MAS6600 Dissertation

Chaos Synchronization and Secure Communication

Submitted by Rashmi Sarwal

Supervisor: Dr. Yi Li

Department of Mathematics and Statistics

March 18, 2024

Acknowledgement

I would like to express my heartfelt gratitude to The University of Sheffield for providing me with the opportunity to pursue my dissertation. This journey would not have been possible without their support and resources.

I am deeply thankful to my supervisor, Dr. Yi Li, for his invaluable guidance, mentorship, encouragement and unwavering support throughout the dissertation period.

I would also like to extend my profound appreciation to Dr. Sam Dolan, Prof. Elizabeth Winstanley, Dr. Paul Mitchener and Dr. Ashley Willis, faculty members at The University of Sheffield for their contributions to my dissertation.

I am also very grateful to my family and friends for their unwavering encouragement and support during this endeavour.

Thank you very much, everyone.

Rashmi Sarwal

Abstract

This study highlights the significance of chaos synchronization as a bridge between chaos theory and practical applications across diverse fields, encompassing both its theoretical foundations and real-world relevance. While investigating various forms of chaos synchronization, the aim is to replicate some results from reference [1], focusing on Complete Synchronization and Phase Synchronization. Later, we will also explore Secure Communication models using different types of synchronization as an application of chaos synchronization.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The world is driven by the complex interplay of dynamical systems, often described by nonlinear equations. Consequently, it becomes essential to deepen our understanding of these equations if we seek insights into the functioning of natural processes in the universe. The intricate nature of nonlinear systems motivated researchers to see nonlinear problems from a quite new perspective. This was introduced by Poincarè through his work on the three-body problem in the late 1800s. He emphasized the qualitative aspects of the system over quantitative questions.

1.1 Nonlinear Dynamics

Nonlinearity has a significant impact on the dynamics of the system. In contrast to linear equations, nonlinear equations allow distinct solutions for the oscillations about equilibrium that are aperiodic in nature. Analytically, this makes nonlinear equations significantly harder to solve. As a result, numerical methods are used to obtain approximate answers, while geometrical reasoning is found quite effective in getting a qualitative understanding of the behaviour of solutions [4].

Mathematically, dynamical systems are classified into two main types: ***differential equations*** (also known as flows) and ***difference equations*** (also known as iterated maps). Differential equations deal with the evolution of dynamical systems in continuous times. We usually express the evolution using differential equations as:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}), \tag{1.1}$$

where $\boldsymbol{f}$ is the *flow*. On the other hand, difference equations are used in problems where time is discretized. For the discrete time 't', we have,

$$\boldsymbol{x}_{t+1} = \boldsymbol{g}(\boldsymbol{x}_t), \tag{1.2}$$

where $\boldsymbol{g}$ is a *map*. These maps are very useful for analyzing the periodic or chaotic solutions of differential equations. They provide an easier way to study complex dynamical systems by simplifying them into simpler systems through the construction of lower-dimensional discrete-time systems from the original systems while preserving the essential properties of the original systems. For instance, in the context of a continuous-time dynamical system, the use of the ***Poincarè map*** (also known as *first return map*) within a lower-dimensional manifold can provide valuable insights into the system. These maps are derived from the *Poincarè section*. A return map is a mathematical representation that relates points in the Poincarè section at successive crossings of a trajectory with that section. In simple words, it describes how the system evolves from one intersection point to the next [2, 5, 6].

1.1.1 The Phase Space, Critical Points and Stability

A dynamical system is described by its ***phase space*** (the set of all possible states of x) and its evolution process (differential equations/difference equations), which describes how the point x evolves in the phase space as time 't' passes. Here, to understand the concept we are going to use a continuous dynamical system as an example. A general dynamical system can be written as a set of ordinary differential equations, we can expand Eq (1.1) as

$$\begin{aligned} \dot{x}_1 &= f_1(x_1, ..., x_n), \\ &. \\ &. \\ &. \\ \dot{x}_n &= f_n(x_1, ..., x_n). \end{aligned} \tag{1.3}$$

The phase space for the general dynamical system comprises coordinates x_1,....,x_n and due to its inherent n-dimensionality, we can designate Eq (1.1) as n-dimensional system or an nth-order system. Thus, n denotes the dimension of the phase space. The solution $(x_1(t), ..., x_n(t))$ corresponds to a point moving along a curve in this space, which is referred to as ***trajectory*** and a representative set of trajectories is known as a ***phase portrait***. Since every point within phase space can serve as an initial condition for a dynamical system, we can say the entire phase space is filled with trajectories. Also, the points for which ***f(x)*** in Eq (1.1) is zero are known as **equilibrium** (or **constant**) **solutions**, often called **critical** (or **fixed**) **points**. Also, to get a better geometrical understanding of the flow, we interpret these differential equations as a *vector field* [2, 3, 5].

Note: *Eq (1.3) is general for all types of systems including nonautonomous (time-dependent) equations. Just there is a key trick which can be easily understood by the following example. Let us consider the equation for a damped harmonic oscillator:* $m\ddot{x} + c\dot{x} + kx = F(t)$. *Here, we let* $x_1 = x$ *and* $x_2 = \dot{x}$ *as usual but now we introduce* $x_3 = t$. *This makes the system equivalent to the 3-dimensional autonomous system. Thus, an nth-order time-dependent equation is a particular case of (n + 1)-dimensional system. This change of variables is useful because it lets us see a phase space with stable trajectories. If we included explicit time dependence, the vectors and paths would continuously change, making it difficult to build the clear geometric picture we want [2].*

The fundamental structure of the flow can undergo alterations as parameters are adjusted. Specifically, fixed points can be either generated or eliminated, and their stability characteristics may undergo modifications. These qualitative shifts in the dynamics of the dynamical system are termed as ***bifurcations***, and the parameter values at which these shifts occur are referred to as ***bifurcation points***.

Now, before we move on to a specific type of nonlinear system, i.e., a chaotic system, which is the primary focus of this dissertation, we will look into some significant results of linear equations that are extremely helpful in understanding nonlinear systems using linear approximations. Here, we are taking a second-order linear homogeneous system with constant coefficients. This type of system takes the form,

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x}, \tag{1.4}$$

where $\boldsymbol{A}$ is a 2×2 constant matrix and $\boldsymbol{x}$ is a 2×1 vector. The solution for such type of equations is of the form $\boldsymbol{x} = \boldsymbol{\xi}e^{rt}$. So, the corresponding eigenvector equation for it is,

$$(\boldsymbol{A} - r\boldsymbol{I})\boldsymbol{\xi} = 0, \tag{1.5}$$

where $\boldsymbol{\xi}$ is an eigenvector and r is an eigenvalue and $\boldsymbol{A}$ is the coefficient matrix. Thus, the eigenvalues correspond to the roots of the polynomial equation,

$$det(\boldsymbol{A} - r\boldsymbol{I}) = 0, \tag{1.6}$$

and eigenvectors are calculated using Eq (1.5) till an arbitrary multiplicative constant. Similarly, here also points for which $\boldsymbol{Ax}=\boldsymbol{0}$ correspond to the equilibrium solutions and are also referred to as critical points. Now, let's assume that $\boldsymbol{A}$ is a non-singular matrix, meaning that $det(\boldsymbol{A}) \neq 0$. Consequently, the only critical point of the system Eq (1.4) is $x = 0$. We will now characterize the differential equation

and look into different types of critical points by examining system Eq (1.4), and this characterization will depend on the nature of the eigenvalues of $\boldsymbol{A}$, helping us to understand its phase portrait (the geometric patterns formed by the trajectories). For this, we will refer to Table (1.1) and Figure (1.1) below which will also provide us with information on the stability of those trajectories [3]:

Table 1.1: Characterization of the linear system $\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x}$ with $det(\boldsymbol{A} - r\boldsymbol{I}) = 0$ and $det(\boldsymbol{A}) = 0$ using different cases for eigenvalues.

Eigenvalues	**Critical Point**	**Stability**
$r_1 > r_2 > 0$	Node (Nodal Source or *repeller*)	Unstable
$r_1 < r_2 < 0$	Node (Nodal Sink or *attractor*)	Asymptotically Stable
$r_2 < 0 < r_1$	Saddle Point	Unstable
$r_1 = r_2 > 0$	Proper or Improper Node	Unstable
$r_1 = r_2 < 0$	Proper or Improper Node	Asymptotically Stable
$r_1, r_2 = \lambda \pm i\mu$	Spiral Point	
$\lambda > 0$		Unstable
$\lambda < 0$		Asymptotically Stable
$r_1 = i\mu, r_2 = -i\mu$	Center	Stable

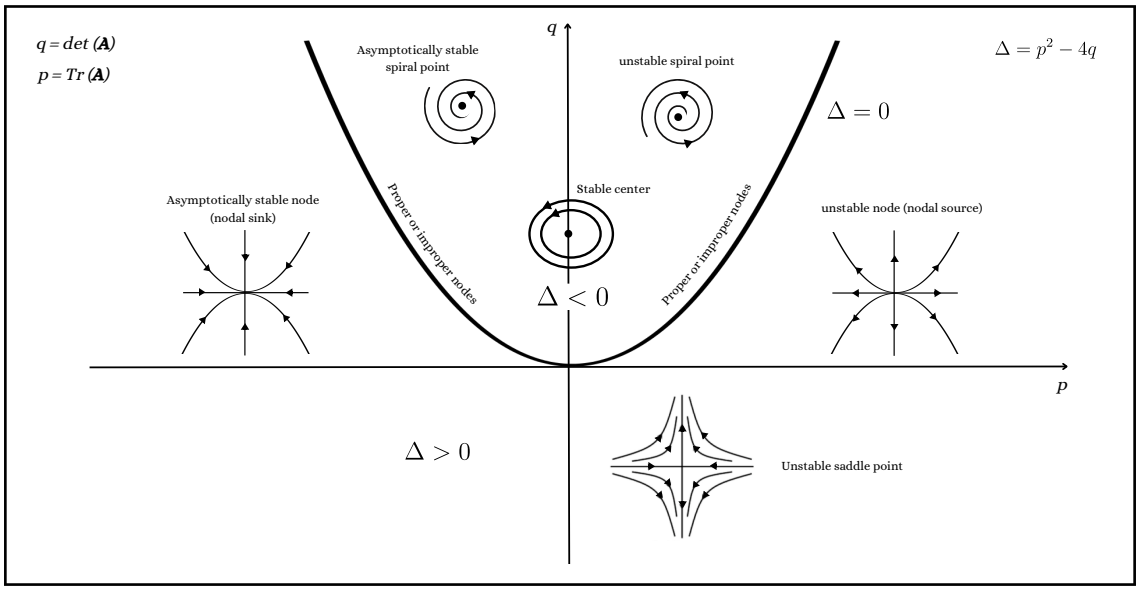


Figure 1.1: *(Original image) Stability diagram or Poincaré diagram: the classification of phase portrait [2, 3].*

Stability: As mentioned earlier, critical points correspond to equilibrium solutions of the system of differential equations. A critical point x_0 of the Eq (1.1) is considered to be ***stable*** if, given any $\epsilon > 0$, there exist a $\delta > 0$ such that every solution $\boldsymbol{x} = \boldsymbol{\phi}(t)$ of the Eq (1.3), which at t = 0 satisfies

$$||\phi(0) - x_0|| < \delta, \tag{1.7}$$

exists for all positive t and satisfies

$$||\phi(t) - x_0|| < \epsilon, \tag{1.8}$$

for all $t \geq 0$. Mathematically, this tells us all solutions that start close enough to x_0 i.e., within the distance δ stay close to x_0 i.e., within the distance ϵ. All the critical points that are not stable are said to be ***unstable*** while a critical point x_0 is said to be ***asymptotically stable*** if it is stable and if there exists δ_0 with $0 < \delta_0 < \delta$, such that if a solution $\boldsymbol{x} = \boldsymbol{\phi}(t)$ satisfies,

$$||\phi(0) - x_0|| < \delta_0, \tag{1.9}$$

then

$$\lim_{t\to\infty} \boldsymbol{\phi}(t) = x_0, \tag{1.10}$$

This means trajectories that start close enough to x_0 must not just remain close, but must eventually approach x_0 as $t \to \infty$. Let S be the set of all points for such trajectories which are attracted by the critical point. The set S is said to be ***basin of attraction*** or ***region of asymptotic stability*** of the critical point. Thus, we define the *basin of attraction* as the region in phase space where initial conditions will lead to a specific attractor [2, 3].

Lyapunov function method: A fundamental strategy for addressing the stability of nonlinear systems involves the utilization of the Lyapunov function method. This method is also known as the direct method since it does not require any prior knowledge of the solution of the system of differential equations. In other words, this method provides a means to determine stability without the necessity of tracing the trajectories of the system, as it involves analyzing the properties of the Lyapunov function [1]. Let's consider an autonomous dynamical system governed by differential equations,

$$\begin{aligned} \dot{x_1} &= F(x_1, x_2), \\ \dot{x_2} &= F(x_1, x_2) \end{aligned} \tag{1.11}$$

Suppose that the point $(x_1 = 0, x_2 = 0)$ is an asymptotically stable critical point such that there exists a domain D containing (0,0) where every trajectory originating

within D ultimately converges to the origin as $t \to \infty$. Now, let's introduce a function V that satisfies the following conditions: $V \geq 0$ for all (x, y) in D, and $V = 0$ only at the origin. Given that each trajectory within D approaches the origin as $t \to \infty$ we observe that, while following any specific trajectory, the function V tends to zero as t increases. But the crucial point is its converse which was proved by Lyapunov: If, for every trajectory, V consistently tends to zero as t increases, then these trajectories will finally approach the origin as $t \to \infty$. Therefore, we can say that the origin is asymptotically stable. There are two theorems (Theorem 1 and Theorem 2) given by Lyapunov regarding stability and instability but before that, we must define the rate of change of V along the trajectory of the system as $\dot{V}$ [3],

$$\dot{V}(x_1, x_2) = V_{x_1}(x_1, x_2)F(x_1, x_2) + V_{x_2}(x_1, x_2)F(x_1, x_2) \tag{1.12}$$

where V_{x_1} and V_{x_2} are partial differentiation along x_1 and x_2 direction respectively.

Theorem 1. *Assume the autonomous system Eq (1.11) possesses an isolated critical point situated at the origin. If there exists a continuous function V with continuous first partial derivatives, which is positive definite, and if the function $\dot{V}$ as defined by Eq (1.12) is negative definite within a certain domain D in the xy-plane containing the point $(0, 0)$, then the origin is regarded as an asymptotically stable critical point. On the other hand, if $\dot{V}$ is negative semidefinite, then the origin is considered a stable critical point [3].*

Theorem 2. *Suppose we have an isolated critical point at the origin within the autonomous system (6). Let V be a continuous function with continuous first partial derivatives such that V(0, 0) = 0, and in the neighbourhood of the origin, there is at least one point where V is either positive or negative. If there exists a domain D containing the origin such that the function $\dot{V}$, as defined by Eq (1.12), exhibits positive definiteness or negative definiteness within D, then we can conclude that the origin represents an unstable critical point [3].*

1.1.2 Limit Cycle and Strange Attractor

The periodic solutions of second-order autonomous systems Eq (1.1) satisfies,

$$x(t + T) = x(t), \tag{1.13}$$

for all time t, where T is the *period* which is a non-negative constant and the corresponding trajectories for it are *closed curves* in the phase plane. Also, a critical point (equilibrium solution) for the autonomous system is a special case of periodic solutions. A periodic solution is a closed trajectory in the phase plane. A closed trajectory within the phase plane such that other neighbouring non-closed trajectories spiral toward or away from it, as $t \to \infty$, is called a ***limit cycle***.

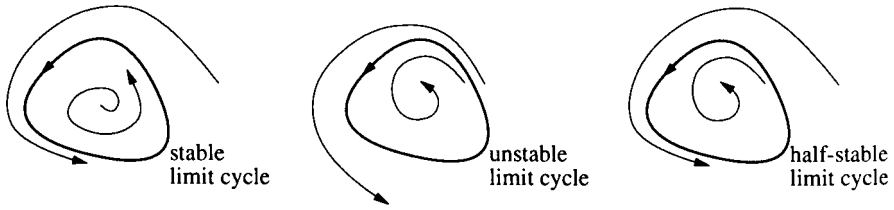


Figure 1.2: *Limit Cycle [2].*

If all nearby trajectories lead to the limit cycle, it is called a *stable* or *attracting* limit cycle. If not, it's *unstable*, or in some rare cases, *half-stable*. Limit cycles are fundamentally nonlinear phenomena. In the case of linear systems, closed orbits can exist, but neighbouring trajectories in such systems do not spiral towards or away from the limit cycle.

