using two nodes within the reservoir, and it turns out that the reservoir's attractor is similar regardless of which type of reservoir dynamics is chosen. The black lines are reservoir states $\boldsymbol{r}(t)$ generated from the noiseless data $\boldsymbol{u}_{true}(t)$, and the green lines are generated from the noisy data $\boldsymbol{u}(t)$.

An interesting consequence in this investigation is that it seems that the order of data being presented is what is truly important to both these reservoirs, not so much the presence of noise [47]. There is some point, if the signal-to-noise ratio is very poor, where the structure of the attractor is completely clouded out and estimation cannot be properly done. But short of that, the differential reservoir seems to be fairly robust to the presence of noise relative to the discrete reservoir. The prediction capabilities of the continuous reservoir far out-performs that of the discrete reservoir in this example. The discrete reservoir has not been able, in our limited attempts, to perform as well as the continuous reservoirs when it comes to noisier data. The conclusion here being that the differential reservoir being better suited to deal with a noisier data set. However, then the data is much less noisy, the discrete reservoir is able to match the performance of the continuous reservoir, and even outperform it in certain situations.

These advantages are not free. It turns out that there is a small price to pay when using continuous reservoirs, coming in two forms. The first is that the continuous reservoir is order of magnitudes slower than their discrete counterpart, depending on the order of time-integrator chosen; more about that in the next section. The continuous reservoir also requires the addition of a time-constant hyperparameter $\gamma$, with the ability to add even more hyperparameters if desired. All this results in a more complicated structure and a larger search space for reservoir hyperparameters. Everything considered, it seems that the continuous reservoir is worth the extra trouble except if sheer speed of computation is required. The additional hyperparameters were easily managed and the prediction quality is rather insensitive to the hyperparameter $\gamma$.

### 8.4.2 Form of Training Layer

For the entirety of this work, we have assumed that a linear basis will fit, on average, the reservoir trajectory to the input data within the given time window, i.e. $\boldsymbol{u}(t_i) = W_{out}\boldsymbol{r}(t_i), \forall t_i \in [0, T]$. This is achieved performing the following minimization routine to find the appropriate $W_{out}$. Another successful option is to use an extended basis involving the squared values of the reservoir states, see Subsection 7.3.3 for specifics.

It seems, most generally, that one has to find a candidate function $\boldsymbol{\Psi}[\,\cdot\,]$ such that $\boldsymbol{u}(t_i) = \boldsymbol{\Psi}[W_{out}\boldsymbol{r}(t_i)]$. Such a search is typically done by imposing a parameterized form for $\boldsymbol{\Psi}[\,\cdot\,]$ then solving for said parameters. Our choice within this work happens to be $\boldsymbol{u}(t_i) = W_{out}\boldsymbol{r}(t_i)$, but there can be many more options. This projection function $\boldsymbol{\Psi}[\,\cdot\,]$ is the inverse of the generalized synchronization

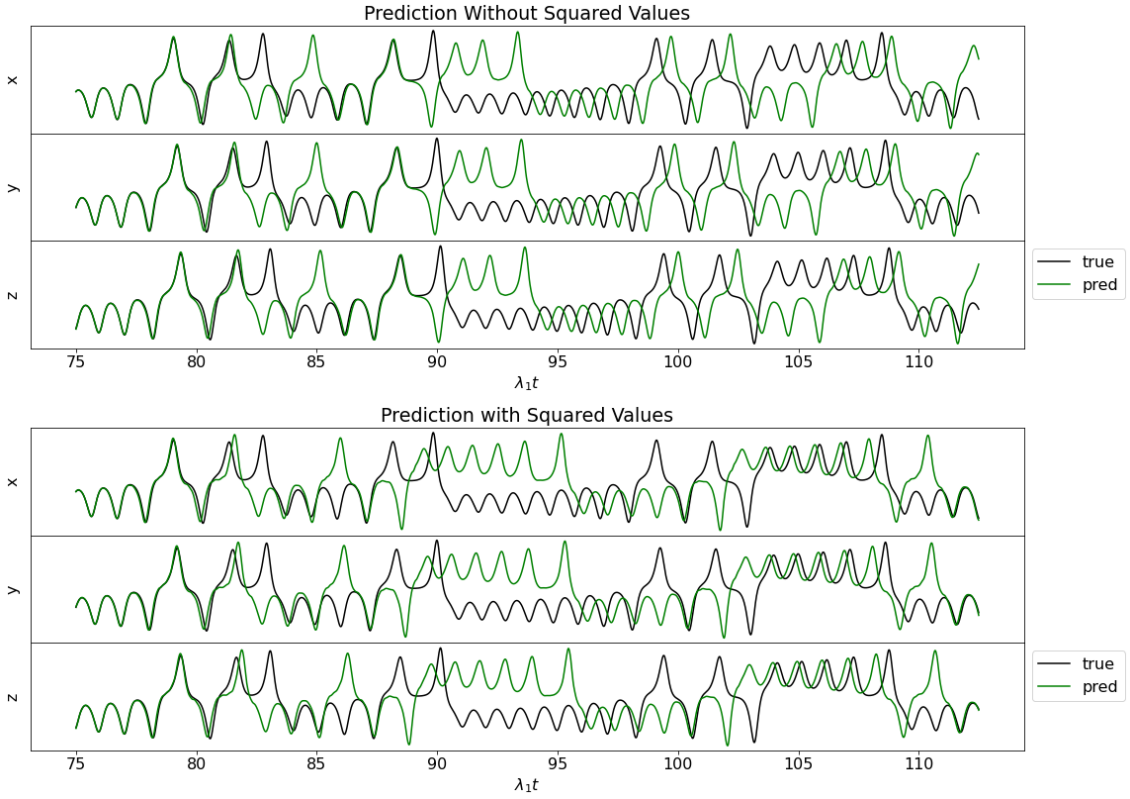


**Figure 8.7:** In comparing the prediction capabilities of the time-delay reservoir, changing nothing else, it appears that there is no advantage nor disadvantage. The same is observed for full state reservoir and slightly different reservoir structures. Using squared terms hardly affects the overall run time of the code since the linear fit is itself one of the quickest parts of the code.

function $\mathbf{\Phi}^{-1}[\cdot] = \mathbf{\Psi}[\cdot]$. In parallel to this work, some testing was done using the form $\boldsymbol{u}(t_i) = \widetilde{W}_{out}\boldsymbol{q}(t_i)$ where $\boldsymbol{q}(t) = [\boldsymbol{r}(t), \boldsymbol{r}^2(t)]$. Changing nothing but the presence of the squared terms, the resulting prediction capabilities for both methods were much the same, as shown in Figure 8.7. It is possible that more complicated training layers could contribute to better predictions, but this is not observed on average and requires further study. The training phase of the reservoir is not the dominant contributor to the code's overall run time, so there is plenty of leeway to pursue more computationally intensive training.

There is also the option of enforcing sparsity, with or without the presence of the square term. There have been many attempts to hard-code in various *a priori* structures. Additionally, using the Euclidean L2 norm (as does this work) is not the only viable approach. There may be potential in using the L1 norm, which is sparse in the nodes, and this might give insight as to which nodes are the sole/dominant contributors in the synchronization step.

### 8.4.3 Form of Embedding

The reservoir dimension $N$ is much larger than the input data dimension $D$. As such, we have to embed the initial data into this higher dimensional space. The method of choice is to use a stochastically generated input weights matrix $W_{in}$ to embed $\boldsymbol{u}(t)$ as $W_{in}\boldsymbol{u}(t)$. In a sense, this is the opposite process of the training step because, here, we are embedding a low dimensional object into a higher dimension space, whereas the training layer aims to project a high dimensional object into the low dimensional space.

$$\dot{\boldsymbol{r}}(t) \equiv \boldsymbol{g}(\boldsymbol{r}, \boldsymbol{u}) = \gamma(-\boldsymbol{r}(t) + \text{activ}[A\boldsymbol{r}(t) + \boldsymbol{\Theta}[\boldsymbol{u}(t)]) \qquad \boldsymbol{\Theta}[\boldsymbol{u}(t)] \equiv \sigma W_{in}\boldsymbol{u}(t) \tag{8.1}$$

We shall call this arbitrary or random embedding $\boldsymbol{\Theta}[\,\cdot\,]$ and is closely related to the generalized synchronization function $\boldsymbol{\Phi}[\,\cdot\,]$ discussed in Section 9. The exact nature of this relation was not explored in this work, but should be a very interesting and important topic for further research. Within this work, this function was chosen as $\boldsymbol{\Theta}[\boldsymbol{u}(t)] = \sigma W_{in}\boldsymbol{u}(t)$ with no variation whatsoever. This can be generalized in many of the same ways that the training layer's projection function $\boldsymbol{\Psi}[\,\cdot\,]$, but keeping in mind the relative dimensionality of these two functions $\boldsymbol{\Psi}[\,\cdot\,]$ and $\boldsymbol{\Theta}[\,\cdot\,]$.

Earlier, we discussed the arbitrariness of the reservoir dynamics. Since there are no true restrictions (that we know of) when it comes to constructing the function $\boldsymbol{g}(\boldsymbol{r}, \boldsymbol{u})$, this should also apply to the embedding function $\boldsymbol{\Theta}[\,\cdot\,]$. One could even replace the $A\boldsymbol{r}(t)$ term with a more general form. There can even be more 'mixing' between $\boldsymbol{r}(t)$ and $\boldsymbol{r}(t)$ if desired, but there seems to be no real justification to do so at this point.

There is also a very specific sparsity structure that is imposed on the input weights $W_{in}$. The structure is such that each of the $N$ nodes of the reservoir receives input from 1 of the $D$ input variables, and the strength of this connection is randomly chosen from a uniform distribution in the range of -1 to 1. This is how we chose to embed the input data given the previous successful attempts in the literature, but there is nothing explicitly special about our chosen sparsity structure – at least from empirical attempts.

### 8.4.4 Forward Integrator

If one chooses to use the continuous form of the reservoir, there is a question of which integrator to use. The usual suspects here are the explicit Runge-Kutta family (e.g. RK2 and RK4) of method, and perhaps even Heun's rule (sometimes called the explicit trapezoidal rule). This work uses the RK4 method exclusively, with RK2 being tried in passing to no extra effects aside from shorter run times.

## 8.5 Further Directions

Hence concludes our exposition of reservoir computing as a model-free means of making predictions. We will say a few words regarding possible research directions in general. It is extremely tempting to make a project out of turning all the knobs and pressing all the buttons in a reservoir, only to find out that they all work much the same. A sweep of all possible reservoir structures will unlikely offer additional insights, since most reservoir structure are able to make good predictions already. In this pursuit, one will likely find that reservoirs are notoriously receptive to the incoming data, regardless of their structure. A better understanding of how reservoir works is the recommended direction, not more ways to design a reservoir work. There are other approaches that also bridge RC with existing time series methods, where its ties to Volterra series and time delay methods are intriguing [48, 49].

So far, most of the work on reservoir computing has been done with computer simulation data, where information of state variables is perfect, and the noise is definitively Gaussian. There is a true need to find out how well these methods apply to real-world data.

## 8.6 Acknowledgments

Chapter 8, in part, uses material and results that appears in Robust Forecasting Using Predictive Generalized Synchronization in Reservoir Computing, submitted in Chaos: An Interdisciplinary Journal of Nonlinear Science , 31(12), 123118. Platt, Jason A., Wong, Adrian S., Clark, Randall, Penny, Stephen G., and Abarbanel, Henry D. I. (2021). The dissertation author was one of the co-authors of this paper.

# 9

# Synchronization in Reservoir Computing

Perhaps the most striking and interesting property of reservoir is its ability to reconstruct the input data $\boldsymbol{u}(t)$ and predict its future behavior, having no explicit information about the equations that generated $\boldsymbol{u}(t)$. Once reservoir computers were fed time-series data from chaotic physical systems, this relationship became much more apparent. It also raises the question of whether synchronization plays any role in reservoir computing. It seems that the listening phase of reservoir computing is mechanistically identical to the concept of Generalized Synchronization (GS), specifically the one-way or drive-response configuration of two systems. We briefly discuss GS before diving into its role in reservoir computing.

## 9.1 Generalized Synchronization

Earlier work on synchronization used identical systems, where the same variable from each copy the systems are coupled together with a scalar coupling constant. Generalized Synchronization was an attempt to generalize these early concepts of synchronization, specifically when the two systems are completely or physically different systems, and even when the multiple variables are being coupled [50]. In reservoir computing, the input data $\boldsymbol{u}(t)$ is used as a forcing, driving, or control term whereas the reservoir state $\boldsymbol{r}(t)$ is the response system. After some initial transient time $T$, the reservoir becomes synchronized to the input data if the reservoir state $\boldsymbol{r}(t)$ can be uniquely determined as some function of the input data, i.e $\boldsymbol{r}(t) = \boldsymbol{\Phi}[\boldsymbol{u}(t)]\ \forall t > T$. This relation will hold exactly and for all time

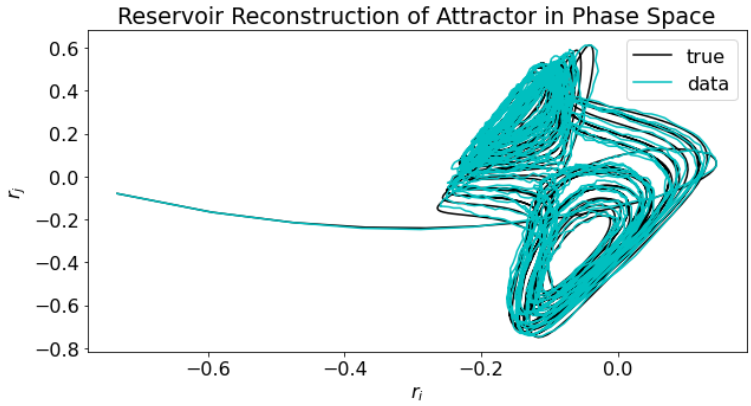


**Figure 9.1:** The reservoir is able to reconstruct the general topology of the Lorenz attractor. The blue line is the reservoir state generated using noisy data $\boldsymbol{u}(t)$, in comparison to the black line which was generated from noiseless true trajectories $\boldsymbol{u}_{true}(t)$ – an exercise only done here for illustration. The point here is that the reservoir is robust to noisy input data and, counter-intuitively, even performs better with slight noise rather than no noise. The outgrowth on the left is from transient trajectories – an inconsequential remnant of initializing the reservoir at $t = t_0$ that will be address separately.

$t > T$, as long as the input data $\boldsymbol{u}(t)$ continues to be presented to the reservoir as it did before. Once the stimulus is no longer presented, the two systems are no longer guaranteed to be synchronized.

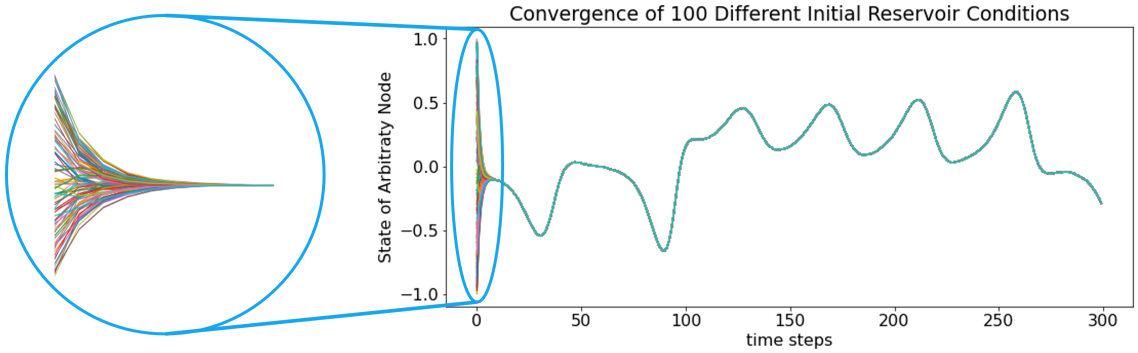


**Figure 9.2:** The auxiliary systems here are different reservoirs, each with a different initial state $\boldsymbol{r}(t_0)$ but otherwise identical. The quick convergence of trajectories is evidence of GS.

The synchronization relationship $\boldsymbol{r}(t) = \boldsymbol{\Phi}[\boldsymbol{u}(t)]$ does not need to be found explicitly, showing its existence is sufficient for GS to be have occurred. There are many ways to demonstrate the existence of such a function $\boldsymbol{\Phi}[\boldsymbol{u}(t)]$, such as original proposal of Mutual False Nearest Neighbors. That calculation, however, is computationally expensive as it has to search for nearest neighbors in both the driving signal space $\boldsymbol{u}(t)$ and the response signal's space $\boldsymbol{r}(t)$. Another test, called the auxiliary system method, was proposed and this works almost as well. The only catch here is that the auxiliary systems method is a necessary condition, but insufficient to guarantee GS. In this work, we

have made sure that the pathologies that cause the auxiliary system method to not hold are avoided, specifically by tuning $\gamma$ so that the frequency spectrum of $\boldsymbol{u}(t)$ and $\boldsymbol{r}(t)$ are similar. A quick but illustrative example is that when once-periodic driving signal causes a twice-periodic response, then the relationship $\boldsymbol{r}(t) = \boldsymbol{\Phi}[\boldsymbol{u}(t)]$ necessary for GS to occur would immediately break down.

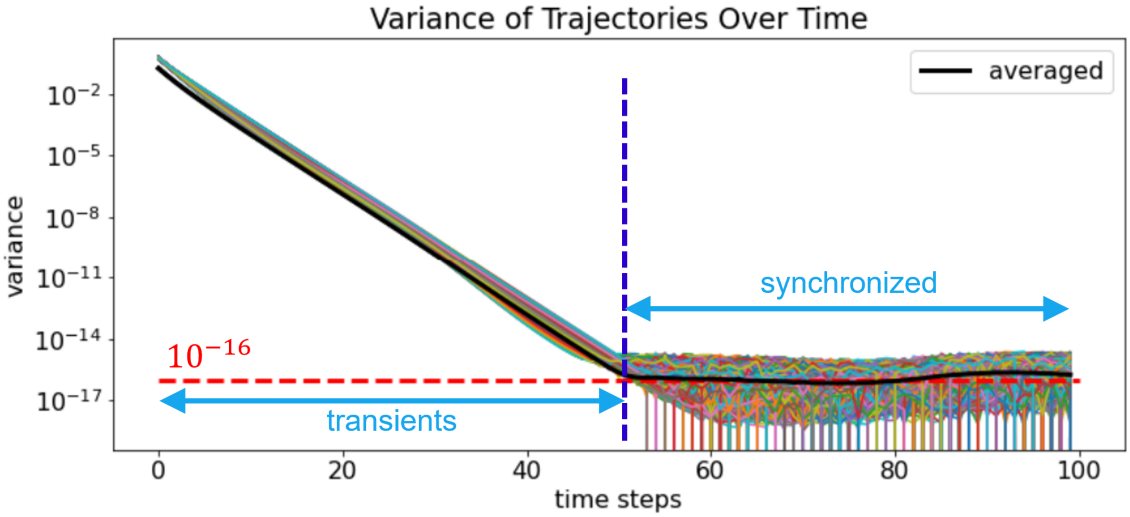


**Figure 9.3:** Each of the colored lines are one of the $N$ nodes in the reservoir. The black line is the unweighted average of there $N$ lines, where the average is taken over the $N$ nodes at every slice in time. We see that the trajectories are numerically indistinguishable after roughly 50 steps. The norm of the standard deviation can be used as well, reaching the same conclusion that the trajectories.

