Jacobian Analysis of a Parameter Aware Reservoir Computer

M.Sc Thesis

By

Namit R Krishna

Discipline of Physics
Indian Institute of Technology Indore

May 21, 2025

Jacobian Analysis of a Parameter Aware Reservoir Computer

A Thesis

Submitted in partial fulfilment of the
requirements for the award of the degree
of
Master of Science

By

Namit R Krishna
2303151022

DISCIPLINE OF PHYSICS
INDIAN INSTITUTE OF TECHNOLOGY INDORE

May 21, 2025

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **Jacobian analysis of a Parameter aware Reservoir Computer** in the partial fulfillment of the requirements for the award of the degree of MASTER OF SCIENCE and submitted in the **DISCIPLINE OF PHYSICS**, **Indian Institute of Technology Indore**, is an authentic record of my own work carried out during the time period from July 2023 to May 2025 under the supervision of Dr. Sarika Jalan, Professor, Indian Institute of Technology Indore.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

21/05/2025

Signature of the student with date
(Namit R Krishna)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

21.05.2025

Signature of the supervisor with date
(Dr. Sarika Jalan)

Namit R Krishna has successfully given his M.Sc. Oral Examination held on 13th May 2025.

Signature of DPGC

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to everyone who supported me throughout the course of this thesis. First and foremost, I am sincerely thankful to my guide, Dr. Sarika Jalan, for her unwavering support, insightful guidance, and constant encouragement. I extend my heartfelt thanks to the members of the Complex Systems Lab at IIT Indore for the stimulating discussions, the shared learning experiences, and the camaraderie that made research both enriching and enjoyable. I am also grateful to my professors at IIT Indore, whose teachings have helped build the foundation on which this work stands. To my friends and peers, thank you for being my sounding board, my motivators, and my companions through the ups and downs of research life. Your presence made the challenging days easier and the good days even better. Lastly, and most importantly, I owe everything to my family. Their unconditional love, patience, and belief in me have been my strongest pillars of support. This journey would not have been possible without them. To everyone who has been part of this journey, directly or indirectly—thank you.

Abstract

This thesis delves into the intersection of non-linear dynamics and network science to gain deeper insights into parameter-aware reservoir computing—a machine learning algorithm renowned for its ability to predict time series data of non-linear dynamical systems at different parameter values. Despite its success, reservoir computing is often regarded as a black box, with its internal mechanisms and theoretical foundations remaining largely unexplored. This work seeks to shed light on how the algorithm processes information and makes predictions, bridging the gap between empirical success and theoretical understanding.

The work conducted thus far focuses on developing a parameter-aware reservoir computing (PARC) model and leveraging it to predict the behaviour of various nonlinear dynamical systems- Logistic map, Higher Order Kuramoto oscillator network and coupled Stuart Landau oscillators. A Bayesian optimisation algorithm has been developed to efficiently determine the optimal hyperparameters for the PARC model to make successful predictions. Additionally, the PARC model has been analysed as a network of interconnected maps. jacobian analysis of the reservoir network is conducted to understand the mechanism utilised by the machine learning model to predict the behaviour of the systems in different parameter regimes, we do this by mapping the bifurcation seen in the RC network to known bifurcations in map networks.

Future work may focus on analysing the behaviour of the PARC network when predicting more complex datasets, such as period-2 to period-4 dynamics, and identifying the underlying network dynamics that enable accurate predictions. Additionally, a better understanding of how the machine learning network works internally can help build a stronger theoretical foundation for creating more efficient algorithms. It can also make it easier to find the best hyperparameters to improve its overall performance.

Table of Contents

List of Figures

Chapter 1

Introduction

Nonlinear dynamical systems lie at the heart of numerous natural and engineered processes, ranging from neuronal signaling and climate dynamics to financial systems and coupled oscillators. These systems are governed by complex, often chaotic behaviors that challenge traditional analytical methods due to their sensitive dependence on initial conditions and intricate parameter landscapes. While classical tools in dynamical systems theory, such as bifurcation analysis and stability criteria, offer foundational insights, they often fall short in handling high-dimensional or data-driven scenarios.

In parallel, machine learning has revolutionised our ability to identify patterns and predict outcomes in systems where explicit models are difficult to construct. Among its diverse approaches, Reservoir Computing (RC) has proven particularly adept at capturing the temporal evolution of chaotic and nonlinear systems, thanks to its ability to transform dynamic inputs into rich internal representations. This architecture enables accurate short-term prediction and reconstruction of complex attractors without requiring full knowledge of the governing equations.

Building on this, Parameter-Aware Reservoir Computing (PARC) introduces a crucial extension: the ability to generalize across different parameter regimes. By embedding system parameters directly into the training process, PARC enables a single model to learn the dynamical structure of a system across a range of behaviors—from periodic to chaotic—while preserving predictive fidelity.

1.1 Motivation

A core motivation for this work stems from the absence of a concrete theoretical framework guiding the construction and analysis of parameter-aware reservoir computing (PARC) models. Despite their empirical success in modeling and predicting the behavior of nonlinear dynamical systems across varying parameter regimes, PARC architectures remain largely heuristic in nature. One of the fundamental challenges lies in the selection and tuning of hyperparameters—such as input scaling, spectral radius, feedback strength, and

reservoir size—that govern the reservoir's ability to retain memory, respond nonlinearly, and generalize across parameters. Currently, these hyperparameters are tuned through trial and error or black-box optimization methods, without a clear understanding of how they influence the internal dynamics of the reservoir during the prediction phase.

To address this gap, this thesis models the predictive stage of a trained PARC network as a system of coupled map equations. This abstraction allows us to treat the reservoir itself as a dynamical system whose evolution reflects the underlying structure of the target system it aims to predict. By applying Jacobian analysis to this reservoir map network, we investigate how different input parameters induce structural and bifurcatory changes in the reservoir's internal dynamics. This analysis reveals the presence of well-known bifurcations—such as saddle-node, period-doubling, and Neimark–Sacker bifurcations—within the reservoir, suggesting that the network undergoes phase transitions that mirror the behavior of the input system. Such insights are not only intellectually compelling, but they also pave the way for a more principled theory of reservoir construction—potentially enabling the design of task-specific reservoirs with built-in dynamical competence and interpretable hyperparameter roles. Ultimately, this work bridges the empirical power of PARC models with dynamical systems theory, offering a path toward more transparent and theoretically grounded machine learning frameworks for modelling complex systems.

1.2 Structure of Thesis

- Chapter 2: introduces the core concepts of nonlinear dynamics, bifurcations, and network-based Jacobian analysis. It also introduces the basics of Machine learning and its different architectures.
- Chapter 3: explores the concept of reservoir computing and its extension, parameter-aware reservoir computing (PARC). It delves into the algorithms that power these models, highlighting their ability to effectively capture and predict the dynamics of complex systems.
- Chapter 4: This chapter details the systems under study, along with the training and testing data derived from solving these systems. It also outlines the bifurcation undergone by these systems.
- Chapter 5: This chapter shows the results of our analysis by plotting the time series, the bifurcation plot of our predicted and actual data of the systems we are studying, along with the eigenvalue analysis
- Chapter 6: Summarizes the work done and also discusses future prospects.
- Appendix: contains the supplementary information for the reader's reference.

Chapter 2

Literature Review

2.1 Non-Linear Dynamics

Dynamical systems describe how a system evolves over time based on a set of mathematical rules. These systems can be classified as linear or nonlinear, depending on whether their governing equations satisfy the condition that the output is directly proportional to the input. Nonlinear dynamical systems do not obey this condition, leading to rich and complex behaviours such as bifurcations, chaos, and self-organization.

Nonlinear dynamics is essential for understanding real-world systems, which often display complex behaviors like chaos, bifurcations, and unpredictability. In chaotic systems small changes in initial conditions can lead to vastly different outcomes, as seen in weather, climate, and financial markets. Nonlinear models are crucial for analyzing phenomena such as turbulence, neural activity, and population dynamics. They also enable effective control in robotics, power grids, and complex networks. In finance, nonlinear dependencies and feedback loops demand advanced modeling for accurate analysis. In the following sections, we explore key concepts like chaos, bifurcations, and their role in machine learning and complex system modeling.

2.2 Mathamatical Foundations

There are two main types of dynamical systems: differential equations called **flows** and iteration maps which are simply called **maps**. Differential equations describe the evolution of systems in continuous time, whereas iterated maps arise in problems where time is discrete. The general framework of both in one dimension is given by:

$$\text{Flows}: \dot{x} = f(x) \quad , \quad \text{Maps}: x_{t+1} = f(x_t)$$

If the function $f(x)$, which governs the evolution of this system over time, exhibits nonlinear behavior—such as $f(x) = x^2$ or $f(x) = 1 - x^3$—then the system is classified as a nonlinear dynamical system, if not it is a linear dynamical system.

2.2.1 Fixed Point

In a dynamical system, a fixed point (or equilibrium point) is a point in the phase space where the system does not change over time. More formally, a fixed point x^* is a state of the system where, if the system starts at x^*, it will remain at $x^*$for all future times.The condition for fixed points is given by :

$$\text{Flows}: \ f(x^*) = 0 \ \ , \ \ \text{Maps}: \ f(x^*) = x^*$$

Fixed points can be classified as stable or unstable based on how the system behaves when it is slightly perturbed away from the fixed point:

- A **stable fixed point** (or attractor) is a point where, if the system is slightly perturbed (moved away from the fixed point), it will return to the fixed point over time. The fixed point is stable if:

$$Maps : |f'(x^*)| < 1, Flows : |f'(x^*)| < 0$$

- An **unstable fixed point** (or repeller) is a point where, if the system is slightly perturbed, it will move away from the fixed point over time. The fixed point is unstable if:

$$Maps : |f'(x^*)| > 1, Flows : |f'(x^*)| > 0$$

- A **saddle point** is a point where stability depends on the direction of perturbation; some directions are stable while others are unstable. The fixed point is a saddle point if:

$$Maps : |f'(x^*)| = 1, Flows : |f'(x^*)| = 0$$

2.2.2 Bifurcations

In the study of nonlinear dynamical systems, **bifurcation** refers to a phenomenon where a small change in the system's control parameter leads to a sudden qualitative change in its behavior. For example,In a pendulum, when a small periodic force is applied, it oscillates smoothly around its stable equilibrium point. However, as the driving force increases, the pendulum's motion becomes more complex, transitioning from regular oscillations to chaotic and unpredictable behaviour. This sudden qualitative change in the system's dynamics due to a small change in the external force is known as bifurcation.

Types of Bifurcations

In this section, we explore some fundamental types of bifurcations that commonly occur in dynamical systems.[3]

Saddle-Node Bifurcation

The saddle-node bifurcation is the basic mechanism by which fixed points are *created* and *destroyed*. As a parameter is varied two fixed points move toward each other ,collide,and mutually annihilate. The prototypical example of the saddle-node bifurcation is given by the first-order system

$$\dot{x} = r + x^2 \tag{2.1}$$

r is a parameter, which may be positive, negative, or zero. When r is negative, there are two fixed points, one stable and one unstable As r approaches 0 from below, the parabola

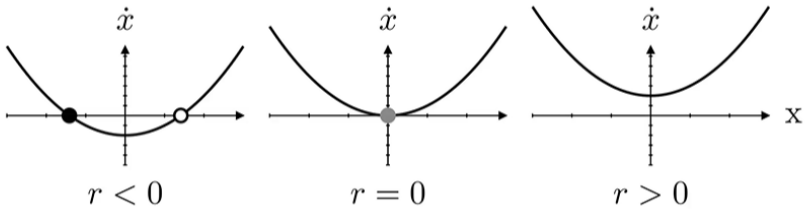
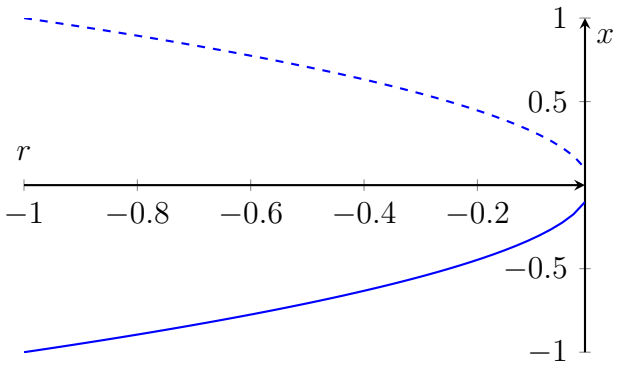


Figure 2.1: Bifurcation diagram illustrating the saddle-node bifurcation. The stable fixed point is represented by a black filled dot, the unstable fixed point by an unfilled dot, and the saddle point by a grey dot. For $r < 0$, two fixed points exist: one stable and one unstable. As $r \to 0$, the fixed points collide and form a single half-stable fixed point at $x^* = 0$. For $r > 0$, the fixed points annihilate, leaving no fixed points.

moves up and the two fixed points move toward each other. When $r = 0$, the fixed points coalesce into a half-stable fixed point at $x^* = 0$. As soon as $r > 0$ it vanishes and now there are no fixed points at all. We could see how the behaviour of the system changes with the control parameter by plotting the bifurcation diagram.Where, the x-axis represents the control parameter (such as the growth rate in the logistic map), while the y-axis represents the asymptotic behavior of the system's state variable.



Transcritical Bifurcation

In certain scientific scenarios, a fixed point is guaranteed to exist for all values of a parameter and can never be annihilated. However, as the parameter varies, the *stability of this fixed point can change*. This phenomenon, where there is an exchange of stability between two fixed points, is known as a transcritical bifurcation. The normal form of a transcritical bifurcation is given by the equation

$$\dot{x} = rx - x^2 \tag{2.2}$$

For $r < 0$, there is an unstable fixed point at $x^* = r$ and a stable fixed point at $x^* = 0$. As r increases, the unstable fixed point approaches the origin and coalesces with it when $r = 0$. Finally, when $r < 0$, the origin has become unstable, and $x^* = r$ is now stable.

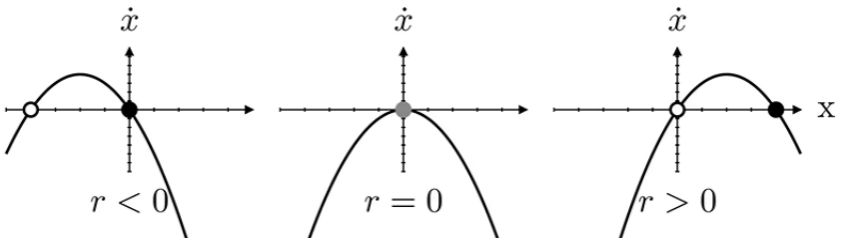
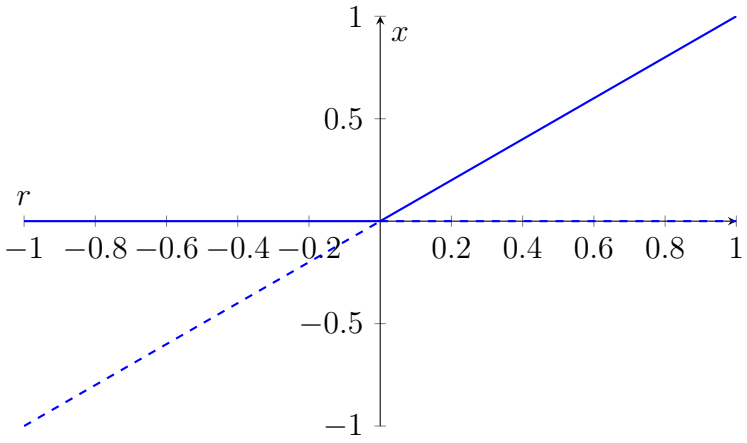


Figure 2.2: Bifurcation diagram illustrating the transcritical bifurcation. As the parameter r is varied, the two fixed points collide and exchange stability at $r = 0$. For $r < 0$, the stable fixed point lies above the unstable one. As $r \to 0$, the fixed points intersect, forming a half-stable fixed point at $x^* = 0$. For $r > 0$, the stability of the fixed points is exchanged, with the previously unstable fixed point becoming stable and vice versa.



Above there is a visualization of a transcritical bifurcation, where two fixed points collide and exchange stability as the control parameter is varied.