Unlike the limit cycle, there is a ***strange attractor*** which is a subset of the phase space of the dynamical system characterized by its complex, non-periodic, and unpredictable behaviour. Trajectories near a strange attractor remain confined to its vicinity over time, exhibiting sensitive dependence towards initial conditions. These attractors have intricate, often fractal-like geometric structures [2, 7]. A strange attractor is at the heart of chaos in a dynamical system, but there are certain types of dynamical systems for which it is possible to have attractors which are strange but not chaotic [8]. Thus, we need a proper measure to identify chaotic behaviour which we will explore in the Subsection (1.1.3). In a chaotic system with a strange attractor, the system can dissipate disturbances at a much faster rate. These systems exhibit high sensitivity to initial conditions, which might initially suggest an inability to dissipate disturbances. However, when the system possesses a strange attractor it makes all trajectories functionally acceptable, this initial-condition sensitivity becomes the most efficient means of dissipating disturbance [7].

Note: *A system evolving in a limit cycle can change to a chaotic attractor due to modification in parameters. The common route to chaos in nonlinear dynamical systems is mostly due to* ***period-doubling bifurcation*** *of limit cycles. As a parameter is gradually changed, the period of an oscillating system can double repeatedly, eventually leading to chaotic behaviour. We will see an example of such behaviour in Section (2.2) [4].*

1.1.3 Chaos and Lyapunov Exponent

Chaos in nature often manifests as a fascinating interplay between deterministic systems and apparent randomness. It implies the unpredictable nature of a deterministic system which has a sensitive dependence on the initial conditions of the system. It indicates if we have similar systems with slightly perturbed initial conditions, the trajectories of both systems will diverge from each other exponentially [9, 10]. Chaotic systems are nonlinear dynamical systems or infinite-order linear systems which are bounded in phase space. In the context of these chaotic systems, the phase space plot exhibits a distinctive behaviour. Instead of converging to closed curves or settling into equilibrium, they follow intricate and entangled trajectories that never repeat themselves. While, in non-chaotic systems nearby paths either converge very quickly or, at worst, move apart more slowly. This means that, in theory, we can make predictions about what will happen over a long time but this is not the case with chaotic systems [7].

Lyapunov Exponents are the rates of the orbital divergence or convergence of nearby trajectories. They measure the exponential growth rates of infinitesimal perturbations, sometimes also known as Lyapunov Characteristic Numbers. The values of the Lyapunov Exponents play an important role in determining chaos. *Positive Lyapunov Exponents* imply orbital divergence and chaos, they determine the time frame in which predicting the state of the system is viable. If one or more of the Lyapunov Exponents for the system are positive then the system is said to be ***chaotic***. On the other hand, *Negative Lyapunov Exponents* establish the time frame for the decay of transients or state perturbations. If all the Lyapunov Exponents are negative for the dynamical system, then the system shows non-chaotic behaviour. Lyapunov Exponent for the dynamical system can be calculated using one-dimensional maps Eq (1.2). Let $\boldsymbol{x}_0$ be the initial condition and let $\boldsymbol{x}_0 + \boldsymbol{\delta}_0$ be a neighbouring point with $\boldsymbol{\delta}_0$ as the very small initial separation. Now, let $\boldsymbol{\delta}_n$ be separation for n iteration where $\boldsymbol{\delta}_n = f^n(\boldsymbol{x}_0 + \boldsymbol{\delta}_0) - f^n(\boldsymbol{x}_0)$. Suppose $|\boldsymbol{\delta}_n| \approx |\boldsymbol{\delta}_0| e^{\lambda n}$, then λ is the ***Leading*** (or ***Maximum***) ***Lyapunov Exponent*** where,

$$\lambda = \lim_{n\to\infty} \left\{ \frac{1}{n} \sum_{i=0}^{n-1} \ln |f'(x_i)| \right\} \tag{1.14}$$

We are interested in Leading Lyapunov Exponent (LLE) for the system because being the largest it dominates over other Lyapunov Exponents and decides the nature of the system. For stable fixed points and cycles $\lambda < 0$ and for chaotic attractors $\lambda > 0$. There is one more way to define Lyapunov Exponents. When dealing with a nonlinear equation, we can examine the characteristics of a specific solution by approximating the dynamics through linearization around the trajectory. This gives

us a linear nonautonomous equation,

$$\dot{\boldsymbol{u}} = \boldsymbol{J}(t)\boldsymbol{u}, \tag{1.15}$$

where $\boldsymbol{u}$ is an $\boldsymbol{N}$ dimensional vector and $\boldsymbol{J}$ is a time-dependent N×N constant matrix. Generally, for deterministic dynamical systems, $\boldsymbol{J}$ is the ***Jacobian*** of an appropriate velocity field $\boldsymbol{F}$, determined along a trajectory $\boldsymbol{x}(t)$ that satisfies the ordinary differential equation,

$$\dot{\boldsymbol{x}} = \boldsymbol{F}(\boldsymbol{x}), \tag{1.16}$$

Let $\boldsymbol{x}(t) = \boldsymbol{x}_0$ be a solution such that $\boldsymbol{F}(\boldsymbol{x_0}) = \boldsymbol{0}$, then the eigenvalues of the Jacobian matrix $\boldsymbol{J}$ quantify the stability of this fixed point. Here, the *Lyapunov Exponents λ_i are the real parts of the eigenvalues.* In a chaotic system, there are as many Lyapunov Exponents as the dimension (d) of the phase space, known as ***Spectrum of Lyapunov Exponents*** defined by $\lambda_1, \lambda_2, ..., \lambda_d$. Lyapunov Exponents are *invariant* of the system because it is independent of the initial and final states of the system due to the limit $t \to \infty$ [5, 7, 11, 12, 13].

1.2 Synchronization

Synchronization is a phenomenon that appears intrinsic to the universe. It reveals itself in a multitude of natural occurrences, illustrating the intrinsic order of our world. From the mesmerizing coordination of thousands of fireflies, flashing in perfect unison, to the rhythmic pulses of our hearts, regulated by hundreds of pacemaker cells, and even in the complex alignment of neural activities, synchronization emerges as a fundamental principle. One particularly clear example can be seen in the harmonic marching of trillions of electrons within a superconductor, allowing electricity to flow through the material without encountering any resistance. Various such examples show how deeply synchronization is embedded in the universe and how synchronicity arises spontaneously in complex systems, providing us with a glimpse into the intricate and harmonious workings of the natural world [14].

In terms of classical physics, synchronisation is the coincidence in the rhythms of autonomous periodic systems caused by the weak interaction between them. It is coordination in the time of different processes. The moderation in the behaviour of the oscillating system to synchronise with another system can be explained in terms of phase locking and frequency locking [15].

Synchronization is also crucial in the world of communication and information technology. When a phone call is made or a message is sent over the internet, our device relies on precise synchronization of data transmission. Without synchronization making data communication is impossible [16].

There are various types of synchronization which are being observed and studied. When two or more identical coupled systems synchronize completely, it is referred to as complete synchronization or Identical synchronization. Phase synchronization occurs when the coupled systems maintain a constrained phase difference while keeping their amplitudes uncorrelated. Whenever there is a time lag in synchronized behaviour between coupled systems that is the case of lag synchronization. Finally, there is generalized synchronization in which we have synchronized behaviour between non-identical systems that are related by a function [17]. Many complex dynamical systems synchronize so why it's unlikely for chaotic systems to synchronise? This is due to their positive Lyapunov Exponents (Subsection 1.1.3) because of which isolated chaotic systems resist synchronization with anything and the variations in the system states increase exponentially with time. Therefore, the synchronization of chaotic systems already become quite intriguing.

Chapter 2

Lorenz and Rössler Chaotic Systems

Chaotic systems are dynamic systems with high complexity. These systems are characterized by their high sensitivity to initial conditions, leading to unpredictable behaviour over time. The thing which makes these systems intriguing is that they demonstrated that even simple mathematical models can exhibit extraordinarily complex and irregular behaviour. Two such well-known chaotic mathematical models are the Lorenz and the Rössler systems, which we will go through in this chapter.

2.1 Lorenz System

The Lorenz system was introduced by meteorologist Edward Lorenz in the 1960s during his research on atmospheric convection. He developed this simplified dissipative model for general atmosphere circulation which concerns the movement of a fluid layer as in Earth's atmosphere which is warmer at the bottom as compared to the top.

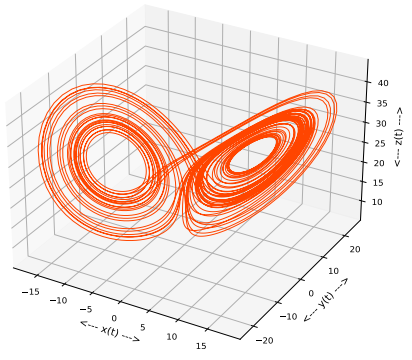


Figure 2.1: *(Original image) Lorenz Attractor in 3-dimensional phase space.*

He explained it using 3-dimensional ordinary nonlinear differential equations:

$$\begin{aligned} \dot{x} &= \sigma(y - x), \\ \dot{y} &= x(\rho - z) - y, \\ \dot{z} &= -\beta z + xy, \end{aligned} \tag{2.1}$$

where σ, ρ and β are parameters of the dynamical systems which depend on the geometrical and materialistic properties of the fluid layer and x, y, and z are the coordinates of the state of the system. Lorenz chose σ=10, ρ=28 and β =8/3

according to the earth's atmosphere for which the system showed chaotic behaviour [2, 3, 18]. When the solution was visualized as a trajectory in phase space, a beautiful butterfly pattern appeared, known as *Lorenz attractor* (strange attractor for Lorenz system).

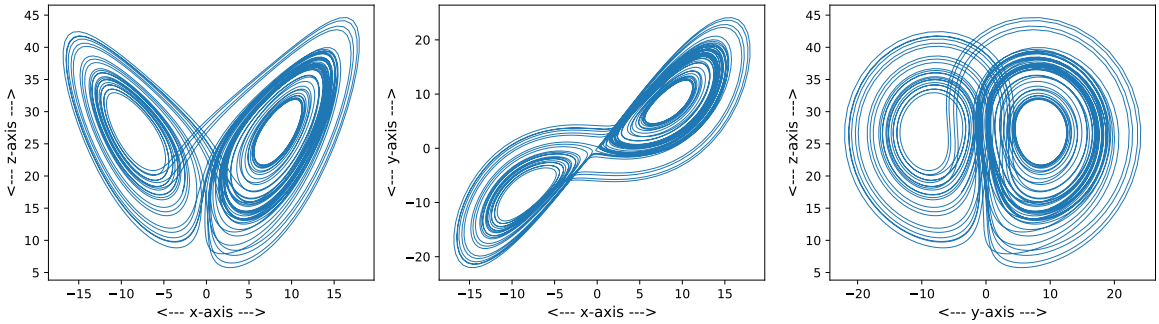


Figure 2.2: *(Original image) Projection of the strange attractor for the Lorenz system in x-z, x-y and y-z plane respectively.*

Now, we will analyse the Lorenz equations using the concepts from Section (1.1). To calculate the critical points we will make the left-hand side of Eq (2.1) equal to zero,

$$\begin{aligned} \sigma(y - x) &= 0, \\ x(\rho - z) - y &= 0, \\ -\beta z + xy &= 0. \end{aligned} \tag{2.2}$$

From the first equation of Eq (2.2) we get $y = x$. Then by substituting $y = x$ in other two equations in Eq (2.2) we get,

$$x(\rho - z - 1) = 0, \tag{2.3}$$

$$-\beta z + x^2 = 0. \tag{2.4}$$

One solution of Eq (2.3) is $x = 0 \implies y = 0$ and thus, from Eq (2.4), $z = 0$. Therefore, $(0, 0, 0)$ is one critical point and let's denote it by O. Another solution for Eq (2.3) is $z = \rho - 1$. Thus, from Eq (2.4) we get $x = \pm\sqrt{b(\rho - 1)}$ which implies $y = \pm\sqrt{b(\rho - 1)}$ and we get two critical points let's say C_1 and C_2 respectively, $(\sqrt{b(\rho - 1)}, \sqrt{b(\rho - 1)}, \rho - 1)$ and $(-\sqrt{b(\rho - 1)}, -\sqrt{b(\rho - 1)}, \rho - 1)$. We can notice that x and y are real only for $\rho \geq 1$ and there is only one critical point O for $\rho < 1$. The point to notice is that at $\rho = 1$ all three critical points coincide and as the value of ρ increases beyond the value 1, the critical point O bifurcates at the origin which leads to the existence of the other two critical points C_1 and C_2.

For the critical point O all the eigenvalues of the system are negative for $\rho < 1$ thus, from Table (1.1) we can say critical point O which is the origin here, is asymptotically stable for $\rho < 1$ and all solutions will approach the origin as $t \to \infty$. But at $\rho = 1$ the origin becomes unstable which depicts the starting of the convective flow because one of the eigenvalues is positive and it continues for $\rho > 1$. Therefore, for $\rho > 1$ the origin (O) is unstable. But for $\rho > 1$ the other two critical points, C_1 and C_2 are asymptotically stable which means all the nearby solutions will approach one of these two critical points exponentially. And for $\rho > \rho^*(\approx 24.737)$ all three critical points are unstable and most of the solutions near the critical points C_1 and C_2 spiral away from them [3, 7].

2.2 Rössler System

Rössler system is even simpler than the Lorenz system with only one nonlinear term yet shows quite complex behaviour. This system was introduced by Otto Rössler in 1976 and explained using the following ordinary differential equations:

$$\begin{aligned} \dot{x} &= -y - z, \\ \dot{y} &= x + ay, \\ \dot{z} &= b + z(x - c), \end{aligned} \tag{2.5}$$

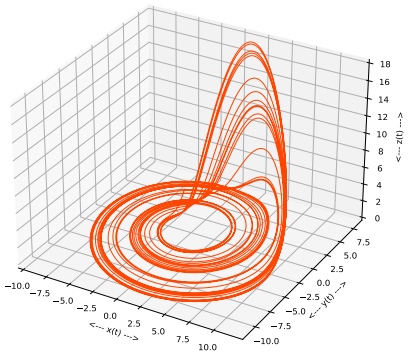


Figure 2.3: *(Original image) Rössler Attractor in 3-dimensional phase space.*

where a, b, and c are parameters of the Rössler system and x, y and z are the coordinates representing the state of the system. When the parameter c is set to 2.3, the system exhibits a simple limit cycle as its attractor, shown in Figure (2.4(a)). However, as we increment the parameter c to reach 2.9 an interesting transformation occurs, the previously observed limit cycle now completes two full cycles before returning to its initial state and its period is approximately twice the original cycle, shown in Figure (2.4(b)). This distinctive behaviour is the *period-doubling bifurcation of the limit cycles* mentioned in the Subsection(1.1.2). It signifies that at some point within the parameter range from $c = 2.3$ to $c = 2.9$, a period-doubling bifurcation event must have occurred.

Finally, after an infinite number of successive period-doubling, we get the strange attractor for the Rössler system at $c = 5.7$, shown in Figure (2.4(d)). Because of the presence of oscillatory components, Rössler system is one of those chaotic attractors which can undergo phase synchronization [2, 19].

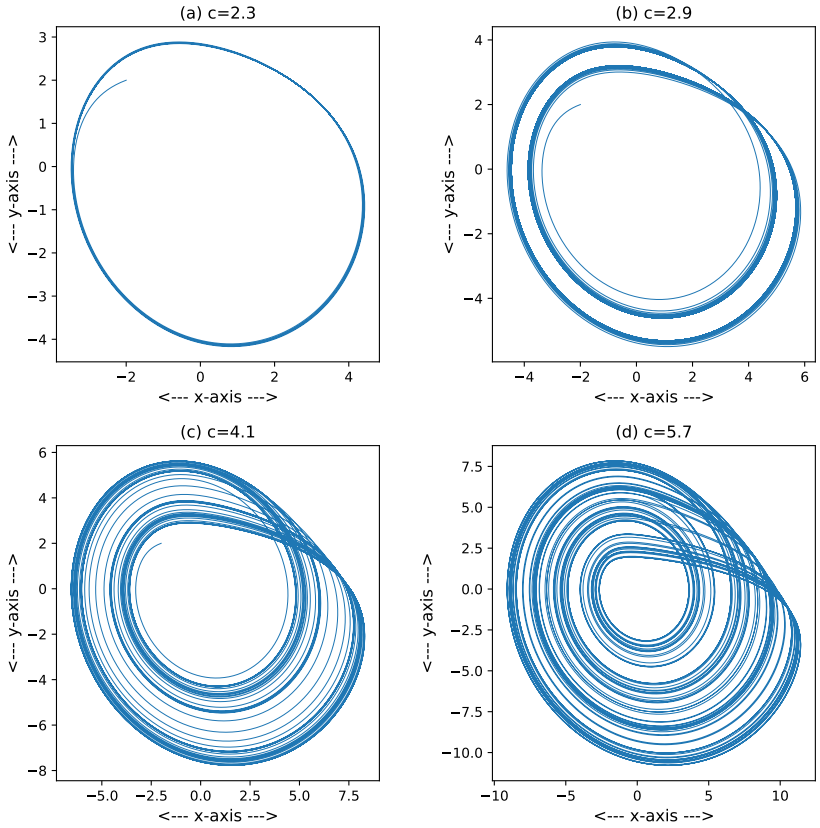


Figure 2.4: *(Orignal image) Projection of the attractor for the Rössler system in x-y plane for difference values of parameter c. For c=2.3, we found a limit cycle as the attractor. For c=2.9 we can see the period-doubling bifurcation of the limit cycle which leads to the strange attractor of the Rössler system at c=5.7.*

Chapter 3

Chaos Synchronization

Chaos and synchronization are both counter-intuitive concepts on their own. Synchronization discusses the synchronized behaviour of coupled systems, in contrast to the chaos, which portrays total disorder and unpredictable behaviour of complex systems. However, that two or more chaotic systems may be brought into synchronization is quite astounding. Synchronization of chaotic systems is one of the fundamental aspects of nonlinear dynamics, which is the convergence of the trajectories of the chaotic systems onto one. Chaotic systems that appear to oppose synchronization due to their sensitive dependence on the initial conditions explained by their positive Lyapunov exponents, can be synchronized with the help of proper coupling between them and the requirements for the synchronization can be described mathematically. This coupling allows us to change the Lyapunov Spectrum, either reducing or increasing the number of positive Lyapunov Exponents [20].