Here, we propose and use an expanded and more reliable version of the auxiliary system method where dozens of systems are used rather than just two. The original auxiliary systems method works but driving two copies of a response system $\boldsymbol{z}_1(t)$ and $\boldsymbol{z}_2(t)$ with the same input $\boldsymbol{u}(t)$. Both $\boldsymbol{z}_1(t)$ and $\boldsymbol{z}_2(t)$ have the same dynamics and parameters, which is to say that they both evolve obeying the equation $\dot{\boldsymbol{z}}_{1,2} = \boldsymbol{g}(\boldsymbol{z}_{1,2})$. The only difference between these two systems are their initial conditions $\boldsymbol{z}_1(t_0)$ and $\boldsymbol{z}_2(t_0)$. If these two almost-identical systems converge to the same trajectory after a finite number of steps, as measured by their Euclidean norm at every time point $t_i$, then it demonstrates that all conditional Lyapunov exponents of the system must be negative[1]. Though seldom framed in such a manner, a response system may only have a one-to-many relationship to the inputs, past a certain time horizon $T$, if all conditional Lyapunov exponents of the response system are negative. We stress again that this is a necessary but insufficient condition. Since we do observe this convergence, there may be some functional relationship between the two, i.e. $\boldsymbol{r}(t) = \boldsymbol{\Phi}[\boldsymbol{u}(t)]$.

[1]The prefix *conditional* here means that the systems $\boldsymbol{z}_1(t)$ and $\boldsymbol{z}_2(t)$ are conditioned on them both being driven by the same input $\boldsymbol{u}(t)$.

The expansion to the auxiliary system method (just called the auxiliary systems approach) is to use many dozens of systems rather than just two. Additionally, we also use the standard deviation of trajectories at every time point $t_i$ rather than the Euclidean norm between two arbitrary trajectories. Convergence of this generalized method is defined as the standard deviation reaching machine-precision zero. Everything else between the two methods remain the same. We come to the conclusion that GS does indeed occur within these reservoirs, which gives us the added comfort knowing that whatever results we get is independent of the initial state $\boldsymbol{r}(t_0)$.

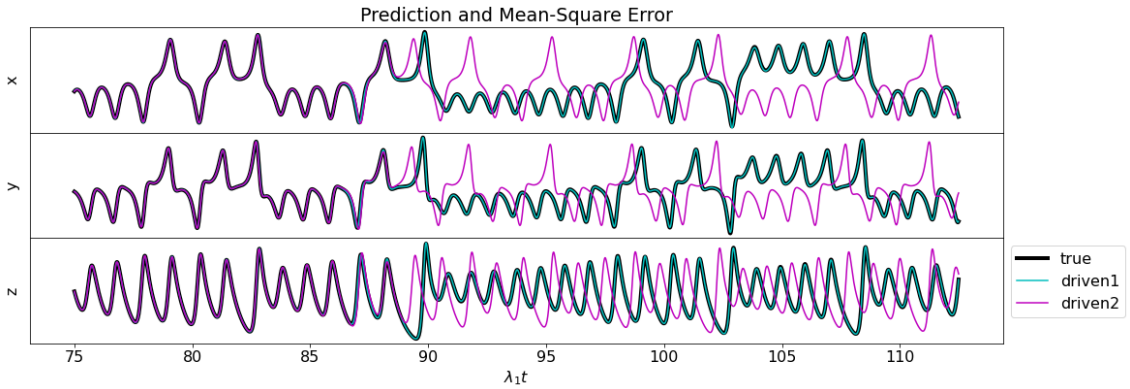


**Figure 9.4:** The same reservoir, after training for $W_{out}$, is able to make good prediction for data that it has never encountered before. 'True' refers to the noiseless data $\boldsymbol{u}_{true}(t)$ of the true trajectory. 'Driven1' refers to the reservoir driven by the 'true' data $\boldsymbol{u}_{true}(t)$. 'Driven2' refers to when the reservoir is driven with noisy data $\boldsymbol{u}(t)$, which is true data plus noise. Most remarkably, the reservoir driven with $\boldsymbol{u}_{true}(t)$ ('driven1') continues to stay synchronized indefinitely even though the reservoir was trained on noisy data. This is proof that synchronization has occurred. 'Driven2' performs better than the autonomous reservoir in the previous figure, roughly doubling the prediction capability, but strays from the true trajectory eventually. This implies that the synchronization is partially due to the training step, and that this synchronization seems to be permanent if a noiseless driving signal $\boldsymbol{u}_{true}(t)$ continues to be fed in. If the noisy driving signal $\boldsymbol{u}(t)$ is provided instead, then the synchronization is temporary.

Further investigation shows that reservoirs, in general, are naturally inclined to synchronize with the input data $\boldsymbol{u}(t)$. One has to try to proactively attempt to tune the reservoir such that it does not synchronize, for example by using extremely large or extremely small hyperparameters. It is also noted that GS is a necessary condition for producing good predictions, but is ultimately insufficient to guarantee even decent predictions. The conclusion here is that the presence of GS is not a good indicator of a reservoir's ability to make predictions. Nevertheless, the fact that GS is a necessary condition is an extremely interesting phenomenon and further study could reveal some better metrics for determining prediction quality.

## 9.2 Bridging to Reservoir Computing

The question of why and how RC works exactly remains poorly understood. Thankfully, at the very least, we do have a few necessary conditions for reservoir computers to work as intended.

- **Echo State Property** is a statement that trajectories of the reservoir converge upon being driven by a time-varying and bounded signal, which we have called $\boldsymbol{u}(t)$ thus far. There are other conditions that are worth mentioning briefly.
- **Fading Memory Property** states that the initial conditions of the reservoir are 'forgotten' as the reservoir evolves in time. This condition is directly implied by the ESP, since convergence of trajectories necessitates that initial conditions do not have any effect [49, 51].
- **Separation Property** states that a unique input $\boldsymbol{u}(t)$ will result in a corresponding unique reservoir trajectory $\boldsymbol{r}(t)$. Effectively, this is a statement regarding the desire for the RC to be a unique but non-degenerate filter of the data.

We want to focus for a moment on the ESP. The test for ESP is to run a large number of reservoirs, each with a different initial condition, and observe that the trajectories converge eventually. This test is, word for word, identical to the auxiliary systems method that we previously covered. To put this bluntly, the test for ESP is exactly the same as the auxiliary systems test of GS. Both ESP and the auxiliary systems method are necessary condition for RC and GS to work, which suggests a potential relationship between the two.

But there are differences between the two as well that need to be reconciled. The statement that GS grants is that if GS is present, then the relationship $\boldsymbol{r}(t) = \boldsymbol{\Phi}[\boldsymbol{u}(t)]$ exists after some transients, but nothing is guaranteed about the inverse relationship $\boldsymbol{u}(t) = \boldsymbol{\Phi}^{-1}[\boldsymbol{u}(t)]$. The inverse relationship is only established through mutual GS where the dynamics of $\boldsymbol{u}(t)$ is also driven by $\boldsymbol{r}(t)$. The reservoir only displays drive-response GS due to $\boldsymbol{r}(t)$ being driven by $\boldsymbol{u}(t)$ only. Therefore, even though we can say that the functional relationship $\boldsymbol{\Phi}[\boldsymbol{u}(t)]$ exists, it is never used explicitly by the reservoir. Instead, the training phase solves for $\boldsymbol{u}(t) \simeq W_{in}\boldsymbol{r}(t)$, which seems to be an attempt to establish the inverse relationship $\boldsymbol{u}(t) = \boldsymbol{\Phi}^{-1}[\boldsymbol{r}(t)] \equiv W_{out}\boldsymbol{r}(t)$.

Only mutually coupled systems, such that $\boldsymbol{r}(t)$ is used to generate $\boldsymbol{u}(t)$ **and** vice versa, guarantee the existence $\boldsymbol{\Phi}[\,\cdot\,]$ and $\boldsymbol{\Phi}^{-1}[\,\cdot\,]$. Yet, $\boldsymbol{\Phi}^{-1}[\,\cdot\,]$ exists and knowledge about its precise form is absolutely crucial to the predictive capabilities of the reservoir. When working with higher dimensions

objects, it is easy to find a unique projection to a lower dimensional object. There is reason to believe that the ratio of dimensions being high allows for $\mathbf{\Phi}^{-1}[\cdot]$ to occur. In this setup, the dimension of $\boldsymbol{r}(t)$ is order of magnitudes larger than the dimension of $\boldsymbol{u}(t)$. Usually, the dimension of $\boldsymbol{r}(t)$ is 200 to 1000 times the dimension of $\boldsymbol{u}(t)$ and this seems to work well and is standard practice. In the build-up of this work, values between 200 to 600 were typically used. Some work has been done to push this to a multiple of 10. It is unclear if it is necessary for $\boldsymbol{r}(t)$ to be in a much higher dimension, but there is also certainly something deeper that is generating this phenomenon. Nevertheless, the question of the roles of dimensionality, RC, GS, and their relationships to one another seem to be fundamental in understanding reservoir computing methods [52]. This warrants significant further investigation if one hopes to build efficient, reliable, and useful reservoirs [53].

## 9.3 Acknowledgments

Chapter 9, in part, uses material and results that appears in Robust Forecasting Using Predictive Generalized Synchronization in Reservoir Computing, submitted in Chaos: An Interdisciplinary Journal of Nonlinear Science , 31(12), 123118. Platt, Jason A., Wong, Adrian S., Clark, Randall, Penny, Stephen G., and Abarbanel, Henry D. I. (2021). The dissertation author was one of the co-authors of this paper.

# Part III

# Appendix

# A

# Path Integral Formulation

Path integrals trace their conception to as early as Norbert Wiener and Paul Dirac. However, it took decades before a young doctoral student, by the name of Richard Feynman, managed to formulate it in an elegant and digestible manner in his dissertation. His piercing insight was the time-slicing approach and 'sum of paths' interpretation, which served as an alternative formulation or interpretation to quantum mechanics. Regardless, the path integral formulation was crucial to the development of theoretical physics as a whole, not just to the quantum realm, and it established itself as a staple in the toolbox of physicists.

To fully derive and appreciate the path integral, at least from the physicists points of view, requires some basic understanding of quantum mechanics which will not be covered here for brevity (and sanity) sake. The notation used should be widely understood by most graduate students in physics, but we encourage unfamiliar readers to go to XYZ before tackling this topic. The path integral formulation can be stated in one elegant line using the Heisenberg picture.

$$\langle x_f, t_f | x_i, t_i \rangle = \int \mathcal{D}x(t) \exp\left(\frac{i}{\hbar}\mathcal{S}[x(t)]\right) \tag{A.1}$$

This should be interpreted as an integral over every possible path of $x(t)$ which the endpoints, $x(t_i) \equiv x_i$ and $x(t_f) \equiv x_f$, are fixed. It is worth noting here that $x(t)$ is a $N$-dimensional vector that varies in time and that there is a normalizing factor built-in to $\mathcal{D}x(t)$ that depends on the dimension $N$ and various fundamental constants. This integral is over a **huge** space. There are only a few trivial

cases where the integral can be evaluated exactly, but most cases rely on several approximations. For the purposes of this chapter, we will omit the vector notation and simply assert that every vector exists in an $N$-dimensional space. The scalar object $\mathcal{S}$ here is called the quantum mechanical action. It so happens that $\mathcal{S}$ is almost identical to the classical action except for the operator nature of position and momentum, which is a somewhat remarkable result. It is only somewhat remarkable since the quantum formulation must hold in the classical limit ($\hbar \to 0$). In that limit, it is the classical trajectory that ends up dominating the integral, since the variations around this classical trajectory end up canceling one another out on average. To see this, the stationary phase approximation is used, and one would expand around the stationary points of the action $\mathcal{S}$ given by the Euler-Lagrange equations. Hence, there is no substantial reason to differentiate between the quantum and classical actions and we will use the word *action* to refer to both.

## A.1 Propagator

Starting with the wave equation at a particular time $\psi(x_f, t_f)$, we can insert our choice of an identity matrix $\mathbb{1} = \int dx_m |x_m, t_m\rangle\langle x_m, t_m|$ and define, in the **Heisenberg picture**[1], how we arrive at this wave equation from an initial state $\psi(x_i, t_i)$.

$$\begin{aligned} \psi(x_f, t_f) &= \langle x_f, t_f | \psi \rangle \\ &= \int dx_i \langle x_f, t_f | x_i, t_i \rangle \langle x_i, t_i | \psi \rangle \\ &= \int dx_i K(x_f, t_f; x_i, t_i)\, \psi(x_i, t_i) \end{aligned} \tag{A.2}$$

Here $K(x_f, t_f; x_i, t_i) = \langle x_f, t_f | x_i, t_i \rangle$ is the full propagator between the initial and final state. This equation tells us that we need to integrate $\psi(x_i, t_i)$ over all initial positions $x_i$ along with the propagator $K(x_f, t_f; x_i, t_i)$ in order to arrive at the final state $\psi(x_f, t_f)$ and is, so far, unsurprising. Yet, the above equation almost begs to be further decomposed as there seems to be little reason to only look at the initial and final states; there are an infinite number of steps in between. Let us divide the problem up $M$ times into $M+1$ uniform and small time-intervals $\Delta t$ such that $x(t) = \{x_1, x_2, \cdots, x_M\}$, where $x_1 \equiv x_i$ and $x_M \equiv x_f$. This allows the decomposition of the above equation, essentially by

[1]The Heisenberg picture has it that the state kets $|\psi\rangle$ do not depend on time, but the basis kets $|x\rangle$ and operators carry the time dependence. A noteworthy result here is that the basis kets do not transform the same way that state kets do; in fact, they transform with the Hermitian conjugate instead. The work is neater when using the Heisenberg picture here, but we will switch later on. The Heisenberg picture is a more natural place to start for the path integral formulation because it does not require us to specify the time evolution of the system nor the usage of $\hbar$.

inserting $M$ identity matrices rather than just one.

$$\begin{aligned}&\int dx_1 \langle x_M, t_M | x_1, t_1 \rangle \langle x_1, t_1 | \psi \rangle = \\ &\quad \int dx_{M-1} \cdots \int dx_1 \langle x_M, t_M | x_{M-1}, t_{M-1} \rangle \cdots \langle x_1, t_1 | x_1, t_1 \rangle \langle x_1, t_1 | \psi \rangle\end{aligned} \tag{A.3}$$

Using the substitution $\langle x_1, t_1 | \psi \rangle = \psi(x_1, t_1)$ and absorbing relevant terms into a product, we can compactly rewrite the above equation, keeping in mind that the integral is evaluated at every time interval.

$$\psi(x_M, t_M) = \int \left[ \prod_{m=1}^{M} dx_{m-1} \underbrace{\langle x_m, t_m | x_{m-1}, t_{m-1} \rangle}_{\widetilde{K}(x_m, t_m; x_{m-1}, t_{m-1})} \right] \psi(x_1, t_1) \tag{A.4}$$

The object $\widetilde{K}(x_m, t_m; x_{m-1}, t_{m-1})$ can be seen as a local propagator that brings the system one step forward in time. Therefore, the product of all these local propagators should give us the full propagator from the initial to final state, once integrated properly. These local (or time-sliced) propagators will be useful later on in revealing the form of propagator and its relationships with the classical action.

$$\begin{aligned}K(x_M, t_M; x_1, t_1) &= \int dx_{M-1} \cdots \int dx_2 \int dx_1 \times \\ &\qquad \widetilde{K}(x_M, t_M; x_{M-1}, t_{M-1}) \cdots \widetilde{K}(x_2, t_2; x_1, t_1) \widetilde{K}(x_1, t_1; x_0, t_0) \\ &= \langle x_M, t_M | x_1, t_1 \rangle\end{aligned} \tag{A.5}$$

From this perspective, we can (hopefully) more clearly see that the propagator $K(x_M, t_M; x_1, t_1)$ contains all the information required to fully propagate the system in time, hence its name. The key takeaway here is that even the propagator itself is an integral made up of smaller, local propagators and a lot of integral evaluation is necessary in order to arrive at the full propagator. At this point, the Heisenberg picture exhausts its convenience and generality, so we will change to the **Schrodinger picture**.[2]

[2]The Schrodinger picture has it that all kets have explicit time dependence but the operators are constant. From here on, the Schrodinger picture makes more sense as we will need the explicit form of the Hamiltonian.