Pitchfork Bifurcation

This bifurcation is common in physical problems that have a *symmetry*. A pitchfork bifurcation is a type of bifurcation in dynamical systems where a single fixed point splits into multiple fixed points as a control parameter is varied. It is named "pitchfork" because the bifurcation diagram resembles the shape of a pitchfork. There are two types of pitchfork bifurcation - Supercritical and Subcritical pitchfork bifurcation

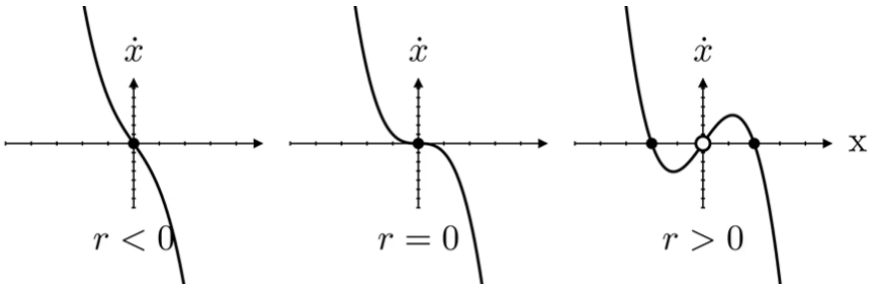
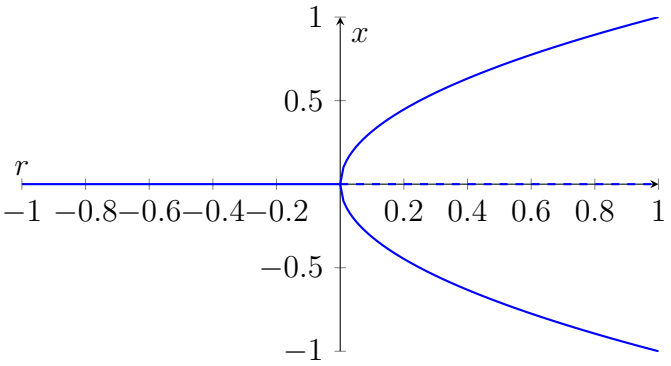


Figure 2.3: Bifurcation diagram illustrating the Supercritical pitchfork bifurcation. For $r < 0$, a single stable fixed point exists at $x^* = 0$. As $r \to 0$, the fixed point loses stability. For $r > 0$, the system exhibits symmetry breaking with two stable fixed points and one unstable fixed point at the origin. For subcritical pitchfork bifurcation the system will follow an opposite flow where two unstable fixed points will come and join and give a single unstable point.

1. **Supercritical Pitchfork Bifurcation**

Occurs when a stable fixed point loses stability and gives rise to two new stable fixed points while the original fixed point becomes *unstable*. The normal form of this bifurcation is

$$\dot{x} = rx - x^3 \tag{2.3}$$

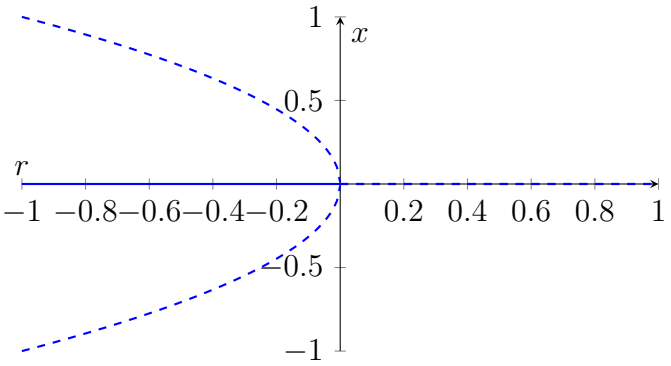


When $r < 0$, the origin is the only fixed point, and it is stable. When $r = 0$, the origin is still stable, but much more weakly so, since the linearization vanishes. Finally, when$r > 0$, the origin has become unstable. Two new stable fixed points appear on either side of the origin, symmetrically located at $x^* = \pm\sqrt{r}$.

2. **Subcritical Pitchfork Bifurcation**

Occurs when an unstable fixed point gives rise to two new unstable fixed points, while the original fixed point remains stable. The normal form of this bifurcation is

$$\dot{x} = rx + x^3 \tag{2.4}$$



The nonzero fixed points $x^* = \pm\sqrt{-r}$ are unstable and exist only for $r < 0$, which is why this bifurcation is termed **subcritical**. More importantly, the origin is stable for $r < 0$ and becomes unstable for $r > 0$.

Period Doubling Bifurcation

As a control parameter r is varied, the system undergoes a transition where a stable fixed point loses its stability, and a new periodic orbit of period $2T$ emerges, where T is the original period. With a further increase in the parameter, the new periodic orbit can also become unstable, and a period-4 orbit emerges, followed by a period-8 orbit, and so on.This sequence of period doubling events eventually leads to chaos (we would learn more about this in further sections).

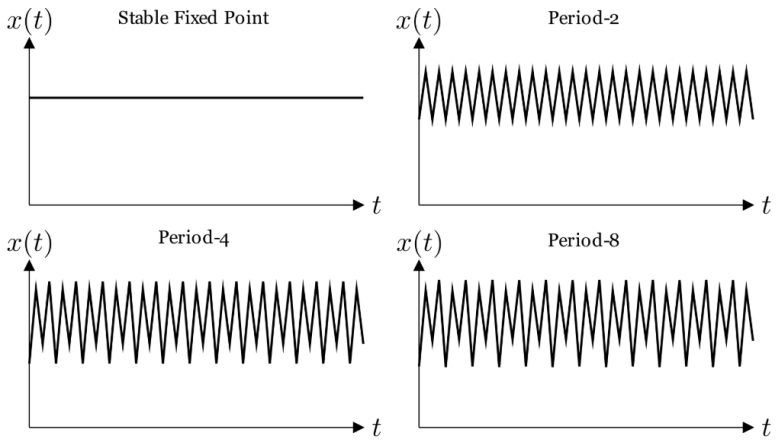


Figure 2.4: Time series plot illustrating period-doubling bifurcation in the logistic map. The trajectory transitions from a stable fixed point to period-2, period-4, and period-8 orbits as the parameter r increases.

If we consider a discrete map, such as the logistic map:

$$x_{n+1} = rx_n(1 - x_n) \tag{2.5}$$

For low values of r, the system converges to a fixed point.At a critical value r_c, the fixed point becomes unstable, and a stable period-2 orbit emerges.As r increases, the period keeps doubling eventually leading to chaos.The bifurcation diagram for the same is given below.

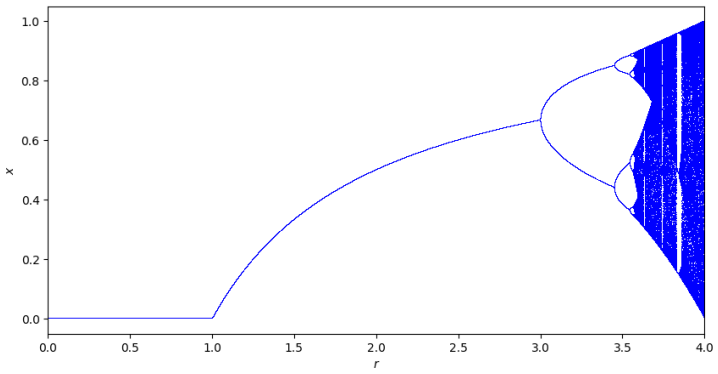


Figure 2.5: Bifurcation diagram of the logistic map showing the transition from a stable fixed point to periodic orbits through successive period doubling, eventually leading to chaotic behaviour,as r increases

Neimark-Sacker Bifurcation:

A Neimark-Sacker bifurcation is a type of bifurcation in discrete dynamical systems that leads to a change in the stability of a fixed point and causes the system to transition from a stable fixed point to a quasi-periodic behavior on a torus. This bifurcation is the discrete-time counterpart of the Hopf bifurcation, which occurs in continuous systems. Let's consider the following 2D discrete system as an example to illustrate the Neimark-Sacker bifurcation:

$$x_{n+1} = (r + \alpha y_n)x_n - \beta y_n, \tag{2.6}$$
$$y_{n+1} = (r + \alpha x_n)y_n + \beta x_n, \tag{2.7}$$

where r, α, β are parameters.

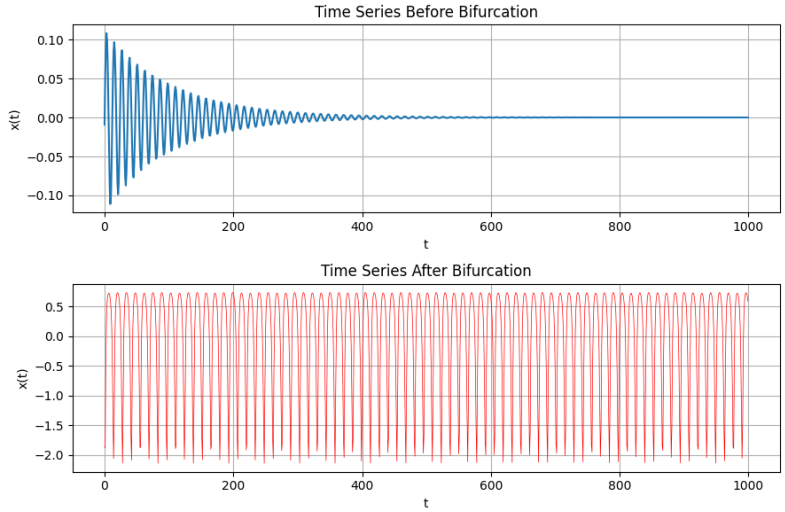


Figure 2.6: Time series plot illustrating the Neimark-Sacker bifurcation. The top panel shows the time series settling to a stable fixed point behavior before the bifurcation, while the bottom panel shows the emergence of quasiperiodic oscillations after the bifurcation.

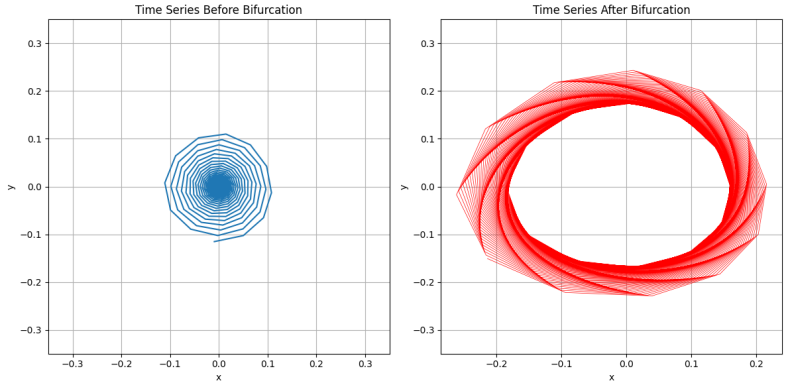


Figure 2.7: Here, we present the phase space diagram of the 2D discrete system to illustrate its behavior before and after the Neimark-Sacker bifurcation. Figure (a) depicts the system before the bifurcation, where trajectories converge to a stable fixed point after a transient phase. Figure (b) illustrates the system after the bifurcation, where it no longer settles into a fixed point but instead evolves into a quasi-periodic orbit, forming a torus in phase space.

2.2.3 Chaos Theory

Chaos theory is a branch of mathematics that studies the behaviour of dynamical systems that are highly sensitive to initial conditions. This phenomenon is famously known as the butterfly effect, where small changes in the initial state can lead to drastically different outcomes over time.

In nonlinear dynamical systems, chaos emerges when the system becomes unpredictable despite being governed by deterministic rules. This unpredictability arises due to the system's extreme sensitivity to small perturbations, making long-term forecasting impossible.

Key Features of Chaos

- **Sensitive Dependence on Initial Conditions:** Two trajectories that start very close to each other will eventually diverge exponentially.
- **Nonlinearity:** The system's equations are nonlinear, meaning the output is not proportional to the input.
- **Aperiodicity:** The system never settles into a fixed point or a periodic orbit.

A chaotic attractor is a set of points toward which a nonlinear dynamical system evolves over time, even though it is under chaotic conditions. In such systems, the trajectory never settles to a fixed point or a periodic orbit but instead follows a complex, non-repeating path within a bounded region of phase space. This path is known as a strange attractor, which exhibits fractal geometry, meaning it has self-similar patterns at different scales.

Example of a Chaotic Attractor: The Lorenz Attractor

The Lorenz system, derived from equations modeling atmospheric convection, is a famous example of a chaotic attractor. It is governed by the following set of nonlinear differential equations:

$$\frac{dx}{dt} = \sigma(y - x) \tag{2.8}$$

$$\frac{dy}{dt} = x(\rho - z) - y \tag{2.9}$$

$$\frac{dz}{dt} = xy - \beta z \tag{2.10}$$

where:

- x, y, z represent different physical quantities (such as fluid flow).
- σ, ρ, and β are system parameters.

For certain values of σ, ρ, and β, the system exhibits **chaotic behavior**. The trajectory never settles into a fixed point or periodic orbit but remains confined within a butterfly-shaped region in phase space, known as the **Lorenz attractor**.

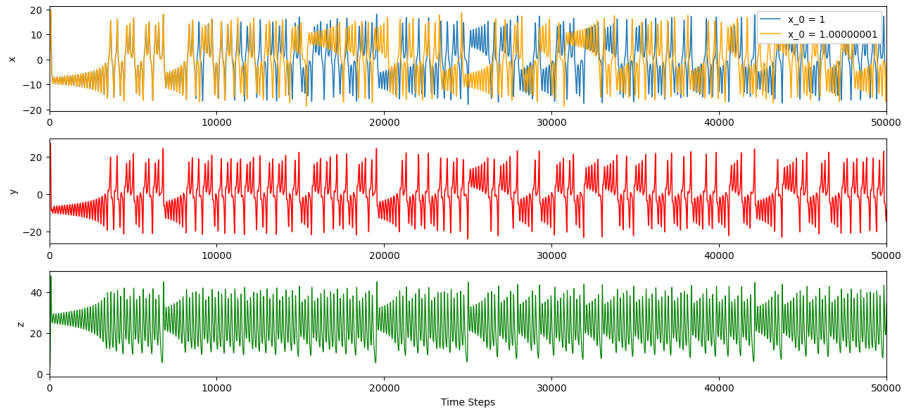


Figure 2.8: Time evolution of x, y, and z variables, where the x data is plotted for two different initial conditions, highlighting the system's sensitivity to initial conditions.

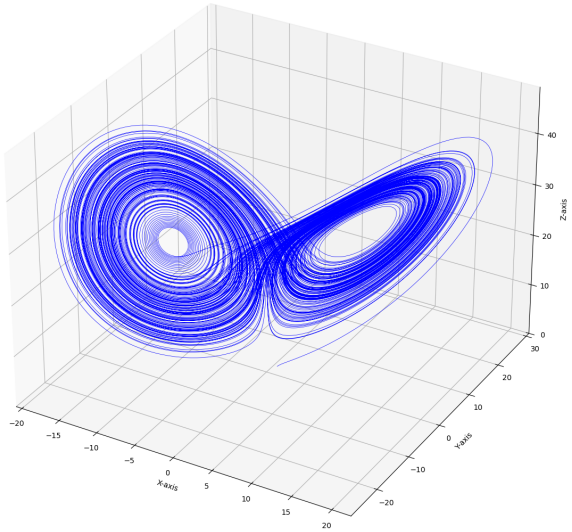


Figure 2.9: The Lorenz Attractor visualized in a 3D phase space, demonstrating chaotic dynamics with trajectories that never intersect but follow a structured, butterfly-shaped pattern.

2.3 Networks

In Complex system science, a network is a mathematical representation of a complex system, where elements (nodes) interact through relationships (edges). Networks help analyze how components of a system are connected, how information or influence spreads, and how the overall system behaves under different conditions. A network consists of:

- Nodes (Vertices): Representing individual components, such as people in a social network, stocks in a financial network, or neurons in a brain network.
- Edges (Links): Representing interactions between components, such as friendships in social networks, transactions in financial systems, or synaptic connections in brain

networks.

The common way of representing a network is by using an adjacency matrix where its element (i, j) is one if the i^{th} and j^{th} nodes are connected otherwise it is zero.

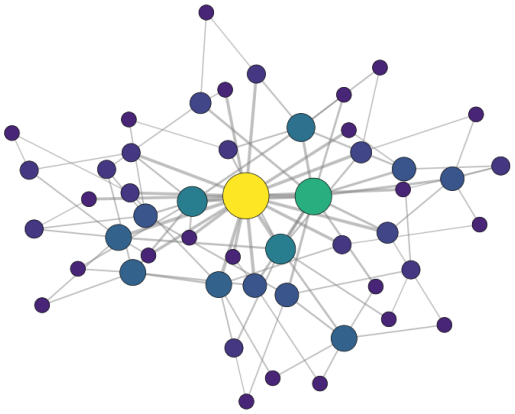


Figure 2.10: A simple network representation where the edges are shown by circles and the edge between two nodes is shown by a line connecting them. The colour and size of the nodes represent how strongly it is interconnected.

Complex systems science helps understand systems with many interacting parts, such as financial markets, brain networks, and social dynamics. It is used to study chaos, bifurcation, and emergent behavior in physics, predict market crashes in finance, model epidemic spread in biology, and analyze complex networks like the internet or power grids. Some useful network topologies are:

1. **Erdős–Rényi (ER) Random Network**:a type of random graph where edges between nodes are assigned randomly with a fixed probability.