Coupling between dynamical systems can be done in two ways: *unidirectional coupling* and *bidirectional coupling*. In the unidirectional case, a primary system consists of two subsystems operating in a drive-response configuration. Here, the evolution of one subsystem drives the behaviour of the other, compelling the response system to copy the dynamics of the drive system. The drive system serves only as an external yet chaotic forcing on the response system. External synchronisation is produced in this instance and the best example of this is secure communication based on synchronization of chaotic systems which will be discussed in Chapter-4. Conversely, in the bidirectional coupling, the two subsystems are interconnected to one another and the coupling factor causes their rhythms to be adjusted to a shared synchronised behaviour. An example of this is the interaction between neurons [21].

Various types of synchronization for coupled chaotic systems have been studied and explained both mathematically and experimentally, including Complete Synchroniza-

tion, Phase Synchronization, Lag Synchronization, and Generalized Synchronization. We will explore these in more detail in upcoming sections. All of these synchronizations can be grouped together under a single term: *Time-Scale Synchronization* [17].

3.1 Chaos Synchronization in Nature

Synchronization of chaotic dynamics in natural systems displays a remarkable occurrence in which apparently random and unrelated systems discover a way to harmonize their behaviour, providing a unique view into the complex connections that influence the natural world.

In the context of ecological systems, chaos synchronization in population dynamics can have profound impacts on ecosystems, influencing species abundance, diversity, and overall stability. Chaotic population dynamics can lead to boom-and-bust cycles, which depict exponential changes in the abundance of species within an ecosystem. These seemingly unpredictable fluctuations often result from complex interactions among species, resource availability, and environmental conditions. Chaos synchronization in the ecosystem like the population dynamics of predators and prey can lead to alternating cycles of abundance. For instance, when the prey population surges, it triggers an increase in the predator population, eventually causing a decline in prey numbers. This synchronized behaviour of predator-prey dynamics can significantly shape the composition and structure of the entire ecosystem, examples of which are the Lynx and Hare cycle of Canada, outbreaks of an insect pest 'Thrips imaginis', etc [7].

Complex systems in biology, such as neurons and immune systems, can also exhibit chaos synchronization under certain conditions. A compelling example of this synchronization is found within the neural networks of the brain. Neurons, the fundamental units of the nervous system, engage in complicated electrical impulses that appear quite complicated at first glance. These apparently irregular neural firings, however, synchronize their activity to enhance the transmission of information, enabling crucial cognitive functions such as memory formation, decision-making, and sensory perception [21].

Chaos synchronization in climate dynamics, such as El Niño and the Southern Oscillation (ENSO), which appear to be unrelated climate phenomena, actually synchronize, having a tremendous influence on global weather patterns [22].

Chaos synchronization can also be observed in nonlinear mechanical systems, like coupled pendulums or chaotic oscillators. Complex electrical circuits or laser sys-

tems, when driven by chaotic signals, can also exhibit chaos synchronization [21].

3.2 Complete Synchronization

Complete synchronization is observed in identical chaotic systems. It is also known as *Identical Synchronization*. We can achieve complete synchronization in two ways: using the coupling factor and using master-slave configuration. The systems will synchronize when the signs of the Lyapunov exponents for the subsystems are all negative and the trajectories of one of the systems will converge to the other.

3.2.1 Using Coupling Strength

Complete synchronization using the coupling function occurs when the difference between state vectors or trajectories of coupled identical chaotic systems converges to zero as time (t) tends to infinity, for the large enough coupling strengths. If isolated identical chaotic systems are diffusively coupled, then the coupling disappears with time, and both systems oscillate in alignment at all coupling strengths and times [1, 17].

Let's consider two identical non-linear n-dimensional systems which are fully diffusively coupled [1]:-

$$\begin{aligned} \dot{x_1} &= f(x_1) + \alpha \boldsymbol{H}(x_2 - x_1) \\ \dot{x_2} &= f(x_2) + \alpha \boldsymbol{H}(x_1 - x_2) \end{aligned} \tag{3.1}$$

where $\boldsymbol{f}$ is general non-linear function, α is coupling strength parameter and $\boldsymbol{H}$ is a smooth coupling function. Here, we assume $\boldsymbol{H(0) = 0}$ to attain invariant synchronized subspace $\boldsymbol{x_1 = x_2}$ for all coupling strength α. For sufficiently strong coupling, $x_1(t)$–$x_2(t) \to 0$ as $t \to \infty$. Let's consider $z := x_1 - x_2$ such that synchronization is attained if

$$\lim_{t \to \infty} z(t) = 0.$$

For the simplest case, $\boldsymbol{H = I}$ (Identity matrix) was considered. Following mathematical analysis, it was discovered that synchronisation occurs when $\alpha > \alpha_c$ [1], where α_c is critical coupling strength for synchronisation and $\alpha_c := \Lambda/2$, where Λ is the Leading Lyapunov Exponent (LLE) for trajectory $x_1(t)$ which measures the rate of infinitesimal asymptotic divergence near this trajectory. This is the case because,

$$||z(t)|| \leq Ce^{(\Lambda - 2\alpha)t} \tag{3.2}$$

where C is constant and $C > 0$. Therefore, values of coupling strength α that are above α_c would result in synchronization of the coupled identical chaotic systems [1].

Now, let's understand the above concept with the help of an example of the Lorenz systems. Consider two coupled Lorenz systems Eq (2.1), as in Eq (3.1) Using numerical analysis it was found that $\alpha_c \approx 0.453$ for two coupled Lorenz systems [1]. In Figure(3.1) below it is shown that indeed, for $\alpha < \alpha_c$ we see no synchronization and for $\alpha > \alpha_c$ there is synchronization in the trajectories of two coupled Lorenz systems.

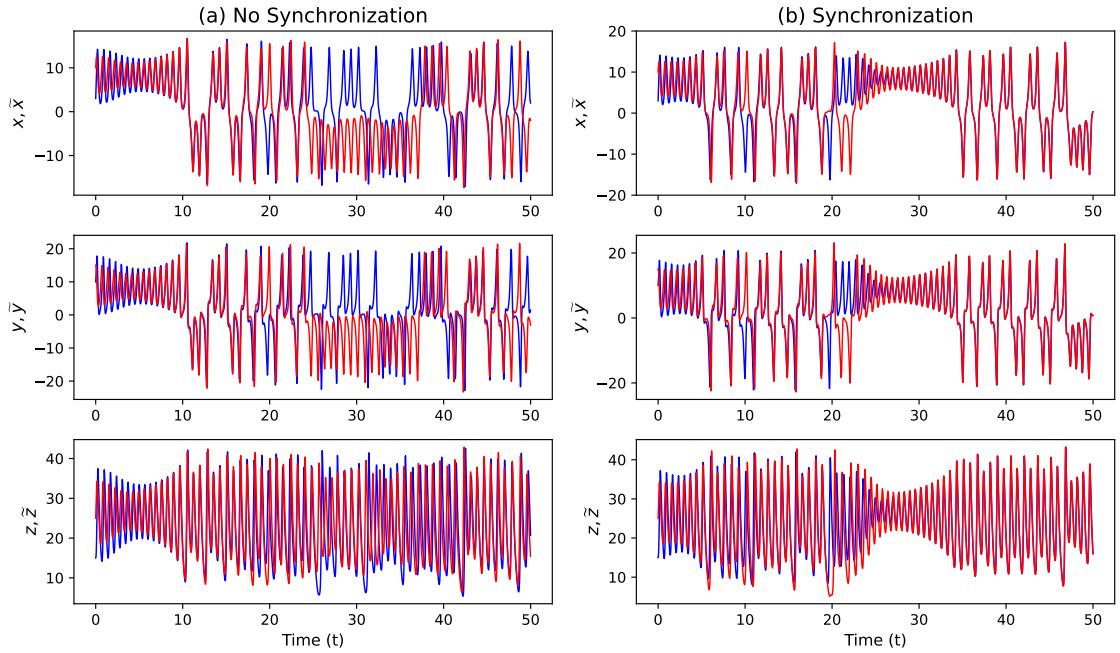
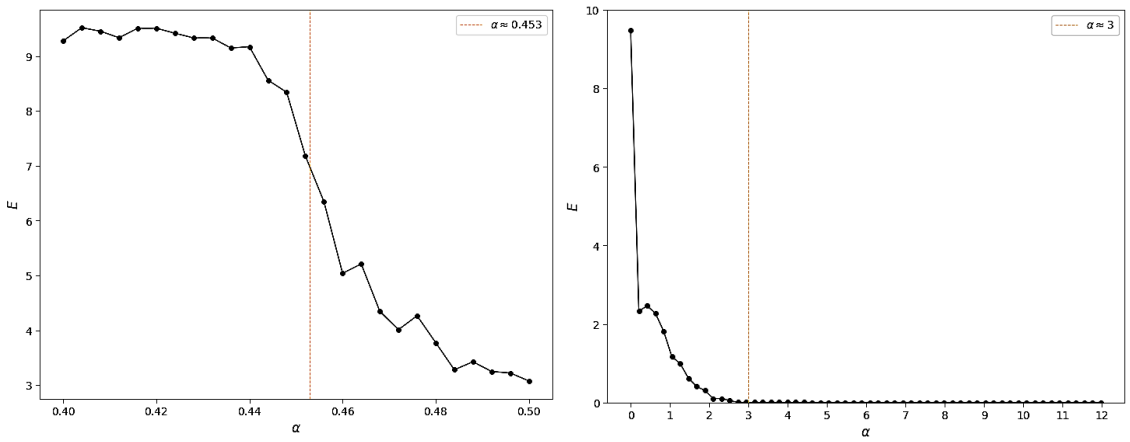


Figure 3.1: *(Original image) Two coupled Lorenz systems with initial conditions selected as (x,y,z) = (3,10,15) and $(\tilde{x}, \tilde{y}, \tilde{z})$ = (10,15,25), with parameter values σ=10, ρ=28, β=8/3. (a) When α=0.4 < α_c, there is no synchronization seen between coupled Lorenz systems. (b) For α=0.5 > α_c, there is synchronization of the trajectories of the coupled Lorenz systems.*

For a time-interval of length T, let's consider the average deviation from synchronization (E) as [1]:

$$E = \frac{1}{T}\int_{t=0}^{T} ||x_1(t) - x_2(t)||dt \tag{3.3}$$

In Figure (3.2) the relation between average deviation from synchronisation (E) and coupling strength α is analysed for two coupled Lorenz systems with coupling matrix $\boldsymbol{H} = \boldsymbol{I}$. Since the synchronisation error is influenced by the initial conditions, the synchronisation graph has been estimated by averaging it over several realizations.



(a) Computed synchronization errors E for range of α from 0.4 to 0.5

(b) Computed synchronization errors E for range of α from 0 to 12

Figure 3.2: *(Original Image) Average deviation from synchronisation represented by synchronization error (E) versus coupling strength α for coupling function $\boldsymbol{H} = \boldsymbol{I}$. The synchronization error E was averaged over 300 simulations. Each realization is run for 2000 seconds with a time step of 0.01 using the 4th-order Runge-Kutta (RK4) numerical method.*

Observed Deviations from Expected Results: There are some cases that have been encountered for some initial conditions; even when setting the value of the coupling factor greater than the critical coupling $\alpha_c \approx 0.453$ as given in reference [1], there is no synchronization seen. An example of this is shown below in Figure (3.3). These observed deviations have been thoroughly studied in reference [23] where it is shown that the system will show non-uniform chaos for α greater than $\alpha_c \approx 0.453$ so, there will be some inconsistencies. According to reference [23], these inconsistencies will be there till $\alpha_c^* \approx 11.827$ beyond which the system will always synchronize and this can be seen from Figure (3.2b) and (3.4).

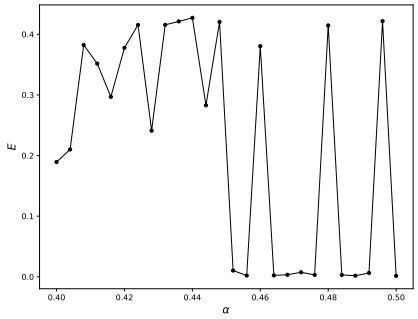


Figure 3.3: *(Original Image) Deviation from the expected result for $\alpha > \alpha_c \approx 0.453$.*

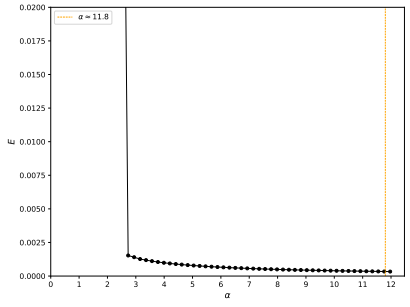


Figure 3.4: *(Original Image) Zoom of Figure (3.2b) around $\alpha \approx 11.8$.*

We can also choose a different coupling function $\boldsymbol{H}$ for the chaotic systems. For example, let's take $\boldsymbol{H}$ as x-coupling matrix for the same coupled systems defined in the above example,

$$H = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

The synchronization of the chaotic systems is observed for coupling strength greater than ~ 3.80 which can be seen in Figure (3.5).

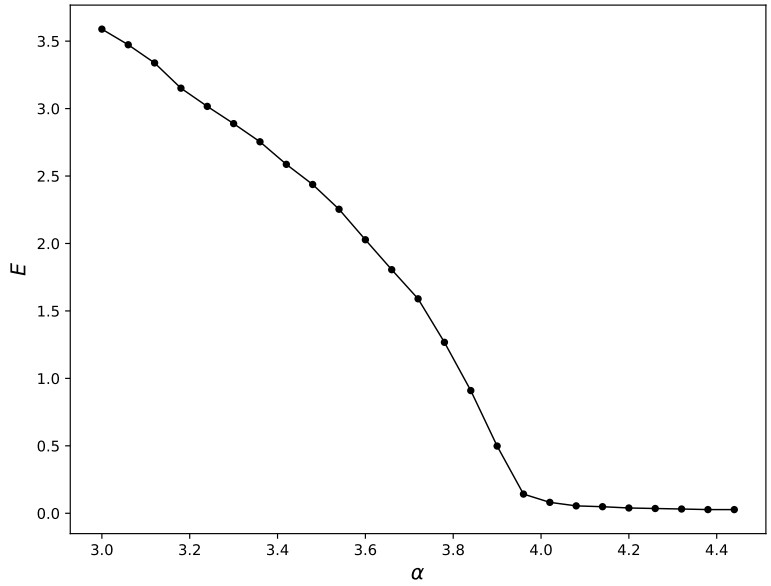


Figure 3.5: *(Original image) Synchronization error (E) versus coupling strength α for coupling function $\boldsymbol{H}$ as x-coupling matrix. The synchronization error E was averaged over 300 simulations. Each realization is run for 2000 seconds with a time step of 0.01 using the 4th-order Runge-Kutta (RK4) numerical method.*

3.2.2 Using Drive-Response (or Master-Slave) Configuration

In this configuration, the drive (transmitter) system sends a signal from one of its components to the response (receiver) system where the receiver was missing the part of the system which was compensated for by using the received signal from the drive system. In this situation, complete synchronisation is possible only if all of the Lyapunov exponents of the response system under the action of the drive system (the conditional Lyapunov exponents) are negative [10, 21]. Let's consider a case

in which the x-component of the response system is replaced with that of the drive system. The equations for such a coupled Lorenz system are given by,

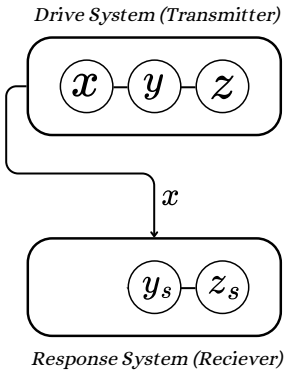


Figure 3.6: *(Original image) Schematic diagram for drive-response configuration*

$$\begin{aligned} \dot{x} &= \sigma(y-x), \\ \dot{y} &= x(\rho-z)-y, \quad \dot{y}_s = x(\rho-z_s)-y_s, \\ \dot{z} &= -\beta z + xy, \qquad \dot{z}_s = -\beta z_s + xy_s, \end{aligned} \tag{3.4}$$

where (x, y, z) are components of the drive system and y_s, z_s are components of the response system. In Figure (3.7), the simultaneous variation of the trajectories is being tracked to check their behaviour by $\Delta y(t) = y(t) - y_s(t)$ and $\Delta z(t) = z(t) - z_s(t)$ and for given initial conditions $(x, y, z, y_s, z_s) = (10.1, 10.1, 10.1, 0.1, 0.1)$, Δy and Δz tends to zero. Hence, after some time systems get synchronized.