## A.2 Time-Slicing into a Path Integral

In the above section, we used the Heisenberg picture for notational and conceptual convenience. More importantly, the Heisenberg picture allows us to avoid specifying how time evolution happens, just accepting that it somehow does happen is sufficient. From here on however, we do need to look at the generator of time evolution, given by the Hamiltonian operator. To do so neatly and intuitively requires the **Schrodinger picture**, the equivalence to the Heisenberg picture is given, roughly speaking, by the equation below.

$$\overbrace{\langle x_{m+1}, t_{m+1}|x_m, t_m\rangle}^{\text{Heisenberg Picture}} \Longrightarrow \overbrace{\langle x_{m+1}|e^{-i\hat{H}\Delta t/\hbar}|x_m\rangle}^{\text{Schrodinger Picture}} \tag{A.6}$$

In the representation above, $\hat{H}$ is the Hamiltonian operator and $\Delta t$ is the uniform time-interval between two consecutive states. The Hamiltonian operator is the infinitesimal generator of time-evolution, as seen by the exponentiation of the operator, which means that it is solely responsible for moving the system state forward in time. This exponential solution is given by the Schrodinger equation, which can be solved appropriately to reproduce the exponential term above.

$$i\hbar\frac{\partial}{\partial t}\psi(x,t) = \hat{H}\psi(x,t) \tag{A.7}$$

The propagator can then be subdivided in the specified window with a method known as time-slicing.

$$\begin{aligned} K(x_M, t_M; x_1, t_1) &= \langle x_M|e^{-i\hat{H}(t_M-t_1)/\hbar}|x_1\rangle \\ &= \langle x_M|\underbrace{e^{-i\hat{H}\Delta t/\hbar}\cdots e^{-i\hat{H}\Delta t/\hbar}}_{M\text{-times}}|x_1\rangle \end{aligned} \tag{A.8}$$

Between each of these $e^{-iH\Delta t}$ we can insert $M-1$ copies of the identity function $\mathbb{1} = \int dx_m|x_m\rangle\langle x_m|$, evaluated at different times. Note that the integration is only over $x_1$ through $x_{M-1}$ since the endpoints $x_1$ and $x_M$ are held fixed.

$$\begin{aligned} \langle x_M|e^{-i\hat{H}(t_M-t_1)/\hbar}|x_1\rangle = \int dx_{M-1}\cdots\int dx_2\int dx_1\times \\ \langle x_M|e^{-i\hat{H}\Delta t/\hbar}|x_{M-1}\rangle\cdots\langle x_2|e^{-i\hat{H}\Delta t/\hbar}|x_1\rangle\langle x_1|e^{-i\hat{H}\Delta t/\hbar}|x_0\rangle \end{aligned} \tag{A.9}$$

Everything here, including the time-slicing is, has been exact so far. Once, however, the

Hamiltonian is split into its constituent parts of kinetic energy and potential energy. We shall assume a separable Hamiltonian and that the non-relativistic form of the kinetic energy is given by $\hat{T} = \frac{\hat{p}^2}{2m}$. Nothing about the form of the potential $\hat{V}$ is assumed. Going forward with the calculation, it is important to keep the $\Delta t$ factor as it informs us about the relative importance of each term. This Lie product or Trotter product limit is the foundation of the time-slicing method.[3]

$$\hat{H} = \hat{T} + \hat{V} \qquad \Longrightarrow \qquad e^{\hat{H}\Delta t} = \lim_{\Delta t \to 0} \left[ e^{\hat{T}\Delta t} e^{\hat{V}\Delta t} \right] = e^{\hat{T}\Delta t} e^{\hat{V}\Delta t} + \mathcal{O}(\Delta t^2) \tag{A.10}$$

Since $\hat{T}$ and $\hat{V}$ do not necessarily commute, one needs to work in the limit that $\Delta t$ is sufficiently small such that the approximation holds. Fortunately, these technical aspects are well-studied and are of no detriment to the derivation or validity of the path integral formulation. The local propagator from time $t_m$ to $t_{m+1}$ can be written an insightful way.

$$\begin{aligned} \langle x_{m+1}|e^{-i\hat{H}\Delta t/\hbar}|x_m\rangle &= \int dp \langle x_{m+1}|p\rangle\langle p|e^{-i\hat{H}\Delta t/\hbar}|x_m\rangle \\ &\simeq \int \frac{dp}{(2\pi\hbar)^N} \exp\left[\frac{ip}{\hbar}(x_{m+1}-x_m) - \frac{i\Delta t}{\hbar}\left(\frac{p^2}{2m} + \hat{V}\right)\right] \\ &= \left(\frac{m}{2\pi i\hbar\Delta t}\right)^{\frac{N}{2}} \exp\left[\frac{i\Delta t}{\hbar}\left(\frac{m}{2}\left[\frac{x_{m+1}-x_m}{\Delta t}\right]^2 - \hat{V}\right)\right] \end{aligned} \tag{A.11}$$

Since we are working in the limit that $\Delta t \to 0$, it should be abundantly clear that $\frac{x_{m+1}-x_m}{\Delta t}$ is in fact the velocity of the system, written as $\dot{v}$ in Lagrangian coordinates. Also, the exponent is clearly equivalent to the classical Lagrangian $\mathcal{L}(x_m, \dot{x}_m)$ except that the above Lagrangian is discrete in time. There is some ambiguity when specifying discrete Lagrangian, but in the infinitesimal limit of $\Delta t$ it will suffice to say that the Lagrangian is evaluated at time $t_m$.[4]

$$\begin{aligned} \langle x_M|e^{-i\hat{H}(t_M-t_1)}|x_1\rangle &= \int \left[\prod_{m=1}^{M-1} dx_m\right] \left[\prod_{m=1}^{M-1} \langle x_{m+1}|e^{-i\hat{H}\Delta t}|x_m\rangle\right] \\ &= \int \underbrace{\left[\left(\frac{m}{2\pi i\hbar\Delta t}\right)^{\frac{MN}{2}} \prod_{m=1}^{M-1} dx_m\right]}_{\mathcal{D}x(t)} \exp\left[\frac{i}{\hbar} \underbrace{\sum_{m=1}^{M-1} \mathcal{L}(x_m, \dot{x}_m)\Delta t}_{\mathcal{S}[x(t)]}\right] \end{aligned} \tag{A.12}$$

In homage to the usual physics fashion, any lingering error terms of the discretization can

[3] There are various ways to approach these operators that are collectively know of Hamiltonian splitting methods or symplectic integrators, one notable example is $e^{\hat{H}\Delta t} = e^{\hat{T}\Delta t/2} e^{\hat{V}\Delta t} e^{\hat{T}\Delta t/2} + \mathcal{O}(\Delta t^3)$ which is equivalent to the Stormer-Verlet method of integrating energy-preserving dynamical systems. We will not be dealing with the technical implementations for the purposes of this work.

[4] There are many ways to discretize a continuous time system, $\frac{x_{m+1}-x_m}{\Delta t}$ is one of them. But $\frac{x_m - x_{m-1}}{\Delta t}$ and

be swept into previous approximations. Something to note in the above equation is the mismatch between the number of terms of the two products. One can make sense of this by either considering that the (local or full) propagator should have units of inverse position, or noticing that there is one more integral over $dx_1$ to be done when evaluating the final state.

$$K(x_M, t_M; x_1, t_1) = \int \mathcal{D}x(t) \exp\left(\frac{i}{\hbar}\mathcal{S}[x(t)]\right) \tag{A.13}$$

Finally, we arrive at the form of the path integral formulation and some interpretation is in order. The above equation states concisely that the propagator is acquired by integrating over all possible paths that a particle can take between two fixed endpoints. When using the propagator to evolve the system forward in time, we are effectively saying that the particle travels every possible path, and the weight or phase factor of each path is given by the action $\mathcal{S}$. As we look toward the classical limit ($\hbar \to 0$), the phase of these weight factors $e^{i\mathcal{S}[x(t)]/\hbar}$ oscillate rapidly. The dominant contribution of the path integral then comes from looking at the stationary points of the action, which is inevitably given by the (classical) Euler-Lagrange equation. This is none other than the classical trajectory of the system following Newton's laws.

## A.3 Recovering the Schrodinger equation

The Schrodinger equation was used in the above formulation, but we should quickly check if we can recover the Schrodinger equation from the above path integral. Starting at the

$$\begin{aligned}\psi(x, t+\Delta t) &= \int dy\, K(x, t+\Delta t; y, t)\, \psi(y, t) \\ &\int \frac{dy}{A} \exp\left[\frac{i}{\hbar}\mathcal{L}(x, y)\Delta t\right] \psi(y, t) \\ &= \int \frac{dy}{A} \exp\left[\frac{i\Delta t}{\hbar}\left\{\frac{m}{2}\left(\frac{x-y}{\Delta t}\right)^2 - V\left(\frac{x+y}{2}\right)\right\}\right] \psi(y, t)\end{aligned} \tag{A.14}$$

We will also be expanding around $x$ and switching to the variable $\varepsilon$.

$$y = x + \varepsilon \qquad\qquad dy = d\varepsilon \tag{A.15}$$

$\frac{x_{m+1}-x_{m-1}}{2\Delta t}$ are just as viable and, in certain cases, even preferable for stability issues. Since we are working in the limit $\Delta t \to 0$, they are fortunately all equivalent.

$$\psi(x,t+\Delta t)=\int\frac{d\varepsilon}{A}\exp\left[\frac{i\Delta t}{\hbar}\left\{\frac{m}{2}\left(\frac{\varepsilon}{\Delta t}\right)^2-V\left(x+\frac{\varepsilon}{2}\right)\right\}\right]\psi(y,t) \tag{A.16}$$

We also Taylor expand $\psi(x,t+\Delta t)$ on the LHS in orders of $\Delta t$, discarding anything higher than first-order.

$$\psi(x,t+\Delta t)=\psi(x,t)+\Delta t\frac{\partial\psi}{\partial t}+\mathcal{O}(\Delta t^2) \tag{A.17}$$

On the RHS, **three** Taylor expansions are used. The first is on the exponential is also Taylor expanded in order of $\Delta t$. Second, the potential is also Taylor expanded $V(x+\frac{\varepsilon}{2})=V(x)+\mathcal{O}(\varepsilon)$. Combined with the $\Delta t$ term, these two expansions give an $\mathcal{O}(\varepsilon\Delta t)$ term that disappears under integration the lone $\varepsilon$ term.

$$\begin{aligned}&\int\frac{d\varepsilon}{A}\exp\left[\frac{i\Delta t}{\hbar}\left\{\frac{m}{2}\left(\frac{\varepsilon}{\Delta t}\right)^2-V\left(x+\frac{\varepsilon}{2}\right)\right\}\right]\psi(y,t)\\&\qquad=\int\frac{d\varepsilon}{A}\exp\left[\frac{im\varepsilon^2}{2\hbar\Delta t}\right]\left\{1-\frac{i\Delta t}{\hbar}V(x)\right\}\psi(y,t)+\mathcal{O}(\Delta t^2)+\mathcal{O}(\varepsilon\Delta t)\end{aligned} \tag{A.18}$$

The last Taylor expansion is on $\psi(y,t)$ in orders of $\varepsilon$. Here, we keep terms up to $\mathcal{O}(\varepsilon^2)$ since, as we will soon see, terms of order $\varepsilon$ become trivial under integration.

$$\psi(y,t)=\psi(x,t)+\varepsilon\frac{\partial\psi}{\partial x}+\frac{\varepsilon^2}{2}\frac{\partial^2\psi}{\partial x^2}+\mathcal{O}(\varepsilon^3) \tag{A.19}$$

From here on, the arithmetical accounting becomes rather tedious. The following terms will be evaluated in some form or another, and any terms not shown below should be assumed to be negligible in the appropriate limit.

$$\int\frac{d\varepsilon}{A}\exp\left[\frac{im\varepsilon^2}{2\hbar\Delta t}\right]=1 \tag{A.20}$$

$$\int\frac{d\varepsilon}{A}\varepsilon\exp\left[\frac{im\varepsilon^2}{2\hbar\Delta t}\right]=0 \tag{A.21}$$

$$\int\frac{d\varepsilon}{A}\frac{\varepsilon^2}{2}\exp\left[\frac{im\varepsilon^2}{2\hbar\Delta t}\right]=\frac{i\hbar\Delta t}{2m} \tag{A.22}$$

Collecting terms and keeping the notation $\psi\equiv\psi(x,t+\Delta t)$ and keeping the potential constant $V(x)\equiv V$, we can recover the Schrodinger equation.

$$\psi = \psi - \Delta t \frac{\partial \psi}{\partial t} + \frac{i\hbar \Delta t}{2m} \frac{\partial^2 \psi}{\partial x^2} - \frac{i\Delta t}{\hbar} V\psi \Longrightarrow \underbrace{i\hbar \frac{\partial}{\partial t}\psi = \left( -\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + V \right)\psi}_{\text{Schrodinger Equation}} \tag{A.23}$$

There is something to be said here about the mathematical rigor in this section, or rather the lack thereof. In fact, there is probably a lot to say about the lack of mathematical rigor in this entire dissertation. There are certainly more rigorous approaches to this section, but this trades some of the physical intuition for lengthy and technical discussion.

## A.4 Wick Rotating into Statistical Mechanics

One of the more important objects in quantum mechanics is $\exp\left[-i\hat{H}t/\hbar\right]$, affectionately called the time-evolution operator. The Hamiltonian operator $\hat{H}$ has eigenvalues that are the energy of the system, and it also happen to be the infinitesimal generator of the time-invariant Lie group. All states generated using $\hat{H}$ will have the same energies and thus live on the same manifold. The Schrodinger equation then describes specifically what the structure of $\hat{H}$ would look like.

$$i\hbar \frac{\partial \psi}{\partial t} = \hat{H}\psi = \left[ \frac{-\hbar^2}{2m}\boldsymbol{\nabla}^2 + V \right]\psi. \tag{A.24}$$

This operator bears resemblance to another object $\exp[-\beta E]$ from statistical mechanics, which describes the (unnormalized) probability of observing a state with energy $E$ given the inverse temperature $\beta = 1/k_B T$. If we just make the substitution of $t \to -i\hbar\beta$ into the time-evolution operator before, it becomes apparent that these two objects are very similar.

$$\exp\left[-i\hat{H}t/\hbar\right] \to \exp[-\beta E] \tag{A.25}$$

Informally speaking, we have just performed simple substitution. This is often called a *Wick rotation*, inspired by how a phase of imaginary unit $i$ corresponds to a 90-degree rotation in a complex space. Strictly speaking, it is not a rotation in a 4-dimensional space or 4-dimensional space-time. Formally, it is called an analytic continuation and it is curious to see that quantum mechanics and statistical mechanics can be related to one another with this analytic continuation. Even in the path integral formulation of quantum mechanics, which is just an alternative to the Hamiltonian approach to quantum mechanics, we see that the quantum mechanical action $\mathcal{S}[x(t)]$ gets replaced by

a Hamiltonian-like function which we call $\bar{H}[x(\tau)]$ here. Conceptually, this new function $\bar{H}[x(\tau)]$ is very similar to the total energy of the path in 'imaginary' time $\tau$.

$$\begin{aligned}
\frac{i}{\hbar}\mathcal{S}[x(t)] &= \frac{i}{\hbar}\int \mathcal{L}\left(x, \frac{\mathrm{d}x}{\mathrm{d}t}\right) dt \\
&= \frac{i}{\hbar}\int \left[\frac{m}{2}\frac{\mathrm{d}x}{\mathrm{d}t}^2 - \hat{V}(x)\right] dt \\
&\simeq \frac{i\Delta t}{\hbar}\sum\left(\frac{m}{2}\left[\frac{x_{m+1}-x_m}{\Delta t}\right]^2 - \hat{V}(x_m, x_{m+1})\right) \\
&\Rightarrow \frac{\Delta\tau}{\hbar c}\sum\left(-\frac{mc^2}{2}\left[\frac{x_{m+1}-x_m}{\Delta\tau}\right]^2 - \hat{V}(x_m, x_{m+1})\right) \\
&= -\frac{1}{\hbar c}\int \left[\frac{mc^2}{2}\frac{\mathrm{d}x}{\mathrm{d}\tau}^2 + \hat{V}(x)\right] d\tau \\
&\equiv -\frac{1}{\hbar c}\bar{H}[x(\tau)]
\end{aligned} \tag{A.26}$$

Yet, the effects of this analytic continuation do not end here. There are additional similarities of such a move. We start by looking at the basic Minkowski metric of special relativity for a flat spacetime. Quantum mechanics and special relativity are still compatible when we use such a metric with the $(-1, 1, 1, 1)$ signature.

$$ds^2 = d(ict)^2 + dx^2 + dy^2 + dz^2 \qquad \Longrightarrow \qquad ds^2 = d\tau^2 + dx^2 + dy^2 + dz^2 \tag{A.27}$$

The substitution $t \to -i\tau/c$ moves us from the $(-1, 1, 1, 1)$ Minkowski signature to the $(1, 1, 1, 1)$ Euclidean signature, where we treat time as if it is just another spatial variable. The previously 'real' time $t$ is replaced by a 'imaginary' time $\tau$ scaled accordingly, and we are free to go back and forth between the two metrics as well.

$$i\hbar\frac{\partial\psi}{\partial t} = \frac{-\hbar^2}{2m}\boldsymbol{\nabla}^2\psi \qquad \Longrightarrow \qquad \frac{\partial\psi}{\partial\tau} = \frac{\hbar c}{2m}\boldsymbol{\nabla}^2\psi \tag{A.28}$$

The basic Schrodinger equation for a free particle also transforms in an interesting way with this simple substitution $t \to -i\tau/c$. In fact, it looks identical in form to the diffusion equation as shown above. Evolution according to the Schrodinger equation begins to look very similar to that of diffusion, which is interesting to consider. Steven Hawking famously called this 'imaginary time', but there is nothing really fictitious about it. Just like how the imaginary unit $i$ is not a fictitious number. It is by a historical malfeasance that Descartes, disliking the concept of $i$, called them imaginary

numbers – yet the name stuck. Perhaps a better way of thinking about the concept of imaginary time is just to treat it as another spatial dimension.

In the Hamiltonian setting, this mechanism allows us to trade a $D$-dimensional dynamical problem for a $(D+1)$-dimensional statistical one, and vice-versa. In the path integral setting, this means that we treat the $D$-dimensional trajectory as a $(D \times M)$-dimensional spatial vector, which is how we approach our data assimilation problem in Chapter 2. Whether this relationship or whether it carries some deeper relationship depends on one's personal inclination towards metaphysics. At the very least, it is a tool that we can leverage in the proper setting.

# B

# Lagrange and Hamilton

## B.1 Stationary Action

For any dynamical problem, one can define a quantity $\mathcal{L}$ called the Lagrangian that is a function of positions $x(t)$, velocities $\dot{x}(t)$, and time $t$. Both the position and velocity should be understood to be $D$-dimensional vectors that vary in time $t$, though the notation below may not always suggest it for simplicity sake. The time integral of the Lagrangian $\mathcal{L}$ along the path $x_a$ to $x_b$ is called the action $\mathcal{S}$.

$$\begin{aligned} \mathcal{S}[x(t)] &= \int_{t_a}^{t_b} \mathcal{L}(x, \dot{x}, t)\, dt \\ x_a \equiv x(t_a) &\qquad x_b \equiv x(t_b) \end{aligned} \tag{B.1}$$

Now that we have the action $\mathcal{S}$, we can recover the equations of motion for the system by invoking the **Principle of Least Action** and looking for stationary points. Many will suggest here that the principle is a misnomer, and the proper name should be the **Principle of Stationary Action** since the original name suggests that we are simply looking for minima. Extremizing the action $\mathcal{S}$ involves using the calculus of variations. This means that the first variation of $\mathcal{S}$ must be

zero, i.e. $\delta\mathcal{S} = 0$, which is to say that the action is non-changing up to first-order variations $\delta x$.

$$\begin{aligned}\delta\mathcal{S} &= \int_{t_a}^{t_b} \left( \frac{\partial \mathcal{L}}{\partial x}\delta x + \frac{\partial \mathcal{L}}{\partial \dot{x}}\delta\dot{x} \right) dt \\ &= \int_{t_a}^{t_b} \left( \frac{\partial \mathcal{L}}{\partial x} - \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{x}} \right) \delta x \; dt + \left[ \frac{\partial \mathcal{L}}{\partial \dot{x}}\delta x \right]_{x_a}^{x_b}\end{aligned} \tag{B.2}$$

The only way to get a stationary action $\delta\mathcal{S} = 0$ is for the first integrand to be uniquely zero. Integration by parts is used within the integrand to decompose the $\delta\dot{x}$ term into two terms. Usually, the endpoints $x_a$ and $x_b$ are kept fixed, which means that the square bracket above is equal to zero once evaluated. Necessarily, the second integrand above will also have to be zero.

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{x}} - \frac{\partial \mathcal{L}}{\partial x} = 0 \tag{B.3}$$

## B.2 Lagrangian Mechanics

In order for the Stationary Action Principle to hold, a very specific set of conditions have to hold. These conditions are given the name of **Euler-Lagrange equations**, which are a set of $N$ second-order Ordinary Differential Equations (ODE) defined in terms of the canonical position coordinates $x$. When used in this way, it gives rise to what is known as Lagrangian mechanics.