2. **Multiplex and Multilayer Networks**:These networks have multiple types of interactions between the same set of nodes or across different layers.

2.3.1 Coupled Dynamics on Networks

A dynamical network is a system where the nodes (individual units) interact with each other based on specific dynamical rules, often described by differential or difference equations. These networks are widely used to model real-world systems where individual components evolve over time while influencing one another.

A network of Map Equations

A map network equation represents the evolution of a networked system where nodes interact according to discrete-time dynamics. The form of the equation depends on the

type of network and interactions. A common framework for networked dynamical systems is a coupled map lattice, where each node follows a discrete-time map and interacts with neighbors. A general form is:

$$x_i(t+1) = f(x_i(t)) + \epsilon \sum_{j \in K_i} A_{ij} g(x_j(t)) \tag{2.11}$$

- $x_i(t)$ is the state of node i at time t,
- $f(x)$ is a local map,
- $g(x)$ is a interaction function,
- A_{ij} is the adjacency matrix of the network(connectivity structure like ER,BA etc),
- K_i is the set of neighbours to node i,
- ϵ is the coupling strength.

2.3.2 Jacobian Analysis

Jacobian analysis provides crucial insights into the key dynamics of networks by examining how their behavior in phase space evolves with respect to parameter variations. By analyzing the eigenvalues of the Jacobian matrix, we can determine the stability of equilibrium points, detect transitions between different dynamical regimes, and identify the nature of bifurcations occurring in the system. The way these eigenvalues change helps us understand whether a system undergoes a saddle-node bifurcation, a Hopf bifurcation, or other critical transitions, revealing deeper insights into the underlying nonlinear dynamics of the network.[3]

To understand this further, we compute the fixed point x^* of Eq (2.11) and then evaluate the Jacobian matrix $\mathcal{J}$ at x^*:

$$\mathcal{J} = \begin{bmatrix} \frac{dx_1[t+1]}{dx_1[t]} & \frac{dx_1[t+1]}{dx_2[t]} & \cdots & \frac{dx_1[t+1]}{dx_m[t]} \\ \frac{dx_2[t+1]}{dx_1[t]} & \frac{dx_2[t+1]}{dx_2[t]} & \cdots & \frac{dx_2[t+1]}{dx_m[t]} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{dx_m[t+1]}{dx_1[t]} & \frac{dx_m[t+1]}{dx_2[t]} & \cdots & \frac{dx_m[t+1]}{dx_m[t]} \end{bmatrix}_{(x_1^*, x_2^*, \cdots, x_m^*)} \tag{2.12}$$

we get the eigenvalues as $[\lambda_1, \lambda_2, \cdots, \lambda_m]$. We then analyse how the eigenvalues of $\mathcal{J}$ vary with changes in some parameter value say ϵ.In the context of a coupled map (or any discrete dynamical system), the behaviour of the eigenvalues tells us what kind of bifurcation occurs to the fixed points.

2.3.3 Bifurcation associated with different eigenvalue behaviour

We plot all the eigenvalues of the Jacobian matrix in the complex plane with a unit circle as shown in the figure. Most of the eigenvalues will be centred around the origin and their behaviour is not important for the analysis [11]

Saddle-Node Bifurcation : If one of the eigenvalues has a magnitude greater than one when we vary the parameter value then we could say a saddle node bifurcation has taken place. Visually this could be seen by a single eigenvalue crossing the unit circle along the positive side of the real axis as in fig 2.12.

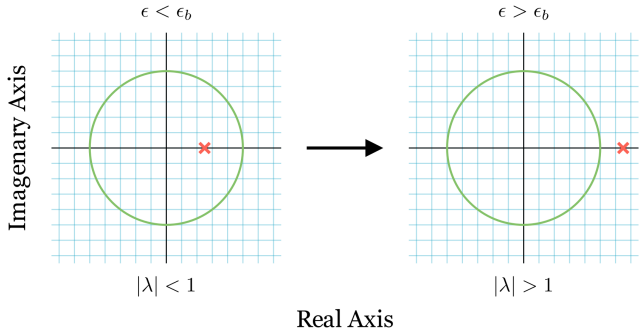


Figure 2.11: Here the cross represents the main eigenvalue whose value varies wrt to the parameter value and is plotted in the complex plane. We see when the parameter value exceeds the bifurcation point ϵ_b it crosses the unit circle along the positive side of the real axis.

Period-doubling Bifurcation: If one of the eigenvalues has a magnitude greater than one when we vary the parameter value and this crossing occurs along the negative side of the real axis then the network goes from a stable fixed point to period-doubling behaviour.

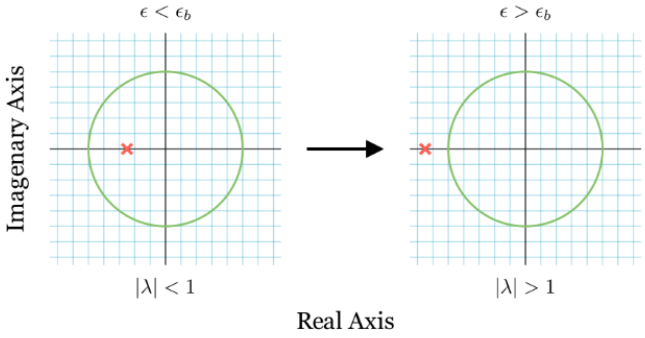


Figure 2.12: Here the cross represents the main eigenvalue whose value varies wrt to the bifurcation parameter value and is plotted in the complex plane. We see when the parameter value exceeds the bifurcation point ϵ_b it crosses the unit circle along the negative side of the real axis.

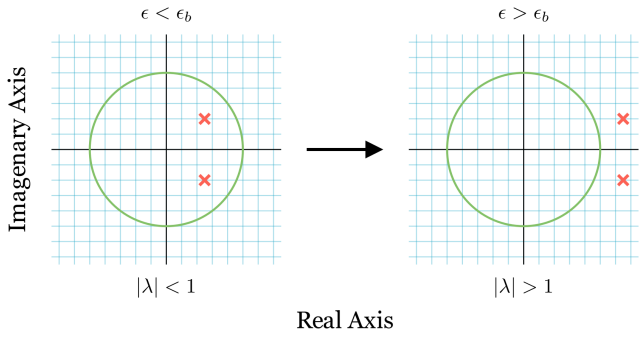


Figure 2.13: Here, the two main eigenvalues that are complex conjugates and whose values vary wrt to the bifurcation parameter are plotted.We see when the parameter value exceeds the bifurcation point ϵ_b they cross the unit circle .

Neimark-Sacker Bifurcation: If a pair of complex conjugate eigenvalues cross the unit circle i.e. their magnitude becomes greater than one then the network behaviour goes from a stable fixed point to oscillating in a complex way. The time series wiggles in a complex way, but without settling into a repeating cycle. The values never return to the same exact point, but they stay within a certain range.

2.4 Introduction to Machine Learning

Machine Learning is a branch of artificial intelligence (AI) that allows computers to learn patterns from data and make predictions or decisions without being explicitly programmed. In simple terms, instead of writing rules manually, the system learns from data and improves its performance over time.

In the study of nonlinear dynamical systems, traditional analytical methods often fail due to the complexity and chaotic nature of the system. Machine learning provides a powerful alternative to model, predict, and analyze such systems as it can handle complex, high-dimensional, and nonlinear data, which traditional mathematical models struggle with.

Main Steps in the Machine Learning Process

In machine learning, the goal is to learn a function that maps input data X to output y, capturing the underlying patterns.We aim to learn a function $f(X, \theta)$ such that:

$$y = f(X, \theta) \tag{2.13}$$

where X is the input data, θ are the model parameters (e.g., weights), and f is the function learned during training [10].Training involves finding optimal θ that minimize

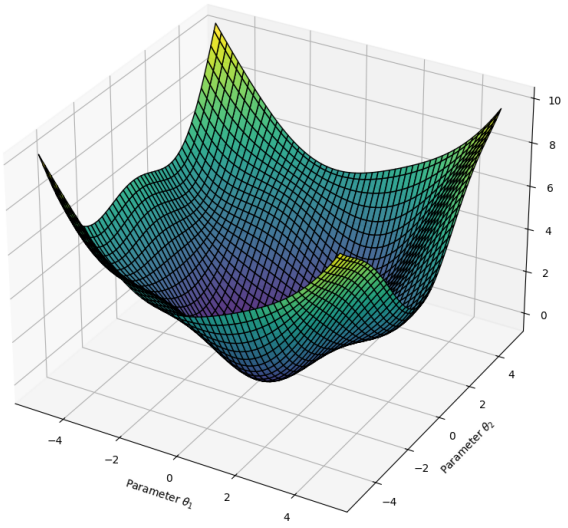


Figure 2.14: A 3D visualisation of a loss function landscape in machine learning. While this plot depicts a loss function dependent on only two parameters, real-world models often involve hundreds or even thousands of parameters. The surface illustrates how different parameter values influence the loss, highlighting the challenges of optimization in complex, high-dimensional, and non-convex spaces.

the difference between predicted and actual outputs.The error is quantified using a loss function, typically:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} (y_i - f(X_i, \theta))^2 \tag{2.14}$$

where N is the number of data points.To minimize the loss, we update parameters using:

$$\theta \leftarrow \theta - \eta \nabla_\theta L(\theta) \tag{2.15}$$

with η as the learning rate and $\nabla_\theta L(\theta)$ the gradient of the loss.

Gradient Descent begins by initializing the parameters θ randomly. The model then computes the loss between its predictions and the actual outputs, along with the gradient of the loss with respect to θ. Using this gradient, the parameters are updated in the direction that reduces the loss. This process is repeated iteratively until the model converges to a set of parameters that minimize the error.

6. Intuition

Gradient Descent is like descending a hill: the model follows the steepest path down the loss landscape to reach the lowest error.

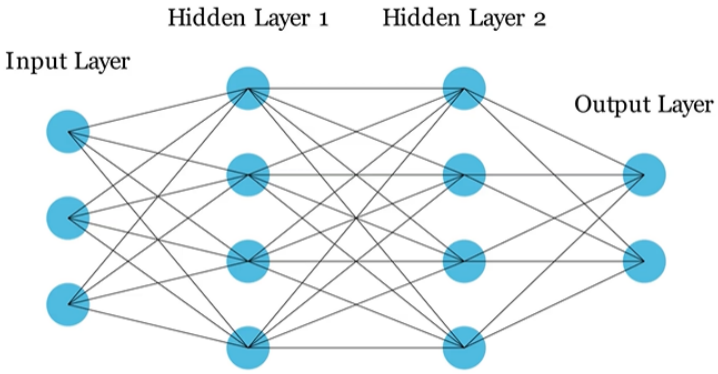


Figure 2.15: This figure illustrates a fully connected neural network with an input layer, two hidden layers, and an output layer. Each node (neuron) processes information and passes it through weighted connections to the next layer, representing how deep learning models learn complex patterns from data.

2.5 Neural Networks (NNs)

Neural networks are computational models inspired by the structure and function of biological neural networks. They consist of layers of interconnected artificial neurons that process and transform input data to learn complex relationships.A typical feedforward neural network consists of three main types of layers:

- **Input Layer**: This layer receives raw data as input. Each neuron in this layer represents a feature of the input.
- **Hidden Layers**: These layers perform non-linear transformations on the input using learned weights and activation functions. Each neuron computes a weighted sum of inputs and applies an activation function to introduce non-linearity.
- **Output Layer**: The final layer produces predictions. For classification tasks, the output neurons often use the softmax activation function, while for regression tasks, they may use a linear activation function.

Mathamatical Foundation

Each neuron computes an activation based on its inputs[10]:

$$z_i = \sum_j w_{ij} x_j + b_i \tag{2.16}$$

$$a_i = \sigma(z_i) \tag{2.17}$$

wherein a neural network, the inputs to a neuron are denoted as x_j, which are multiplied by corresponding weights w_{ij}. Additionally, each neuron has an associated bias term b_i, which allows the model to learn shifts in the data distribution. The weighted sum of inputs and bias gives the pre-activation value z_i, which is then passed through an activation function $\sigma(z)$ to introduce non-linearity into the network, enabling it to learn complex patterns in the data.

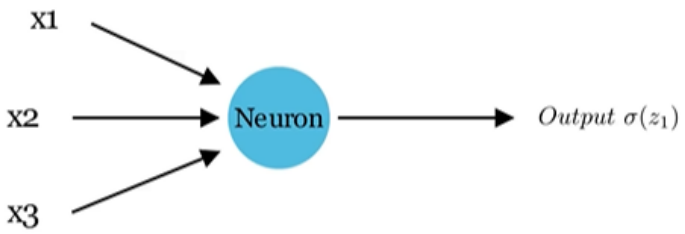


Figure 2.16: A visual representation of a single neuron in a neural network. The neuron receives three weighted inputs (x_1, x_2, x_3) with corresponding weights (w_1, w_2, w_3). The weighted sum is processed by the neuron, generating an output. The diagram illustrates the fundamental computational unit of artificial neural networks.

2.6 Recurrent Neural Networks (RNNs)

While feedforward neural networks work well for static data, they do not consider the temporal dependencies present in sequential data (such as time series, speech, or text). Recurrent Neural Networks (RNNs) address this limitation by incorporating feedback connections that allow information to persist across time steps.

Key Idea of RNNs

Unlike feedforward networks, RNNs have a hidden state that acts as memory, enabling them to capture dependencies across sequences. At each time step t, the hidden state is updated based on the previous hidden state and the current input:

$$h_t = f(W_h h_{t-1} + W_x x_t + b) \tag{2.18}$$

The hidden state at time t, denoted as h_t, represents the memory of the network, while the input at time t is given by x_t. The weight matrices W_h and W_x determine how the previous hidden state and the current input influence the new hidden state. Additionally, b is the bias term, and the activation function f, typically chosen as tanh or ReLU, introduces non-linearity to the transformation. The output is then computed as:

$$y_t = g(W_y h_t + b_y) \tag{2.19}$$

where,W_y and b_y are the output weights and biases.

Challenges of RNNs

- **Vanishing Gradient Problem:** During backpropagation, gradients can become very small, making it difficult to train long-range dependencies.
- **Exploding Gradient Problem:** In some cases, gradients grow exponentially, leading to unstable training.
- **Difficulty in Learning Long-Term Dependencies:** Standard RNNs struggle with capturing relationships over long sequences.

To overcome these issues, architectures like Long Short-Term Memory (LSTM) , Gated Recurrent Units (GRUs) and Reservoir Computing were introduced, which use gating mechanisms to selectively retain or forget information.

Chapter 3

Parameter Aware Reservoir Computing

3.1 Reservoir Computing

Reservoir Computing (RC) is a machine learning paradigm designed to efficiently process and predict time series data, particularly for complex, nonlinear, and chaotic systems. It is a form of Recurrent Neural Network (RNN) but differs from traditional deep learning approaches in how it processes information. The key idea behind RC is to use a fixed, randomly initialized reservoir that maps input data into a high-dimensional space, where patterns and dependencies become more easily recognizable. A simple linear readout layer is then trained to make predictions [2].

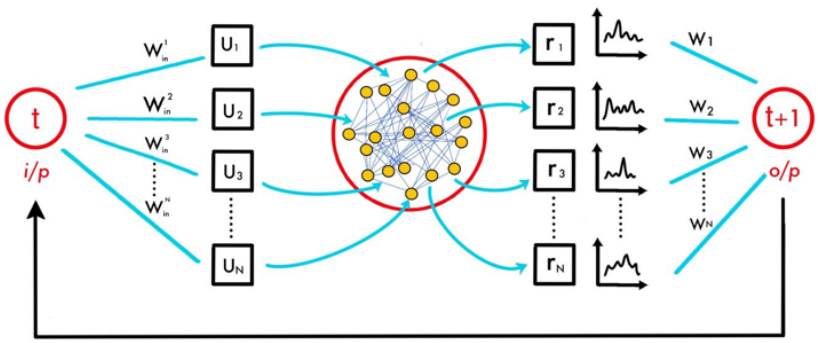


Figure 3.1: Schematic representation of a Reservoir Computing network. The input signal is mapped into a high-dimensional space through a fixed input layer, which connects to a dynamic reservoir of recurrent nonlinear nodes. The reservoir processes temporal dependencies and transforms the input into a complex state representation. A readout layer, typically a simple linear regression model, is trained to extract relevant features and produce the final output, enabling efficient time series prediction and pattern recognition.

Reservoir computing simplifies the training of recurrent networks by fixing the recurrent connections and training only the output weights. This reduces computational

complexity while still capturing temporal dynamics effectively. The reservoir computing model works because of three main reasons:

- Nonlinearity: Enables the system to capture complex, nonlinear patterns in the input that linear models cannot.
- Memory:Allows the reservoir to retain information about past inputs, making it suitable for time-dependent tasks like time series prediction.
- High-Dimensional Embedding:Transforms input into a rich feature space where patterns become linearly separable, enabling simple readout layers to perform well.