In reference [1] it is shown that for this particular subsystem, a Lyapunov function (Subsection 1.1.1) for the displacements Δy and Δz can be constructed. Using Eq (3.4) we get,

$$\begin{aligned} \dot{\Delta} y &= -\Delta y - x\Delta z, \\ \dot{\Delta} z &= x\Delta y - \beta\Delta z. \end{aligned} \tag{3.5}$$

Now, let's consider the Lyapunov function

$$V = \frac{1}{2}(\Delta y^2 + \Delta z^2) \tag{3.6}$$

and it is shown that $\dot{V} = -\Delta y^2 - \beta\Delta z^2$. Since V is positive and $\dot{V}$ is negative, Δy and Δz will converge to zero. As a result, the response subsystem will exhibit the same dynamics as the drive system.

In Figure (3.7), the drive system sends the x signal to the response system, which

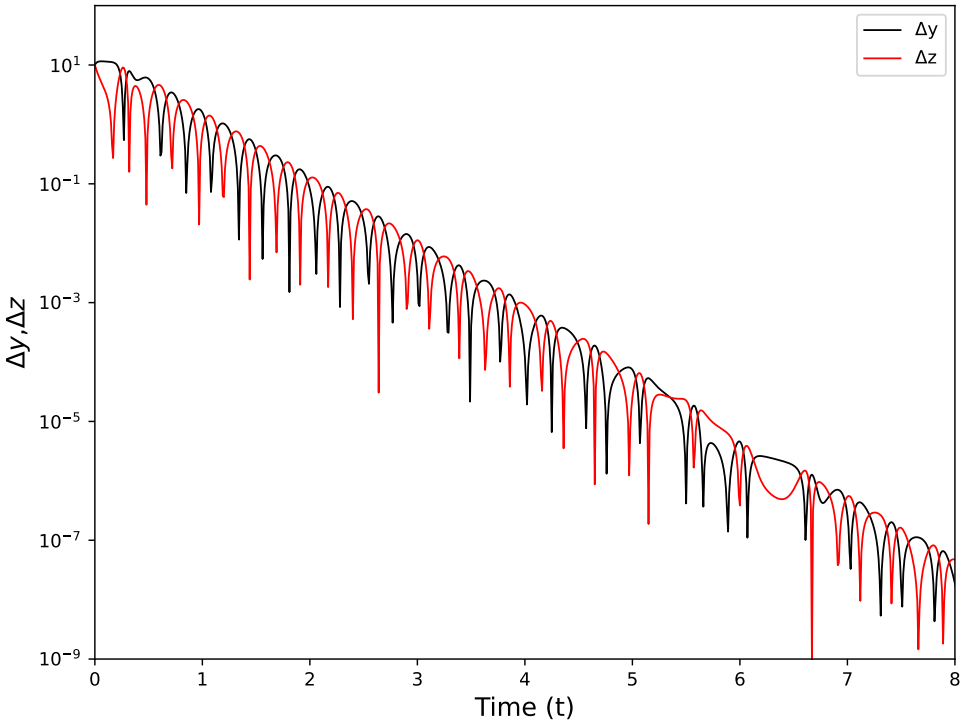


Figure 3.7: *(Original image) Simulation of drive-response coupling for two Lorenz systems for which x is the drive system. The given initial conditions are $(x, y, z, y_s, z_s) = (10.1, 10.1, 10.1, 0.1, 0.1)$. The graph is simulated using the 4th-order Runge-Kutta (RK4) numerical method [1]*

doesn't have the x component but is otherwise similar to the drive, i.e., y_s and z_s subsystems are identical to the drive's y and z subsystem. And for the systems to synchronize the conditional Lyapunov exponents must be negative [1, 20, 21]. These Lyapunov exponents are called ***Conditional Lyapunov Exponents*** because of their dependence on the driving signal, here the x signal in Figure (3.7). It is important to note that not all feasible driving signal selections result in a synchronised state.

3.3 Phase Synchronization

Phase synchronisation is the locking of coupled chaotic systems phases which lead to the frequency entrainment. Whenever for two coupled chaotic dynamical systems $x_{1,2}$ the phase difference $\Delta\phi$ for instantaneous phase $\phi_{1,2}$ is bounded by some constant, we get phase synchronization between the systems i.e., $|\phi_1(t) - \phi_2(t)| <$ constant.

Because of the chaotic nature of the system, the phase difference will not be exactly zero. In phase synchronization of coupled chaotic dynamical systems, only phases of the subsystems are locked, while the dynamics remain hyper-chaotic. Note that phase synchronization does not imply complete synchronization because amplitudes of the coupled chaotic systems remain uncorrelated. In general cases, it is challenging to introduce a phase for a chaotic attractor, but it is feasible for a suitable class of chaotic attractors which possess unstable periodic orbits like Rössler system. [1, 17, 21].

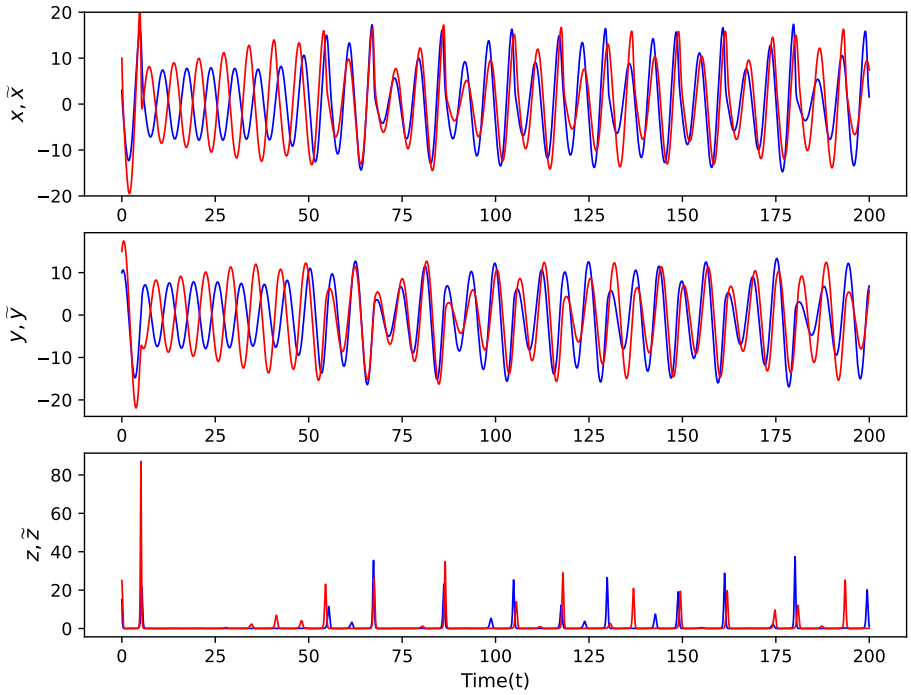


Figure 3.8: *(Original Image) Phase Synchronization*

Using the phase angle $\phi_{1,2}(t)$ it is possible to define a mean rotation frequency,

$$\Omega_{1,2} = \lim_{t\to\infty} \frac{\phi_{1,2}(t) - \phi_{1,2}(0)}{t} \tag{3.7}$$

where,

$$\phi_{1,2}(t) = \arctan\left(\frac{y_{1,2}}{x_{1,2}}\right) \tag{3.8}$$

$$A_{1,2}(t) = \sqrt{x_{1,2}^2 + y_{1,2}^2} \tag{3.9}$$

Here, $A_{1,2}(t)$ are the amplitudes of the oscillations and the *frequency mismatch* is defined as $\Delta\Omega = \Omega_2 - \Omega_1$. Thus, for phase synchronization to occur the mean rotation frequency must be the same for the drive and the response system i.e., $\Delta\Omega = 0$. This leads to the frequency entrainment known from coupled periodic oscillations which is a temporal locking of the motion or signal frequency of one dynamical system which is retained by the motion or frequency of another dynamical system [24]. An example of this is shown below with the help of Figure (3.9) using two unidirectionally coupled Rössler systems which is coupled over x-components. The equation for two non-identical Rössler systems is given by,

$$\begin{aligned} \dot{x}_{1,2} &= -\omega_{1,2} y_{1,2} - z_{1,2} + \alpha(x_{2,1} - x_{1,2}) \\ \dot{y}_{1,2} &= \omega_{1,2} x_{1,2} + a y_{1,2} \\ \dot{z}_{1,2} &= b + z_{1,2}(x_{1,2} - c) \end{aligned} \tag{3.10}$$

where ω is the *mismatch parameter* which makes the two systems non-identical, α is the coupling constant, $a = 0.165, b = 0.2$ and $c = 10$ are the constant parameters for the Rössler system. Given, $\omega_{1,2} = \omega_0 \pm \Delta$ where $\omega_0 = 0.97$ and $\Delta = 0.02$, the synchronization is observed for $\alpha > \alpha_c (\sim 0.04)$ [1, 21, 24].

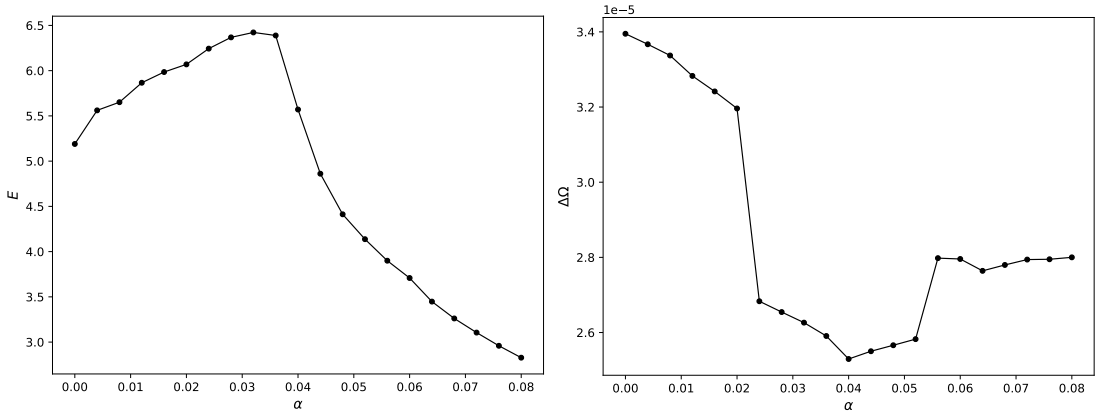


Figure 3.9: *(Orignal Image) The synchronisation error E and frequency mismatch $\Delta\Omega$ relation with coupling strength α. Phase synchronization occurs for $\alpha > \alpha_c (\sim 0.04)$ [1]. The synchronization error E was averaged over 300 simulations. Each realization is run for 2000 seconds with a time step of 0.01 using the 4th-order Runge-Kutta (RK4) numerical method.*

The Rössler system show non-uniform chaotic behaviour for some initial conditions because of which we can see some irregularities in Figure (3.9). Also, if coupling strength α is small then the amplitudes of the chaotic systems will show chaotic behaviour but the phase will sync by bounded phase difference.

3.4 Lag Synchronization

In lag synchronization, two chaotic dynamical systems achieve identical states through a time shift, represented as $x_1(t+\tau) = x_2(t)$. The time lag denoted as τ, diminishes as the coupling strength between the oscillators increases, ultimately leading to complete synchronization. Lag synchronization is mainly investigated in scenarios involving symmetrically coupled but non-identical dynamical systems or systems with time delays [25]. When dealing with identical coupled systems, complete synchronization represents a special case of lag synchronization, occurring when τ equals zero. The determination of lag synchronization often relies on a similarity function which quantitatively assesses how similar or correlated the states of two chaotic dynamical systems are with respect to the time lag τ.[17, 21].

Furthermore, recent studies have unveiled an intriguing phenomenon in dissipative chaotic systems with time-delayed feedback. In this context, the driven systems anticipate the behaviour of the driver systems by synchronizing with their future states, where y(t) $\approx x(t+\tau)$. For a more comprehensive analysis of lag synchronization, please refer to the following references: [21, 25].

3.5 Generalized Synchronization

Generalized synchronization is the synchronization between two non-identical chaotic dynamical systems. It is introduced for drive-response configuration. The chaotic dynamics of the driving system are independent of the response system parameters. For generalized synchronization to occur coupled chaotic systems must have a functional relationship, i.e., $x_2(t) = F[x_1(t)]$. The case of generalised synchronisation is more challenging since the functional relation F[.] is involved. Yet, there are numerous approaches to discover the synchronized behaviour of coupled chaotic systems, such as *the auxiliary system approach* which links the generalized synchronization problem to the complete synchronisation problem and *the method of the nearest neighbourhood* which determines when closeness in response space implies closeness in driving space. The synchronisation condition in generalized synchronization means that the drive alone determines the response dynamics [1, 17, 21, 26].

For unidirectionally coupled dynamical systems, the drive x and the response y systems are coupled as

$$\begin{aligned} \dot{x} &= f(x) \\ \dot{y} &= g(y, h(x)) \end{aligned} \tag{3.11}$$

where x $\in R^n$, y $\in R^m$ and h(x) is the coupling, and for certain coupling strengths, the dynamics of system y are completely determined by the dynamics of system x.

That is, x solutions can be mapped into y solutions.

$$y = \Psi(x) \tag{3.12}$$

where Ψ is the functional relationship between coupled chaotic systems This results in generalized synchronization between these two systems. When Ψ is the identity, *complete synchronization* is a special case of generalized synchronization [1].

Now, let's review the methodologies stated above for discovering the synchronised behaviour of connected chaotic dynamical systems.
Auxiliary System Approach:-
In this, we have an auxiliary system which is a copy of the response (or slave) system and the drive (or master) system drives the slave system and an auxiliary system. If the auxiliary and response systems exhibit complete synchronization then the drive and response systems are in generalized synchronization. A detailed study about this approach can be found in reference [1].

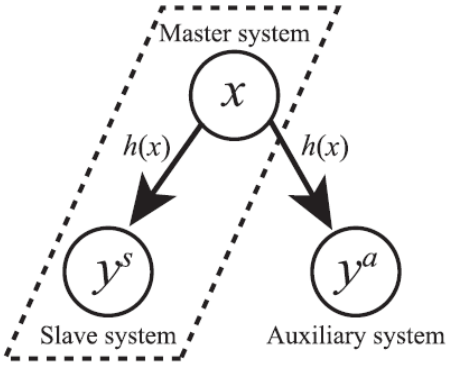


Figure 3.10: *Schematic diagram for auxiliary system approach for generalized synchronization. If there is complete synchronization between y^s and y^a, then the generalized synchronization occurs between x and $y^{a,s}$ [1].*

The method of the nearest neighbourhood:-
The key idea is to look at how close points are mapped through the dynamics. The existence of the mapping Ψ can be inferred by evaluating the attributes of surrounding points [1]. Let D be the phase space of the driving system x and R be the phase space for y. If there exists a transformation Ψ from the trajectories of the attractor in D space to the trajectories in R space then we have synchronization between the x and y systems. This approach detects the presence of the continuous transformation Ψ and hence distinguishes between synchronised and unsynchronized behaviour in coupled chaotic dynamical systems. In simple words, in this approach, we are trying to find some geometric relation between the driving and response systems which do

not alter the identity of neighbourhoods in state space. It is kind of finding the correlation between driving and response systems. This approach has been thoroughly studied in reference [26].