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{x}} = \frac{\partial \mathcal{L}}{\partial x} \tag{B.4}$$

The Euler-Lagrange equations are both necessary **and** sufficient conditions for stationary points of the action. Whether or not each stationary point is a minimum, maximum, or saddle point warrants further investigation, usually on the determinant of the Hessian $\frac{\partial^2 \mathcal{L}}{\partial x_i \partial x_j}$ of the Lagrangian. Effectively, the functional $\mathcal{S}$ is stationary **if and only if** the corresponding Euler-Lagrange equations (defined only on the function $\mathcal{L}$) are satisfied. The Euler-Lagrange equations gives the equations of motion for a classical system, but is also used in quantum systems since the classical limit must also hold true.

$$\begin{aligned}\mathcal{L} &= T(\dot{x}) - V(x) \\ &= \frac{1}{2}m\dot{x}^2 - V(x)\end{aligned} \tag{B.5}$$

## B.3 Hamiltonian Mechanics

The Hamiltonian formulation for classical mechanics is also another way to approach dynamical problems. These two formulations are equivalent descriptions of the same problem, just that the mathematical structures that underpin these two formulations differ. We can perform a Legendre transform to form a Hamiltonian which is defined in $(x, p)$ rather than $(x, \dot{x})$. Defining the canonical momentum $p$ as:

$$p \equiv \frac{\partial \mathcal{L}}{\partial \dot{x}} \tag{B.6}$$

Exercising some shorthand, we can define the Legendre transform that brings us from the Lagrangian $\mathcal{L}$ to the Hamiltonian $\mathcal{H}$ with the following equation.

$$\mathcal{H} \equiv \dot{x}p - \mathcal{L} \tag{B.7}$$

Using the Legendre transform, we look at how this new Hamiltonian function $\mathcal{H}$ varies with respect to its constituents in the equation above.

$$\begin{aligned} d\mathcal{H} &= \dot{x}\, dp + p\, d\dot{x} - d\mathcal{L} \\ &= \dot{x}\, dp + p\, d\dot{x} - \frac{\partial \mathcal{L}}{\partial \dot{x}} d\dot{x} - \frac{\partial \mathcal{L}}{\partial x} dx - \frac{\partial \mathcal{L}}{\partial t} dt \\ &= \dot{x}\, dp - \frac{\partial \mathcal{L}}{\partial x} dx - \frac{\partial \mathcal{L}}{\partial t} dt \end{aligned} \tag{B.8}$$

We know that Euler-Lagrange equations must hold for and contains the term $\frac{\partial \mathcal{L}}{\partial x} dx$ directly. Upon some substitution and the definition of $p \equiv \frac{\partial \mathcal{L}}{\partial \dot{x}}$ earlier, we get the following equation.

$$d\mathcal{H} = \dot{x}\, dp - \dot{p}\, dx - \frac{\partial \mathcal{L}}{\partial t} dt \tag{B.9}$$

In parallel, we can also directly evaluate the variation of the Hamiltonian $\mathcal{H}$ as a function of $(x, p)$.

$$d\mathcal{H} = \frac{\partial \mathcal{H}}{\partial p} dp + \frac{\partial \mathcal{H}}{\partial x} dx + \frac{\partial \mathcal{H}}{\partial t} dt \tag{B.10}$$

To reconcile these two equations, we have the following relations for how the variables $(x, p)$ evolve in time. As usual, we assume that the Lagrangian $\mathcal{L}$ has no explicit time dependence, which translates to the Hamiltonian also not possessing any explicit time dependence. These are called

**Hamilton's equations**, which are an alternative but equivalent approach to Lagrangian and Newtonian mechanics.

$$\dot{x} \equiv \quad \frac{\partial \mathcal{H}}{\partial p} \qquad \dot{p} \quad \equiv -\frac{\partial \mathcal{H}}{\partial x} \qquad \frac{\partial \mathcal{H}}{\partial t} = -\frac{\partial \mathcal{L}}{\partial t} = 0 \tag{B.11}$$

## B.4 Poisson Brackets and Hamiltonian Flow

The mathematical structure of Hamiltonian mechanics does not end with the evolution of the canonical coordinates $x$ and$p$, it holds much more generally. We can look at any function $f(x, p, t)$ that is lives on the (symplectic) manifold defined by $\mathcal{H}$ and examine how it evolves in time. We use the above substitution of Hamilton's equations to generate a Poisson bracket relationship defined by $\mathcal{H}$.

$$\begin{aligned} \frac{df}{dt} &= \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial p}\frac{dp}{dt} + \frac{\partial f}{\partial t} \\ &= \frac{\partial f}{\partial x}\frac{d\mathcal{H}}{dp} - \frac{\partial f}{\partial p}\frac{d\mathcal{H}}{dq} + \frac{\partial f}{\partial t} \\ &= \{f, \mathcal{H}\} + \frac{\partial f}{\partial t} \end{aligned} \tag{B.12}$$

From this, we can see that the Hamiltonian is the continuous generator of time-evolution. Focusing for a moment on functions that do not have an explicit time-dependence, we can recover the time dynamics of any function that lives on the manifold by simply evaluating the Poisson bracket.

$$\dot{f} = \{f, \mathcal{H}\} = X_{\mathcal{H}} f \tag{B.13}$$

Moreover, the Poisson bracket is effectively as a linear operator such that $\{\,\cdot\,, \mathcal{H}\} = X_{\mathcal{H}}$. As a word of clarification, $X_{\mathcal{H}}$ is linear in the sense that it is an operator. There are undoubted going to be $x$ and $p$ dependence in the specific structure of $X_{\mathcal{H}}$ for the general case scenario. As usual, time-invariant, conserved quantities, or first-integral can be identified with this, and their Poisson bracket under the Hamiltonian returns zero.

$$\{f, \mathcal{H}\} = 0 \quad \Rightarrow \quad f(x, p, t) = const. \tag{B.14}$$

The appearance of a Poisson bracket is emblematic of the deep mathematical structure that arises from Hamiltonian mechanics. It is a worthwhile pursuit to go deeper into the field of symplectic

geometry, but is alas not within the scope of this work. As we mentioned earlier, the Lagrangian formalism and Hamiltonian formalism are in fact equivalent. There are certain classes of problems where the Lagrangian formalism is better suited for handling, and others where the Hamiltonian formalism is better suited.

## B.5 Symplectic Structure

The position variable $x$ is a $D$-dimensional vector, so its corresponding momentum $p$ is also a $D$-dimensional vector. Together, this forms the phase space of the system. If we take the variables $(x, p)$ and create an extended vector space $z = (x, p)$, then Hamilton's equations can be condensed into a more compact form.

$$\dot{z} = J\boldsymbol{\nabla}_z \mathcal{H}(z) \tag{B.15}$$

The operator $\boldsymbol{\nabla}_z \equiv (\frac{\partial}{\partial x}, \frac{\partial}{\partial p})$ is the gradient in the extended state space of $z = (x, p)$. The matrix $J$ can be inferred by noticing that, in the original Hamilton's equations, $x$ and $p$ appear only once in each equation, always on the opposing sides. This is to say, in crude terms, that the block off-diagonal elements of $J$ should be identity matrices (of size $D \times D$) whereas the block diagonal elements should be the zero matrix (of consistent size).

$$J = \begin{bmatrix} 0 & \mathbb{1} \\ -\mathbb{1} & 0 \end{bmatrix} \tag{B.16}$$

Oddly, this matrix $J$ has no formal name and is often referred to in textbooks and literature simply as "**the** skew/anti-symmetric matrix". It has the expected property of $J^T = -J$, but more remarkably that $J^2 = -\mathbb{1}$ and $\det(J) = 1$.

# C

# An Attempt at Fixing Backpropagation

## C.1 The Derivations of Backpropagation

Historically, there are two main approaches towards deriving the backpropagation algorithm. The first, simplest, and most famous is Rumelhart, Hinton, and Williams (1985) [54]. The derivation is straightforward but elegantly simple, enough that a first-year university student can understand it and implement in code. It seems that, for these reasons, Rumelhart's derivation is the dominant and more pervasive one.

On the other hand, there is also LeCun's derivation (1988) [55]. Even though LeCun is a famous scientist in his own right, his derivation of backpropagation is very seldom mentioned. This derivation stems from dynamic programming and control theory, specifically from the calculus of variations. Functionally, it achieves the same result as Rumelhart's but is slightly more complicated [56].

Hidden in LeCun's derivation is the fact that he imposes a strict equality constraint on the network dynamics. This corresponds to a hard-constraint, but the method can be further generalized with a soft-constraint. Specifically, we can start from a very relaxed (soft) constraint and gradually enforce the network's structure iteratively, arriving at the original hard-constraint. This is known as *annealing* in the physics and optimization community, or as a *penalty method* in other communities. We discuss this is some detail in Chapter 5, though that treatment focused more on our use case for the data assimilation action, which is effectively our cost function. There are tremendous similarities

between the data assimilation framework and the deep learning framework [57]. Once you treat the layers of a deep network as if it was a unfolding in time, then the connection becomes more apparent. The biggest differences are that the deep network has a different structure at each layer, and that the equations of the model have no physical basis.

## C.2 Deep Neural Network Loss Function

When using a Deep Neural Network (DNN), the task of classification of, say, images usually uses a loss function that takes on the following form.

$$\mathcal{L}(\mathcal{W};\mathcal{Y}) = \frac{1}{2N}\sum_{n\in\mathcal{Y}}\left[x_L^{(n)} - y_L^{(n)}\right]^2 \tag{C.1}$$

$\mathcal{Y}$ is the training set consisting of $N$ labeled images, indexed by $n$. A particular image (think of an array of pixels) has index $n$ and is represented above as $y_0^{(n)}$. Its respective label is represented as $y_L^{(n)}$. The zero index of $y_0^{(n)}$ is indicative of the image being an input to the network. The last index $L$ of $y_L^{(n)}$ tells us that $y_L^{(n)}$ is related to the output of the final layer of the network. So, $\left\{y_0^{(n)}, y_L^{(n)}\right\}$ forms an input-output pair, where we have access to $N$ such pairs in our training set.

$\mathcal{W}$ is a set of weights connecting the nodes of our multi-layered network of depth $L$. The use of an image classification network is chosen for conceptual understanding and reader familiarity, but this loss function is generalized enough for any classification task. This loss function $\mathcal{L}(\mathcal{W};\mathcal{Y})$ will be large when there is a huge mismatch between the network's ability to classify an image, hence it is a justified cost function to minimize.

$$\min_{\mathcal{W}} \mathcal{L}(\mathcal{W};\mathcal{Y}) \tag{C.2}$$

Of critical importance is $x_L^{(n)}$, which should be thought of as *the output of a network given* $y_0^{(n)}$. Keep in mind that the network output $x_L^{(n)}$ is dependent on the weights $\mathcal{W}$. The goal of the network is to generate an estimated label $x_L^{(n)}$, which should be as close as possible to the original label $y_L^{(n)}$, given only $y_0^{(n)}$. That is to say, we want a network that can output the correct label of a given image *most of the time*. The internal or hidden states of the network $\left\{x_1^{(n)}, x_2^{(n)}, \cdots, x_{L-1}^{(n)}\right\}$ are largely unimportant, except for the fact that these states are stationary with respect to the input-output pairs, which we will explore in depth later. To train the network, our task is to tune the weights $\mathcal{W}$ such that the above loss function is minimized.

## C.3 Rumelhart's Backpropagation

We have the following equations that describes the fully deterministic relationship between the layers, written as a discrete forward mapping from layer $\ell - 1$ to layer $\ell$. As per tradition, $f_\ell(\cdot)$ here represents any reasonable nonlinear activation function, most often chosen to be one of the many sigmoid functions. The weight at a particular layer is given by $W_\ell$ and the subscript $\ell$ on the function is to account for the situation where the activation function might be slightly different between layers, considering operations like convolution and pooling operations etc.

$$x_\ell = f_\ell(W_\ell x_{\ell-1}) \qquad ; \qquad \ell \in \{1, 2, \cdots, L\} \tag{C.3}$$

This mapping gives the following equations when taking derivatives with respect to the arguments $W_\ell$ and $x_{\ell-1}$. For the matrix calculus, we are using row-vector notation for these equations. Hence, these equations below are all matrix-vector products. The relationship between layers give us the following equations, which we will be using very soon.

$$\frac{\partial x_\ell}{\partial W_\ell} = f'_\ell(W_\ell x_{\ell-1})\, x_{\ell-1} \tag{C.4}$$

$$\frac{\partial x_\ell}{\partial x_{\ell-1}} = f'_\ell(W_\ell x_{\ell-1})\, W_\ell \tag{C.5}$$

Here we have an energy or cost function $\mathcal{L}$ associated with the network, previously discussed. This energy function is associated with the cost of a prediction, having high cost for *very wrong* predictions and low cost for *less wrong* predictions.

$$\mathcal{L}(\mathcal{W}; \mathcal{Y}) = \frac{1}{2N} \sum_{n \in \mathcal{Y}} \left[ x_L^{(n)} - y_L^{(n)} \right]^2 \tag{C.6}$$

For the time being, we will ignore the fact that the labels come from a batch of input-output pairs. Hence, the $N$ and $(n)$ terms in the equation above will be dropped. The derivative of the loss function with respect to the internal states $x_\ell$ are not directly relevant because the states themselves take on any value that it needs to satisfy the structural constraints of the network. The derivatives of the weights $\mathcal{W}$ on the other hand is of chief importance, since it tells us how to update $\mathcal{W}$, which dictates the relationship between the layers. Here we look at the derivative of the cost function with

respect to the weights of the final layer as a first pass calculation.

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial W_L} &= \frac{\partial \mathcal{L}}{\partial x_L}\frac{\partial x_L}{\partial W_L} \\ &= (x_L - y_L)\, f'_\ell(W_\ell x_{\ell-1})\, x_{\ell-1}\end{aligned} \tag{C.7}$$

As we look at how the weights of the shallower ($\ell \to 1$) layers change, a simple functional form of the equation appears in terms of a sequence of matrix products. This simple form of the derivatives is one of the many reasons for the popularity and widespread use of backpropagation.

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial W_\ell} &= \frac{\partial \mathcal{L}}{\partial x_L}\frac{\partial x_L}{\partial W_\ell} \\ &= \frac{\partial \mathcal{L}}{\partial x_L}\frac{\partial x_L}{\partial x_{L-1}}\frac{\partial x_{L-1}}{\partial x_{L-2}}\cdots\frac{\partial x_{\ell+1}}{\partial x_\ell}\frac{\partial x_\ell}{\partial W_\ell} \\ &= \frac{\partial \mathcal{L}}{\partial x_L}\left[\prod_{k=\ell}^{L-1}\frac{\partial x_{k+1}}{\partial x_k}\right]\left[\frac{\partial x_\ell}{\partial W_\ell}\right]\end{aligned} \tag{C.8}$$

## C.4 Vanishing Gradient and Lyapunov Exponents

The goal of the minimization is to find a set of weights $\mathcal{W}$ such that $\frac{\partial \mathcal{L}}{\partial W_\ell}$ is individually near zero for all $\ell$. The above function tells us how to move in this high dimensional space such that we can minimize $\mathcal{L}$, at which point $\frac{\partial \mathcal{L}}{\partial W_\ell}$ will be close to zero. What this equation recognizes is that the weights at different layers are completely independent of one another, seeing that there are no terms like $\frac{\partial W_{\ell+1}}{\partial W_\ell}$ appearing in the equations. We run into a problem then at, say a certain layer $\ell'$, when the gradient $\frac{\partial \mathcal{L}}{\partial W_\ell}$ approaches an incredibly small value at one such layer $\ell > \ell'$ that occurs after $\ell'$. This effectively bottlenecks the search/training for an optimal set of weights $\mathcal{W}$, since updates can no longer be pushed into a previous layer. This structure of repeated matrix multiplication is the primary source of the vanishing gradient problem. This is because, as a network's architecture gets deeper, all odds of one or more terms approaching zero increases. This will *artificially* make the update procedure $\frac{\partial \mathcal{L}}{\partial W_\ell}$ small.

We also see the appearance of a product of terms $\frac{\partial x_{k+1}}{\partial x_k}$, which arises from the fact that $x_k$ is used to generate $x_{k+1}$. The product term $\prod_{k=\ell}^{L-1}\frac{\partial x_{k+1}}{\partial x_k}$ is often encountered in the context of dynamical systems, specifically related to the Lyapunov spectrum of a discrete time system. The Lyapunov spectrum is a set of numbers (individually called Lyapunov exponents) that, roughly speaking, measure of the exponential growth rate of errors in different directions as the dynamical system progressing.

The Lyapunov exponents can be zero or negative. For a Lyapunov exponent that is exactly zero, two close-by trajectories stay a fixed distance away from one another, for every time-slice of the trajectory. Negative Lyapunov exponents mean that small errors in certain directions diminish over time.

Assuming that neither $\frac{\partial \mathcal{L}}{\partial x_L}$ nor $\frac{\partial x_\ell}{\partial W_\ell}$ is exactly zero, it would mean that the resulting product $\prod_{k=\ell}^{L-1} \frac{\partial x_{k+1}}{\partial x_k}$ should be small, but not exactly zero either. This is similar to minimizing the Lyapunov spectrum of the apparent network with respect to the weights $\mathcal{W}$, effectively creating a more stable network[1]. After all, we don't want to flip one pixel of a photo of a cat and have it labeled as a dog by our network. It seems that deep learning and dynamical system share a lot of the same structure, where approaching deep networks from the perspective of dynamical systems may alleviate the vanishing gradient problem.

## C.5 Search Space and Alternative Cost Function

In the prevailing view of backpropagation, the search space is over the set of weights $\mathcal{W} = \{W_1, W_2, \cdots, W_L\}$. However, the way that this search is conducted is by fixing the weights at a particular layer $\ell$, then solving for the weights in the previous layer $\ell - 1$. Such a practice means that erroneous information in the latest layer are also propagated to the earlier layers, which causes the instability in the training process.

$$\mathcal{L}(\mathcal{W}; \mathcal{Y}) = \frac{1}{2N} \sum_{n \in \mathcal{Y}} \left[ x_L^{(n)} - y_L^{(n)} \right]^2 + \frac{\varepsilon}{2(L-1)} \sum_{\ell=1}^{L-1} \left[ x_\ell - f_\ell(W_\ell x_{\ell-1}) \right]^2 \tag{C.9}$$

The controls/weights $\mathcal{W}$ are adjusted, and the entire system dynamics is integrated forward, which is known in this field as the forward pass. The instability is due to the Hamiltonian nature of the forward pass. Instead, we can look for the set of weights that creates a stationary loss function $\mathcal{L}(\mathcal{W}; \mathcal{Y})$ which is the Lagrangian formulation. This is the dual formulation to the backpropagation method, which is inherently more stable. This stability comes at a cost, of course, which is that the parameters have to be individually adjusted. The search space of these networks is enormous compared to the backpropagation, but these may see some use in the more recent *implicit deep learning* architectures,

[1] The idea of 'minimizing the Lyapunov spectrum' is an extremely crude way of describing the idea. As for exactly what is meant by this, I must admit that I am not exactly sure what it means either. A first-pass description of the procedure might be something like the following: we want to change the controls or weights of the system such that the system is more stable in certain directions, but unstable in the directions that matter. Minimizing some of the Lyapunov exponents is a way to create a network that may be stable in some directions, but is it unclear how this affects the instability of the other directions.

where the weights of each layer are held fixed, since they are perfectly compatible. This compatibility is easy to notice since states that satisfy $x_\ell - f_\ell(W_\ell x_{\ell-1}$ do not contribute at all to the sum. In some sense, this loss function almost suggests the jump from the usual deep learning architecture to the implicit version.