3.1.1 Algorithm

- A reservoir computing machine projects an n-dimensional input channel $u(t)$ into a higher m-dimensional space through an input weight matrix W_{in}. The reservoir network's structure is represented by an adjacency matrix A of size $m \times m$, which introduces memory effects into the reservoir state. Both W_{in} and A are chosen at initialization and remain fixed throughout the training process. The entries of W_{in} are sampled from a uniform random distribution over the range $[-b, b]$. Commonly, A is generated as the adjacency matrix of an Erdős-Rényi network with a connection probability σ and scaled such that its spectral radius is ρ, achieved by adjusting A so its largest eigenvalue equals ρ.
- The reservoir state $r(t)$ is updated for N_t time steps using the following equation:

$$r[i+1] = (1-\alpha)r[i] + \alpha \tanh(Ar[i] + W_{in}u[i+1]), \tag{3.1}$$

  The parameter α, known as the leakage rate, balances the influence of the previous reservoir state and the current input state in determining the updated reservoir state.
- During training, the objective is to minimize the error between the predicted output and the actual target output. The reservoir states $r(t)$ are stored and stacked to form a matrix R of dimensions $m \times N_t$. To introduce asymmetry in the system and enhance training, only the odd-numbered rows of R are squared. The actual target output is also stacked to form a matrix U of dimensions $n \times N_t$. The output weight matrix W_{out} is computed using Tikhonov regularization:

$$W_{out} = UR^T(RR^T + \beta I)^{-1}, \tag{3.2}$$

  where β is a regularisation parameter.

- In the testing phase, the output of the current state serves as the input for the next time step. The output $v(t)$ at each time step is then given by

$$v(t) = W_{out} r(t). \tag{3.3}$$

- Bayesian optimization is employed to efficiently tune the hyperparameters of the Reservoir Computing (RC) model. By modeling the performance landscape as a probabilistic function, it balances exploration and exploitation to find optimal parameter combinations—such as spectral radius, input scaling, and regularization strength—that minimize prediction error. This approach significantly reduces the computational cost compared to grid or random search while improving model performance.

3.1.2 Predicting using Reservoir Computing

In this subsection, we demonstrate the capability of the Reservoir Computing model in predicting chaotic time series generated from the Lorenz system. While the model reliably captures the dynamics for up to 400 time steps, it successfully preserves the qualitative behaviour of the chaotic system throughout the prediction horizon.

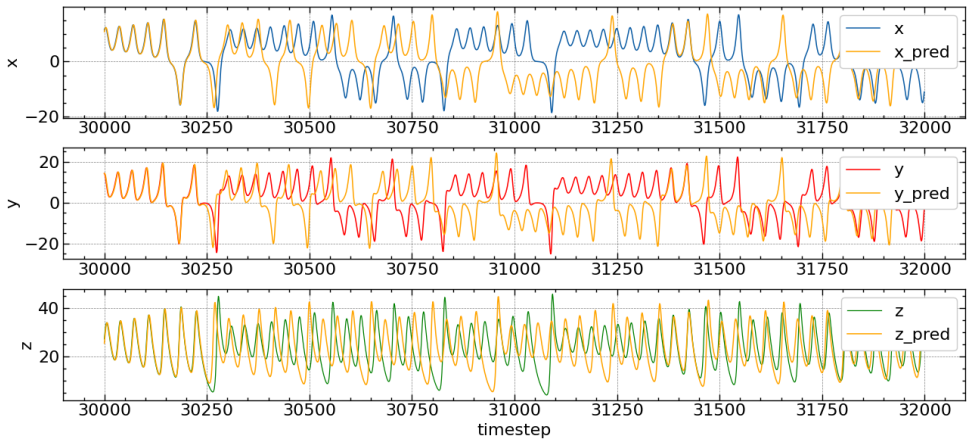


Figure 3.2: Comparison of the actual and predicted time series of the Lorenz system using the Reservoir Computing model. The model closely follows the true trajectory up to approximately 400 time steps, beyond which the predicted values begin to diverge due to the system's inherent chaotic nature.

To further evaluate the model's performance, we plot the phase space trajectories of both the predicted and true time series. This visual comparison provides intuitive insight into how closely the model replicates the underlying dynamics of the chaotic system. By examining the reconstructed attractor, we can assess not only the short-term accuracy but also the extent to which the model captures the long-term qualitative behavior of the Lorenz system.

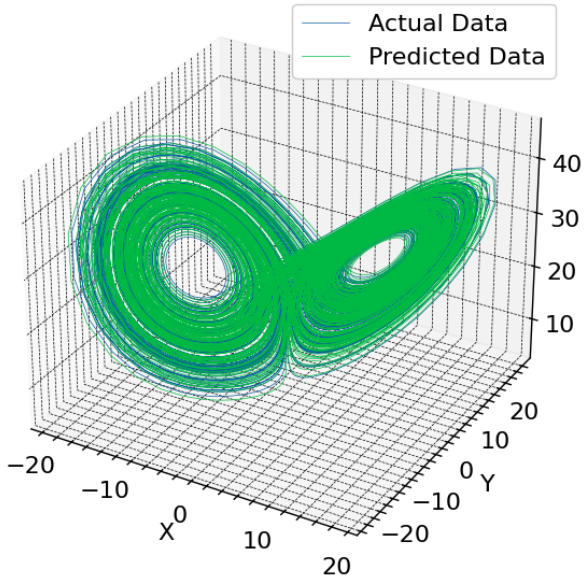


Figure 3.3: Phase space comparison between the predicted trajectory from the Reservoir Computing model and the actual trajectory of the Lorenz system. The close overlap in the initial region highlights the model's ability to capture the system's underlying dynamics, while deviations in later time steps reflect the sensitivity of chaotic systems to initial conditions.

Some other interesting Reservoir Computing architectures involve the single-node RC [5] and the next-gen RC [4], but we will focus on the Parameter-aware architecture.

3.2 Parameter-Aware RC

Parameter-Aware Reservoir Computing Network (PARC) is an advanced variant of the reservoir computing framework that enhances the traditional reservoir computing model by incorporating the dynamics of the input parameters into the reservoir's architecture. By incorporating the system parameter as an additional input, it has been demonstrated that a PARC can be trained for a set of different values of the bifurcation parameter before the critical transition and correctly predict the system's behaviour at a different parameter value.

$$r[i+1] = (1-\alpha)r[i] + \alpha tanh(Ar[i] + W_{in}u[i+1] + k_b W_b(\epsilon - \epsilon_b)) \tag{3.4}$$

Here, ϵ represents the system parameter, while K_b is a constant that determines the strength of its influence. where an additional input channel provides information about the system's bifurcation parameter ϵ, both k_b and ϵ_0 are hyperparameters. The matrix W_b is the associated weight matrix that projects this parameter into all reservoir states and is initialised with values from a uniform distribution over $[-b, b]$ and remains fixed during training.[1]

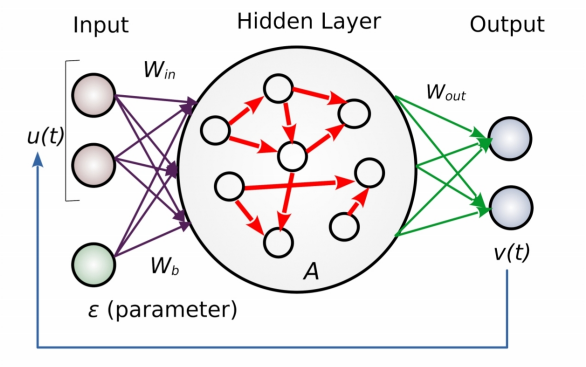


Figure 3.4: Schematic diagram of parameter-aware architecture having an additional input parameter channel ε. W_{in}, W_b are the weight matrices for input u and ε, respectively. A is the adjacency matrix of the reservoir. W_{out} is determined after training and during the testing phase, current output serves as the input for the next time step.

3.2.1 Jacobian Analysis of the PARC Map Network

In the testing phase, the current state's output serves as the input for the next state[1]. We modify Eq (3.4) as follows:

$$r[i+1] = (1-\alpha)r[i] + \alpha \ \tanh(Ar[i] + W_{in}W_{out}r[i] + k_b W_b(\epsilon - \epsilon_0)), \tag{3.5}$$

which simplifies to a map equation:

$$r[i+1] = (1-\alpha)r[i] + \alpha \ \tanh(\Lambda r[i] + \Omega), \tag{3.6}$$

where $\Lambda = A + W_{in}W_{out}$ and $\Omega = k_b W_b(\epsilon - \epsilon_0)$. This defines a system of m autonomous map equations, with $r[i]$ and Ω as m-dimensional column vectors and Λ as an $m \times m$ matrix. Successful prediction indicates that W_{out} enables the map to replicate the original system's bifurcation behaviour.

The final time series output at a desired bifurcation parameter value is a linear combination of reservoir states, given by $v[i] = W_{out}r[i]$. Thus, analyzing Eq (3.6) reveals the system's dynamics.

To understand this further, we compute the fixed point r^* of Eq (3.6) and then evaluate the Jacobian matrix $\mathcal{J}$ at r^*:

$$\mathcal{J} = \begin{bmatrix} \frac{dr_1[i+1]}{dr_1[i]} & \frac{dr_1[i+1]}{dr_2[i]} & \cdots & \frac{dr_1[i+1]}{dr_m[i]} \\ \frac{dr_2[i+1]}{dr_1[i]} & \frac{dr_2[i+1]}{dr_2[i]} & \cdots & \frac{dr_2[i+1]}{dr_m[i]} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{dr_m[i+1]}{dr_1[i]} & \frac{dr_m[i+1]}{dr_2[i]} & \cdots & \frac{dr_m[i+1]}{dr_m[i]} \end{bmatrix}_{(r_1^*, r_2^*, \cdots, r_m^*)} \tag{3.7}$$

We then analyse how the eigenvalues of $\mathcal{J}$ vary with changes in ϵ.In the context of a coupled map (or any discrete dynamical system), the theorem linear stability analysis

states that if even one of the real values of the eigenvalues of the Jacobian matrix evaluated at a fixed point crosses 1 in magnitude (i.e.,$|\lambda| > 1$), the fixed point becomes unstable. This marks the loss of stability and often signals the onset of bifurcations, leading to more complex dynamical behaviour.

This is because the deviations given to a system in a fixed point change according to the equation

$$\delta x_t = \mathcal{J}\delta x_{t-1} \tag{3.8}$$

and when $|\lambda| > 1$ the deviations propogate.

Chapter 4

Methodology

The primary objective of this study is to analyze the bifurcation structure underlying the Parameter-Aware Reservoir Computing (PARC) Map Network and its ability to replicate the dynamics of a given input system. By training the reservoir on different parameter regimes, we investigate how the network adapts its internal map structure to capture key dynamical transitions, such as fixed points, periodic orbits, and chaotic behaviour. A crucial aspect of this work involves examining the bifurcation mechanisms within the reservoir, determining whether it undergoes period-doubling, Neimark-Sacker, or other bifurcations to match the input system's behaviour. Through Jacobian eigenvalue analysis,we systematically explore the reservoir's dynamical flexibility and its capacity to generalize across nonlinear systems.

Some Non-Linear systems that we have worked with are

- The Logistic map
- The Higher-order Kuramoto model
- Coupled Stuart-Landau Oscillators

4.1 Logistic Map

The Logistic Map is a simple mathematical model used to describe how populations change over time in an idealised environment, its's equation is given by:

$$x_{n+1} = \mu x_n(1 - x_n) \tag{4.1}$$

In this context, x_n represents the population at the n^{th} generation, scaled between 0 and 1. The parameter μ is the **growth parameter**, which controls the rate at which the population grows.

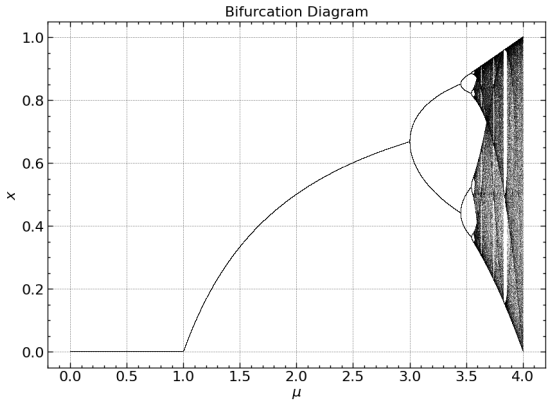


Figure 4.1: The bifurcation graph of the logistic map wrt μ

The behaviour of the logistic map varies with respect to the control parameter μ for $0 < \mu < 2$ the map only has a single stable fixed point but for values $\mu > 2$ the equation shows period doubling behaviour as seen in figure 4.1.

4.1.1 Obtaining Data

We obtain the behaviour of x_n for the logistic for different values of μ values 3.1, 3.2, 3.3, 3.4 and 2.9. From Figure (4.2) we can see that for the first three values of epsilon the Logistic map shows period doubling And for $\mu = 2.9$ the map shows fixed-point

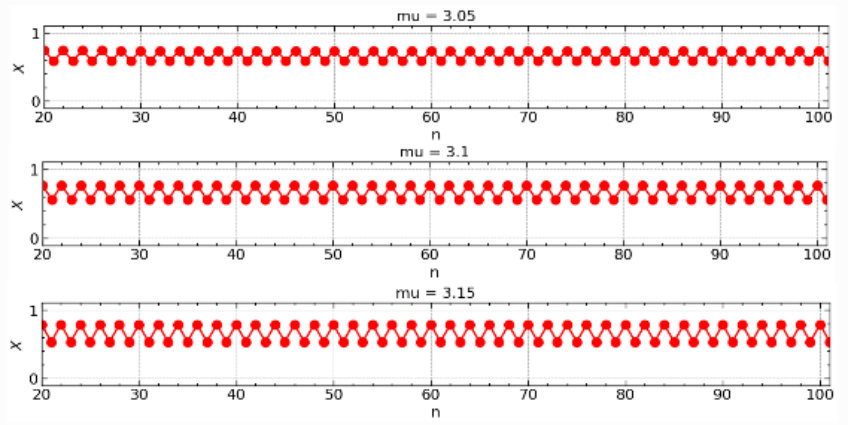


Figure 4.2: Training data for Logistic map

behaviour,as seen below.We train the PARC using the training data and make it capable of predicting the testing data

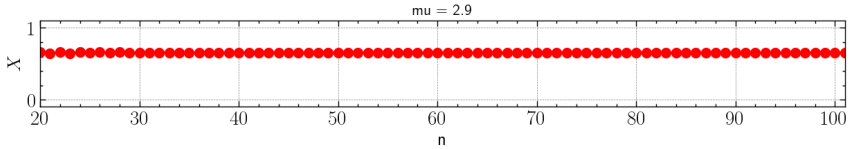


Figure 4.3: Testing data for Logistic map

4.2 Kuramoto model

The **Kuramoto model**, introduced by Yoshiki Kuramoto in the 1970s, is a key framework for studying synchronization in networks of coupled oscillators. Each oscillator has a natural frequency and interacts with others through phase differences. The model is defined as:

$$\dot{\theta}_i = \omega_i + \frac{K}{N}\sum_{j=1}^{N}\sin(\theta_j - \theta_i) \tag{4.2}$$

where θ_i is the phase, ω_i the intrinsic frequency, and K the coupling strength. At low K, oscillators behave independently, while higher K leads to synchronization.

4.2.1 Higher-order Kuramoto Model

The Higher-order Kuramoto model was developed to model more complex interactions that go beyond pairwise coupling. Here, additional nonlinear terms or higher-order coupling effects are incorporated. The model that we will be working with is the one mentioned in [7]. Here the higher-order interactions are triangular(three oscillators interact with each other at the same time) and tetrahedral (four oscillators interact with each other at the same time) in nature. The equation of motion for the same is given by

$$\begin{aligned}\dot{\theta}_i = \omega_i + \frac{K_1}{N}\sum_{j=1}^{N}\sin(\theta_j - \theta_i) + \frac{K_2}{N^2}\sum_{j=1}^{N}\sum_{l=1}^{N}\sin(2\theta_j - \theta_l - \theta_i) \\ + \frac{K_3}{N^3}\sum_{j=1}^{N}\sum_{l=1}^{N}\sum_{m=1}^{N}\sin(\theta_j + \theta_l - \theta_m - \theta_i)\end{aligned} \tag{4.3}$$

Here K_2 and K_3 are the coupling strengths of the triangular and tetrahedral interactions respectively. In the respective mean fields, the dynamical evolution equations could be written as

$$\dot{\theta}_i = \omega_i + K_1\, r_1\, \sin(\psi_1 - \theta_i) + K_2\, r_1\, r_2\, \sin(\psi_2 - \psi_1 - \theta_i) + K_3\, r_1^3\, \sin(\psi_1 - \theta_i) \tag{4.4}$$

with the help of complex order parameters defined as

$$z_n = r_n e^{i\psi_n} = \frac{1}{N}\sum_{j=1}^{N} e^{in\theta_j} \tag{4.5}$$

which measures the strength of the global synchronization of the oscillators with $0 \leq r_1 \leq 1$, Where $r_1 \sim 0$ indicates a complete incoherent state, whereas $r_1 \sim 1$ indicates global synchronization and ψ_1 measures the mean phase of all the oscillators.