3.6 Applications of Chaos Synchronization

Chaos synchronization is a phenomenon which is characterized by its ability to bring order to seemingly disorderly systems and has its influence in a wide range of applications in various fields. We will look into some such applications in brief in the following:

- Chaos synchronisation is an exciting area of research in cryptography. Since, chaotic signals are noise-like and broadband, it is difficult to read the message and thus, masking information in chaotic signals provides an innovative way for secure communication which we will see in detail in Chapter (4).
- Because natural systems are generically heterogeneous, and the coupling is typically complex and weak, complete synchronisation may be exceptional but still, many natural systems can be explained with the help of other types of synchronization. Stable biological systems like the dynamics of the pacemaker cells where millions of the heart's pacemaker cells fire in unison can be modelled by phase synchronization. Phase synchronization methods have also been used to study spatial synchronization of oscillations in blood distribution systems and synchronization of biological neuron activities [21].
- In ecological systems, Chaos synchronization techniques are useful in understanding and predicting population dynamics in ecosystems. It can be also used to gain insights into the resilience and adaptability of ecosystems, as well as to inform conservation efforts aimed at maintaining the health and diversity of natural environments. [7]
- *Parameter estimation and prediction:* Anticipating future behaviours or data in system analysis plays a pivotal role across various research domains. However, a significant challenge arises in determining the essential parameters needed for accurate predictive models. Now, chaos synchronization techniques are also being utilized for this purpose. Here, the goal is to adjust the mathematical model's parameters so that it behaves similarly to the observed data. Chaos synchronization ensures that the model's dynamics synchronize the dynamics of the real system. Finally, the adjusted parameters represent estimates of the actual parameters governing the system [1].
- Chaos synchronization also proves to be highly useful in climate science. It plays an essential role in improving both the accuracy and dependability of

weather forecasting as well as climate change modeling. By analyzing the synchronization patterns in ocean-atmosphere interactions, such as El Niño and La Niña events, scientists can gain a better understanding of how to anticipate shifts in regional weather patterns and seasonal climate anomalies [7, 27].

- Scientists have used phase synchronization methods to study solar activity. With the help of phase synchronization methods, they have discovered that there's a connection between the timing of sunspots and a fast component of the sun's inertial motion. This connection is statistically significant and provides some evidence for the idea that gravity might have a slight influence on solar activity [21].

Chapter 4

Secure Communication

In the field of secure communication, the utilization of chaotic systems has emerged as an innovative approach. Chaos, with its inherent unpredictability and complexity, provides a unique foundation for encoding and decoding sensitive information. Incorporating chaotic systems into secure communication not only improves encryption but also adds a layer of resistance to standard hacking techniques. In upcoming sections, we will look into how different types of chaos synchronization are used to develop secure communication models.

4.1 Secure Communication Using Complete Synchronization

Secure communication (SC) based on complete synchronization utilizes chaos synchronization between the drive and response chaotic subsystems. This approach requires that all the Conditional Lyapunov Exponents of the response subsystems be negative. Here, the parameters of the chaotic system play an important role as the encryption keys for the secure communication of messages. Now, we will see how different types of signals (messages) can be transmitted using secure communication.

4.1.1 For Analog Signal

In the context of transmitting analog signals, we assume the amplitude of the chaotic signal is much larger than the amplitude of the analog signal. The important point here is to add the original signal (message) $m(t)$ to the chaotic signal to generate a new signal $s(t)$. Let's assume Lornez subsystems as drive and response chaotic systems with x-coupling. Hence, $s(t) = m(t) + x(t)$ will be the new signal where $x(t)$ is the x-component of the drive system. This process is known as masking of the information.

Usually, masking of the information on carrier signals does not require any kind of encryption. But here in secure communication using complete synchronization, the key point is to have a transmitter (drive system) and a receiver (response system) such that they synchronise and as we know from earlier, for complete synchronisation to occur between the drive and the response systems they must have the same parameters. Thus, the parameters of the chaotic system act as encryption keys here and hence, the message can only be recovered if the parameters are known [1].

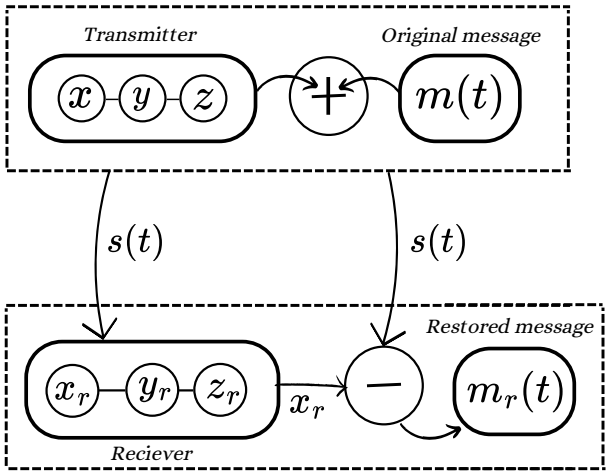


Figure 4.1: *(Original image) Schematic diagram for secure communication using complete synchronization of the chaotic systems in drive-response configuration for analog signal.*

Let's see this with the help of an example where message signal $m(t)$ is given by,

$$m(t) = 0.1\frac{\sin\left(1.2\pi\sin^2(t)\right)}{\pi\sin^2(t)}\cos\left(10\pi\cos(0.9t)\right) \tag{4.1}$$

Since we have taken Lornez subsystems as drive and response chaotic systems with x-coupling, the message s(t) drive the response system (receiver) as,

$$\begin{aligned} \dot{x}_r &= \sigma(y_r - x_r), \\ \dot{y}_r &= s(\rho - z_r) - y_r, \\ \dot{z}_r &= -\beta z_r + s y_r \end{aligned} \tag{4.2}$$

where $\sigma = 16.0$, $\beta = 4.0$ and $\rho = 45.2$ are the chosen parameters for the Lorenz subsystems. Finally, the original message can be recovered by $m(t) = s(t) - x_r(t)$ which can be seen in Figure (4.2). The recovered signal has few irregularities because of the presence of noise. As long as the system experiences small perturbations

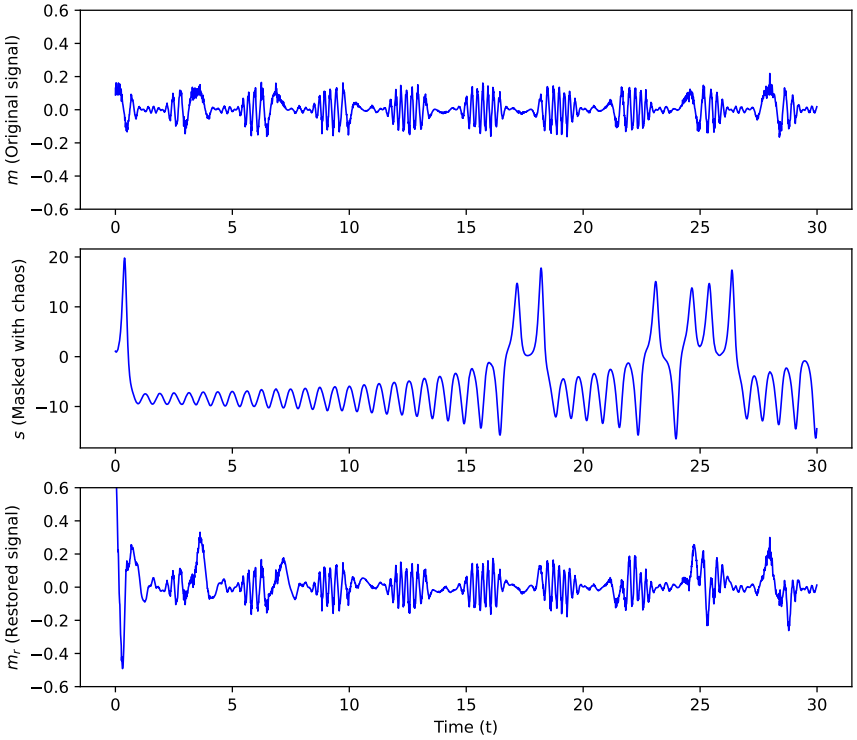


Figure 4.2: *(Orignal image) Restoring the original message after secure communication of the analog signal using complete synchronization of the Lorenz subsystems in drive-response configuration with x-coupling. Here, the paramters are taken as $\sigma = 16.0$, $\beta = 4.0$ and $\rho = 45.2$.*

(low-amplitude noise), the synchronization of chaotic systems remains exponentially stable, allowing for continued synchronization to occur. The reason why these coupled chaotic systems do not diverge due to small perturbations is because of the stability of synchronized states which is discussed in detail in references [21, 23]. These small perturbations slightly deviate the synchronized states but do not lead to exponential divergence, making secure communication feasible using the synchronization of chaotic systems.

4.1.2 For Digital Signal

For secure communication of digital signals (binary signals), we use the modulation of paramters technique. Here, the masking of information is done using the parameter of chaotic subsystems. The transmitter incorporates an adjustable parameter $\sigma_a(t) = \sigma + \delta m(t)$ such that transmitter and receiver systems synchronize (or be in tune) when $m(t) = 0$ and desynchronize (or out of tune) when $m(t) = 1$. We restore

the original message, m(t), based on the synchronization and desynchronization patterns.

Now let's see an example of this Figure (4.3) using Lorenz subsystems with x-coupling where the dynamics of the transmitter and the receiver are given by,

$$\begin{aligned}
\dot{x}_1 &= \sigma_a(y_1 - x_1), \\
\dot{y}_1 &= x_1(\rho - z_1) - y_1, \\
\dot{z}_1 &= -\beta z_1 + x_1 y_1. \\
\dot{x}_2 &= \sigma_a(y_2 - x_2), \\
\dot{y}_2 &= x_1(\rho - z_2) - y_2, \\
\dot{z}_2 &= -\beta z_2 + x_1 y_2
\end{aligned} \tag{4.3}$$

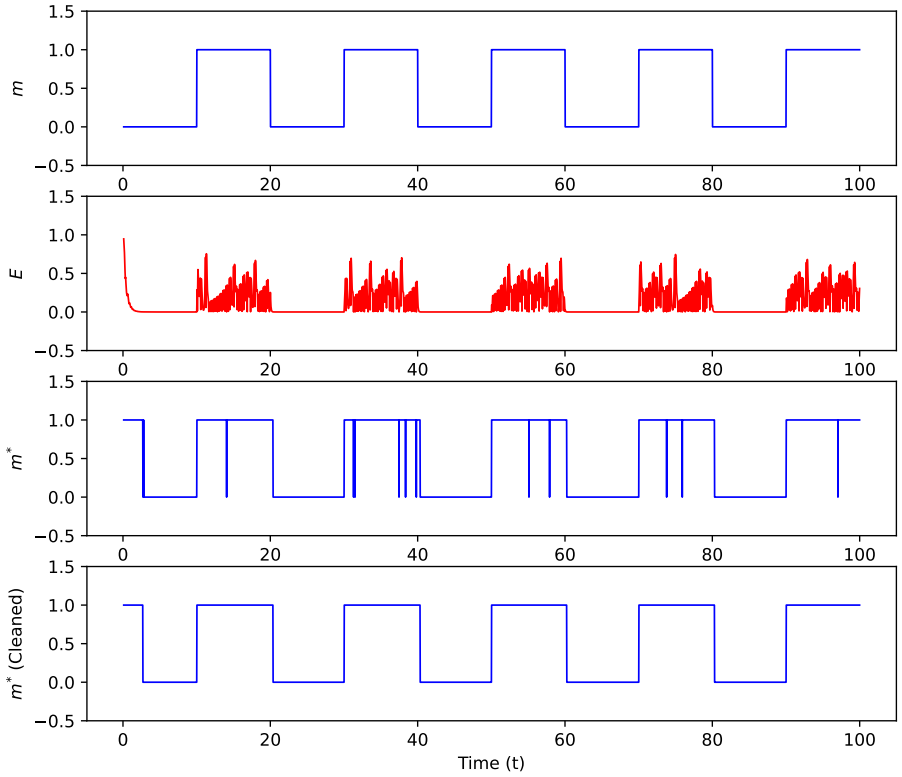


Figure 4.3: *(Orignal image) Restoring the original message after secure communication of the digital signal based on complete synchronization of the Lorenz subsystems using modulation of parameter technique. The restored message has been cleaned by using the majority voting algorithm. The accuracy of the majority voting algorithm for the cleaned restored data came to approximately 99.87%.*

where $\sigma_a(t)$ is the adjustable parameter defined above and δ is the mismatch. If the transmitter and receiver have the same parameters, the system will synchronize when $m(t)$=0 and desynchronize when $m(t) = 1$. When there is a significant mismatch, δ, it results in a synchronization error $E > 0$, which is represented as a binary signal 1 by setting $m(t) = 1$. Thus, this pattern of synchronization and desynchronization can be used to retrieve the original digital (binary) signal from the transmitted signal. Also, the restored message has some irregularities due to the presence of noise which has been cleaned here in Figure (4.3) by using the majority voting algorithm since it is a binary signal. In the majority voting algorithm for data cleaning, the data points are considered, and the most common or frequently occurring value is selected as the cleaned data point. Hence, we get our final cleaned restored message [1].

4.1.3 Limitations and Solutions

The major limitation of secure communication using complete synchronization is that it is possible to break secure communication using a special type of Poincáre section known as the return maps (Section (1.1)). This process does not involve the complete restoration of the sender's dynamics but the message can be recovered by analyzing the evolution of the signal on the attracting sets of these maps [28, 29].

Another constraint to consider is that the analog signals can only be effectively transmitted in low amplitude which makes the communication range very limited. Also, the quality of the synchronized signal may degrade over long communication channels or in the presence of noise, affecting the reliability of secure communication.

One way to enhance the security of secure communication is to use phase synchronization instead of complete synchronization, which we will see in detail in Section(4.2).

Another efficient way is discussed in the reference [30] in which it has been shown that the secure transmission of messages with a low bit error rate using the chaos pass filter configuration or even error-free transmission utilising the chaos modulation in the synchronisation of two chaotic systems can be achieved by using the bidirectional coupling for secure communication instead of the unidirectional coupling which we have used in Section (4.1).

4.2 Secure Communication Using Phase Synchronization

As we know from Section (3.2) only a particular class of chaotic attractors show phase synchronization. Thus, here we will see secure communication based on the phase synchronization approach with the help of Rössler system as an example. The secure communication model consists of three chaotic Rössler systems ($x_{1,2,3}$) to improve the security of communication, since in Subsection (4.1.3) we saw that there are certain methods that have been enhanced and reported to break the secure communication models based on complete synchronization [1, 31].

This approach of phase locking involves the mean of the phases ϕ_1 and ϕ_2 from two systems in the transmitter can serve as a spontaneous phase signal ϕ_m, which couples to the third system, as described in Equation (4.4). This increases the security way more since the return maps of the phase ϕ_m exhibit a significantly higher level of complexity compared to ϕ_1 (or ϕ_2), making it nontrivial to decode [31]. The transmitter is composed of two weakly coupled systems x_1 and x_2 which are identical to each other and the receiver x_3 is slightly different and these systems are defined by,

$$\begin{aligned}
\dot{x}_{1,2} &= -(\omega + \Delta\omega)y_{1,2} - z_{1,2} + \epsilon(x_{2,1} - x_{1,2}),\\
\dot{y}_{1,2} &= (\omega + \Delta\omega)x_{1,2} + ay_{1,2},\\
\dot{z}_{1,2} &= b + z_{1,2}(x_{1,2} - c).\\
\dot{x}_3 &= -y_3 - z_3 + \alpha(r_3 \cos\phi_m - x_3),\\
\dot{y}_2 &= x_3 + ay_3,\\
\dot{z}_2 &= b + z_3(x_3 - c)
\end{aligned} \tag{4.4}$$

where the mismatch paramter $\omega = 1$ and the parameters of the Rössler system are given by $a = 0.15$, $b = 0.2$ and $c = 10$ and the coupling constant between x_1 and x_2 is given by $\epsilon = 5 \times 10^{-3}$. The coupling constant α is between transmitter and receiver and r_3 is the amplitude of the receiver system given by $A_3 = \sqrt{x_3^2 + y_3^2}$. Here, $\Delta\omega$ is an adjustable mismatch parameter which will be used to mask the original binary message (digital signal) for secure communication [1]. It is defined as,

$$\Delta\omega = \begin{cases} 0.01 & \text{if bit digit } =1 \\ -0.01 & \text{if bit digit } =0 \end{cases} \tag{4.5}$$

Equation (3.8) provides the phase definition for the Rössler system. The phase synchronization will occur between ϕ_m and ϕ_3. Thus, the phase difference between ϕ_m and ϕ_3 will be used to retrieve the sent signal. The changes in the adjustable mismatch parameter $\Delta\omega$ will result in the variation of the phase difference between ϕ_m and ϕ_3. Finally, the message can be retrieved from different phase-locking values which can be seen in Figure (4.4) [1].