## C.6 Remarks

Admittedly, the ideas laid out here are lacking somewhat in rigor, specifically in treating terms like $\frac{\partial x_{k+1}}{\partial x_k}$ as scalars rather than matrices. These ideas are also underdeveloped, so they require implementation and comparisons to test their validity. Nevertheless, we believe that these ideas present a different paradigm towards optimizing and interpreting neural networks. A lot of the groundwork here is establish in the recent paper by Abarbanel et al. [58], and the ideas of this appendix are a direct result of our discussions with those authors. The formulation of this alternative approach to deep learning is almost identical to the data assimilation approach used throughout this work. This alternative view on training deep networks is very interesting and a potentially useful direction of research to pursue, if possible.

# D

# Dynamical Initialization

The data assimilation technique explored in Part I of this work searches a large space for trajectories that are consistent with the model dynamics. As an initial guess, the model uses the available data and slowly 'forgets' the initial state as the constraint term of the cost function is steadily increased in magnitude. The convergence of the algorithm and the quality of the solution is strongly dependent on how close our initial guess is to the true state.

There are a variety of reasons which may influence this. One scenario to imagine is that when you are far away from the global minimum, there may be a multitude of local minima that the states encounters as it makes its way towards the global minimum. It is possible, and common, that the state gets stuck in a sufficiently deep local minimum. The state estimate may not ever reach the global minimum in such a situation, not even a local minimum that is comparable in depth as the global minimum (loosely called *superior minima*). Now, starting the initial guess close to the global minimum is an ill-defined task to begin with, since it begs the question of how one knows which points are close to the global minima. If one has information about the global minimum in the first place, there is no reason to have to find the global minimum. However, all we have to do is to propose a better initial condition than a random one, and the bar is set fairly low so we can actually make progress on this.

We will lay out a special case here on how to propose a good initial condition in trajectory space (not state space). The trajectory $x(t)$ is represented as a $D \times M$ matrix, so the distance between two trajectories $x(t)$ and $y(t)$ is given by the Frobenius norm of their different, i.e $\|x(t) - y(t)\|_F^2$. In

our special case scenario, we have a physical system with a $D$-dimensional state space that is set up in a laboratory setting, where we can make measurements of all $D$ dimensions of the system. We also have access to the dynamical equations of this system, which are known to a high degree of confidence. Fortuitously, the instruments that are measuring this system were not properly set up such that some fraction of these instruments/sensors fail to produce coherent and reliable measurements of this system. In other words, we can only rely on the data from a few instruments. How does one 'make up' or reconstruct the missing data?

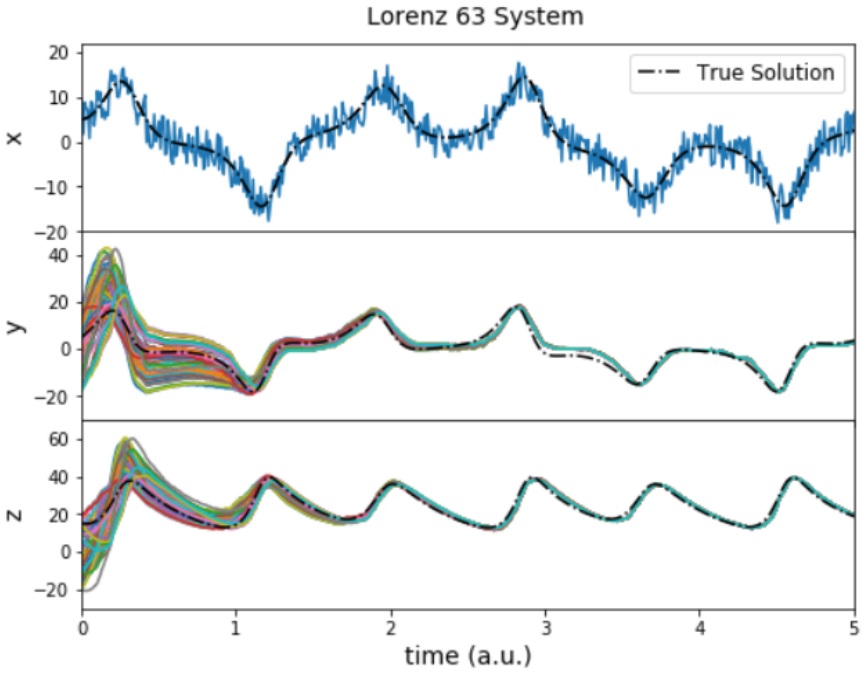


**Figure D.1:** Using dynamical initialization, noisy data of the $x$ variable and randomly chosen initial values of $y$ and $z$ are useful to generate the entire trajectory. This is repeated for the same data for $x$, but 50 different initial values of $y$ and $z$. We observe that all 50 trajectories eventually converge to a wrong but relatively close-by state. This relationship appears to hold indefinitely.

We have found an interesting technique that will apply in such a situation. We shall look at a more concrete example of the situation laid out above, namely measurements of the **Malkus waterwheel** which are governed by the Lorenz equations. In our case, we set up the problem such that the $x$ variable of the Lorenz system is the only reliable measurement, albeit some noise. The measurements of the $y$ and $z$ variables are utterly unreliable and discarded completely. The method that we call *Dynamic Initialization* is to use the noisy time-series measurements of $x$ as the *de facto* input for $x$, for all time, when constructing the vector field of the Lorenz system. On the other hand,

at the first time step we use randomly initiated values for $y_0$ and $z_0$. The system is integrated forward in time one step, say, with the RK4 scheme and only the resulting values of $y_1$ and $z_1$ are kept from this integration step. Whatever value for $x$ that resulted from the integration is discards, instead we use the measurement data of $x_1$, as well as the integrated values of $y_1$ and $z_1$, to generate the $y$ and $z$ variables of the next step. We continue this process until we run out of data. For this example, the parameters used to generate $x$ are exactly the same as the parameters used to perform the integration, and we are aware of how convenient but unlikely this situation is. This wordy explanation might be confusing, so we have included an algorithmic pseudo-code description below.

**Algorithm 2** Dynamic Initialization example for the Lorenz system, with $x(t)$ measured but $y(t)$ and $z(t)$ not measured. Both $y(t)$ and $z(t)$ are set to 0 in this example, but the idea holds more generally.

$x_0 \leftarrow x_0$
$y_0 \leftarrow 0$
$z_0 \leftarrow 0$
**for** $i = [1, M-1]$ **do**
  $\{\dot{x}_i, \dot{y}_i, \dot{z}_i\} \leftarrow f(\mathbf{x}_i)$
  $x_{i+1} \leftarrow x_{i+1}$ ▷ $x(t)$ is **never** updated.
  $y_{i+1} \leftarrow y_i + \dot{y}_{i+1}$
  $z_{i+1} \leftarrow z_i + \dot{z}_{i+1}$

We believe that the mechanism that induces this behavior is related to some form of synchronization, even though we will not be exploring the phenomenon of dynamical initialization any further, at least within the confines of this work. We are also aware that the applicability of dynamical initialization is narrow since we would have access to the exact equations and the state space data. Even so, we have illustrated a situation where dynamical initialization may be used with the Malkus waterwheel and faulty sensors, and we hope that there are more situations like this. Nevertheless, the phenomenon of dynamical initialization warrants further exploration and exposure so that it may be useful to whomever is faced with a scenario similar to what we laid out.

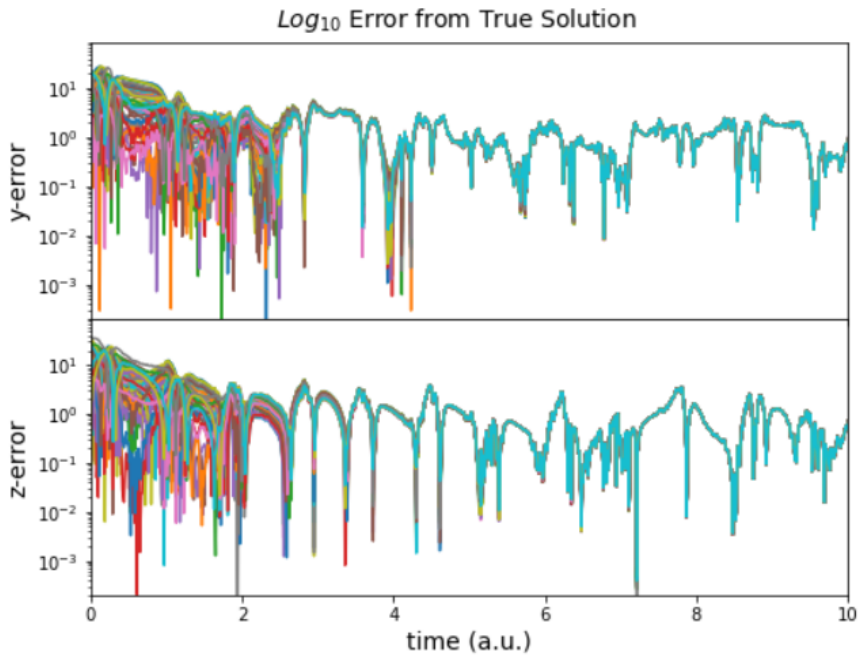


**Figure D.2:** Errors in the $y$ and $z$ variables as time progresses. The errors decay over time until all 50 states converge. The convergence of state implies that there is some form of synchronization that drives this approach.

# E

# Lyapunov Spectrum

## E.1 Introduction

When handling nonlinear systems exhibiting chaos, one should have some tools available that can characterize the chaos in the system. The Lyapunov spectrum is one of these tools, and it refers to the set of Lyapunov exponents that succinctly describes the global behavior of chaos within the system. Fundamentally, they measure the average rate of divergence or convergence of trajectories around the entire attractor. The calculations for the Lyapunov spectrum are inherently statistical in their nature, which arise from the invocation of ergodic theory on the dynamical system, so long-time averages are a fundamental part of the calculation. The full scope of topics that the Lyapunov spectrum touches many branches of mathematics such as differential geometry, topology, fractals, and measure theory – to name but a few. Across all these branches, the Lyapunov spectrum have different interpretations and consequences, leading to some pretty deep questions that span across different disciplines. Regardless of the choice of application, this section of the appendix handles the methods for calculating the Lyapunov spectrum in both continuous and discrete time systems, but only for the case of when there is access to the dynamical equations. Though important, we do not handle the calculation of the Lyapunov spectrum in the case where we only have access to data with no equations. Almost all of the time is dedicated to continuous systems, because the practical problems faced by the modern world are generally continuous – fluid mechanics and electromagnetism chiefly being the two largest culprits. Various methods, each with its own quirks and insights, will be visited.

## E.2 Largest Lyapunov Exponent

No matter what type of problem one wants to approach – continuous or discrete, using equations or using data – it is instructive to start with the largest (most positive) element of the Lyapunov spectrum, which is called the **Largest Lyapunov Exponent** (LLE). This will serve as a soft introduction of the Lyapunov spectrum as it is the simplest to calculate and interpret. We will only be handling autonomous systems that are either represented by differential equations (continuous) or some forward map (discrete). When calculating only the largest exponent with this method, it is irrelevant whether the system is continuous or discrete. If there is no access to the underlying equations that generate the data, simply skip the (trivial) generation step.

$$\dot{x} = f(x) \qquad ; \qquad x_{k+1} = F(x_k) \tag{E.1}$$

We shall be assuming here that the system of interest does exhibit chaotic behavior, otherwise the entire discussion will be moot. Though there is no widely agreed upon definition of chaos, it will suffice here to say that chaotic systems can simply be described as having extreme sensitivity to initial conditions. More practically, the definition can be '*exponential divergence of nearby trajectories up to a bounded size*'. The boundedness of this divergence is important, otherwise a trivial function such as the time exponential $e^t$ will be deemed chaotic. This bound happens to be the size of the attractor. We can compare a simple case where a **close-by initial pair** $x_0^i$ and $x_0^j$ are integrated forward up to some large time $t$. At time $t$, these two final states are $x^i(t)$ and $x^j(t)$ respectively. The LLE is then given by

$$\widetilde{\Lambda}(x_0^i, x_0^j) = \lim_{x_0^i \to x_0^j} \lim_{t\to\infty} \frac{1}{t} \ln \frac{\left\| x^i(t) - x^j(t) \right\|}{\left\| x_0^i - x_0^j \right\|} \equiv \lim_{\delta x_0^{i,j} \to 0} \lim_{t\to\infty} \frac{1}{t} \ln \frac{\left\| \delta x^{i,j}(t) \right\|}{\left\| \delta x_0^{i,j} \right\|}. \tag{E.2}$$

However, this is definition is biased towards the choice of the two initial conditions $x_0^i$ and $x_0^j$. It could incorporate the parts of the attractor that are unusually stable (e.g. within one lobe of Lorenz 63) or unstable (e.g. only capturing the lobe switches of Lorenz 63). To eliminate the bias, we simply repeat the above calculation for a large number of such pairs $x_0^i$ and $x_0^j$ in a brute force manner, and take the unweighted average. Effectively, we are looking at some point $x_0^i$ and many of its neighbors $x_0^j$, then this is repeated for values of $x_0^i$ at other parts of the attractor. It is imperative that the points chosen for $x_0^i$ are on the attractor, and not transients of some far-away initial conditions. The choice of $x_0^j$ also need to be close to the chosen $x_0^i$, generally this means that $x_0^j - x_0^i$ forms a small ball

roughly four to five orders of magnitude smaller than the attractor size. These steps should eliminate the biases as much as reasonably possible.

We also need to consider that there is some point in time $t$ after which the 'exponential divergence of nearby trajectories' is capped by the size of the attractor, after which we no longer want to incorporate these samples into our average. This point in time will be called $T^{i,j}$, and it is a function of the pair of initial conditions. Pausing for a moment to consider what this means; this suggests that the LLE is calculating the local growth rates of the differences between neighboring states.

$$\Lambda = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} \frac{1}{T^{i,j}} \ln \frac{\left\| \delta x^{i,j}(T^{i,j}) \right\|}{\left\| \delta x_0^{i,j} \right\|} \tag{E.3}$$

This formulation hardly the most elegant representation of the concept of Lyapunov exponents, but it is illustrative of the fact that the LLE is an averaged quantity calculated **only** through numerical simulations. A few well-warranted questions come to mind, the first is regarding the size of both $M$ and $N$, to which there is no good *a priori* answer. In practice, both $M$ and $N$ on the order of 200 yield reasonable calculations. The second question pertain to how long of a time window $T$ is necessary. There is a Goldilocks-zone one has to wrestle with. We only want to look at the growth rate of the errors, so too long of a window results in the errors plateauing at attractor size, contaminating our sample pool. On the flip side, too short of a window will results in the improper lly captured. There is really no good answer here as well.

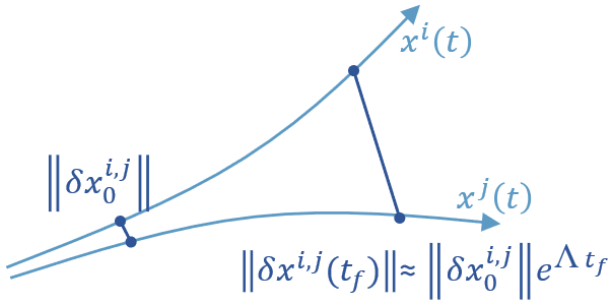


**Figure E.1:** Divergence of different initial conditions is a way to estimate the LLE.

As should be obvious by now, this method is crude in many ways and fraught with imprecision, in addition to it only giving the largest exponent. For starters, the sheer amount of computational power required to perform (E.3) successfully is frighteningly wasteful given the quality of the calcu-

lation. The following sections will deal with the rest of the spectrum, which is handled differently for continuous and discrete time systems, and also differently when one has no access to the underlying equations.

## E.3 Continuous Time Systems

Continuous time systems are represented by differential equations and the continuity refers to the nature of time, not states. These differential equations are integrated by the differential equation specified, using any reasonable numerical integrator, from some initial condition $x(t_0) \equiv x_0$.

$$\dot{x} = f(x) \tag{E.4}$$

These continuous systems always have to be of dimension three or greater in order for chaos to manifest and almost always involves a set of parameters that are not shown here in the notation. This is due to the mathematical limitations of one-dimensional trajectories eventually crossing in two dimensions, but there exists 'another way out' in three dimensions and above. For the entirety of the continuous time section, we will demonstrate our findings using the Lorenz 63 with the parameters $\sigma = 16, \rho = 45.92$, and $\beta = 4$. The numerical integrator used in the 'classic' 4th-order Runge-Kutta method (RK4) and a time-step of $\Delta t = 0.01$. The initial conditions are $x_0 = [19, 20, 50]$, which already lie very close to the attractor.

Omitting the use of the equations is certainly possible because there are methods that can be used to estimate Lyapunov exponents directly from data. These methods turn out be the most general and widely applicable approach since we can always use the equations to generate data. However, having access to the underlying equations, we might as well utilize them to calculate the Lyapunov spectrum. Leveraging the equations also allows us to calculate the Lyapunov exponents to incredibly high accuracy and at significantly faster computation. Without the equations, the quality of the calculation relies heavily on having extremely long histories of the trajectories, and the presence of noise low noise also caused the quality of calculation to deteriorate quickly. Practically speaking, having access to such data sets is rare, but having access to the underlying equations and the parameters is, arguably, equally as rare. As such, we should have methods and approaches for both these cases. If access to both the data and the system are available, then the methods leveraging the equations should definitely be utilized.

### E.3.1 Variational System and Volume Growth

To properly estimate the Lyapunov exponents, we first introduce the concept of a variational system $Y(t)$ which co-evolves alongside the usual system $x(t)$ with the rules

$$\dot{Y} = J(t)Y \qquad ; \qquad J_{ij} \equiv \left.\frac{\partial f_i}{\partial x_j}\right|_{x(t)} \qquad ; \qquad Y(t_0) \equiv Y_0 = \mathbb{1} \tag{E.5}$$

where the $Y(t)$ is a matrix system with arbitrary initial condition $Y_0$, chosen as the $D$-dimensional identity matrix for simplicity and convenience (as will be revealed later). It may be instructional to view the variational system $Y(t)$ as an auxiliary system which tracks certain aspects of the original system $x(t)$. The matrix $J(x)$, called the **Jacobi matrix of the vector field** or simply **Jacobian**, is populated by the elements $J_{ij}$ that are evaluated at every point $x(t)$ along its trajectory [59]. Note that above, $i$ or $j$ are the spatial indices and will remain so for the rest of the paper. The initial basis vector $Y_0$, which forms a *unit box* spanning the entire $D$-dimensional space, is integrated forward to produce $Y(t)$ that tracks the contraction or expansion of spatial dimensions across time.

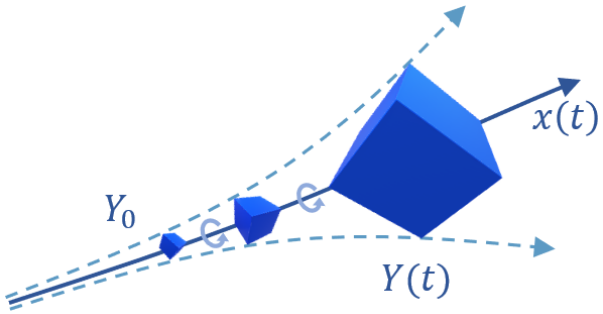


**Figure E.2:** Tracking the exponential growth of a *unit box* in time is an intermediate step towards calculating all the Lyapunov exponents. This box is both stretched and rotated as the system evolves, but only the stretching aspect is of interest. The use of a variational system is an elegant generalization of the previous method which only yielded the largest exponent.