Using Eq (4.4) and Eq (4.5) and applying Ott and Antonsen's ansatz, we would obtain a low-dimensional system that governs the macroscopic dynamics of Eq (4.3). In

particular, by considering the continuum limit of infinitely many oscillators and applying the Ott-Antonsen ansatz (see Appendix B for details), we obtain the amplitude r and angle ψ as simpled differential equations are given below,

$$\dot{r} = -r + \frac{K_1}{2} r(1 - r^2) + \frac{K_{2+3}}{2} r^3(1 - r^2) \tag{4.6}$$

$$\dot{\psi} = 0 \tag{4.7}$$

So by studying the behaviour of Eq (4.6) called the order parameter equation, we can understand the behaviour of the network system i.e. when they are synchronised or showing incoherent behaviour. Eq (4.7) tells us that the mean phase of the oscillators remains constant.

4.2.2 Bifurcation Plot of the order parameter equation

The bifurcation plot of the order parameter equation in the higher-order Kuramoto model, plotted over different values of K_1 and K_2, reveals how synchronization levels and phase transitions change with varying coupling strengths. By observing changes in the order parameter r, which ranges from 0 (no synchronization) to 1 (full synchronization), the plot identifies regions where the system transitions between incoherent, partially synchronized, and fully synchronized states.

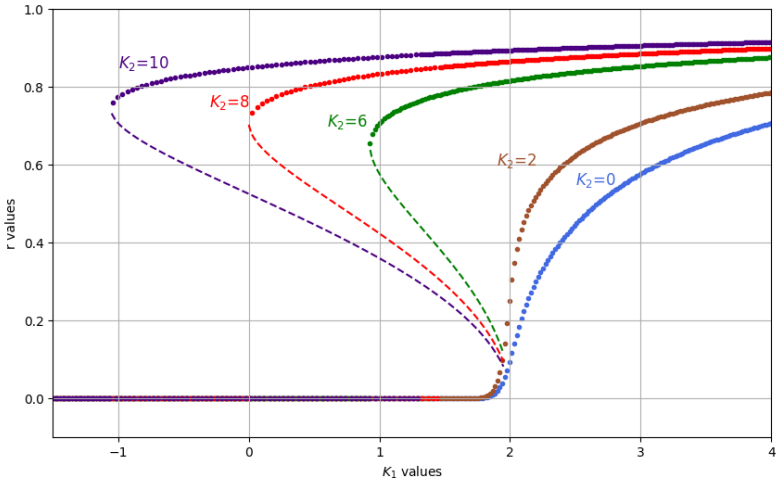


Figure 4.4: Bifurcation plot of the order parameter equation with respect to K_1 for different values of K_2, with stable points represented by dots and unstable points by dashed lines. The plot illustrates that higher values of K_2 can enhance synchronization or lead to multistable states, highlighting the significant influence of non-pairwise interactions on stability and synchronization dynamics. Additionally,we could see higher K_2 values lead to a second-order transition from a coherent to an incoherent state.Reproduced from [7]

4.2.3 Obtaining Data

Transcritical Bifurcation($K_2 = 0$)

We obtain the behaviour of $r(t)$ from numerically solving the order parameter Eq (4.6) for $K_2 = 0$ and K_1 values 2.05,2.1,2.15,2.2,2.25 and 1.99. From Figure (4.5) we can see that for $K_1 > 2$, there is a single stable fixed point with a value greater than 0 and an unstable fixed point at 0. And for $K_1 < 2$,the fixed point at 0 becomes stable while the

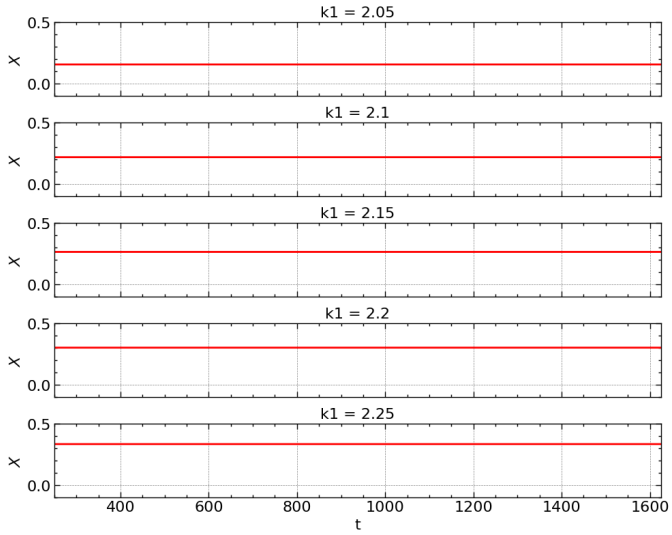


Figure 4.5: Training Data for Order Parameter Equation with $K_2 = 0$ and $K_1 = 2.05, 2.1, 2.15, 2.2, 2.25$. Here there is a single fixed point with a value more than 0.

other fixed point becomes unstable,thus showing a Transcritical Bifurcation We use the

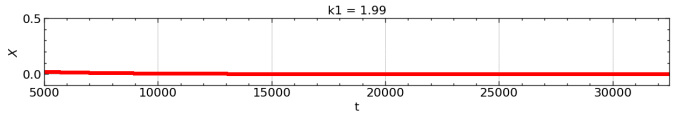


Figure 4.6: Testing Data for Order Parameter Equation with $K_2 = 0$ and $K_1 = 1.99$, here we could see a single fixed point whose value is 0.

behaviour(or time series data) of the first five values of K_1 as our training data after removing the initial transient(removing the data values for the first 100 or so iterations) and checking whether the RC can predict the single fixed point behaviour of the order parameter equation for $K_1 = 1.99$ also with initial transient removed. We the analyse the behaviour of the RC map during this prediction.

Pitchfork Bifurcation($K_2 = 8$)

We obtain the behaviour of $r(t)$ from numerically solving the order parameter Eq.(4.6) for $K_2 = 8$ and K_1 values 0.05,0.1,0.15,0.2,0.25 and -0.05. From Figure 4.7 we can see that for the first five values of K_1, the system has two fixed points. Here the system is

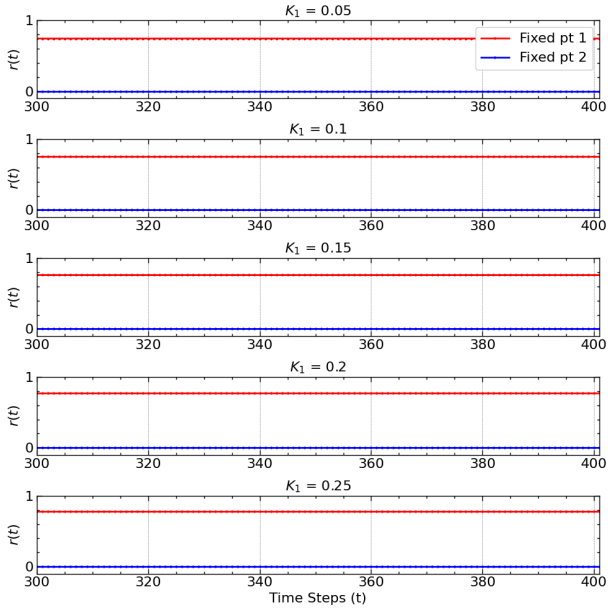


Figure 4.7: Training Data for Order Parameter Equation for $K_2 = 8$ and K_1 values 0.05,0.1,0.15,0.2,0.25.Here, we could see two fixed points

undergoing a Pitchfork bifurcation ,we then do the analysis as the previous case. For the K_1 value -0.05, the system only has a single fixed point. Here the system is undergoing a Pitchfork bifurcationWe then do the analysis as the previous case

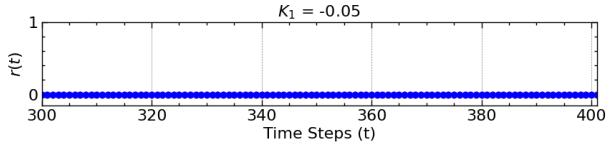


Figure 4.8: Testing data for Order Parameter Equation for $K_2 = 8$ and $K_1 = -0.05$,here we could see only a single fixed point.

4.3 Coupled Stuart-Landau Oscillators

To explore the dynamics of nonlinear coupled oscillatory systems, we consider a minimal yet rich model: a pair of coupled Stuart-Landau oscillators. The Stuart-Landau oscillator arises as the normal form of a supercritical Hopf bifurcation and serves as a canonical model for studying limit-cycle behavior in nonlinear dynamical systems.

The system under study consists of two non-identical Stuart-Landau oscillators with diffusive coupling. Each oscillator is described by a pair of real-valued variables (x_i, y_i), corresponding to the real and imaginary parts of a complex amplitude $Z_i = x_i + iy_i$. The evolution of each oscillator is governed by the following set of nonlinear differential equations:

$$\dot{x}_i = (1 - x_i^2 - y_i^2)x_i - \omega_i y_i + \varepsilon(x_j - x_i) \tag{4.8}$$

$$\dot{y}_i = (1 - x_i^2 - y_i^2)y_i + \omega_i x_i + \varepsilon(y_j - y_i) \tag{4.9}$$

$$\text{where } i, j \in \{1, 2\},\ i \neq j.$$

In this formulation, ω_i is the intrinsic angular frequency of the i^{th} oscillator,ε denotes the coupling strength. In this study, we fix the intrinsic frequencies as $\omega_1 = 2$ and $\omega_2 = 7$, then we study the behaviour of this system for different values of ε.We see there exists a stable fixed point for $1 < \varepsilon < 3.6$ which looses its stability through a Hopf bifurcation and a stable limit cycle appears for $\varepsilon > 3.6$.

4.3.1 Obtaining Data

We obtain the behaviour of $x_1(t), x_2(t), y_1(t)$ *and* $y_2(t)$ by numerically solving the equations Eq (4.8) and Eq (4.9). We find the behaviour of the system at ε values $3.3, 3.65, 3.7$ and 3.75 which gives us our training and testing dataset.

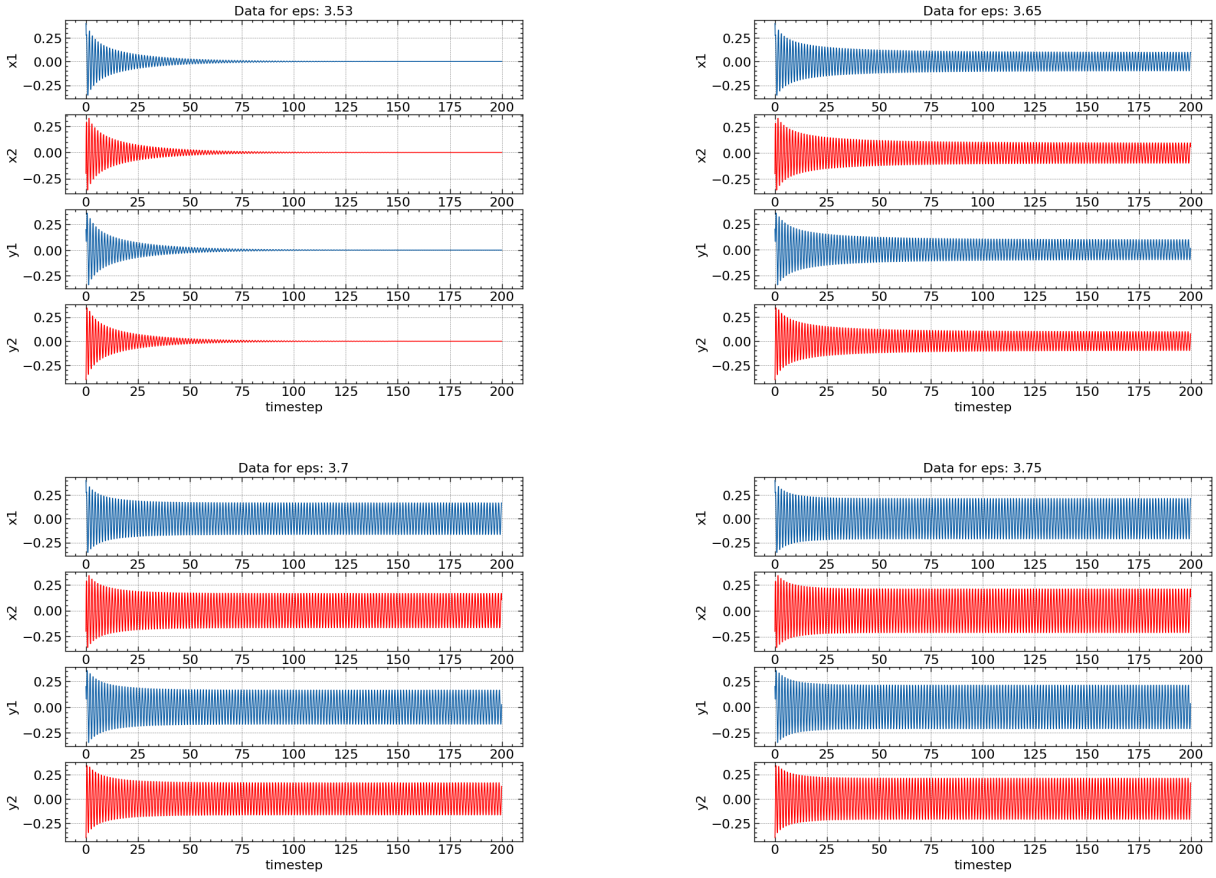
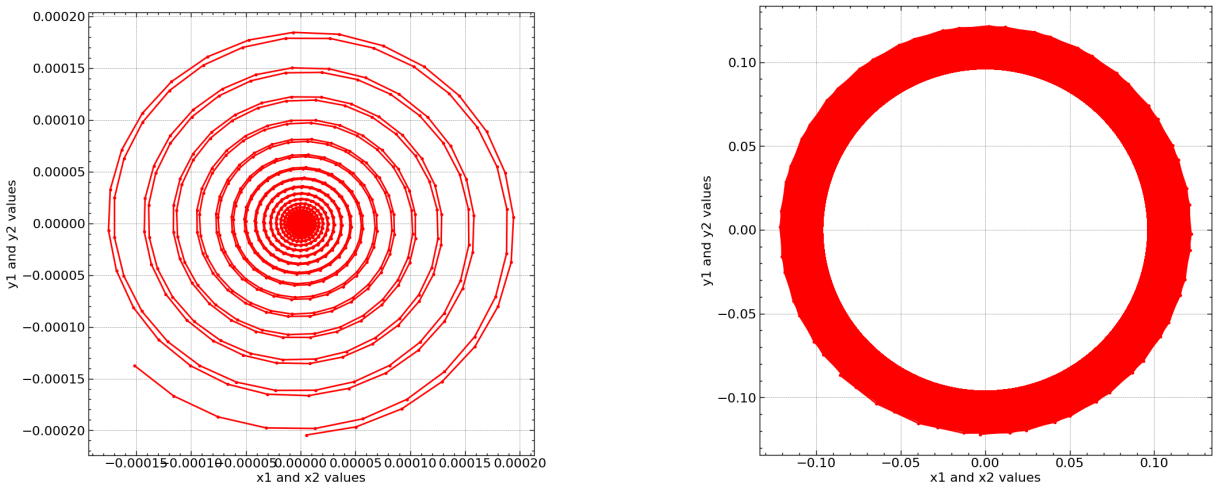


Figure 4.9: Coupled Stuart-Landau oscillator outputs for different coupling strengths. The first three ($\varepsilon = 3.53, 3.65, 3.7$) are training data; the last ($\varepsilon = 3.75$) is testing data.

We could plot the phase space diagram of the system and see how it undergoes Hopf Bifurcation for ϵ values greater than 3.6.



(a) $\epsilon < 3.6$ (b) $\epsilon > 3.6$

Figure 4.10: We have plotted the phase space behaviour of the 4-D coupled Stuart-Landau Oscillator system. Figure (a) shows the system's behaviour for epsilon values less than the bifurcation point, where the system settles down to a stable fixed point at 0. Figure (b) shows the behaviour after it has undergone a Hopf bifurcation, where the system shows a stable closed loop.

Chapter 5

Results

5.1 Logistic Map

We have plotted and compared the predicted values of the PARC with the actual values of the Logistic map for different μ values.