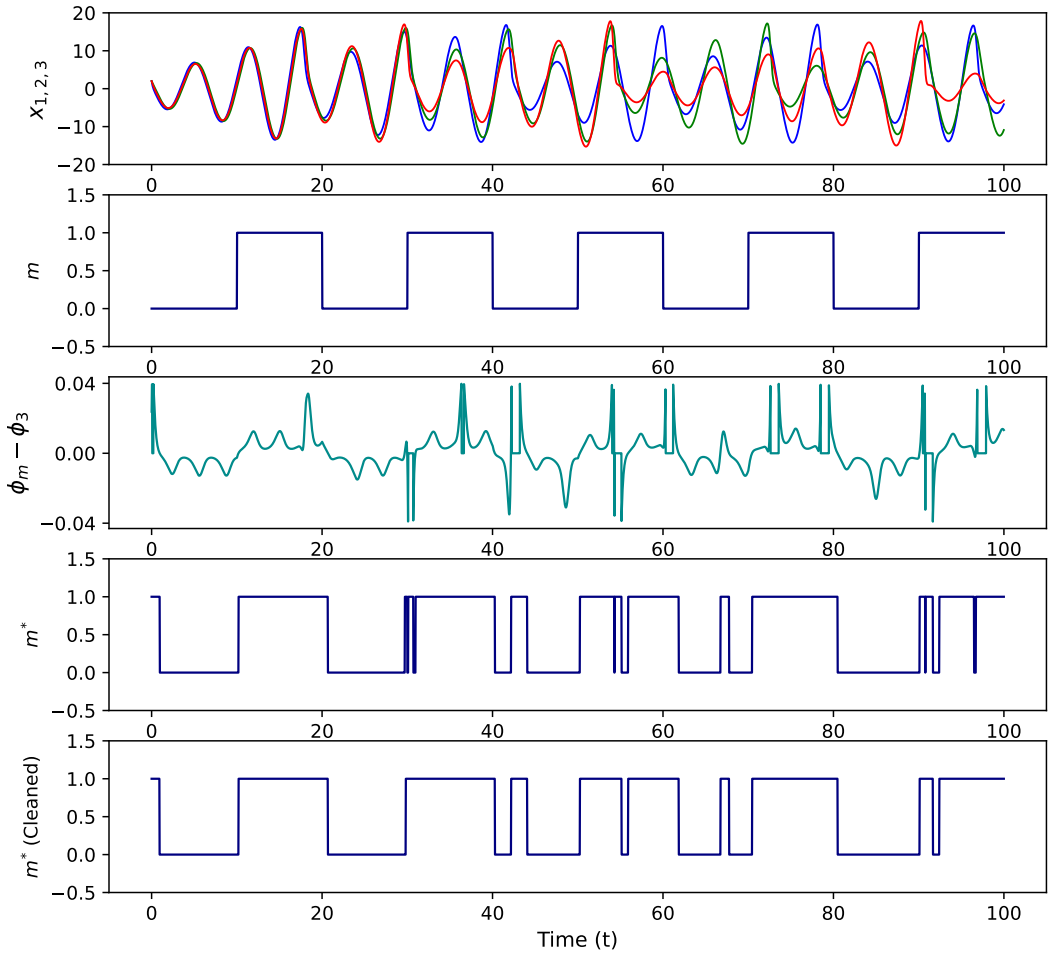


Figure 4.4: *(Orignal image) Restoring the original message after secure communication of the digital signal based on phase synchronization. The restored message has been cleaned by using the majority voting algorithm. The accuracy of the majority voting algorithm for the cleaned restored data came to approximately 99.21%.*

Limitaions: Though phase synchronization-based secure communication is hard to decode using return maps but it has its own limitations. First, it is highly complex to use phase synchronization for analog signals since in phase synchronization the amplitude of the chaotic systems remains chaotic. Secondly, even for digital signals sometimes the noise level in the restored message is too high after cleaning, a glimpse of which we can see in the above Figure (4.4) which decreases the efficiency of secure communication.

Chapter 5

Conclusion and Discussion

This study emphasizes the significance of chaos synchronization, both as a theoretical concept and as a practical phenomenon with wide-ranging implications. The ability to synchronize chaotic systems has far-reaching consequences in several domains, making it an interesting topic for discussion.

In this study, we have explored various forms of chaos synchronization, including Complete Synchronization, Phase Synchronization, Lag Synchronization, and Generalized Synchronization. Focusing on complete and phase synchronization for computational analysis, we attempted to reproduce the results from reference [1]. We found there are some inconsistencies like in Figure (3.3) due to the non-uniform chaotic behaviour of the Lorenz system which results in deviation of plot between synchronization error E and coupling strength α_c. We also got to know that these inconsistencies are present in the Lorenz system for $\alpha_c < 11.827$ mentioned in the Subsection (3.2.1). Similarly, due to the same reason of the non-uniform chaotic behaviour of the Rössler system, in phase synchronization, we found some deviation in the plot between frequency mismatch $\Delta\Omega$ and coupling strength α, shown in Figure (3.9).

We also explored chaos synchronization in drive-response configuration and got the expected graph for synchronization error in y and z directions w.r.t. time as shown in Figure (3.7). Later, we used this concept in secure communications as an application of chaos synchronisation, shown in Figures (4.2 and 4.3). There we were able to send analog and digital signals securely using complete synchronization with minimal error. Likewise, we were also able to send digital messages securely using phase synchronization with some extent of error (Figure (4.4)). The errors were discussed in the limitations and solutions in Sections (4.1 and 4.2).

Furthermore, we also discussed other applications of chaos synchronization in various fields, including parameter estimation and prediction, understanding the dynamics of species populations in ecology and its impact on the stability of ecosystems, functioning of our brain's neural networks, understanding astrophysical bodies like the sun and even improving our ability to model climate change and forecast extreme weather events.

In the future, I would like to study the sensitivity and the real-world practical applications of parameter estimation using the chaos synchronization technique. Since, the recent world is filled with an enormous amount of unexplored data, having a tool to predict parameters of the mathematical model developed for the system's data can help a lot to get insight about that system.

At last, I would like to mention an interesting perspective about the universe discussed in the reference [7]. The author discusses how even in the most complex systems, there is an underlying order and raises the question of how we perceive chaos and whether it might change when observed by a chaotic observer. This leads to a discussion of quantum mechanics and the idea that observer-internal chaos could be the source of quantum uncertainty. In summary, the author explores the possibility that chaos, both in the external world and within the observer, can influence our understanding of quantum mechanics and the inherent uncertainty found in the behaviour of particles on the smallest scales. This concept raises profound questions about the nature of observation, perception, and the fundamental principles of the universe.

Bibliography

[1] D. Eroglu, J. S. W. Lamb, and T. Pereira. Synchronisation of chaos and its applications. *Contemporary Physics*, 58(3):207–243, 2017.

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

[3] W. E. Boyce and R. C. DiPrima. *Elementary Differential Equations and Boundary Value Problems*. Wiley, 2013.

[4] J. M. G. Miranda. *Synchronization and Control of Chaos: An Introduction for Scientists and Engineers*. Imperial College Press, 2004.

[5] A. P. Willis. Topic-1: An introduction to chaotic dynamics. Module: Mathematical modelling of natural systems. *University of Sheffield*, 2023.

[6] C. W. Wu and N. F. Rul'kov. Studying chaos via 1-d maps-a tutorial. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 40(10):707–721, 1993.

[7] A. V. Holden. *Chaos*. Manchester University Press, 1986.

[8] C. Grebogi, E. Ott, S. Pelikan, and J. A. Yorke. Strange attractors that are not chaotic. *Physica D: Nonlinear Phenomena*, 13(1):261–268, 1984.

[9] A. Hastings, C. Hom, S. Ellner, P. Turchin, and C. Godfray. Chaos in ecology: Is mother nature a strange attractor?*. *Annu. Rev. Ecol. Syst.*, 24:1–33, 11 2003.

[10] L. M. Pecora and T. L. Carroll. Synchronization in chaotic systems. *Phys. Rev. Lett.*, 64:821–824, Feb 1990.

[11] A. M. Lyapunov. *General Problem of the Stability Of Motion*, volume 55. CRC Press, 1992.

[12] A. Politi. Lyapunov exponent. *Scholarpedia*, 8(3):2722, 2013.

[13] G. Benettin, L. Galgani, A. Giorgilli, and J. Strelcyn. Lyapunov characteristic exponents for smooth dynamical systems and for hamiltonian systems; a method for computing all of them. part 1: Theory. *Meccanica*, 15:9–20, 1980.

[14] S. H. Strogatz. *SYNC: The Emerging Science of Spontaneous Order*. Hachette Books, 2003.

[15] A. Pikovsky and M. Rosenblum. Synchronization. *Scholarpedia*, 2(12):1459, 2007.

[16] Arkady Pikovsky, Michael Rosenblum, and Jürgen Kurths. *Synchronization: From Simple to Complex*. Cambridge University Press, 1997.

[17] A. E. Hramov and A. A. Koronovskii. An approach to chaotic synchronization. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 14(3):603–610, 2004.

[18] E. N. Lorenz. Deterministic nonperiodic flow. *Journal of atmospheric sciences*, 20(2):130–141, 1963.

[19] K.M. Ibrahim, R. K. Jamal, and F. H. Ali. Chaotic behaviour of the rossler model and its analysis by using bifurcations of limit cycles and chaotic attractors. In *Journal of Physics: Conference Series*, volume 1003, page 012099. IOP Publishing, 2018.

[20] L. M. Pecora and T. L. Carroll. Synchronization of chaotic systems. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 25(9):097611, 2015.

[21] S. Boccaletti, J. Kurths, G. Osipov, D. Valladares, and C. S. Zhou. The synchronization of chaotic systems. *Physics reports*, 366(1-2):1–101, 2002.

[22] N. Jajcay, S. Kravtsov, G. Sugihara, A. A. Tsonis, and M. Paluš. Synchronization and causality across time scales in el niño southern oscillation. *npj Climate and Atmospheric Science*, 1(1):33, 2018.

[23] H. Fujisaka and T. Yamada. Stability theory of synchronized motion in coupled-oscillator systems. *Progress of theoretical physics*, 69(1):32–47, 1983.

[24] U. Parlitz, L. Junge, W. Lauterborn, and L. Kocarev. Experimental observation of phase synchronization. *Physical Review E*, 54(2):2115, 1996.

[25] C. Li and X. Liao. Lag synchronization of rossler system and chua circuit via a scalar signal. *Physics Letters A*, 329(4-5):301–308, 2004.

[26] N.F. Rulkov, M. M. Sushchik, L. S. Tsimring, and H. D. Abarbanel. Generalized synchronization of chaos in directionally coupled chaotic systems. *Physical Review E*, 51(2):980, 1995.

[27] A. Selvam. *Chaotic climate dynamics*. Luniver Press, 2007.

[28] G. Pérez and H. A. Cerdeira. Extracting messages masked by chaos. *Physical Review Letters*, 74(11):1970, 1995.

[29] K. M. Short. Steps toward unmasking secure communications. *International Journal of Bifurcation and Chaos*, 4(04):959–977, 1994.

[30] W. Kinzel, A. Englert, and I. Kanter. On chaos synchronization and secure communication. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 368(1911):379–389, 2010.

[31] J. Y. Chen, K. W. Wong, L. M. Cheng, and J. W. Shuai. A secure communication scheme based on the phase synchronization of chaotic systems. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 13(2):508–514, 05 2003.