Whereas the previous calculation in (E.3) only gives the largest exponent, tracking the variation system $Y(t)$ will give the entire spectrum. Solving for the entire trajectory $Y(t)$ invariably requires numerically integrating the above system, say with the RK4 method. From the set of evolution rules (E.5), we can integrate a solution for $Y(t)$ alongside $x(t)$ by defining an extended system

and its dynamics.

$$\begin{bmatrix} \dot{x} \\ \dot{Y} \end{bmatrix} = \begin{bmatrix} f(x) \\ J(t)Y \end{bmatrix} \qquad ; \qquad \begin{bmatrix} x(t_0) \\ Y(t_0) \end{bmatrix} = \begin{bmatrix} x_0 \\ \mathbb{1} \end{bmatrix} \tag{E.6}$$

As a practical aside, it is important to note here that the integration of both $x(t)$ and $Y(t)$ should be carried out as a single extended system rather than two systems independently. This is just an artifact of order of function evaluations that forward integrators use. Once the entire time series of both $x(t)$ and $Y(t)$ have been calculated, we can observe how the volume of the variational system grows or shrinks. Since vol $Y_0 \equiv 1$, the relative volume of $Y(t)$ is given by the scalar triple product of its columns, which happen to correspond to the determinant of $Y(t)$ as well. It was previously mentioned that we are interested in the exponential divergence, so the exponential form for the volume of $Y(t)$ is imposed. We can now move the determinant operation into the trace of the exponent.

$$\text{vol}\, Y(t) = |\det Y(t)| = \det e^{\widetilde{J}t} = e^{\text{tr}\, \widetilde{J}t}. \tag{E.7}$$

Given the appearance of the trace operation, the connection between eigenvalues of the matrix $\widetilde{J}$ and the volume of the variational system $Y(t)$ become apparent. These eigenvalues of $\widetilde{J}$ turn out to be the Lyapunov exponents or the Lyapunov spectrum of the **original system**, which were co-opted to *grow* the variational system according to the rules in (E.5)[60].

$$\text{tr}\, \widetilde{J} = \sum_{d=1}^{D} \lambda_d = \lim_{t \to \infty} \frac{1}{t} \ln \text{vol}\, Y(t) \tag{E.8}$$

The Lyapunov exponents are always arranged in the descending order $\lambda_1 > \lambda_2 > \cdots > \lambda_D$ where $D$ is the dimension of the state space $x$. Notice that there is a limit of $t \to \infty$ in the equation above to ensure that we are estimating global properties of the attractor, rather than being biased by transient from the local region around the initial condition. So far, we have shown how to compute the trace of the Lyapunov spectrum, but teasing out the individual Lyapunov exponents individually require some additional work in the form of matrix decompositions. The individual Lyapunov exponents can be seen as the growth rate of the principle directions describing the matrix decomposition of choice, as is the topic of the next segment, is the QR decomposition.

### E.3.2 Quick Aside: QR Decomposition

A quick and brief introduction to QR decomposition is in order, covering only the requisite knowledge for our use case. We will be assuming here that $Y$ is real $D \times D$ square matrix, as is the case for our purposes. Given a real square matrix $Y$, there always exists a decomposition

$$Y = QR \tag{E.9}$$

where $Q$ is an orthogonal matrix and $R$ is an upper triangular (or right triangular) matrix. It could be helpful to think about the this as a decomposition of $Y$ into two parts - a *stretching* operation $R$ followed by an *improper rotation* $Q$. The summary of orthogonal square matrices is simply[1]

$$Q^T = Q^{-1} \qquad ; \qquad QQ^T = Q^TQ = \mathbb{1}. \tag{E.10}$$

The vectors spanning $Q$ are all mutually orthogonal and of unit length, which means that they form an orthonormal basis. If $Y$ is invertible (specifically that all eigenvalues are non-zero), then the decomposition is also unique. This decomposition is unusually useful for calculating the determinant, and provides additional benefits later on. With $Q$ being an orthogonal matrix, we have that $|\det Q| \equiv 1$.

$$|\det Y| = |\det QR| = |\det Q| \cdot |\det R| = |\det R| \tag{E.11}$$

The determinant of $Y$ can be calculated directly from that of $R$, and this is the justification for using the QR decomposition. This is more than a quick numerical trick for the determinant, as the added structure of $Q$ and $R$ is going to be useful.

$$|\det Y| = \prod_{d=1}^{D} |\lambda_d(Y)| = \prod_{d=1}^{D} |R_{dd}| \,. \tag{E.12}$$

As another added bonus of triangular matrices, the diagonal elements $R_{dd}$ do not just multiply together to give the determinant, they are themselves the eigenvalues of $R$ – a fact that we will leverage this later on. For how powerful the QR decomposition is, especially in the Lyapunov spectrum calculation, it is surprisingly cheap/fast to run at $\mathcal{O}(D^3)$. It is almost always going to be efficiently

[1] For non-square matrices, only $Q^TQ = \mathbb{1}$ holds and $QQ^T \neq \mathbb{1}$. This is common in high-dimensional problems, where $Q$ is generally expected to be a tall/skinny matrix that spans only a subspace.

implemented and readily available in a good linear algebra library written in any major programming language you chose to work with.

### E.3.3 The Entire Spectrum

Armed with the knowledge of QR decomposition and its usefulness, we pick back up recalling the relationship between the Lyapunov spectrum and volume, that is the volume if given by the product of all Lyapunov exponents. Once we perform the QR decomposition $Y(t) = Q(t)R(t)$, recall that volume of the variational system can be rewritten as

$$\text{vol}\, Y(t) = |\det Y(t)| = \prod_{d=1}^{D} |R_{dd}(t)|\,. \tag{E.13}$$

Plugging in the above equation into (E.8), the product can be moved out of the logarithm and transformed into a sum. Moving away from having a product inside the logarithm means better numerical stability, as extremely small values of $R_{dd}$ will be operated on by the logarithm, which reduces the range of possible values. A smaller range of values leads to better numerical stability because machine precision creeps in when operating on a very large and a very small number.

$$\sum_{d=1}^{D} \lambda_d = \lim_{t\to\infty} \frac{1}{t} \ln \prod_{d=1}^{D} |R_{dd}(t)| = \sum_{d=1}^{D} \lim_{t\to\infty} \frac{1}{t} \ln |R_{dd}(t)| \tag{E.14}$$

Given that the QR decomposition is unique, the set of numbers $R_{dd}(t)$ along the diagonal of $R(t)$ is also unique[2]. Thus, we conjecture that the individual elements of the Lyapunov spectrum are also uniquely determined and given by

$$\lambda_d = \lim_{t\to\infty} \frac{1}{t} \ln |R_{dd}(t)|\,. \tag{E.15}$$

To add some interpretation to the importance of $R(t)$, its diagonal elements $R_{dd}(t)$ characterize the local growth of the basis vectors that span $Y(t)$, which are given by the column vectors of

[2]These diagonal elements $R_{dd}$ are the eigenvalues of $R$, but not the eigenvalues of $Y$. However, the two sets of eigenvalues are closely related. Since we are concerned with the exponential growth of volumes handled by $R$, which is independent from the rotations handled by $Q$, the eigenvalues of $R_{dd}$ do the job of characterizing the exponential growth of volume of $Y(t)$. This justifies their use in calculating the Lyapunov spectrum, rather than the eigenvalues of $Y$ that are complex numbers. These imaginary parts describe some (surprisingly quick) rotations, but are unimportant for tracking volumes.

$Q(t) = [q_1(t), q_2(t), \cdots, q_D(t)]$. The individual elements of Lyapunov spectrum $\lambda_d$ describe how the exponential growth of distances, on average, along the corresponding basis vectors $q_d(t)$.

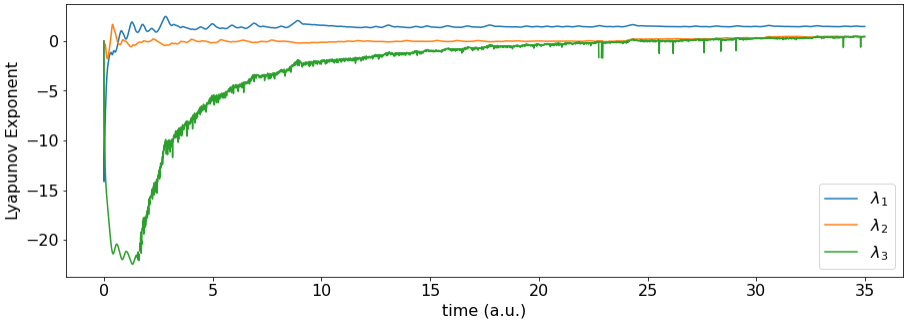


**Figure E.3:** The Lyapunov spectrum of the Lorenz 63 system calculated with the method described by (E.5) and (E.15). $\lambda_1$ holds steady and agrees well with the true value. $\lambda_2$ holds steady at the true value only initially, but gets corrupted by the creeping values of $\lambda_3$ around $t = 25$. $\lambda_3$ manages to hover around the true value (approximately $-22.5$) but very quickly becomes corrupted.

Some might argue that the above definition of the Lyapunov spectrum is not sufficient, with some merit, as some form of ensemble average should be taken to avoid biases in sampling the attractor. This may be true, but there are more pressing issues to deal with. Namely, the smallest Lyapunov exponent quickly deteriorates and begins to corrupt the next smallest exponent, as seen in the figure above. Despite the trouble it took to get to this point, the equation (E.15) only looks elegant but is otherwise unwise to use for practical calculations. The degradation occurs because the smallest Lyapunov exponents is comparatively larger in magnitude than the other exponents, leading to an extremely fast shrinkage in that direction. Computationally, this results in machine precision issues when attempting to keep track of that exponent. This degradation eventually spreads from only affecting the smaller exponent, to affecting the second smaller exponent. It is unclear what *exactly* causes the degradation, but likely it is due to the mixing of the Lyapunov vectors (directions in which the exponents affect) as the variational system evolves in time. Regardless, this is a strict upgrade to (E.3) due to the LLE being well approximated even for long time windows - and the calculation is faster to run too. The main takeaway of the section should be that the Lyapunov spectrum characterizes the average expansion of volume in time, though precise calculations require additional careful steps. There is much improvement to be made in varying directions and will be explored in the following sections. The variational system method here, as it turns out, is similar in nature to the Successive QR method that will be covered later. The one difference between these two methods is that there are redundant calculations in the variational system method, as will be discussed when we talk about

Successive QR. Nevertheless, this variational system method establishes a good conceptual starting point and is preferred from a pedagogical standpoint.

## E.4 Practical Calculations

### E.4.1 Normalized Variational System

In the previous segment, we showed the calculation of the entire Lyapunov spectrum by observing the expansion of volumes of a variational system $Y(t)$. Leveraging the powerful QR decomposition, we show that the fundamental directions of growth are given by the column of $Q(t)$ and the local growth rate in those directions are given by the diagonal elements $R_{dd}(t)$. Yet, we run into issues because the important structures of $Y(t)$ do not hold in long-time simulations [61]. Numerical instabilities arise, yet these are simply artifacts of using finite precision. Otherwise, there is nothing fundamentally wrong with the method above. One simple fix that comes to mind is to avoid taking the limit in (E.15) too literally. Instead, we can break up the full time window of $n$-steps into multiple shorter windows of $m$-steps (such that $n = am$ where $a$ is an integer), where $Y_k$ is normalized to have unit volume at the end of that window. This way, the same calculation can be done without running into numerical issues. We shall call the normalized $Y_k$ as $\widetilde{Y}_k$.

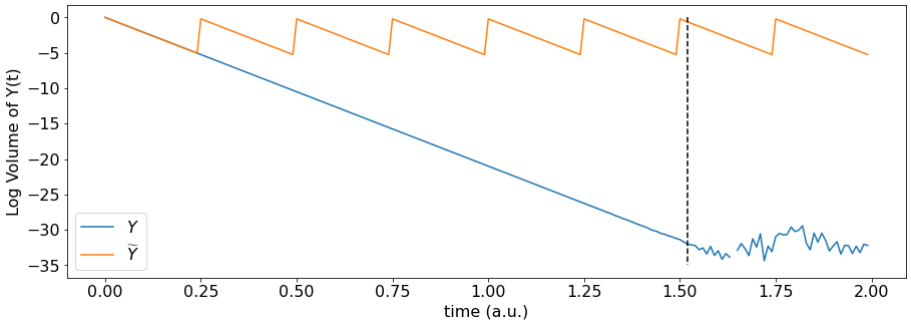


**Figure E.4:** The shrinking log-volume of $\widetilde{Y}$ and $Y$ against time. $\widetilde{Y}$ is normalized as a unit volume ($\ln 1 = 0$) every $m = 25$ steps of $\Delta t = 0.01$. The normalized $\widetilde{Y}$ continues behaving well even after $Y$ is running into stability issues, presumably from numerical precision issues. The average slope of this saw-tooth, excluding the discontinuities, will give the sum of Lyapunov exponents. After computing $\widetilde{Y}_k = Q_k R_k$, the individual exponents can be calculated according to the diagonal elements of $R_k$, as usual.

Every $m$-steps, i.e whenever $k$ is a multiple of $m$, we have to normalize $\widetilde{Y}_k = Q_k R_k$ by setting $\widetilde{Y}_k \leftarrow Q_k$. This constitutes a normalization step because the columns of $Q_k$ are all unit vectors hence $\operatorname{vol} Q_k = |\det Q_k| = 1$. This is yet another clever application of the QR decomposition since

it preserves the direction of growth that is important to keep track of, all while preserving the full Lyapunov spectrum. There is a small price to pay, which is a normalization step every so often, and it results in a minuscule effect on the computational times. So, instead of taking the limit of $t \to \infty$ and dealing with numerical and stability issues, the equation below is better thought of an average quantity in the limit of large sample sizes.

$$\lambda_d = \lim_{n\to\infty} \sum_{\substack{k=1 \\ (k \bmod m)\neq 0}}^{n} \frac{1}{t_k} \ln |R_{dd}(t_k)| \,. \tag{E.16}$$

On the usual topic of how large of a value to use for $m$, Figure E.4 gives us an *a posteriori* clue. Around $t = 1.5$, the smooth slope of the plot gives way to some sort of error that is clearly creeping into our calculation. This hints that any value of $m$ that stops the window before then would likely be a safe choice. This normalization method is incredibly useful and will serve as a basis for next method to be discussed.

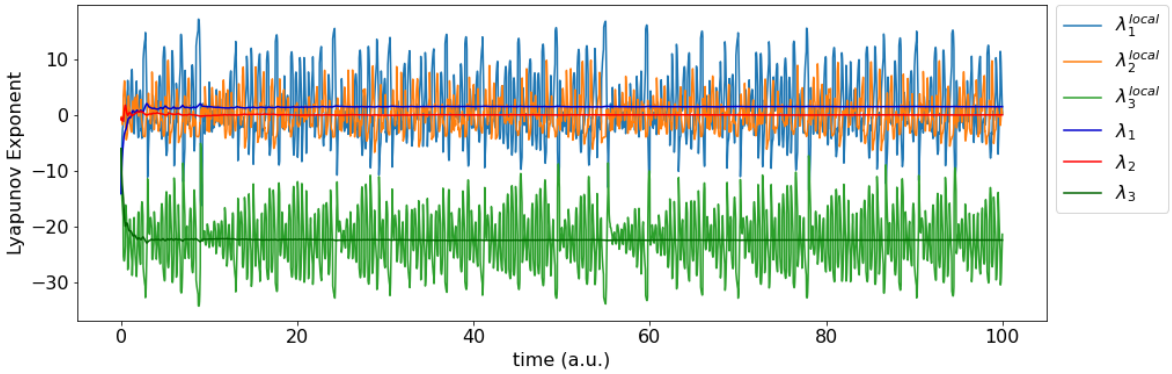


**Figure E.5:** Using the normalized variational system of subsection E.4.1 for Lorenz 63, the local Lyapunov exponents and their averages are calculated. The three lighter colored lines are the local Lyapunov exponents. The three darker colored lines are the rolling average of the local Lyapunov exponents. With normalization, this method converges very quickly to (essentially) the consensus value of $[1.49, 0.00, -22.49]$ and is much more reliable than that of Figure E.3.