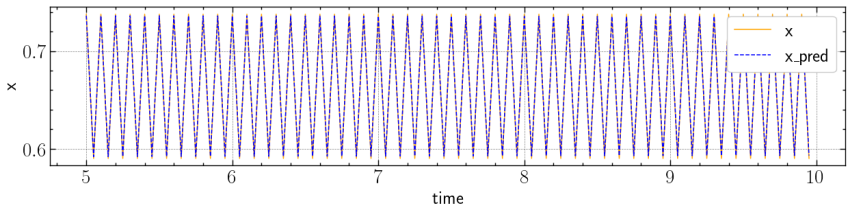


Figure 5.1: Prediction for $\mu = 3.05$, the prediction was successful with a NRMSE of 0.0715

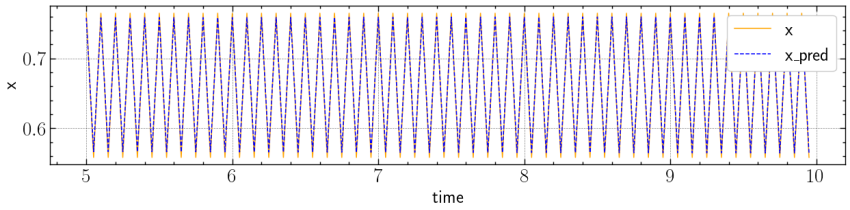


Figure 5.2: Prediction for $\mu = 3.1$, the prediction was successful with a NRMSE of 0.0421

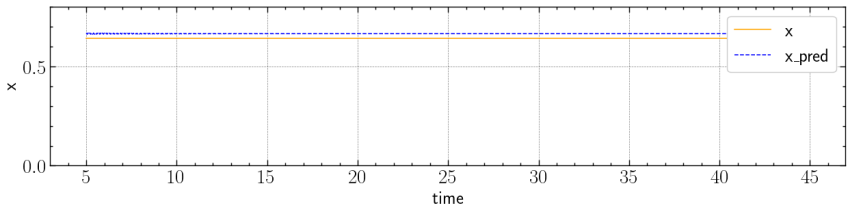


Figure 5.3: Prediction for $\mu = 2.8$ was successful, yielding an NRMSE of 41.08.

We can see that the PARC algorithm is able to successfully predict the transition from fixed point to period doubling behaviour of the logistic map. We could see that high error

arises for $\mu = 2.8$ despite correctly capturing the system's behavior. This is due to a small displacement between predicted and actual values. Since both predictions and true values form straight lines, this displacement persists across all points, contributing to the elevated error.

5.1.1 Jacobian Analysis

Now we do the Jacobian analysis of the trained PARC network that is able to give the correct predictions.For this system, the network is able to predict the period-doubling bifurcation of the logistic map by either undergoing a period-doubling bifurcation or surprisingly enough by undergoing a saddle-node bifurcation also.

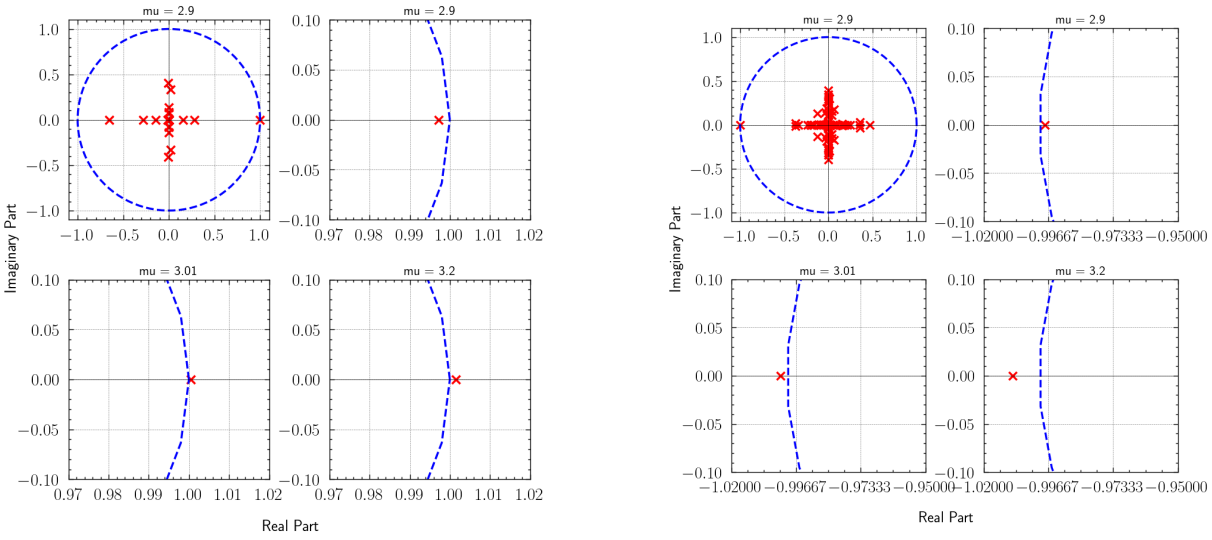


Figure 5.4: Bifurcation mechanisms employed by the PARC map network to predict the transition from a fixed point to period-doubling behaviour at $\mu = 2$. The first image depicts a saddle-node bifurcation, and the second image showcases a period-doubling bifurcation.

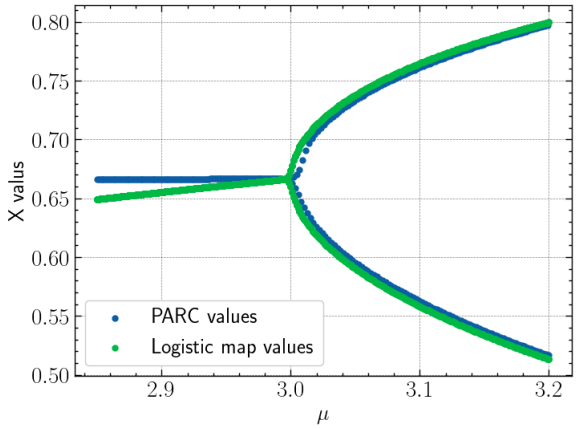


Figure 5.5: A comparative Bifurcation plot between the predicted and actual values to see how well the PARC network mimics the bifurcation behaviour of the Logistic map

5.2 The Higher Order Kuramoto Model

5.2.1 Prediction for $K_2 = 0$

For $K_2 = 0$, the order parameter equation undergoes a pitchfork bifurcation and we see the reservoir network undergoes a saddle-node bifurcation to mimic this behaviour

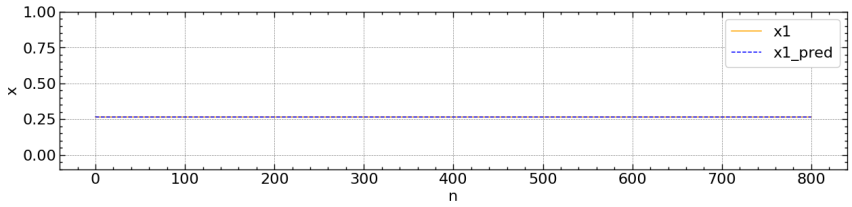


Figure 5.6: Prediction for $K_1 = 2.1$, the prediction was successful with a NRMSE of 5.86

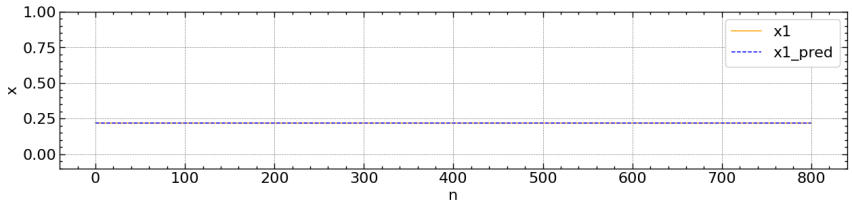


Figure 5.7: Prediction for $K_1 = 2.15$, the prediction was successful with a NRMSE of 3.42

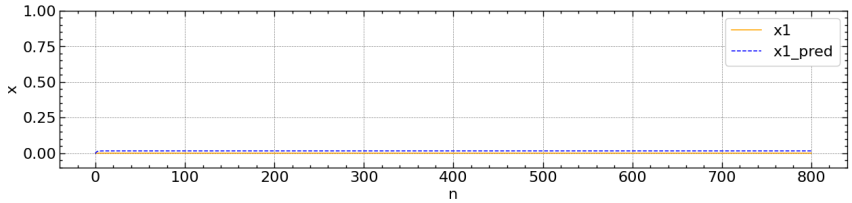


Figure 5.8: Prediction for $K_1 = 1.9$, the prediction was successful with a NRMSE of 30.4

5.2.2 Prediction for $K_2 = 8$

For $K_2 = 8$, the order parameter equation undergoes a supercritical pitchfork bifurcation and we see that the reservoir map network undergoes a saddle-node bifurcation to mimic this behaviour.

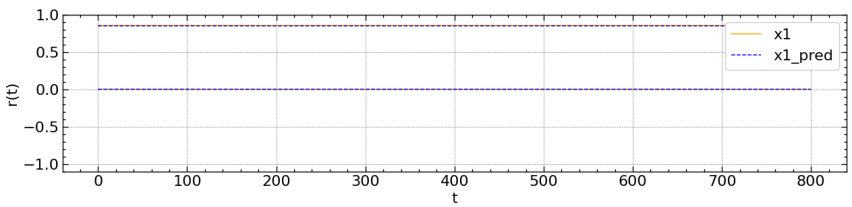


Figure 5.9: Predicted and actual data for $K_1 = 0.13$, with an NRMSE of 0.000002

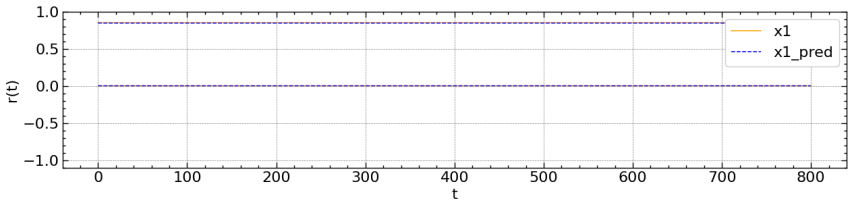


Figure 5.10: Predicted and actual data for $K_1 = 0.23$, with an NRMSE of 0.000002

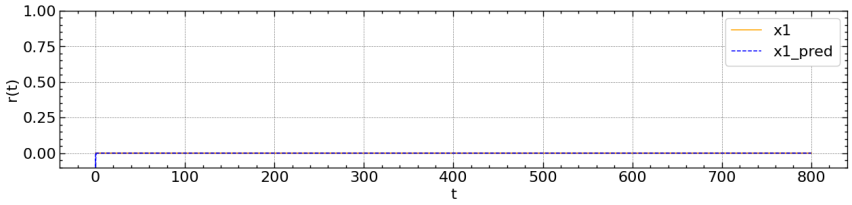
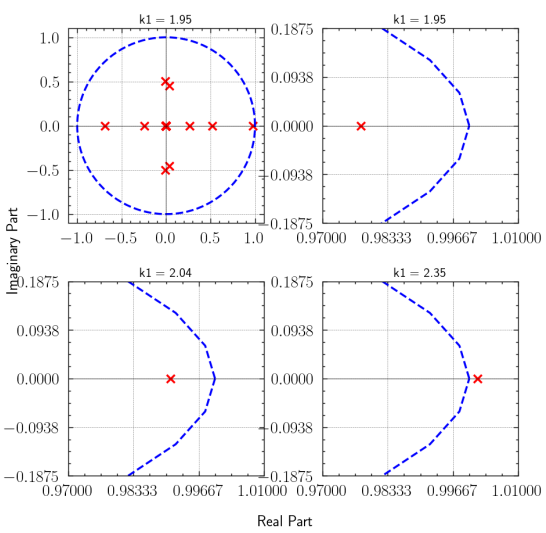
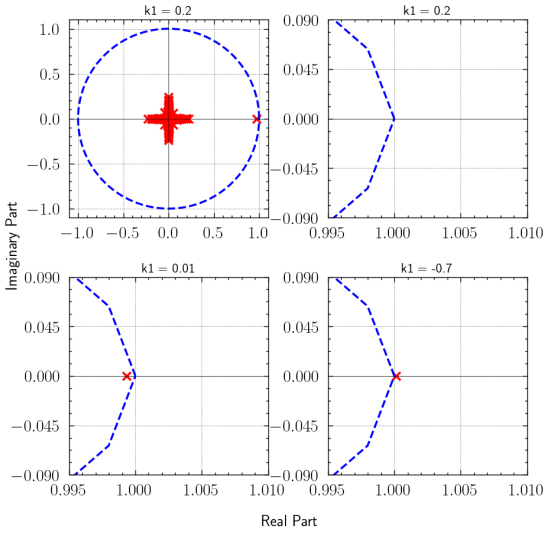


Figure 5.11: Predicted and actual data for $K_1 = -0.1$, with an NRMSE of 1.00

5.2.3 Jacobian Analysis



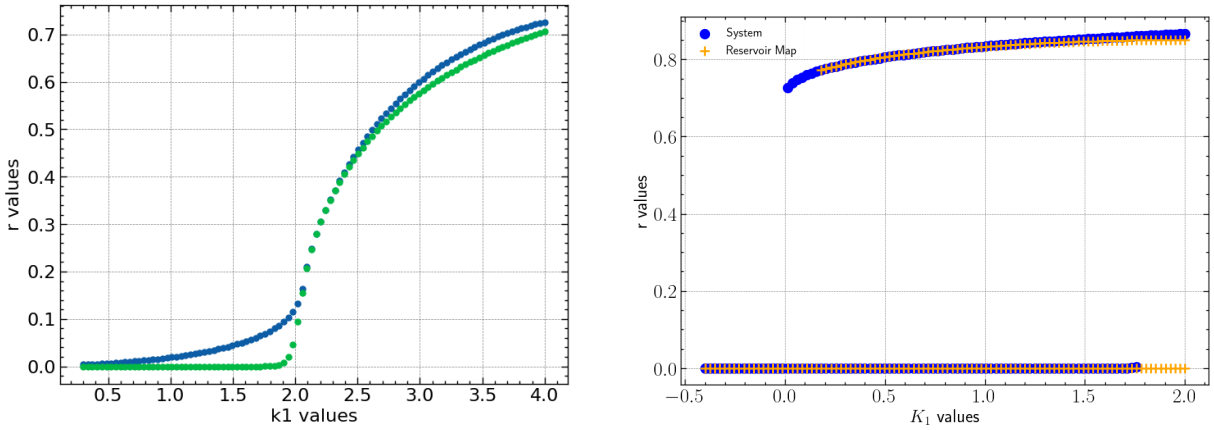
(a) Eigenvalues for $K_2 = 0$



(b) Eigenvalues for $K_2 = 8$

Figure 5.12: Here we have plotted the eigenvalues in the complex plane of the Jacobian matrix of the PARC network used to predict the behaviour of Higher-order Kuramoto oscillator for different K_1 and K_2 values.

In a saddle-node bifurcation, there is no fixed point after the bifurcation point but in a pitchfork bifurcation, there are fixed points before and after the bifurcation point. But interestingly the reservoir network utilizes a saddle-node bifurcation to mimic the pitchfork bifurcation and is able to make successful predictions.This shoes one of the interesting way the map network learns complex dynamics.



(a) Plot for k2=0

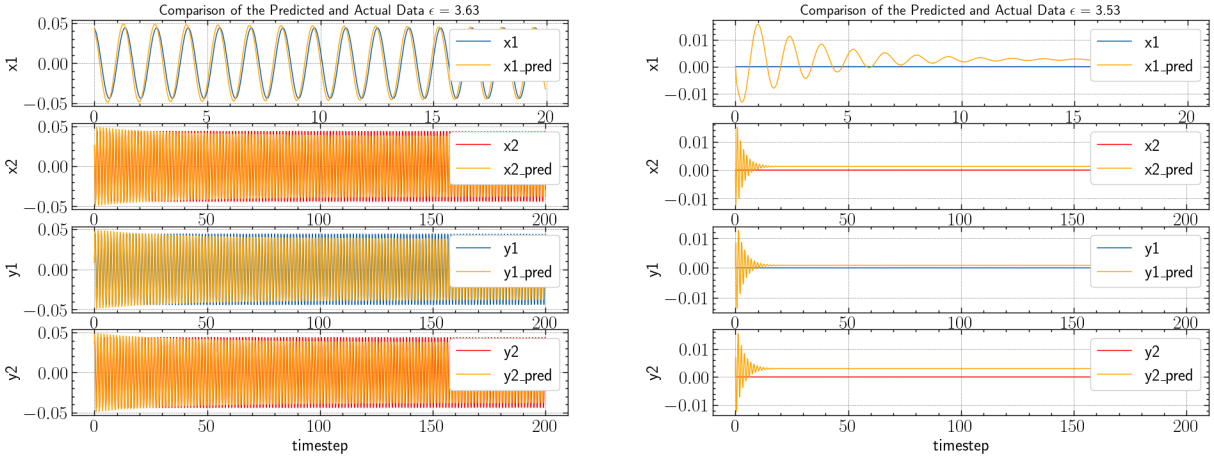
(b) Plot for k2=8

Figure 5.13: Comparative Bifurcation plot between the prediction of the PARC model and the actual system, showing how well the PARC model is able to mimic the bifurcation dynamics of the system it learns.We also see that the predictions fail after we go further away from the bifurcation point we have trained the map in (fig b)

5.3 The Coupled Stuart Landau Oscillator

5.3.1 Prediction

Here we have plotted the predicted and actual values of the coupled oscillator system.