A Appendix

A.1 Python Code

A.1.1 For Lorenz System

```
# Importing Packages
import math, sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time
from ipywidgets import interact, interactive
from scipy import integrate
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import cnames
from matplotlib import animation
import random
from tqdm.notebook import tqdm

# Function for 2-D and 3-D Plot
def lorenz_ode_plot_components (x, y, z):
    plt.figure (figsize = (8.5,6.5))
    plt.plot ( t, x, linewidth = 1, color = 'b' )
    plt.plot ( t, y, linewidth = 1, color = 'r' )
    plt.plot ( t, z, linewidth = 1, color = 'g' )
    plt.grid ( True )
    plt.xlabel ( '<--- Time --->' )
    plt.ylabel ( '<--- x(t), y(t), z(t) --->' )
    plt.title ( 'Lorenz Time Series Plot' )

    plt.legend(['x','y','z'], loc = 2)
    plt.savefig ( 'lorenz_ode_components.png' )
    plt.show ( )
    return

def lorenz_ode_plot_3d (x, y, z):
    fig = plt.figure (figsize = (8.5,8.5))
```

```
    ax = fig.add_subplot(projection='3d')
    ax.plot ( x, y, z, linewidth = 0.5, color = 'orangered' )
    ax.grid ( True )
    ax.set_xlabel ( '<--- x(t) --->' ,fontsize=12)
    ax.set_ylabel ( '<--- y(t) --->' ,fontsize=12)
    ax.set_zlabel ( '<--- z(t) --->' ,rotation=90,fontsize=12)
    plt.savefig ( 'lorenz_ode_3d.pdf', format='pdf' )
    plt.show ( )
    return

# Compute the time-derivative of a Lorenz system

def x_dot(x1, y1, z1, x2, y2, z2, t,alp):
    x1_dot=sigma*(y1 - x1)+alp*(x2-x1)
    x2_dot=sigma*(y2 - x2)+alp*(x1-x2)
    return(x1_dot,x2_dot)

def y_dot(x1, y1, z1, x2, y2, z2, t,alp):
    y1_dot=rho*x1 - y1 - x1*z1+alp*(y2-y1)
    y2_dot=rho*x2 - y2 - x2*z2+alp*(y1-y2)
    return(y1_dot,y2_dot)

def z_dot(x1, y1, z1, x2, y2, z2, t,alp):
    z1_dot=-1*(beta*z1) + x1*y1+alp*(z2-z1)
    z2_dot=-1*(beta*z2) + x2*y2+alp*(z1-z2)
    return(z1_dot,z2_dot)

# 4th order Runge Kutta (RK-4) numerical method
def RungeKutta4(a1,b1,c1,a2,b2,c2,alp,n):
    x1 = np.zeros(n)
    y1 = np.zeros(n)
    z1 = np.zeros(n)

    x2 = np.zeros(n)
    y2 = np.zeros(n)
    z2 = np.zeros(n)
    t = np.zeros(n)

    x1[0] = a1
    y1[0] = b1
    z1[0] = c1

    x2[0] = a2
    y2[0] = b2
    z2[0] = c2
    t[0] = 0
    dt = 0.01
```

```
# Compute the approximate solution at equally spaced times.
for k in tqdm(range(n-1)):

    t[k+1] = t[k] + dt

    k1,u1 = x_dot(x1[k], y1[k], z1[k],x2[k], y2[k], z2[k], t[k],alp)
    l1,v1 = y_dot(x1[k], y1[k], z1[k],x2[k], y2[k], z2[k], t[k],alp)
    m1,w1 = z_dot(x1[k], y1[k], z1[k],x2[k], y2[k], z2[k], t[k],alp)

    k2,u2 = x_dot((x1[k] + 0.5*k1*dt), (y1[k] + 0.5*l1*dt),
     (z1[k] + 0.5*m1*dt),(x2[k] + 0.5*u1*dt), (y2[k] + 0.5*v1*dt),
      (z2[k] + 0.5*w1*dt), (t[k] + dt/2),alp)
    l2,v2 = y_dot((x1[k] + 0.5*k1*dt), (y1[k] + 0.5*l1*dt),
     (z1[k] + 0.5*m1*dt),(x2[k] + 0.5*u1*dt), (y2[k] + 0.5*v1*dt),
      (z2[k] + 0.5*w1*dt), (t[k] + dt/2),alp)
    m2,w2 = z_dot((x1[k] + 0.5*k1*dt), (y1[k] + 0.5*l1*dt),
     (z1[k] + 0.5*m1*dt),(x2[k] + 0.5*u1*dt), (y2[k] + 0.5*v1*dt),
      (z2[k] + 0.5*w1*dt), (t[k] + dt/2),alp)

    k3,u3 = x_dot((x1[k] + 0.5*k2*dt), (y1[k] + 0.5*l2*dt),
     (z1[k] + 0.5*m2*dt),(x2[k] + 0.5*u2*dt), (y2[k] + 0.5*v2*dt),
      (z2[k] + 0.5*w2*dt), (t[k] + dt/2),alp)
    l3,v3 = y_dot((x1[k] + 0.5*k2*dt), (y1[k] + 0.5*l2*dt),
     (z1[k] + 0.5*m2*dt),(x2[k] + 0.5*u2*dt), (y2[k] + 0.5*v2*dt),
      (z2[k] + 0.5*w2*dt), (t[k] + dt/2),alp)
    m3,w3 = z_dot((x1[k] + 0.5*k2*dt), (y1[k] + 0.5*l2*dt),
     (z1[k] + 0.5*m2*dt),(x2[k] + 0.5*u2*dt), (y2[k] + 0.5*v2*dt),
      (z2[k] + 0.5*w2*dt), (t[k] + dt/2),alp)

    k4,u4 = x_dot((x1[k] + k3*dt), (y1[k] + l3*dt), (z1[k] + m3*dt),
     (x2[k] + u3*dt), (y2[k] + v3*dt), (z2[k] + w3*dt), (t[k] + dt),alp)
    l4,v4 = y_dot((x1[k] + k3*dt), (y1[k] + l3*dt), (z1[k] + m3*dt),
     (x2[k] + u3*dt), (y2[k] + v3*dt), (z2[k] + w3*dt), (t[k] + dt),alp)
    m4,w4 = z_dot((x1[k] + k3*dt), (y1[k] + l3*dt), (z1[k] + m3*dt),
     (x2[k] + u3*dt), (y2[k] + v3*dt), (z2[k] + w3*dt), (t[k] + dt),alp)

    x1[k+1] = x1[k] + (dt*(k1 + 2*k2 + 2*k3 + k4) / 6)
    y1[k+1] = y1[k] + (dt*(l1 + 2*l2 + 2*l3 + l4) / 6)
    z1[k+1] = z1[k] + (dt*(m1 + 2*m2 + 2*m3 + m4) / 6)

    x2[k+1] = x2[k] + (dt*(u1 + 2*u2 + 2*u3 + u4) / 6)
    y2[k+1] = y2[k] + (dt*(v1 + 2*v2 + 2*v3 + v4) / 6)
    z2[k+1] = z2[k] + (dt*(w1 + 2*w2 + 2*w3 + w4) / 6)

return x1,y1,z1,x2,y2,z2,t
```

```
# Initial conditions and function call
sigma = 10.0
beta = 8.0/3.0
rho = 28.0
a=0.5
N=5000
x_1, y_1, z_1, x_2, y_2, z_2,t=RungeKutta4(7,10,15,10,15,25,a,N)
x1, y1, z1, x2, y2, z2,t=RungeKutta4(3,10,15,10,15,25,0.4,N)

#Sync Error vs Time forlast vlaue of alpha (=0.5)
plt.plot(t,[math.sqrt(pow(x1-x2,2)+pow(y1-y2,2)+pow(z1 - z2,2))
for x1,x2,y1,y2,z1, z2 in zip(x_1,x_2,y_1,y_2,z_1, z_2)])

plt.figure (figsize = (11,6.5))
plt.subplot(3,2,2)
plt.title('(b) Synchronization', fontsize=14)
plt.plot ( t, x_1, linewidth = 1, color = 'b' )
plt.plot ( t, x_2, linewidth = 1, color = 'r' )
plt.ylim(-20, 20)
plt.ylabel ( r'$x,\widetilde{x}$',fontsize=12)

plt.subplot(3,2,4)
plt.plot ( t, y_1, linewidth = 1, color = 'b' )
plt.plot ( t, y_2, linewidth = 1, color = 'r' )
#plt.ylim(-30, 30)
plt.ylabel ( r'$y,\widetilde{y}$',fontsize=12)

plt.subplot(3,2,6)
plt.plot ( t, z_1, linewidth = 1, color = 'b' )
plt.plot ( t, z_2, linewidth = 1, color = 'r' )
plt.xlabel ( 'Time (t)' ,fontsize=11)
plt.ylabel ( r'$z,\widetilde{z}$',fontsize=12)
#plt.ylim(0, 45)

plt.subplot(3,2,1)
plt.plot ( t, x1, linewidth = 1, color = 'b' )
plt.plot ( t, x2, linewidth = 1, color = 'r' )
plt.title('(a) No Synchronization', fontsize=14)
#plt.ylim(-30, 30)
plt.ylabel ( r'$x,\widetilde{x}$',fontsize=12)

plt.subplot(3,2,3)
plt.plot ( t, y1, linewidth = 1, color = 'b' )
plt.plot ( t, y2, linewidth = 1, color = 'r' )
#plt.ylim(-30, 30)
plt.ylabel ( r'$y,\widetilde{y}$',fontsize=12)
```

```
plt.subplot(3,2,5)
plt.plot ( t, z1, linewidth = 1, color = 'b' )
plt.plot ( t, z2, linewidth = 1, color = 'r' )
plt.xlabel ( 'Time (t)' ,fontsize=11)
plt.ylabel ( r'$z,\widetilde{z}$',fontsize=12)
#plt.ylim(0, 45)
plt.tight_layout()
plt.savefig('Complete_Sync.pdf', format='pdf')
plt.show ( )

lorenz_ode_plot_3d(x_1,y_1,z_1)

plt.figure ( figsize = (16,4))
plt.subplot(1,3,1)
plt.plot ( x_1, z_1, linewidth = 0.5 )
plt.xlabel ( '<--- x-axis --->' ,fontsize=12)
plt.ylabel ( '<--- z-axis --->' ,fontsize=12)

plt.subplot(1,3,2)
plt.plot ( x_1, y_1, linewidth = 0.5)
plt.xlabel ( '<--- x-axis --->' ,fontsize=12)
plt.ylabel ( '<--- y-axis --->' ,fontsize=12)

plt.subplot(1,3,3)
plt.plot ( y_1, z_1, linewidth = 0.5 )
plt.xlabel ( '<--- y-axis --->' ,fontsize=12)
plt.ylabel ( '<--- z-axis --->' ,fontsize=12)
plt.savefig('Lorenz_2d.pdf', format='pdf')

plt.figure ( figsize = (7.5,5.5))
plt.plot ( y_1, z_1, linewidth = 0.5 )
plt.xlabel ( '<--- y-axis --->' ,fontsize=12)
plt.ylabel ( '<--- z-axis --->' ,fontsize=12)
plt.savefig('Lorenz_2d_yz.pdf', format='pdf')

plt.figure ( figsize = (7.5,5.5))
plt.plot ( x_1, y_1, linewidth = 0.5)
plt.xlabel ( '<--- x-axis --->' ,fontsize=12)
plt.ylabel ( '<--- y-axis --->' ,fontsize=12)
plt.savefig('Lorenz_2d_xy.pdf', format='pdf')
```

A.1.2 For Rössler System

```
# Importing Packages
import math, sys
import numpy as np
import pandas as pd
```

```
import matplotlib.pyplot as plt
import time
from ipywidgets import interact, interactive
from scipy import integrate
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import cnames
from matplotlib import animation
import random
from tqdm.notebook import tqdm

# Function for 2-D and 3-D Plot
def Rossler_ode_plot_components (x, y, z):
    plt.figure (figsize = (8.5,8.5))
    plt.plot ( t, x, linewidth = 1, color = 'b' )
    plt.plot ( t, y, linewidth = 1, color = 'r' )
    plt.plot ( t, z, linewidth = 1, color = 'g' )
    plt.grid ( True )
    plt.xlabel ( '<--- Time --->' )
    plt.ylabel ( '<--- x(t), y(t), z(t) --->' )
    plt.title ( 'Rossler Time Series Plot' )

    plt.legend(['x','y','z'], loc = 2)
    plt.savefig ( 'Rossler_ode_components.png' )
    plt.show ( )
    return

def Rossler_ode_plot_3d (x, y, z):
    fig = plt.figure ( figsize = (8.5,8.5))
    ax = fig.add_subplot(projection='3d')
    ax.plot ( x, y, z, linewidth = 0.5, color = 'orangered' )
    ax.grid ( True )
    ax.set_xlabel ( '<--- x(t) --->' )
    ax.set_ylabel ( '<--- y(t) --->' )
    ax.set_zlabel ( '<--- z(t) --->' ,rotation=90)

    plt.savefig ( 'Rossler_ode_3d.png' )
    plt.show ( )
    return

#Compute the time-derivative of a Rossler system
def x_dot(x1, y1, z1):
    x1_dot=-y1 - z1
    return x1_dot
def y_dot(x1, y1, z1):
    y1_dot= x1+a*y1
    return y1_dot
def z_dot(x1, y1, z1,c):
```

```
    z1_dot=b+z1*(x1-c)
    return z1_dot

# 4th order Runge Kutta (RK-4) numerical method
def RungeKutta4(a1,b1,c1,n,c):
    x1 = np.zeros(n)
    y1 = np.zeros(n)
    z1 = np.zeros(n)
    t = np.zeros(n)

    x1[0] = a1
    y1[0] = b1
    z1[0] = c1

    t[0] = 0
    dt = 0.01

    #Come the approximate soion at elly spaced times.
    for k in tqdm(range(n-1)):

        t[k+1] = t[k] + dt

        k1 = x_dot(x1[k], y1[k], z1[k])
        l1 = y_dot(x1[k], y1[k], z1[k])
        m1 = z_dot(x1[k], y1[k], z1[k],c)

        k2 = x_dot((x1[k] + 0.5*k1*dt), (y1[k] + 0.5*l1*dt), (z1[k] + 0.5*m1*dt))
        l2 = y_dot((x1[k] + 0.5*k1*dt), (y1[k] + 0.5*l1*dt), (z1[k] + 0.5*m1*dt))
        m2 = z_dot((x1[k] + 0.5*k1*dt), (y1[k] + 0.5*l1*dt), (z1[k] + 0.5*m1*dt),c)

        k3 = x_dot((x1[k] + 0.5*k2*dt), (y1[k] + 0.5*l2*dt), (z1[k] + 0.5*m2*dt))
        l3 = y_dot((x1[k] + 0.5*k2*dt), (y1[k] + 0.5*l2*dt), (z1[k] + 0.5*m2*dt))
        m3 = z_dot((x1[k] + 0.5*k2*dt), (y1[k] + 0.5*l2*dt), (z1[k] + 0.5*m2*dt),c)

        k4 = x_dot((x1[k] + k3*dt), (y1[k] + l3*dt), (z1[k] + m3*dt))
        l4 = y_dot((x1[k] + k3*dt), (y1[k] + l3*dt), (z1[k] + m3*dt))
        m4 = z_dot((x1[k] + k3*dt), (y1[k] + l3*dt), (z1[k] + m3*dt),c)

        x1[k+1] = x1[k] + (dt*(k1 + 2*k2 + 2*k3 + k4) / 6)
        y1[k+1] = y1[k] + (dt*(l1 + 2*l2 + 2*l3 + l4) / 6)
        z1[k+1] = z1[k] + (dt*(m1 + 2*m2 + 2*m3 + m4) / 6)


    return x1,y1,z1,t

# Initial conditions and function call
a=0.2
```

```
b=0.2
c1=2.3
c2=2.9
c3=4.1
c4=5.7
N= 10000
x_1, y_1, z_1,t=RungeKutta4(-2.0, 2.0, 2.0,N,c1)
x1,y1,z1,t=RungeKutta4(-2.0, 2.0, 3.0,N,c2)
x,y,z,t=RungeKutta4(-2.0, 2.0, 3.0,N,c3)
x_,y_,z_,t=RungeKutta4(1, 0, 0,N,c4)
df = pd.DataFrame(x_)
df.to_csv('x1.csv', index=False)

plt.figure ( figsize = (7.5,5.5))
plt.plot ( x_1, y_1, linewidth = 0.5, color = 'b' )
plt.xlabel ( '<--- x-axis --->' )
plt.ylabel ( '<--- y-axis --->' )
plt.savefig('Rossler_2d.pdf', format='pdf')

plt.figure ( figsize = (10,10))
plt.subplots_adjust(wspace=0.35, hspace=0.25)
plt.subplot(2,2,1)
plt.plot ( x_1, y_1, linewidth = 0.5 )
plt.xlabel ( '<--- x-axis --->' ,fontsize=12)
plt.ylabel ( '<--- y-axis --->' ,fontsize=12)
plt.title("(a) c=2.3")

plt.subplot(2,2,2)
plt.plot ( x1, y1, linewidth = 0.5)
plt.xlabel ( '<--- x-axis --->' ,fontsize=12)
plt.ylabel ( '<--- y-axis --->' ,fontsize=12)
plt.title("(b) c=2.9")

plt.subplot(2,2,3)
plt.plot ( x, y, linewidth = 0.5 )
plt.xlabel ( '<--- x-axis --->' ,fontsize=12)
plt.ylabel ( '<--- y-axis --->' ,fontsize=12)
plt.title("(c) c=4.1")

plt.subplot(2,2,4)
plt.plot ( x_, y_, linewidth = 0.5 )
plt.xlabel ( '<--- x-axis --->' ,fontsize=12)
plt.ylabel ( '<--- y-axis --->' ,fontsize=12)
plt.title("(d) c=5.7")
plt.savefig('Rossler_2d_2.pdf', format='pdf')

Rossler_ode_plot_3d(x_,y_,z_)
```

A.1.3 For Complete synchronization (CS) with coupling function $H = I$

```
# Importing Packages
import math, sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time
from scipy import integrate
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import cnames
from matplotlib import animation
import random
from tqdm.notebook import tqdm

#Compute the time-derivative of a Lorenz system
def x_dot(x1, y1, z1, x2, y2, z2, t,alp):
    x1_dot=sigma*(y1 - x1)+alp*(x2-x1)
    x2_dot=sigma*(y2 - x2)+alp*(x1-x2)
    return(x1_dot,x2_dot)

def y_dot(x1, y1, z1, x2, y2, z2, t,alp):
    y1_dot=rho*x1 - y1 - x1*z1+alp*(y2-y1)
    y2_dot=rho*x2 - y2 - x2*z2+alp*(y1-y2)
    return(y1_dot,y2_dot)

def z_dot(x1, y1, z1, x2, y2, z2, t,alp):
    z1_dot=-1*(beta*z1) + x1*y1+alp*(z2-z1)
    z2_dot=-1*(beta*z2) + x2*y2+alp*(z1-z2)
    return(z1_dot,z2_dot)

# 4th order Runge Kutta (RK-4) numerical method
def RungeKutta4(a1,b1,c1,a2,b2,c2,alp,n):
    x1 = np.zeros(n)
    y1 = np.zeros(n)
    z1 = np.zeros(n)

    x2 = np.zeros(n)
    y2 = np.zeros(n)
    z2 = np.zeros(n)
    t = np.zeros(n)

    x1[0] = a1
    y1[0] = b1
    z1[0] = c1

    x2[0] = a2
```

```
y2[0] = b2
z2[0] = c2
t[0] = 0
dt = 0.01

# Compute the approximate solution at equally spaced times.
for k in tqdm(range(n-1)):

    t[k+1] = t[k] + dt

    k1,u1 = x_dot(x1[k], y1[k], z1[k],x2[k], y2[k], z2[k], t[k],alp)
    l1,v1 = y_dot(x1[k], y1[k], z1[k],x2[k], y2[k], z2[k], t[k],alp)
    m1,w1 = z_dot(x1[k], y1[k], z1[k],x2[k], y2[k], z2[k], t[k],alp)

    k2,u2 = x_dot((x1[k] + 0.5*k1*dt), (y1[k] + 0.5*l1*dt),
     (z1[k] + 0.5*m1*dt),(x2[k] + 0.5*u1*dt), (y2[k] + 0.5*v1*dt),
      (z2[k] + 0.5*w1*dt), (t[k] + dt/2),alp)
    l2,v2 = y_dot((x1[k] + 0.5*k1*dt), (y1[k] + 0.5*l1*dt),
     (z1[k] + 0.5*m1*dt),(x2[k] + 0.5*u1*dt), (y2[k] + 0.5*v1*dt),
      (z2[k] + 0.5*w1*dt), (t[k] + dt/2),alp)
    m2,w2 = z_dot((x1[k] + 0.5*k1*dt), (y1[k] + 0.5*l1*dt),
     (z1[k] + 0.5*m1*dt),(x2[k] + 0.5*u1*dt), (y2[k] + 0.5*v1*dt),
      (z2[k] + 0.5*w1*dt), (t[k] + dt/2),alp)

    k3,u3 = x_dot((x1[k] + 0.5*k2*dt), (y1[k] + 0.5*l2*dt),
     (z1[k] + 0.5*m2*dt),(x2[k] + 0.5*u2*dt), (y2[k] + 0.5*v2*dt),
      (z2[k] + 0.5*w2*dt), (t[k] + dt/2),alp)
    l3,v3 = y_dot((x1[k] + 0.5*k2*dt), (y1[k] + 0.5*l2*dt),
     (z1[k] + 0.5*m2*dt),(x2[k] + 0.5*u2*dt), (y2[k] + 0.5*v2*dt),
      (z2[k] + 0.5*w2*dt), (t[k] + dt/2),alp)
    m3,w3 = z_dot((x1[k] + 0.5*k2*dt), (y1[k] + 0.5*l2*dt),
     (z1[k] + 0.5*m2*dt),(x2[k] + 0.5*u2*dt), (y2[k] + 0.5*v2*dt),
      (z2[k] + 0.5*w2*dt), (t[k] + dt/2),alp)

    k4,u4 = x_dot((x1[k] + k3*dt), (y1[k] + l3*dt), (z1[k] + m3*dt),
     (x2[k] + u3*dt), (y2[k] + v3*dt), (z2[k] + w3*dt), (t[k] + dt),alp)
    l4,v4 = y_dot((x1[k] + k3*dt), (y1[k] + l3*dt), (z1[k] + m3*dt),
     (x2[k] + u3*dt), (y2[k] + v3*dt), (z2[k] + w3*dt), (t[k] + dt),alp)
    m4,w4 = z_dot((x1[k] + k3*dt), (y1[k] + l3*dt), (z1[k] + m3*dt),
     (x2[k] + u3*dt), (y2[k] + v3*dt), (z2[k] + w3*dt), (t[k] + dt),alp)

    x1[k+1] = x1[k] + (dt*(k1 + 2*k2 + 2*k3 + k4) / 6)
    y1[k+1] = y1[k] + (dt*(l1 + 2*l2 + 2*l3 + l4) / 6)
    z1[k+1] = z1[k] + (dt*(m1 + 2*m2 + 2*m3 + m4) / 6)

    x2[k+1] = x2[k] + (dt*(u1 + 2*u2 + 2*u3 + u4) / 6)
    y2[k+1] = y2[k] + (dt*(v1 + 2*v2 + 2*v3 + v4) / 6)
```

```
        z2[k+1] = z2[k] + (dt*(w1 + 2*w2 + 2*w3 + w4) / 6)

    return x1,y1,z1,x2,y2,z2,t

#Synchronization error function
def eAlpha(tmp,a,N):
  tmp1,tmp2,tmp3=0,0,0
  x_1, y_1, z_1, x_2, y_2, z_2, t=[],[],[],[],[],[],[]
  x_1, y_1, z_1, x_2, y_2, z_2, t = RungeKutta4(tmp[0],tmp[1],tmp[2],
                                                tmp[3],tmp[4],tmp[5],a,N)

  e=sum(np.sqrt(pow((x1 - x2),2) +pow((y1 - y2),2) +pow((z1 - z2),2) )
    for x1, x2,y1,y2,z1,z2 in zip(x_1, x_2,y_1,y_2,z_1,z_2)) #total_absolute_difference
  return e #,t,x_1, y_1, z_1, x_2, y_2, z_2

from joblib import Parallel, delayed

# Initial conditions and function call
sigma = 10.0
beta = 8.0/3.0
rho = 28.0
E = {}
N=int(2000/0.01)
for i in tqdm(range(0,300)):
    tmp=random.sample(range(10,50), 6)
    E[i] = []
    results = Parallel(n_jobs=6)(delayed(eAlpha)(tmp,a,N)
                                 for a in np.arange(0.4,0.504,0.004))
    E[i].extend(results)
A=pd.DataFrame(E)
E_new=A.sum(axis=1)
k=len(E.keys())
#E_new

plt.figure (figsize = (8.5,6.5))
plt.plot(np.arange(0.4,0.504,0.004),E_new/k, linewidth = 1,
         color = 'black',marker='o', markersize=4)
plt.xlabel(r"$\alpha$",fontsize=14)
plt.ylabel(r"$E$",fontsize=14)
plt.savefig('CS-Img1.pdf', format='pdf')
plt.show()

#A.to_csv('A_for_E_new.csv', index=False)
#d=pd.read_csv("/content/UploadedA_for_E_new.csv")
```