### E.4.2 QR Exploit

This method exploits the QR method in a way that utilizes many of the important properties of orthogonal and upper-triangular matrices, hence its name. This method is also called the *continuous QR method*, but there appears to be no wide-agreed-upon name due to the relative obscurity of this method. The choice of the word 'continuous' makes sense given that this method is completely agnostic

to the chosen value of $\Delta t$. As long as trajectories (even the shadow trajectories) along the attractor are given, the method should accurately determine the Lyapunov spectrum with no problems. We start by posing that the variational system $Y(t)$ can be QR decomposed at any point in time.

$$Y(t) = Q(t)R(t) \tag{E.17}$$

All three matrices $Y(t)$, $Q(t)$,and $R(t)$ are assumed to have a time dependence to them. So naturally, we can look at their time derivatives. Applying the chain rule, we can write down the equation

$$\dot{Y}(t) = \dot{Q}(t)R(t) + Q(t)\dot{R}(t). \tag{E.18}$$

However, we already have another definition of $\dot{Y}(t)$ from the dynamic equation of the variational system given in (E.5). We further leverage both these definitions to establish the following relation.

$$\dot{Y}(t) = J(x)Y(t) = J(x)Q(t)R(t) \tag{E.19}$$

Lastly, we have the definition of orthogonal matrices, which holds at any point in time. The RHS of the equations below is constant though the LHS is time dependent. Taking a time derivative of the orthogonal property gives a very useful relation, which is one of the cornerstones of this method.

$$\begin{aligned} Q^T(t)Q(t) &= \mathbb{1} \\ Q^T(t)\dot{Q}(t) + \dot{Q}^T(t)Q(t) &= 0 \end{aligned} \tag{E.20}$$

From the above equations, we now have a complete set of interesting equations that underpins this method. From here on, the time dependence of the matrices inside the parentheses will be dropped for clarity and brevity sake. Collecting the time derivatives $\dot{Y}$ from (E.17) and (E.18) gives the following equation.

$$\dot{Q}R + Q\dot{R} = JQR \tag{E.21}$$

This equation is rearranged to get all the matrices involving $R$, which are $\dot{R}R^{-1}$ here, to one side. Both these matrices have special properties that arise from their relation to upper-triangular

matrices, and all the information regarding the Lyapunov spectrum are contained in the upper-triangular matrices of the QR decomposition.

$$\dot{R}R^{-1} = Q^T J Q - Q^T \dot{Q}. \tag{E.22}$$

Two facts about upper-triangular matrices are in order. First, the inverse of upper-triangular matrices is also upper-triangular. Second, the product of two upper-triangular matrices is also upper-triangular. Hence, we see that $\dot{R}R^{-1}$ is still upper-triangular – which we will soon leverage. The importance of the upper-triangular matrix like $\dot{R}R^{-1}$ is due to them preserving the determinant of matrices, as shown in (E.12), which will be apparent soon. Next, we can see from (E.20) that

$$Q^T \dot{Q} = -\dot{Q}^T Q \qquad ; \qquad \dot{Q}^T Q \equiv (Q^T \dot{Q})^T. \tag{E.23}$$

Gathering the above terms, it is revealed that the matrix $Q^T \dot{Q}$ has a surprising structure, that it is actually anti-symmetric.

$$-Q^T \dot{Q} = (Q^T \dot{Q})^T \tag{E.24}$$

As $Q^T \dot{Q}$ is an anti-symmetric matrix, the elements along the diagonal are necessarily all zeros and the subtraction of the $Q^T \dot{Q}$ leaves the diagonal elements of the $\dot{R}R^{-1}$ intact and unchanged. This leads to the diagonal elements of $\dot{R}R^{-1}$ being uniquely supplied by the diagonal elements of $Q^T J Q$, which does not have any time derivatives. The lack of time derivatives is a fascinating and welcomed feature as it avoids potential numerical instabilities in the calculation of $(\dot{R}R^{-1})_{dd}$.

$$(Q^T J Q)_{dd} = (\dot{R}R^{-1})_{dd} = \frac{\dot{R}_{dd}}{R_{dd}} = (R^{-1}\dot{R})_{dd} \tag{E.25}$$

We serendipitously run into another remarkable property of two upper-triangular matrices regarding diagonal elements, hence the second equality in the equation above. Specifically, that the diagonal elements of two upper-triangular matrices are sufficient to determine the diagonal elements of their resulting product. We need not look at the off-diagonal elements of $\dot{R}$ or $R^{-1}$, nor do we need to evaluate the inverse of $R$ explicitly. This is not a property leveraged at all in the method, but it

yet another serendipitous property that results in computational savings. One could also populate all the off-diagonal elements of $Q^T\dot{Q}$ from $Q^TJQ$ if so desired, but those elements will not be useful for our purposes [62]. Now that we have gained some insight on the importance of $\dot{R}R^{-1}$, we table this result and look at (E.22) and its relation to $\dot{R}R^{-1}$. Rearranging for $J$, we see $\dot{R}R^{-1}$ show up in the following way.

$$J = \dot{Q}Q^T + Q\dot{R}R^{-1}Q^T \tag{E.26}$$

We run into a familiar looking object $\dot{Q}Q^T$ on the RHS. Though not trivially related to its cousin $Q^T\dot{Q}$, some simple shuffling of (E.20) will reveal that both these objects are anti-symmetric, but only for square matrices $Q$. For more general cases, which may come up when solving for the top few Lyapunov exponents, one has to deal with rectangular matrices for which $Q^TQ = \mathbb{1} \neq QQ^T$. However, under the operation of a trace, we get more surprising but useful results.

$$\mathrm{tr}\, J = \mathrm{tr}\Big(\dot{Q}Q^T\Big) + \mathrm{tr}\Big(Q\dot{R}R^{-1}Q^T\Big) \tag{E.27}$$

The trace has a useful property regarding cyclic permutations, and we use this to move the orthogonal matrices $Q^T$ to the front.

$$\mathrm{tr}\Big(\dot{Q}Q^T\Big) + \mathrm{tr}\Big(Q\dot{R}R^{-1}Q^T\Big) = \mathrm{tr}\Big(Q^T\dot{Q}\Big) + \mathrm{tr}\Big(Q^TQ\dot{R}R^{-1}\Big) \tag{E.28}$$

In the first term, the anti-symmetric matrix $Q^T\dot{Q}$ appears again. An anti-symmetric matrix definitively has zeros along its diagonal, hence the trace of $Q^T\dot{Q}$ is zero and bears no effect on the trace of $J$. In the second term, we have the appearance of $Q^TQ = \mathbb{1}$ defined to be the identity matrix and plays no part at all in the trace operation. All that is left for determining the trace of $J$ is the trace of the upper-triangular matrix $\dot{R}R^{-1}$.

$$\sum_{d=1}^{D} \lambda_d = \mathrm{tr}\, J = \mathrm{tr}\Big(\dot{R}R^{-1}\Big) \tag{E.29}$$

The matrix $\dot{R}R^{-1}$ is not directly computed. Calculation $\dot{R}$ from time difference can be fraught with issues, and it is not clear how the analytic form for $R^{-1}$ should be approached. Inverting $R$ may be quick and straightforward, but is otherwise useless unless we are able to determine $R^{-1}$ properly.

As a result, neither of these matrices are explicitly calculated. Instead, we use the relation in (E.25). Remember that both $Q$ and $R$ are unique once the square matrix $J$ is prescribed. The trace is also not evaluated, but is left merely as a a tool for simplifications done above. The Lyapunov spectrum is therefore uniquely determined, but here only locally in space – we refer to this as the local Lyapunov spectrum. The spectrum is given by the diagonal element of the resulting matrix $Q^T JQ$.

$$\lambda_d^{\text{local}} = (Q^T JQ)_{dd} \tag{E.30}$$

Since this is only a local exponent, we would need to take an ensemble average at various locations on the attractor to get the (global) Lyapunov exponents.

$$\lambda_d = \lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} \left[Q(t_k)^T J(t_k) Q(t_k)\right]_{dd} \qquad ; \qquad Y(t_k) = Q(t_k)R(t_k) \tag{E.31}$$

However, this calculation also relies on performing QR decomposition on the variational system $Y(t)$ to get $Q(t)$. This will quickly run into numerical rounding issues or other stability issues as $t$ fuels the exponential growth or contraction of the system. Luckily, we already covered the normalization trick of subsection (E.4.1), which can be utilized here in addition to the QR exploit to get much better accuracy. This method seems to be a natural fit for the normalization trick because the calculation suggested for the Lyapunov exponents here do not care about the growth rates $R(t_k)$ of the variational system $Y(t_k)$, just the directions of the growth $Q(t_k)$. As beautifully elegant as this method is, it appears to be agnostic to the issue of path-ordering. Path-ordering is important as the $J(t_k)$ are mutually non-commutative, yet they need to be applied in the correct order for the variations system $Y(t)$ to exhibit the proper growth rates. So far, through empirical trials, this method has proven to be just as good as the others and the issue of ordering may not be important. It might be the case that the ergodic property of the Lyapunov spectrum overcomes the need for sequential applications of $J(t)$.

### E.4.3 Using Matrix Exponents (Successive QR Method)

The method has two key features. The first is the use of a matrix exponent as an integration scheme of the variational system $Y(t)$. The second key feature is a workaround for the normalization issue previously explored. However, we would need to use some features of the QR decomposition

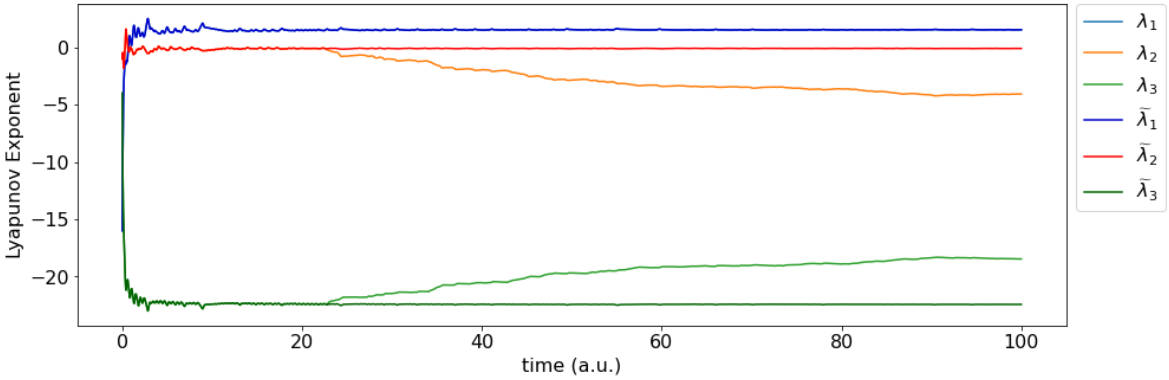


**Figure E.6:** The 'QR exploit' method applied to Lorenz 63 gives us the values of the Lyapunov spectrum as we would expect. A rolling average of the local Lyapunov exponents is used here. However, without normalization, we run into spurious degradation of the lower two exponents. Once normalization is implemented (denoted with superscript $\sim$), the calculation converges to the expected values. The first Lyapunov exponents with and without normalizing are identically overlapped. The degradation happens around the $t = 22$ mark, and affects the lower two Lyapunov exponents symmetrically such that the trace is always preserved. The convergent values here are $[1.51, -0.02. - 22.48]$.

again to fully utilize this workaround. To explore the first feature, we look at the formulation of the variational system (E.5), specifically where $\dot{Y} = J(t)Y$. It would seem like a matrix exponential of $J(t)$ would be a solution in the form of a forward map

$$Y(k\Delta t) \simeq Y_k = e^{J_{k-1}\Delta t} Y_{k-1}, \tag{E.32}$$

denoting $J_k \equiv J(t_k)$ and the time step is $\Delta t \equiv \frac{t}{n}$. Through some analysis, this exponential form holds well when the vector field of the original system $f(x)$ varies very slowly in $x$, specifically that $\left\| f'(x)^2 \right\| \gg \|f''(x)f(x)\|$. This is often the case with continuous systems that have some notion of smoothness implied in their definition. Assuming the smoothness, we can reiterate the map of the above equation $n$-times until we arrive at time $t$. We then pose that $Y(t)$ can be described by some averaged behavior for sufficiently large $t$ such that

$$Y(t) \approx Y_n = \overbrace{e^{J_{n-1}\Delta t} e^{J_{n-2}\Delta t} \cdots e^{J_0 \Delta t}}^{\text{n-times}} Y_0 = e^{\widetilde{J}t} Y_0, \tag{E.33}$$

where $\widetilde{J}$ is the resultant matrix exponent of the above calculation[3]. Notice that the state $Y(t)$

[3]Remember that $\widetilde{J} \neq J_{n-1} + J_{n-2} + \cdots + J_0$ due to the nature of matrix exponents whereby two matrices $J_a$ and $J_b$ do not generally commute. In fact, any two matrices $J_a$ and $J_b$ will almost never commute in this application. Nevertheless, it is still true that $e^{J_{n-1}} e^{J_{n-2}} \cdots e^{J_0} = e^{n\widetilde{J}}$ because that is how we chose to define it. It is impractical, though possible, to use the Baker-Campbell-Hausdorff formula here so we will avoid it completely.

never enters the equation in a nonlinear sense. The above equation treats the variational system as locally-linear (w.r.t. $Y$), with the values of $J(x)$ provided to the $Y(t)$ from the original system $x(t)$[4]. Empirically, equation (E.32) turns out to hold extremely well locally, for small $\Delta t$, given a numerically integrated trajectory $x(t)$. The exponential of the Jacobian matrix serves as a viable alternative to other numerical integrators. This hints that there is little reason to explicitly integrate the extended system $Y(t)$, described in (E.6). Instead, it is sufficient to simply calculate and store the series of matrices $S_k \equiv e^{J_k \Delta t}$ when calculating the Lyapunov spectrum.

$$Y_n = \prod_{k=0}^{n-1} S_k Y_0 = e^{\widetilde{J}t} Y_0, \tag{E.34}$$

Here, we should note that the intention is just to store the sequence of matrices $S_k$, and no matrix-matrix multiplication should be carried out. Naive multiplication of a large numbers of matrices is numerically unstable. One would notice severe numerical precision issues as the condition number (a measure of numerical rounding error for our purposes here) increases exponentially with each subsequent matrix-matrix multiplication. This is a similar issue that subsection E.4.1 is attempting to resolve with normalization. The workaround here is to perform *successive QR decomposition* on each $S_k$ such that the resulting sequence of matrices are in the same basis [63].

$$S_n \cdots S_1 \overbrace{S_0}^{Q_0 R_0} = S_n \cdots \overbrace{S_1 Q_0}^{Q_1 R_1} R_0 \tag{E.35}$$

We start by decomposing the right-most matrix $S_0$ and repeat the iterative/successive steps of decomposition $S_k Q_{k-1}$ in increasing order. Note that each subsequent $S_k$ needs to be right-multiplied by $Q_{k-1}$ before the QR decomposition is utilized. These steps are repeated until we end up with a sequence of $R_k \equiv R(t_k)$ matrices and one $Q_n$ matrix, shown below. This particular matrix multiplication of $Q_k$ that, though done often, will not propagate bad condition numbers. Such a property is due to orthonormal matrices having a low condition number, usually of order unity.

$$\overbrace{S_{n-1} \cdots S_1 S_0}^{n\text{-times}} = Q_n \overbrace{R_{n-1} \cdots R_1 R_0}^{n\text{-times}} \tag{E.36}$$

[4]Linear systems are traditionally defined to have analytic solutions globally. These, on the other hand, are linear in the sense that they locally obey $Y_{k+1} = e^{J_k \Delta t} Y_k$ extremely well, where $J_k$ is time-varying but not explicitly dependent on time. This result in systems that, in general, have no analytic solutions, but they are not nonlinear either! The general solution to equation (E.32) is $Y(t) = \exp \int_0^t J(x(s))ds$.

The $Q_n$ matrix can be seen as the Lyapunov vectors - directions where the growth or shrinkage occurs – but is otherwise not of interest. The matrices $R_k$ can be used to efficiently and stably calculate the Lyapunov spectrum. We use an averaged version of (E.15), in the same spirit as (E.16), to calculate the global Lyapunov spectrum.

$$\lambda_d = \lim_{n\to\infty} \sum_{k=1}^{n} \frac{1}{t_k} \ln |R_{dd}(t_k)| \tag{E.37}$$

One particularly useful way of interpreting the success of the successive QR decomposition here is that it computes an 'agreed upon' orientation $Q_n$ that growth occurs at every step of the system. The intermediate matrices $R_k$ are then reoriented at every step in such a way that the diagonal element preserve the Lyapunov exponents, shown in the equation below.

$$Q_n R_n \cdots R_1 R_0 = Q_n R_n Q_n^T \cdots Q_n R_1 Q_n^T Q_n R_0 \tag{E.38}$$

It might seem that from (E.37) that there is no respect for the path-ordering in the averaging process. The ordering is actually contained within the construction of $R_k$ in (E.35), where the successive QR decomposition is carried out starting from $R_0$ and moving leftward. To readers familiar with this topic of Lyapunov exponents, this method is identical to how one would find the Lyapunov spectrum of discrete time systems. It is identical except for a quick substitution, so this method is sometimes known as the *discrete QR method*. However, the name is somewhat misleading since this method works for continuous systems that have to be discretized for computation. The matrix exponential $e^{J_k \Delta t}$ serves as the discrete time system's equivalent of the Jacobian. The workhorse in calculating the Lyapunov spectrum is a sequence of QR decompositions, so we opt to call both these methods the **successive QR method** or **sequential QR method**.

This method is conceptually identical to the variational system method covered earlier. One can make the case that the explicit RK schemes are actually attempting to approximate the matrix exponential, so invoking the matrix exponential is technically the more general case. The one advantage is that $Y(t)$ does not need to be explicitly calculated. Instead, the growth of $Y(t)$ is implicitly contained in the $e^{J_k \Delta t}$ term, and we trade the price of computing $Y(t)$ with an explicit time integrator with the price of computing the matrix exponent[5], at least for continuous time systems. In terms of computer memory, both methods use the same amount of memory. For discrete time systems, however, the successive QR method is the clear winner as we avoid many unnecessary calculations.

[5]The matrix exponent calculation generally uses some eigenvalue decomposition algorithm as the back-end

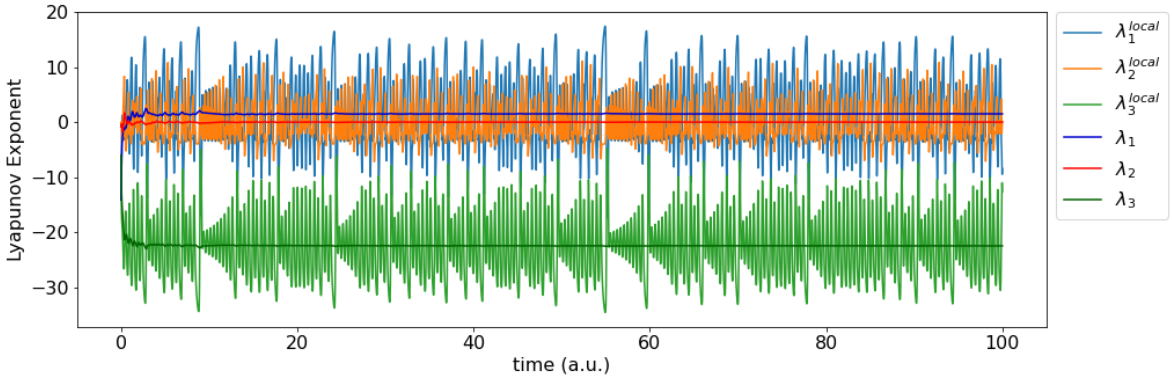


**Figure E.7:** The matrix exponent method with successive QR decomposition, applied to Lorenz 63, estimates the Lyapunov spectrum quite steadily. Refer to Figure E.5 as a direct comparison. The values for the Lyapunov exponents here are $[1.50, 0.00, -22.50]$.

### E.4.4 Comparison of Methods

Our first comparison will be on the computational requirements of the two methods, which is one of the more important and practical questions that one might have when making a comparison. The QR exploit method performs one QR decomposition and two matrix multiplications for each data point included in the calculation. The successive QR method performs one QR decomposition and one matrix multiplication for each data point. There are other operations such as normalization, picking diagonal terms, logarithms, that require an insignificant amount of computational time, so there are not included for our simple comparison. In this respect, the successive QR method has a slight edge in computational speed, but not by much.

The second comparison regarding the computed trajectories of the variational system $Y(t)$. The QR exploit explicitly requires the trajectory of $Y(t)$, so a good integrator and small values of $\Delta t$ is required to generate accurate trajectories of $Y(t)$. The successive QR method does not need explicit trajectories of $Y(t)$ at all, but it uses $\Delta t$ explicitly in the calculations. It also requires a good approximation for $e^{J\Delta t}$, but this is conceptually and computationally equivalent to using a good integrator for $Y(t)$. The moral here is that a good integrator and a small $\Delta t$ is necessary regardless of the method. These were required for the calculation of the state space trajectories $x(t)$ to begin with, and it seems to play an equally pivotal role for the variational system (perhaps unsurprisingly).

---

workhorse. These eigendecomposition algorithm typically take longer than an explicit time integrator such as RK4. However, there exist iterative eigendecomposition algorithms that converge in $\mathcal{O}(N^2)$, which is the same order as the time integrators. Still, the variational system method performs quicker than the matrix exponent method in general if speed is really an issue. There is also the option of using the first order approximation $e^{J_k \Delta t} \simeq \mathbb{1} + J_k \Delta t$, or whatever the chosen order is, to save some computation. This achieved results that just as accurate no matter the approach.

Lastly, we compare the sequential nature of the two calculations. The QR exploit has no respect for ordering, and the successive QR method is completely the opposite. Yet, both seem to be just as accurate holding everything else constant. However, the sequential nature of the successive QR method means that everything needs to be performed in order, thus rendering any notions of parallelization completely moot. The QR exploit method, on the other hand, can be parallelized with little effort once the trajectory of the variational system $Y(t)$ has been calculated. Unfortunately, Lyapunov exponents do not require long histories of trajectories in the first place, which really limits the usefulness of parallelization.