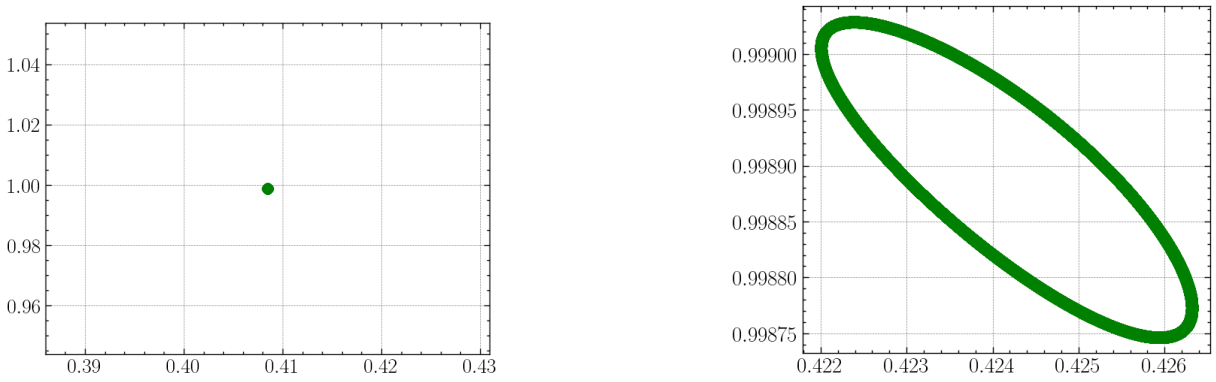


(a) Predicted and actual values for $\epsilon = 3.63$, with NRMSE = 1.42

(b) Predicted and actual values for $\epsilon = 3.53$, with NRMSE = 0.99

Figure 5.14: Comparison of predictions for two values of ϵ in the Stuart-Landau system.

We could plot the phase space diagram of two reservoir nodes to see how their behaviour varies due to different epsilon values we could see the neimark saicker bifurcation happening giving us a clearer understanding of its internal dynamics.



(a) $\epsilon < 3.6$ (b) $\epsilon > 3.6$

Figure 5.15: We have plotted the phase space behavior of two reservoir nodes within the PARC network. The observed dynamics indicate that the system undergoes a Neimark–Sacker bifurcation, the discrete-time analogue of a Hopf bifurcation. This behavior suggests that the PARC network effectively leverages discrete-time dynamics to learn and represent the behavior of both continuous and discrete dynamical systems.

5.3.2 Jacobian analysis

We the conduct an eigen value analysis of the PARC network to see what bifurcation is happening in the network.

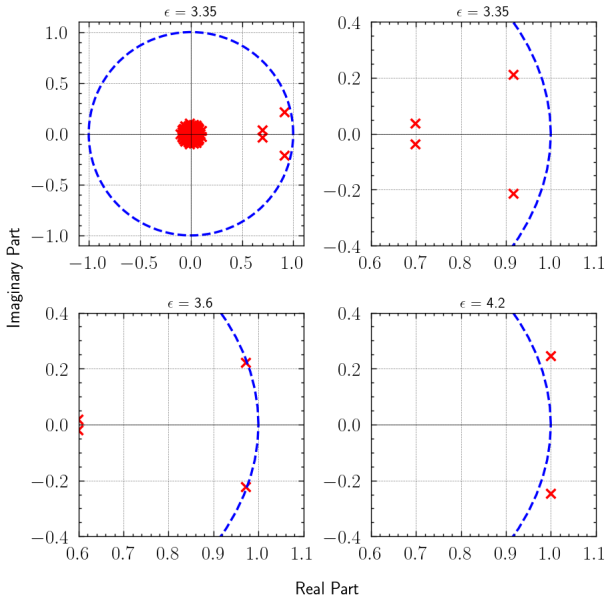


Figure 5.16: The plot of eigenvalues in the complex plane shows how their magnitudes change with respect to ϵ, with the dotted blue lines representing a unit circle. We could see a pair of complex conjugate eigenvalues crossing the unit circle, which is a clear indication of Neimark-Sacker bifurcation.

Chapter 6

Conclusion

In this thesis, we developed and analyzed a Parameter-Aware Reservoir Computing (PARC) model to predict the behavior of various nonlinear dynamical systems across different parameter regimes. By applying the model to systems such as the Logistic map, Higher-Order Kuramoto oscillators, and Coupled Stuart-Landau oscillators, we demonstrated its capability to generalize and adapt to varying dynamical conditions. We then mapped the internal behaviour of the PARC network to known bifurcations seen in map networks to uncover key internal dynamics.We were able to see that the PARC network used

- The reservoir model exhibited both period-doubling and, interestingly, saddle-node bifurcations while learning the dynamics of the Logistic map.
- It captured the behavior of the Higher-Order Kuramoto model through a saddle-node bifurcation in the reservoir network.
- The model reflected a Neimark–Sacker bifurcation while learning the Hopf bifurcation dynamics of the Coupled Stuart–Landau system—remarkably, the Neimark–Sacker bifurcation is the discrete-time analogue of the Hopf bifurcation observed in continuous systems.

The work also culminated in the development of the PaRes.py framework, a user-friendly and versatile tool that enables researchers to apply the PARC machine learning model to a wide range of dynamical systems. The framework includes built-in functionalities for training-test data splitting and incorporates Bayesian optimization for efficient hyperparameter tuning.

Overall, this work contributes meaningfully to bridging the gap between nonlinear dynamical systems and machine learning by uncovering the internal mechanisms through which the PARC model processes and predicts complex system behavior. By revealing how known bifurcations emerge within the reservoir and providing an accessible implementation through the PaRes.py framework, this thesis offers both theoretical insights and practical tools. These advancements not only enhance the interpretability of reservoir

computing but also pave the way for future interdisciplinary research at the intersection of nonlinear science and data-driven modeling.

Future Prospects

During this research, it was shown that the PARC network is able to predict more complicated bifurcation dynamics, such as fixed point to chaos or period 2 to period 4 behaviour. But Jacobian analysis of the normal PARC network is now able to give us answers on how it learns these dynamics. This is because of two main reasons

- There is no behaviour that is shown by the eigenvalues of any map network to map the eigenvalue behaviour shown in the PARC network, which is capable of predicting the fixed point to chaotic behaviour bifurcation.
- Now, for the prediction of period 2 to period 4, if we need to check the bifurcation behaviour, we have to do a Jacobian analysis of the second iterate map, which is $f(f(r))$, which becomes very computationally taxing

One way we could further this research is by analysing the behaviour of the map for such complex dynamics and also finding the second iterate map equation of the PARC network to do further analysis.

Uses in Finance

An exciting and promising area of research involves leveraging the Parameter-Aware Reservoir Computing (PARC) model to predict complex financial time series. Due to the echo state property of the model, it fails in predicting time series that consist of seasonality and trend, such as daily financial stock closing price data across timelines So what we could do is decompose our time series into seasonal, trend and residual parts. Here, the seasonal time series is stationary in nature, and it is similar to the time series we have predicted till now with some stochasticity.

We use a Geometric Brownian Motion equation, which is widely used in quantitative finance to model stock data behaviour.

$$S(t) = S_0 \cdot \exp\left(\left(\mu - \frac{1}{2}\sigma^2\right)t + \sigma W(t)\right) \tag{6.1}$$

Here, $S(t)$ denotes the stock price at time t, S_0 is the initial stock price, μ represents the drift or expected return rate, σ is the volatility (standard deviation of returns), and σ^2 is the variance of returns. The variable t stands for time, and $W(t)$ represents a standard Brownian motion (also known as a Wiener process). We use the above equation to obtain the time series data for different volatility values, we then can use the PARC network to

predict the behaviour of the system at unknown volatility values. The decomposition of the obtained time series into its three parts is shown below.

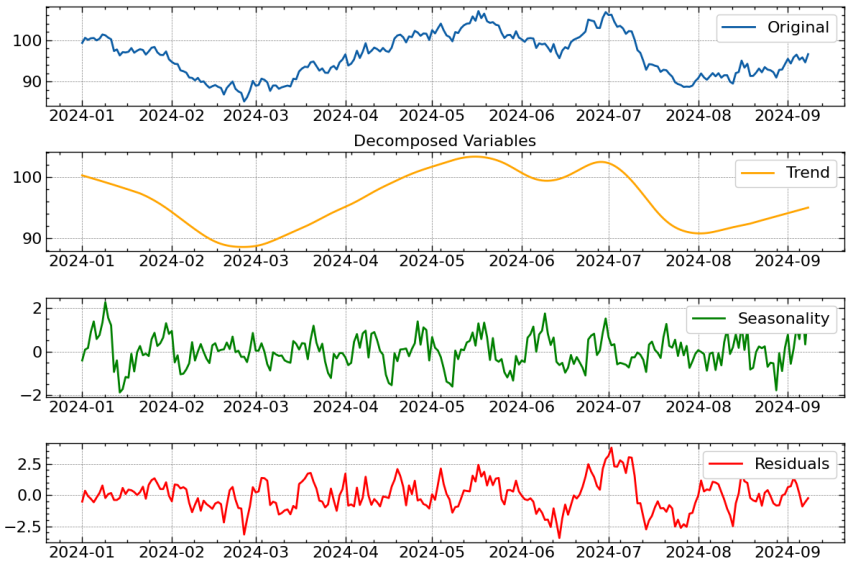


Figure 6.1: Here the first subfigure, we have plotted the original financial timeseries. Then in the next three subfigures, we could see how the original could be decomposed into its seasonal, trend and residual parts.

The result of this analysis is that the model is able to make better predictions compared to the normal financial time series, but work still needs to be done to find the ideal hyperparameters such we are able to generalise the system across different volatility values and further work need to be done here.

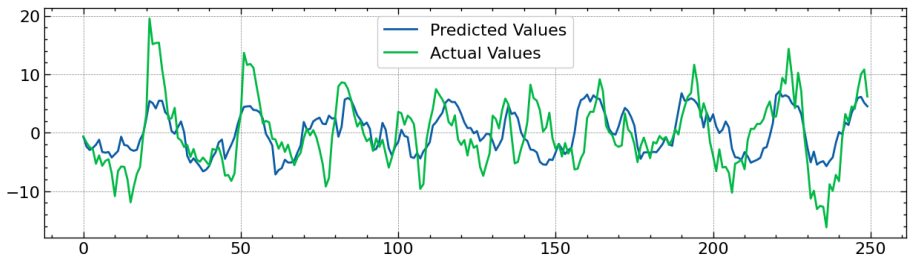


Figure 6.2: Seasonal prediction of financial time series for testing volatility using our PARC model value with an NRMSE of 0.78.We could see the prediction is not fully accurate yet, but it is able to predict the overall trend.

Appendix A

Understanding the PARC code

In this section, we will go through how the `PaRes.py` framework is used to predict the time series data of the logistic map at different μ values and how we could use it to conduct the eigenvalue analysis of the PARC network.

Step 1:

Define a function to calculate the time series of the system you are working with for different bifurcation values. This could be done by solving the system's equation numerically using `Scipy` or, in the case of Logistic map, a simple recursive loop.

```
def logistic_map(mu, x_0):
    data_length = 36000

    t = np.linspace(0, data_length, data_length)
    x = x_0
    x_data = np.zeros_like(t)
    x_data[0] = x_0


    for i in range(1,data_length):
        x_new = mu * x * (1 - x)
        x  = x_new
        x_data[i] = x_new

    return x_data
```

Step 2:

We then select the bifurcation parameters we want to train and test the data with. Here, the data is obtained for each parameter value and is stacked horizontally to have a single list containing the different behaviours of the system.

```
mu_values = [3.05,3.10,3.15,2.95]

data_x_list = []

```

```
for mu in mu_values:
    x_0 = float(0.8)
    x_data = logistic_map(mu, x_0)
    data_x_list.append(x_data)

stacked_x1 = np.transpose(np.hstack(data_x_list))

stacked_total = np.expand_dims(stacked_x1 ,axis=0)
print(stacked_total.shape)
```

Step 3:

We now use the `PaRes.py` framework to split the given data list into training and testing data, where the last value will be our testing data and all others will be our training data. The framework also consists of the Reservoir Computer class, which includes training and predicting methods. We make an object of the class according to our selected hyperparameters and provide it with training data to learn the input system's dynamics. We then check how well the PARC's prediction matches the actual testing data by using the NRMSE metric.

```
from PaResPy import ReservoirComputer

training_data,valid_data,train_without_transient = ReservoirComputer.
    train_test_split(stacked_total)

eps_train = [3.05,3.10,3.15]

dim_reservoir = 100
rho = 0.3037
sigma = 0.00085
k_b = 0.0492
alpha = 1
model = ReservoirComputer(1, dim_reservoir, rho, sigma, k_b, alpha)
model.train(training_data, train_without_transient,eps_train)
predicted_data,_ = model.predict(mu,len(valid_data), valid_data,
    train_without_transient)
total_var = np.var(predicted_data)
NRMSE = np.sqrt(np.mean((valid_data[:] - predicted_data[:]) ** 2) /
    total_var)
print(NRMSE)
```

Step 4:

Here, we provide the Bayesian optimisation code that could help us find the accurate hyperparameters. We start the code block by providing the function a range of hyperparameters in which we believe to find the ideal ones. The function works by finding the set of hyperparameters that minimises the NRMSE .

```
from skopt import gp_minimize
from skopt.space import Real, Integer
mu = 3.1
mu_data = logistic_map(mu, x_0)
valid_data = np.expand_dims(mu_data[-16000:],axis=1)

def f_x(hyperparameters):
    rho, sigma,k_b= hyperparameters
    model = ReservoirComputer(1, 150, rho, sigma, k_b, 1)
    model.train(training_data, train_without_transient)
    predicted_data,_ = model.predict(mu,len(valid_data), valid_data,
   train_without_transient)
    total_var = np.var(predicted_data)
    NRMSE = np.sqrt(np.mean((valid_data[:] - predicted_data[:]) ** 2)
   / total_var)
    return NRMSE

space = [
    Real(0.1,0.9, name='rho'),
    Real(0.0001,0.1, name='sigma'),
    Real(0.001,0.09,name='k_b')
]

result = gp_minimize(f_x, space, n_calls=80, random_state=42)

# Best hyperparameters found
print(f"Best hyperparameters: {result.x}")
print(f"Best score (NRMSE): {result.fun}")
```

Step 5:

We could plot the predicted and actual behaviour using the below code block to see how well the Machine learning model is able to generalise across different parameter values.

```

mu_values = np.array([3.05, 3.1, 2.8])
for mu in mu_values:
    mu_data = logistic_map(mu, x_0)
    valid_data = np.expand_dims(mu_data[-16000:],axis=1)
    predicted_data, Res_test_total = model.predict(mu,len(valid_data),
    valid_data, train_without_transient)

    #Checking NRMSE
    total_var = np.var(predicted_data)
    NRMSE = np.sqrt(np.mean((valid_data[:] - predicted_data[:]) ** 2)
   / total_var)
    print("The NRMSE value of mu value {} :".format(mu), NRMSE)

    #Checking the plots
```

```
    dt = 0.05
    x_actual_data = valid_data
    x_pred_data = predicted_data
    timeaxis = np.arange(0, (valid_data.shape[0])*dt, dt)
    plot_timeseries_figure(x_pred_data[100:900],x_actual_data
   [100:900],timeaxis[100:900])
```

Step 5:

We then import the eigenvalue analysis function from the framework, which will calculate the eigenvalues of the Jacobian matrix of the PARC used for prediction. We plot the eigenvalues in the complex plane to see how they vary for different parameters giving us insight to the internal dynamics of the PARC model.

```

A, W_in, W_out, W_b = model.get_weights()
eb = 0
alpha = 1
Lambda = A + np.matmul(W_in, W_out)
Omega = k_b * W_b

from Parespy import eigenvalues

epsilon_list=np.array([2.9, 3.01, 3.2])
analysis_model = eigenvalues(eb,alpha,Lambda,Omega,dim_reservoir)
eig_array = analysis_model.eigen_values(epsilon_list,int_guess=0.65)

x_lim = [0.97,1.02, 6]
y_lim = [-0.1, 0.1, 5]

analysis_model.Plot_eig_values(eig_array,epsilon_list,x_lim,y_lim)
```