Overall, both these methods are very similar. Both have very similar run times and very similar ease of implementation. The steps and tricks in the QR exploit method is rather involved, and it is not clear exactly what is happening in that calculation except that it

## E.5 Discrete-Time Systems

Discrete systems have distinct features that do not exhibit in their continuous counterparts, but are otherwise nearly identical. To be specific, the discreteness here refers to the nature of time, not a discrete set of states. The most prominent distinction of discrete systems is that chaos can exhibit themselves in dimensions less than three and that system parameters are not required to transition the system into chaos. In fact, discrete maps of dimension one is already capable of exhibiting chaos. This property is attributed to the lack of smoothness in these discrete maps which may generate a sudden 'kick' to the system. Discrete maps are generally described by a forward map, starting from some initial condition $x_0$.

$$x_{k+1} = F(x_k) \tag{E.39}$$

As discussed in section E.4.3, we do not need to use a variational system in order to calculate the Lyapunov spectrum, but the formulation of the variational system for discrete time systems is useful for understanding and interpreting the calculation. Note that below, the index $k$ is for time progression and the $i$ or $j$ are the spatial indices. These will be the augmented dynamics that describes our variational system.

$$Y_{k+1} = L_k Y_k \qquad ; \qquad (L_k)_{ij} \equiv \left.\frac{\partial F_i}{\partial x_j}\right|_{x_k} \qquad ; \qquad Y_0 = \mathbb{1} \tag{E.40}$$

Previously, however, we had the following equations describing our variation system for the continuous time case. These equations are nearly identical and involve the vector field $\dot{x} = f(x)$ rather than the forward map $x_{k+1} = F(x_k)$. The equations for the continuous system repeated here for ease of comparison.

$$Y_{k+1} = e^{J_k \Delta t} Y_k \qquad ; \qquad J_{ij} \equiv \left. \frac{\partial f_i}{\partial x_j} \right|_{x(t)} \qquad ; \qquad Y_0 = \mathbb{1} \tag{E.41}$$

Calculating the Lyapunov spectrum for discrete time systems still involves the same intuition, that is we are looking for the exponential growth of neighboring trajectories within the system. The (normalized) variational system method in E.4.1 and the successive QR method in E.4.3 will both work, except for a simple replacement of with $e^{J_k \Delta t} \to L_k$. Since discrete systems have direct access to the forward map of the system as $Y_{k+1} = L_k Y_k$, we can use successive QR method directly on $L_k$ without using it to generate the trajectories $Y_k$. This renders the variational system method completely redundant for discrete systems. When dealing with discrete time forward maps, it seems that the successive QR method is the only game in town. Due to the time constraints and the limited use case for the author's work, we intentionally do no got into further detail on the discrete time calculation of the Lyapunov spectrum. We also do not delve into the Lyapunov spectrum calculations when only data is available [64].

# F

# Annealed HMC from a KAM Perspective

In this section, we will talk about KAM theory and then some of its implications from applied to HMC and the data assimilation action in our approach. The full treatment of KAM theory is very esoteric and has details that are far beyond the scope of its use for the purposes of this work. Some of the details regarding the inequality bounds of KAM are very technical and almost impenetrable to non-experts (including the author). However, we will be skipping over most of these details of the theory as they are irrelevant to our use case. We will only focus on a small fraction of its implications. As far as our use of KAM theory, it will suffice to summarize our use of the theory in one sentence:

> "Almost all small perturbations to an integrable Hamiltonian dynamical system will result in trajectories with persistent quasi-periodic motion."

There is admittedly a lot of important details that are glossed over in this statement, and we will not even attempt to address them all. It shall suffice to say that the magnitude of the perturbation affects the fraction of trajectories that become quasi-periodic. These quasi-periodic trajectories trace out the entire surface of a fixed torus. In comparison, the trajectories that fail to become quasi-periodic do not trace out a fixed surface. They tend to have a phase space that is not conserved, which means that these other trajectories reach a fix point or diverge. Such trajectories are *destroyed* in the context of KAM theory. We shall pick up where we left off in Chapter 4 with the HMC action including the annealing term $\varepsilon$.

$$\begin{aligned}\mathcal{H}[\boldsymbol{X},\boldsymbol{P}|\boldsymbol{Y}] &= \mathcal{S}[\boldsymbol{X}|\boldsymbol{Y}] + \frac{1}{2}\boldsymbol{P}^\top \boldsymbol{M}^{-1}\boldsymbol{P} \\ &= \underbrace{\frac{1}{2}\|\boldsymbol{X}-\boldsymbol{Y}\|^2_{\boldsymbol{R}_{\text{meas}}} + \frac{1}{2}\left\|\boldsymbol{P}^2\right\|_{\boldsymbol{M}^{-1}}}_{\mathcal{H}_0[\boldsymbol{X},\boldsymbol{P}]} + \underbrace{\frac{1}{2}\left\|L^+\boldsymbol{X} - \boldsymbol{F}(L^-\boldsymbol{X})\right\|^2_{\boldsymbol{R}_{\text{model}}}}_{\varepsilon\mathcal{H}_1[\boldsymbol{X}]}\end{aligned} \tag{F.1}$$

Recall the Hamiltonian that we constructed from the original data assimilation action $\mathcal{S}[\boldsymbol{X}|\boldsymbol{Y}]$ by adding a term akin to kinetic energy. We split this Hamiltonian $\mathcal{H}[\boldsymbol{X},\boldsymbol{P}|\boldsymbol{Y}]$ into two parts, which we call $\mathcal{H}_0[\boldsymbol{X},\boldsymbol{P}]$ and $\varepsilon\mathcal{H}_1[\boldsymbol{X}]$. The $\varepsilon$ factor is artificially extracted from the definition of the norm in order to draw similarities between KAM theory and the annealed HMC action. This dimensionless factor $\varepsilon$ can be viewed as the inverse characteristic length scale of the problem.

$$\mathcal{H}_0[\boldsymbol{X},\boldsymbol{P}] = \frac{1}{2}\|\boldsymbol{X}-\boldsymbol{Y}\|^2_{\boldsymbol{R}_{\text{meas}}} + \frac{1}{2}\left\|\boldsymbol{P}^2\right\|_{\boldsymbol{M}^{-1}} \tag{F.2}$$

$$\varepsilon\mathcal{H}_1[\boldsymbol{X}] = \frac{1}{2}\varepsilon\left\|L^+\boldsymbol{X} - \boldsymbol{F}(L^-\boldsymbol{X})\right\|^2_{\boldsymbol{R}_{\text{model}}} \tag{F.3}$$

The resulting equation of motions is linear if taking only $\mathcal{H}_0[\boldsymbol{X},\boldsymbol{P}]$ into account, which is to say that the trajectories are known for all time once initial conditions are specified. Specifically, these linear dynamics correspond to noisy variations of trajectory $\boldsymbol{X}$ around the measured trajectories $\boldsymbol{Y}$, one of these trajectories matches the 'true' trajectory. Through imposing the nonlinear term $\mathcal{H}_1[\boldsymbol{X}]$, by gradually increasing $\varepsilon$, it becomes more clear which values of $\boldsymbol{X}$ best fits the data $\boldsymbol{Y}$.

$$\dot{\boldsymbol{X}} = \frac{\partial\mathcal{H}}{\partial\boldsymbol{P}} = \frac{\partial\mathcal{H}_0}{\partial\boldsymbol{P}} \tag{F.4}$$

$$\dot{\boldsymbol{P}} = -\frac{\partial\mathcal{H}}{\partial\boldsymbol{X}} = -\frac{\partial\mathcal{H}_0}{\partial\boldsymbol{X}} - \varepsilon\frac{\partial\mathcal{H}_1}{\partial\boldsymbol{X}} \tag{F.5}$$

From the point of view of KAM theory, our annealing technique increases $\varepsilon$, thereby increasing the nonlinear coupling of the system. For values of $\varepsilon$ sufficiently small[1], most of the trajectories are integrable, meaning that they will continue to explore phase space in a stable manner. Not only is this annealing method stable from the KAM perspective, but the theory also states that the resulting trajectories are quasi-periodic hence well-mixing.

[1] We do not provide an estimate in this work.

There seems to be some relationship between annealing and KAM theory that warrants additional exploration. One could plausibly explain the effectiveness of the annealing family of methods, on a very specific set of problems, using KAM theory. At the very least, this provides some evidence as the effectiveness of the annealing method applied to our data assimilation action.

# Bibliography


[1] E. N. Lorenz, "Predictability: A problem partly solved," in *Proc. Seminar on predictability*, vol. 1, 1996.

[2] G. Evensen, *Data assimilation: the ensemble Kalman filter*, vol. 2. Springer, 2009.

[3] K. P. B. Chandra and D.-W. Gu, "Nonlinear filtering," *Cham, Switzerland: Springer*, 2019.

[4] H. Abarbanel, *Predicting the future: completing models of observed complex systems*. Springer, 2013.

[5] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[6] J. Pathak, A. Wikner, R. Fussell, S. Chandra, B. R. Hunt, M. Girvan, and E. Ott, "Hybrid forecasting of chaotic processes: Using machine learning in conjunction with a knowledge-based model," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 28, no. 4, p. 041101, 2018.

[7] E. Kalnay, *Atmospheric modeling, data assimilation and predictability*. Cambridge university press, 2003.

[8] O. Talagrand, "Assimilation of observations, an introduction," *Journal of the Meteorological Society of Japan*, vol. 75, no. 1B, pp. 191–209, 1997.

[9] H. D. Abarbanel, R. Brown, J. J. Sidorowich, and L. S. Tsimring, "The analysis of observed chaotic data in physical systems," *Reviews of modern physics*, vol. 65, no. 4, p. 1331, 1993.

[10] D. Dürr and A. Bach, "The onsager-machlup function as lagrangian for the most probable path of a diffusion process," *Communications in Mathematical Physics*, vol. 60, no. 2, pp. 153–170, 1978.

[11] C. Gardiner, *Stochastic methods*, vol. 4. Springer Berlin, 2009.

[12] R. P. Feynman, A. R. Hibbs, and D. F. Styer, *Quantum mechanics and path integrals*. Courier Corporation, 2010.

[13] J. C. Quinn, *A path integral approach to data assimilation in stochastic nonlinear systems*. University of California San Diego, 2010.

[14] J. Bröcker, "What is the correct cost functional for variational data assimilation?," *Climate Dynamics*, vol. 52, no. 1, pp. 389–399, 2019.

[15] J. M. Restrepo, "A path integral method for data assimilation," *Physica D: Nonlinear Phenomena*, vol. 237, no. 1, pp. 14–27, 2008.

[16] O. Zeitouni and A. Dembo, "A maximum a posteriori estimator for trajectories of diffusion processes," *Stochastics: An International Journal of Probability and Stochastic Processes*, vol. 20, no. 3, pp. 221–246, 1987.

[17] N. Kadakia, D. Rey, J. Ye, and H. D. Abarbanel, "Symplectic structure of statistical variational data assimilation," *Quarterly Journal of the Royal Meteorological Society*, vol. 143, no. 703, pp. 756–771, 2017.

[18] A. Dembo and O. Zeitouni, "Onsager-machlup functionals and maximum a posteriori estimation for a class of non-gaussian random fields," *Journal of multivariate analysis*, vol. 36, no. 2, pp. 243–262, 1991.

[19] G. Evensen, "Sequential data assimilation with a nonlinear quasi-geostrophic model using monte carlo methods to forecast error statistics," *Journal of Geophysical Research: Oceans*, vol. 99, no. C5, pp. 10143–10162, 1994.

[20] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The journal of chemical physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[21] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, vol. 57, pp. 97–109, 04 1970.

[22] A. Gelman, W. R. Gilks, and G. O. Roberts, "Weak convergence and optimal scaling of random walk metropolis algorithms," *The annals of applied probability*, vol. 7, no. 1, pp. 110–120, 1997.

[23] D. Ceperley, "Metropolis methods for quantum monte carlo simulations," in *AIP Conference Proceedings*, vol. 690, pp. 85–98, American Institute of Physics, 2003.

[24] S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth, "Hybrid monte carlo," *Physics letters B*, vol. 195, no. 2, pp. 216–222, 1987.

[25] M. Betancourt, "A conceptual introduction to hamiltonian monte carlo," *arXiv preprint arXiv:1701.02434*, 2017.

[26] D. J. MacKay, D. J. Mac Kay, *et al.*, *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[27] R. M. Neal, *Bayesian learning for neural networks*, vol. 118. Springer Science & Business Media, 2012.

[28] R. M. Neal *et al.*, "Mcmc using hamiltonian dynamics," *Handbook of markov chain monte carlo*, vol. 2, no. 11, p. 2, 2011.

[29] H. Goldstein, C. Poole, and J. Safko, "Classical mechanics," 2002.

[30] B. Leimkuhler and S. Reich, *Simulating hamiltonian dynamics*. Cambridge university press, 2004.

[31] N. Kadakia, "Hybrid monte carlo with chaotic mixing," *arXiv preprint arXiv:1604.07343*, 2016.

[32] H. Yoshida, "Construction of higher order symplectic integrators," *Physics letters A*, vol. 150, no. 5-7, pp. 262–268, 1990.

[33] Z. Fang, A. S. Wong, K. Hao, A. J. Ty, and H. D. Abarbanel, "Precision annealing monte carlo methods for statistical data assimilation and machine learning," *Physical Review Research*, vol. 2, no. 1, p. 013050, 2020.

[34] J. Ye, D. Rey, N. Kadakia, M. Eldridge, U. I. Morone, P. Rozdeba, H. D. Abarbanel, and J. C. Quinn, "Systematic variational method for statistical nonlinear state and parameter estimation," *Physical Review E*, vol. 92, no. 5, p. 052901, 2015.

[35] J. E. Marsden and M. West, "Discrete mechanics and variational integrators," *Acta Numerica*, vol. 10, pp. 357–514, 2001.

[36] D. Rey, M. Eldridge, M. Kostuk, H. D. Abarbanel, J. Schumann-Bischoff, and U. Parlitz, "Accurate state and parameter estimation in nonlinear systems with sparse observations," *Physics Letters A*, vol. 378, no. 11-12, pp. 869–873, 2014.

[37] J. Ye, N. Kadakia, P. J. Rozdeba, H. D. Abarbanel, and J. C. Quinn, "Improved variational methods in statistical data assimilation," *Nonlinear Processes in Geophysics*, vol. 22, no. 2, pp. 205–213, 2015.

[38] W. G. Whartenby, J. C. Quinn, and H. D. Abarbanel, "The number of required observations in data assimilation for a shallow-water flow," *Monthly weather review*, vol. 141, no. 7, pp. 2502–2518, 2013.

[39] J. Pathak, Z. Lu, B. R. Hunt, M. Girvan, and E. Ott, "Using machine learning to replicate chaotic attractors and calculate lyapunov exponents from data," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 27, no. 12, p. 121102, 2017.

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

[41] L. Grigoryeva and J.-P. Ortega, "Echo state networks are universal," *Neural Networks*, vol. 108, pp. 495–508, 2018.

[42] J. Pathak, B. Hunt, M. Girvan, Z. Lu, and E. Ott, "Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach," *Physical review letters*, vol. 120, no. 2, p. 024102, 2018.

[43] J. Huke, "Embedding nonlinear dynamical systems: A guide to takens' theorem," 2006.

[44] T. S. Parker and L. O. Chua, "Chaos: A tutorial for engineers," *Proceedings of the IEEE*, vol. 75, no. 8, pp. 982–1008, 1987.

[45] Z. Lu, J. Pathak, B. Hunt, M. Girvan, R. Brockett, and E. Ott, "Reservoir observers: Model-free inference of unmeasured variables in chaotic systems," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 27, no. 4, p. 041102, 2017.

[46] L. Grigoryeva, A. Hart, and J.-P. Ortega, "Learning strange attractors with reservoir systems," *arXiv preprint arXiv:2108.05024*, 2021.

[47] Z. Lu, B. R. Hunt, and E. Ott, "Attractor reconstruction by machine learning," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 28, no. 6, p. 061104, 2018.

[48] D. J. Gauthier, E. Bollt, A. Griffith, and W. A. Barbosa, "Next generation reservoir computing," *Nature communications*, vol. 12, no. 1, pp. 1–8, 2021.

[49] S. Boyd and L. Chua, "Fading memory and the problem of approximating nonlinear operators with volterra series," *IEEE Transactions on circuits and systems*, vol. 32, no. 11, pp. 1150–1161, 1985.

[50] L. Kocarev and U. Parlitz, "Generalized synchronization, predictability, and equivalence of unidirectionally coupled dynamical systems," *Physical review letters*, vol. 76, no. 11, p. 1816, 1996.

[51] L. Gonon and J.-P. Ortega, "Fading memory echo state networks are universal," *Neural Networks*, vol. 138, pp. 10–13, 2021.

[52] T. Lymburn, D. M. Walker, M. Small, and T. Jüngling, "The reservoir's perspective on generalized synchronization," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 29, no. 9, p. 093133, 2019.

[53] T. L. Carroll and L. M. Pecora, "Network structure effects in reservoir computers," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 29, no. 8, p. 083130, 2019.

[54] D. E. Rumelhart and J. L. McClelland, *Learning Internal Representations by Error Propagation*, pp. 318–362. MIT Press, 1987.

[55] Y. LeCun, D. Touresky, G. Hinton, and T. Sejnowski, "A theoretical framework for back-propagation," in *Proceedings of the 1988 connectionist models summer school*, vol. 1, pp. 21–28, 1988.

[56] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[57] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[58] H. D. Abarbanel, S. Shirman, D. Breen, N. Kadakia, D. Rey, E. Armstrong, and D. Margoliash, "A unifying view of synchronization for data assimilation in complex nonlinear networks," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 27, no. 12, p. 126802, 2017.

[59] L. Dieci, R. D. Russell, and E. S. Van Vleck, "On the compuation of lyapunov exponents for continuous dynamical systems," *SIAM journal on numerical analysis*, vol. 34, no. 1, pp. 402–423, 1997.

[60] Z.-M. Chen, K. Djidjeli, and W. Price, "Computing lyapunov exponents based on the solution expression of the variational system," *Applied Mathematics and Computation*, vol. 174, no. 2, pp. 982–996, 2006.

[61] M. Sandri, "Numerical calculation of lyapunov exponents," *Mathematica Journal*, vol. 6, no. 3, pp. 78–84, 1996.

[62] K. Geist, U. Parlitz, and W. Lauterborn, "Comparison of different methods for computing lyapunov exponents," *Progress of theoretical physics*, vol. 83, no. 5, pp. 875–893, 1990.

[63] H. D. Abarbanel, R. Brown, and M. B. Kennel, "Local lyapunov exponents computed from observed data," *Journal of Nonlinear Science*, vol. 2, no. 3, pp. 343–365, 1992.

[64] A. Wolf, J. B. Swift, H. L. Swinney, and J. A. Vastano, "Determining lyapunov exponents from a time series," *Physica D: nonlinear phenomena*, vol. 16, no. 3, pp. 285–317, 1985.