Appendix B

Dimension Reduction of Higher-order Kuramoto Model

The HIgher- Order Kuramoto Model is given by:

$$\dot{\theta}_i = \omega_i + \frac{K_1}{N}\sum_{j=1}^{N}\sin(\theta_j - \theta_i) + \frac{K_2}{N^2}\sum_{j=1}^{N}\sum_{l=1}^{N}\sin(2\theta_j - \theta_l - \theta_i) + \frac{K_3}{N^3}\sum_{j=1}^{N}\sum_{l=1}^{N}\sum_{m=1}^{N}\sin(\theta_j + \theta_l - \theta_m - \theta_i) \tag{B.1}$$

Here K_1 and K_2 are the coupling strengths of the triangular and tetrahedral interactions respectively. We then introduce a complex Order Parameter z_n given by:

$$z_n = r_n e^{i\psi_n} = \frac{1}{N}\sum_{j=1}^{N} e^{in\theta_j} \tag{B.2}$$

which measures the strength of the global synchronization of the oscillators with $0 \leq r \leq 1$, Where $r \sim 0$ indicates a complete incoherent state, whereas $r \sim 1$ indicates global synchronization and ψ measures the mean phase of all the oscillators.
Using n=1:

$$z_1 = r_1 e^{i\psi_1} = \frac{1}{N}\sum_{j=1}^{N} e^{i\theta_j} \tag{B.3}$$

On multiplying both sides with $e^{-i\theta_i}$, we get

$$r_1 e^{i(\psi_1 - \theta_i)} = \frac{1}{N}\sum_{j=1}^{N} e^{i(\theta_j - \theta_i)} \tag{B.4}$$

Multiplying Eq(A.4) with Eq(A.3) with dummy indices l and m, we get

$$r_1^3\ e^{i(\psi_1+\psi_1-\psi_1-\theta_i)} = \frac{1}{N^3}\sum_{j=1}^{N}\sum_{l=1}^{N}\sum_{m=1}^{N} e^{i(\theta_j+\theta_l-\theta_m-\theta_i)} \tag{B.5}$$

which simplifies too

$$r_1^3\ e^{i(\psi_1-\theta_i)} = \frac{1}{N^3}\sum_{j=1}^{N}\sum_{l=1}^{N}\sum_{m=1}^{N} e^{i(\theta_j+\theta_l-\theta_m-\theta_i)} \tag{B.6}$$

Using n=2:

$$z_2 = r_2\ e^{i\psi_2} = \frac{1}{N}\sum_{j=1}^{N} e^{i2\theta_j} \tag{B.7}$$

On multiplying both sides with $e^{-i\theta_i}$ and then multiplying it with eq (A.3) using the dummy index l, we get

$$r_2 r_1 e^{i(\psi_2-\psi_1-\theta_i)} = \frac{1}{N^2}\sum_{j=1}^{N}\sum_{l=1}^{N} e^{i(2\theta_j-\theta_l-\theta_i)} \tag{B.8}$$

Using the result that $sin(\theta) = \frac{e^{i\theta}-e^{-i\theta}}{2i}$, We could open up the equation (A.1) and then substitute the results obtained in equations (A.4),(A.6)and (A.8). We then obtain the mean field form, which is given by:

$$\dot{\theta}_i = \omega_i + K_1\ r_1 \sin(\psi_1-\theta_i) + K_2\ r_1\ r_2 \sin(2\psi_2-\psi_1-\theta_i) + K_3\ r_1^3\ \sin(\psi_1-\theta_i) \tag{B.9}$$

Using the result for $sin(\theta)$ again, we could open up the above equation

$$\begin{aligned}\dot{\theta}_i &= \omega_i + \frac{K_1\ r_1}{2i}(e^{i(\psi_1-\theta_i)} - e^{-i(\psi_1-\theta_i)}) + \frac{K_2\ r_1\ r_2}{2i}(e^{i(2\psi_2-\psi_1-\theta_i)} - e^{-i(2\psi_2-\psi_1-\theta_i)}) \\ &\quad + \frac{K_3\ r_1^3}{2i}(e^{i(\psi_1-\theta_i)} - e^{-i(\psi_1-\theta_i)}) \qquad \text{(B.10)} \\ \dot{\theta}_i &= \omega_i + \frac{1}{2i}[(K_1\ r_1\ e^{i\psi_1} + K_2\ r_2\ e^{i\psi_2}\ r_1\ e^{-i\psi_1} + K_3(r_1\ e^{i\psi_1})^2\ r_1\ e^{-i\psi_1}).e^{-i\theta_i} \\ &\quad - (K_1\ r_1\ e^{-i\psi_1} + K_2\ r_2\ e^{-i\psi_2}\ r_1\ e^{i\psi_1} + K_3(r_1\ e^{-i\psi_1})^2\ r_1\ e^{i\psi_1}).e^{i\theta_i}] \qquad \text{(B.11)}\end{aligned}$$

Using $z_1 = r_1 e^{i\psi_1}$ and $z_2 = r_2 e^{i\psi_2}$ and remembering that z^* is the complex conjugate, we could write down the equation (A.9) as,

$$\dot{\theta}_i = \omega_i + \frac{1}{2i}\left(H.e^{-i\theta_i} - H^*.e^{i\theta_i}\right) \tag{B.12}$$

Where, $H = K_1 z_1 + K_2 z_2 z_1^* + K_3 z_1^2 z_1^*$, we could drop the index i.
In the thermodynamic limit $N \to \infty$, the state of the network can be described by a

density function $f(\theta, \omega, t)$ which measures the density of oscillators with phase θ and $\theta + d\theta$ having natural frequency lying between ω and $\omega + d\omega$ at time t. This density function is defined as

$$\int_{-\infty}^{\infty} \int_{0}^{2\pi} f(\theta, \omega, t)\, d\theta\, d\omega = 1 \tag{B.13}$$

Since the number of oscillators is conserved, the density function will satisfy the continuity equation.

$$0 = \frac{\partial f}{\partial t} + \frac{\partial (f\dot{\theta})}{\partial t} \tag{B.14}$$

The density function can be expanded into Fourier series w.r.t to θ as

$$f(\theta, \omega, t) = A\left[1 + \sum_{n=1}^{\infty} f_n(\omega, t) e^{in\theta} + \sum_{n=1}^{\infty} f_n^*(\omega, t) e^{-in\theta}\right] \tag{B.15}$$

Where A is the normalisation constant and $f_n(\omega, t)$ is the n^{th} Fourier component. let's define $\int_0^{2\pi} f(\theta, \omega, t)\, d\theta = g(\omega)$, where we assume the natural frequency ω of each oscillator is drawn from a distribution $g(\omega)$. On integrating both sides,

$$\int_{-\infty}^{\infty} \int_{0}^{2\pi} f(\theta, \omega, t)\, d\theta\, d\omega = \int_{-\infty}^{\infty} \int_{0}^{2\pi} A\left[1 + \sum_{n=1}^{\infty} f_n(\omega, t) e^{in\theta} + \sum_{n=1}^{\infty} f_n^*(\omega, t) e^{-in\theta}\right] \tag{B.16}$$

$$\begin{aligned}
\int_0^{2\pi} g(\omega) d\omega &= \int_{-\infty}^{\infty} \int_0^{2\pi} A d\omega d\theta + A \int_{-\infty}^{\infty} \int_0^{2\pi} \sum_{n=1}^{\infty} f_n(\omega, t) e^{in\theta} d\omega d\theta \\
&\quad + A \int_{-\infty}^{\infty} \int_0^{2\pi} \sum_{n=1}^{\infty} f_n^*(\omega, t) e^{-in\theta} d\omega d\theta \\
&= \int_{-\infty}^{\infty} \int_0^{2\pi} A d\omega d\theta + A \int_{-\infty}^{\infty} \sum_{n=1}^{\infty} f_n(\omega, t) \int_0^{2\pi} e^{in\theta} d\theta d\omega \\
&\quad + A \int_{-\infty}^{\infty} \sum_{n=1}^{\infty} f_n^*(\omega, t) \int_0^{2\pi} e^{-in\theta} d\theta d\omega \\
&= \int_{-\infty}^{\infty} 2\pi A d\omega
\end{aligned} \tag{B.17}$$

On comparing the LHS and RHS we obtain, $A = g(\omega)/2\pi$. So the Fourier expansion will be,

$$f(\theta, \omega, t) = \frac{g(\omega)}{2\pi}\left[1 + \sum_{n=1}^{\infty} f_n(\omega, t) e^{in\theta} + \sum_{n=1}^{\infty} f_n^*(\omega, t) e^{-in\theta}\right] \tag{B.18}$$

Next, we use the Ott-Antonsen [9] anstanz which assumes all Fourier modes decay geometrically, i.e. $f_n(\omega, t) = (\alpha(\omega, t))^n$ for some α which is analytic in the complex ω plane.

Then,

$$\frac{\partial f}{\partial t} = \frac{g(\omega)}{2\pi}\left(1 + \sum_{n=1}^{\infty} n\alpha^{n-1}(\omega,t)\dot{\alpha}e^{in\theta} + \sum_{n=1}^{\infty} n\alpha^{*n-1}(\omega,t)\dot{\alpha^*}e^{-in\theta}\right) \tag{B.19}$$

$$\frac{\partial f}{\partial \theta} = \frac{g(\omega)}{2\pi}\left(\sum_{n=1}^{\infty} \alpha^{n}(\omega,t)\ in\ e^{in\theta} - \sum_{n=1}^{\infty} \alpha^{*n}(\omega,t)\ in\ e^{-in\theta}\right) \tag{B.20}$$

$$\frac{\partial \dot{\theta}}{\partial \theta} = \frac{-1}{2i}\left(iHe^{-i\theta} + iH^*e^{i\theta}\right) \tag{B.21}$$

We then substitute this density function in the continuity equation,

$$\frac{g(\omega)}{2\pi}\left(1 + \sum_{n=1}^{\infty} n\ \alpha^{n-1}(\omega,t)\ \dot{\alpha}\ e^{in\theta} + \sum_{n=1}^{\infty} n\ \alpha^{*n-1}(\omega,t)\ \dot{\alpha^*}(\omega,t)\ e^{-in\theta}\right) + \tag{B.22}$$

$$\frac{g(\omega)}{2\pi}\left(\sum_{n=1}^{\infty} \alpha^{n}(\omega,t)\ in\ e^{in\theta} - \sum_{n=1}^{\infty} \alpha^{*n}(\omega,t)\ in\ e^{-in\theta}\right).\left(\omega + \frac{1}{2i}\left(H\ e^{-i\theta_i} - H^*\ e^{i\theta_j}\right)\right)$$

$$-\frac{1}{2}\left(H\ e^{-i\theta} + H^*\ e^{i\theta}\right).\frac{g(\omega)}{2\pi}\left[1 + \sum_{n=1}^{\infty} \alpha^{n}(\omega,t)\ e^{in\theta} + \sum_{n=1}^{\infty} \alpha^{*n}(\omega,t)\ e^{-in\theta}\right] = 0$$

As the RHS is 0, the coefficients of the exponent terms should also add up to zero. So, on opening up the brackets and comparing the coefficients of $e^{i\theta}$, The dynamics of the N-dimensional Kuramoto model collapses into a single differential equation,

$$\dot{\alpha} = -i\omega\alpha + \frac{1}{2}\left[H^* - H\alpha^2\right] \tag{B.23}$$

where $H = K_1 z_1 + K_2 z_2 z_1^* + K_3 z_1^2 z_1^*$. In the thermodynamic limit, the order parameter can be written as $z_1 = r_1\ e^{i\psi_1} = \int\int f(\theta,\omega,t)e^{i\theta}\,d\theta\,d\omega$, which after inserting the Fourier decomposition of f becomes,

$$z_1 = \int_{-\infty}^{\infty} \alpha^*(\omega,t)g(\omega)\ d\omega \tag{B.24}$$

If we choose $g(\omega)$ to be a Lorentzian frequency distribution,

$$g(\omega) = \frac{\Delta}{\pi[\Delta^2 + (\omega - \omega_0)^2]} \tag{B.25}$$

where, ω_0 is the mean frequency and we take it to be 0, Δ is a constant.

$$z_1 = \int_{-\infty}^{\infty} \alpha^*(\omega, t) \frac{\Delta}{\pi[\Delta^2 + \omega^2]}\, d\omega \tag{B.26}$$

$$= \int_{-\infty}^{\infty} \alpha^*(\omega, t) \frac{\Delta}{\pi(\Delta + i\omega)(\Delta - i\omega)} \tag{B.27}$$

On applying contour integration in the negative half-contour plane of ω, we obtain

$$z_1 = \frac{\Delta}{\cancel{\pi}}\, 2\cancel{\pi i} \lim_{z \to -i\Delta} \cancel{(\Delta + i\omega)}\, \frac{\alpha^*(\omega, t)}{(\Delta - i\omega)\cancel{(\Delta + i\omega)}} \tag{B.28}$$

$$= \cancel{2i\Delta}\, \frac{\alpha^*(-i\Delta, t)}{\cancel{2i\Delta}} = \alpha^*(-i\Delta, t) \tag{B.29}$$

So we obtain, $z_1 = r_1 e^{i\psi_1} = \alpha^*(-i\Delta, t)$ thus $z_1^* = r_1 e^{-i\psi_1} = \alpha(-i\Delta, t)$. Similarly, if we do the same steps for $z_2 = r_2 e^{i\psi_2}$ we would obtain$z_2 = r_2 e^{i\psi_2} = (\alpha^*(-i\Delta, t))^2 = (r_1 e^{i\psi_1})^2$.Taking $\Delta = 1$, and dropping the index as all are same. We obtain,

$$\dot{(re^{-i\psi})} = -i\omega r e^{-i\psi} + \frac{1}{2}\left[H^* - H r^2 e^{-i2\psi}\right] \tag{B.30}$$

$$\begin{aligned} \dot{r}e^{-i\psi} - ire^{-i\psi}\dot{\psi} = -i\omega r e^{-i\psi} + \frac{1}{2}[K_1\, r e^{-i\psi} + K_2 r^2 e^{-i\psi} + K_3 r^3 e^{-i\psi} \\ - r^2 e^{-2i\psi}(K_1\, r e^{i\psi} + K_2 r^2 e^{i\psi} + K_3 r^3 e^{i\psi})] \end{aligned} \tag{B.31}$$

$$\dot{r} - ir\dot{\psi} = -i\cancel{\omega r} + \frac{-i}{2}[K_1\, r + K_2\, r^3 + K_3\, r^3 - r^3\, K_1 + K_2\, r^5 + K_3\, r^5] \tag{B.32}$$

$$\dot{r} - ir\dot{\psi} = -r + \frac{K_1}{2}(r - r^3) + \frac{K_2 + K_3}{2}(r^3 - r^5) \tag{B.33}$$

As there is i in the LHS but not in the RHS, $r\dot{\psi} = 0$, but as r can't be zero, $\dot{\psi} = 0$, so we get the final differential equation for the real order parameter as

$$\boxed{\dot{r} = -r + \frac{K_1}{2}(r - r^3) + \frac{K_2 + K_3}{2}(r^3 - r^5)} \tag{B.34}$$

References

[1] D. Sisodia, S. Jalan, "Dynamical Analysis of a Parameter-Aware Reservoir Computer", *Physical Review E*, accepted for publication on September 12, 2024.

[2] T. L. Carroll, "Using reservoir computers to distinguish chaotic signals", *Physical Review E*, vol. 98, no. 5, Article 052209, 2018.

[3] S.H. Strogatz, *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*, 2nd ed., Westview Press, 2015.

[4] D. J. Gauthier, E. Bollt, A. Griffith, and W. A. S. Barbosa, "Next generation reservoir computing", *Nature Communications*, vol. 12, Article 5564, 2021.

[5] L. Appeltant, M. C. Soriano, G. Van der Sande, J. Danckaert, S. Massar, J. Dambre, B. Schrauwen, C. R. Mirasso, and I. Fischer, "Information processing using a single dynamical node as complex system", *Nature Communications*, vol. 2, Article 468, 2011.

[6] M. Cucchi, S. Abreu, G. Ciccone, D. Brunner, and H. Kleemann, "Hands-on reservoir computing: a tutorial for practical implementation", *Neuromorphic Computing and Engineering*, vol. 2, no. 3, Article 032002, 2022.

[7] P. S. Skardal, A. Arenas, "Higher order interactions in complex networks of phase oscillators promote abrupt synchronization switching", *Communications Physics*, vol. 3, Article 218, 2020.

[8] 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*, vol. 27, no. 12, Article 121102, 2017.

[9] E. Ott, T. M. Antonsen, "Low dimensional behavior of large systems of globally coupled oscillators", *Chaos*, vol. 18, no. 3, Article 037113, 2008.

[10] M. A. Nielsen, *Neural Networks and Deep Learning*, Determination Press, 2015.

[11] Y. A. Kuznetsov, *Elements of Applied Bifurcation Theory*, 2nd ed., Applied Mathematical Sciences, vol. 112, Springer, 1998.