A.1.4 For CS with coupling function H as x-coupling matrix

```
# Importing Packages
import math, sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time
from scipy import integrate
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import cnames
from matplotlib import animation
import random
from tqdm.notebook import tqdm

#Compute the time-derivative of a Lorenz system
def x_dot(x1, y1, z1, x2, y2, z2, t,alp):
    x1_dot=sigma*(y1 - x1)+alp*(x2-x1)
    x2_dot=sigma*(y2 - x2)+alp*(x1-x2)
    return(x1_dot,x2_dot)

def y_dot(x1, y1, z1, x2, y2, z2, t,alp):
    y1_dot=rho*x1 - y1 - x1*z1
    y2_dot=rho*x2 - y2 - x2*z2
    return(y1_dot,y2_dot)

def z_dot(x1, y1, z1, x2, y2, z2, t,alp):
    z1_dot=-1*(beta*z1) + x1*y1
    z2_dot=-1*(beta*z2) + x2*y2
    return(z1_dot,z2_dot)

# 4th order Runge Kutta (RK-4) numerical method
def RungeKutta4(a1,b1,c1,a2,b2,c2,alp,n):
    x1 = np.zeros(n)
    y1 = np.zeros(n)
    z1 = np.zeros(n)

    x2 = np.zeros(n)
    y2 = np.zeros(n)
    z2 = np.zeros(n)
    t = np.zeros(n)

    x1[0] = a1
    y1[0] = b1
    z1[0] = c1

    x2[0] = a2
```

```
y2[0] = b2
z2[0] = c2
t[0] = 0
dt = 0.01

# Compute the approximate solution at equally spaced times.
for k in tqdm(range(n-1)):

    t[k+1] = t[k] + dt

    k1,u1 = x_dot(x1[k], y1[k], z1[k],x2[k], y2[k], z2[k], t[k],alp)
    l1,v1 = y_dot(x1[k], y1[k], z1[k],x2[k], y2[k], z2[k], t[k],alp)
    m1,w1 = z_dot(x1[k], y1[k], z1[k],x2[k], y2[k], z2[k], t[k],alp)

    k2,u2 = x_dot((x1[k] + 0.5*k1*dt), (y1[k] + 0.5*l1*dt),
     (z1[k] + 0.5*m1*dt),(x2[k] + 0.5*u1*dt), (y2[k] + 0.5*v1*dt),
      (z2[k] + 0.5*w1*dt), (t[k] + dt/2),alp)
    l2,v2 = y_dot((x1[k] + 0.5*k1*dt), (y1[k] + 0.5*l1*dt),
     (z1[k] + 0.5*m1*dt),(x2[k] + 0.5*u1*dt), (y2[k] + 0.5*v1*dt),
      (z2[k] + 0.5*w1*dt), (t[k] + dt/2),alp)
    m2,w2 = z_dot((x1[k] + 0.5*k1*dt), (y1[k] + 0.5*l1*dt),
     (z1[k] + 0.5*m1*dt),(x2[k] + 0.5*u1*dt), (y2[k] + 0.5*v1*dt),
      (z2[k] + 0.5*w1*dt), (t[k] + dt/2),alp)

    k3,u3 = x_dot((x1[k] + 0.5*k2*dt), (y1[k] + 0.5*l2*dt),
     (z1[k] + 0.5*m2*dt),(x2[k] + 0.5*u2*dt), (y2[k] + 0.5*v2*dt),
      (z2[k] + 0.5*w2*dt), (t[k] + dt/2),alp)
    l3,v3 = y_dot((x1[k] + 0.5*k2*dt), (y1[k] + 0.5*l2*dt),
     (z1[k] + 0.5*m2*dt),(x2[k] + 0.5*u2*dt), (y2[k] + 0.5*v2*dt),
      (z2[k] + 0.5*w2*dt), (t[k] + dt/2),alp)
    m3,w3 = z_dot((x1[k] + 0.5*k2*dt), (y1[k] + 0.5*l2*dt),
     (z1[k] + 0.5*m2*dt),(x2[k] + 0.5*u2*dt), (y2[k] + 0.5*v2*dt),
      (z2[k] + 0.5*w2*dt), (t[k] + dt/2),alp)

    k4,u4 = x_dot((x1[k] + k3*dt), (y1[k] + l3*dt), (z1[k] + m3*dt),
     (x2[k] + u3*dt), (y2[k] + v3*dt), (z2[k] + w3*dt), (t[k] + dt),alp)
    l4,v4 = y_dot((x1[k] + k3*dt), (y1[k] + l3*dt), (z1[k] + m3*dt),
     (x2[k] + u3*dt), (y2[k] + v3*dt), (z2[k] + w3*dt), (t[k] + dt),alp)
    m4,w4 = z_dot((x1[k] + k3*dt), (y1[k] + l3*dt), (z1[k] + m3*dt),
     (x2[k] + u3*dt), (y2[k] + v3*dt), (z2[k] + w3*dt), (t[k] + dt),alp)

    x1[k+1] = x1[k] + (dt*(k1 + 2*k2 + 2*k3 + k4) / 6)
    y1[k+1] = y1[k] + (dt*(l1 + 2*l2 + 2*l3 + l4) / 6)
    z1[k+1] = z1[k] + (dt*(m1 + 2*m2 + 2*m3 + m4) / 6)

    x2[k+1] = x2[k] + (dt*(u1 + 2*u2 + 2*u3 + u4) / 6)
    y2[k+1] = y2[k] + (dt*(v1 + 2*v2 + 2*v3 + v4) / 6)
```

```
        z2[k+1] = z2[k] + (dt*(w1 + 2*w2 + 2*w3 + w4) / 6)

    return x1,y1,z1,x2,y2,z2,t

#Synchronization error function
def eAlpha(tmp,a,N):
  tmp1,tmp2,tmp3=0,0,0
  x_1, y_1, z_1, x_2, y_2, z_2, t=[],[],[],[],[],[],[]
  x_1, y_1, z_1, x_2, y_2, z_2, t = RungeKutta4(tmp[0],tmp[1],tmp[2],
                                                tmp[3],tmp[4],tmp[5],a,N)

  e=sum(np.sqrt(pow((x1 - x2),2) +pow((y1 - y2),2) +pow((z1 - z2),2) )
  for x1, x2,y1,y2,z1,z2 in zip(x_1, x_2,y_1,y_2,z_1,z_2)) #total_absolute_difference
  return e

from joblib import Parallel, delayed

# Initial conditions and function call
sigma = 10.0
beta = 8.0/3.0
rho = 28.0
E = {}
N=int(2000/0.01)
for i in tqdm(range(0,300)):
    tmp=random.sample(range(-20,30), 6)
    E[i] = []

    results = Parallel(n_jobs=12)(delayed(eAlpha)(tmp,a,N)
                                  for a in np.arange(3,4.5,0.06))
    E[i].extend(results)

A=pd.DataFrame(E)
E_new=A.sum(axis=1)
k=len(E.keys())

#Plot
plt.figure (figsize = (8.5,6.5))
plt.plot(np.arange(3,4.5,0.06),E_new/k, linewidth = 1,
         color = 'black',marker='o', markersize=4)
plt.xlabel(r"$\alpha$",fontsize=14)
plt.ylabel(r"$E$",fontsize=14)
plt.savefig('Img1.pdf', format='pdf')
plt.show()

#A.to_csv('A_for_E_new.csv', index=False)

#d=pd.read_csv("/content/sample_data.csv")
```

A.1.5 For CS in drive-response configuration using RK4 method

```
# Importing Packages
import math, sys
import numpy as np
import matplotlib.pyplot as plt
import time
from ipywidgets import interact, interactive
from scipy import integrate
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import cnames
from matplotlib import animation
import random
from tqdm.notebook import tqdm

#Compute the time-derivative of a Lorenz system
def x_dot(x1, y1, z1):
    x1_dot=sigma*(y1 - x1)
    return x1_dot

def y_dot(x1, y1, z1, y2, z2):
    y1_dot=rho*x1 - y1 - x1*z1
    y2_dot=rho*x1 - y2 - x1*z2
    return(y1_dot,y2_dot)

def z_dot(x1, y1, z1, y2, z2):
    z1_dot=-1*(beta*z1) + x1*y1
    z2_dot=-1*(beta*z2) + x1*y2
    return(z1_dot,z2_dot)

# 4th order Runge Kutta (RK-4) numerical method
def RungeKutta4(a1,b1,c1,b2,c2,n):

    x1 = np.zeros(n)
    y1 = np.zeros(n)
    z1 = np.zeros(n)


    y2 = np.zeros(n)
    z2 = np.zeros(n)
    t = np.zeros(n)

    x1[0] = a1
    y1[0] = b1
    z1[0] = c1

    y2[0] = b2
```

```
    z2[0] = c2
    t[0] = 0
    dt = 0.01

    #Compute the approximate solution at equally spaced times.
    for k in range (n-1):

        t[k+1] = t[k] + dt

        k1 = x_dot(x1[k], y1[k], z1[k])
        l1,v1 = y_dot(x1[k], y1[k], z1[k], y2[k], z2[k])
        m1,w1 = z_dot(x1[k], y1[k], z1[k], y2[k], z2[k])

        k2 = x_dot((x1[k] + 0.5*k1*dt), (y1[k] + 0.5*l1*dt), (z1[k] + 0.5*m1*dt))
        l2,v2 = y_dot((x1[k] + 0.5*k1*dt), (y1[k] + 0.5*l1*dt), (z1[k] + 0.5*m1*dt),
         (y2[k] + 0.5*v1*dt), (z2[k] + 0.5*w1*dt))
        m2,w2 = z_dot((x1[k] + 0.5*k1*dt), (y1[k] + 0.5*l1*dt), (z1[k] + 0.5*m1*dt),
         (y2[k] + 0.5*v1*dt), (z2[k] + 0.5*w1*dt))

        k3 = x_dot((x1[k] + 0.5*k2*dt), (y1[k] + 0.5*l2*dt), (z1[k] + 0.5*m2*dt))
        l3,v3 = y_dot((x1[k] + 0.5*k2*dt), (y1[k] + 0.5*l2*dt), (z1[k] + 0.5*m2*dt),
         (y2[k] + 0.5*v2*dt), (z2[k] + 0.5*w2*dt))
        m3,w3 = z_dot((x1[k] + 0.5*k2*dt), (y1[k] + 0.5*l2*dt), (z1[k] + 0.5*m2*dt),
         (y2[k] + 0.5*v2*dt), (z2[k] + 0.5*w2*dt))

        k4 = x_dot((x1[k] + k3*dt), (y1[k] + l3*dt), (z1[k] + m3*dt))
        l4,v4 = y_dot((x1[k] + k3*dt), (y1[k] + l3*dt), (z1[k] + m3*dt), (y2[k] + v3*dt),
         (z2[k] + w3*dt))
        m4,w4 = z_dot((x1[k] + k3*dt), (y1[k] + l3*dt), (z1[k] + m3*dt), (y2[k] + v3*dt),
         (z2[k] + w3*dt))

        x1[k+1] = x1[k] + (dt*(k1 + 2*k2 + 2*k3 + k4) / 6)
        y1[k+1] = y1[k] + (dt*(l1 + 2*l2 + 2*l3 + l4) / 6)
        z1[k+1] = z1[k] + (dt*(m1 + 2*m2 + 2*m3 + m4) / 6)

        y2[k+1] = y2[k] + (dt*(v1 + 2*v2 + 2*v3 + v4) / 6)
        z2[k+1] = z2[k] + (dt*(w1 + 2*w2 + 2*w3 + w4) / 6)

    return x1,y1,z1,y2,z2, t

# Initial conditions and function call
sigma = 16
beta = 4
rho = 45.2
N=8000
x_1, y_1, z_1, y_2, z_2, t = RungeKutta4(-10.1,10.1,10.1,0.1,0.1,N)
```

```
#Plot
plt.figure (figsize = (6.5,6.5))
fig, axs = plt.subplots(2,1)

axs[0].plot ( t, y_1, linewidth = 1, color = 'b' )
axs[0].plot ( t, y_2, linewidth = 1, color = 'r' )
#plt.ylim(-30, 30)
plt.ylabel ( r'$y,\bar{y}$')
axs[1].plot ( t, z_1, linewidth = 1, color = 'b' )
axs[1].plot ( t, z_2, linewidth = 1, color = 'r' )
plt.xlabel ( '<--- Time --->' )
plt.ylabel ( r'$z,\bar{z}$')
#plt.ylim(0, 45)
plt.show ( )

Y,Z=[],[]
for y1, y2 in zip(y_1, y_2):
    Y.append(abs(y1 - y2))
for z1, z2 in zip(z_1, z_2):
    Z.append(abs(z1 - z2))

#Sync Error vs Time
plt.plot(t,[math.sqrt(pow(y1-y2,2)+pow(z1 - z2,2))
        for x1,y1,y2,z1, z2 in zip(x_1,y_1,y_2,z_1, z_2)])
plt.ylabel(r"$E$",fontsize=14)
plt.xlabel("t",fontsize=14)

#Plot
plt.figure (figsize = (8.5,6.5))
ax = plt.axes(yscale='log')
ax.plot( t,Y, linewidth = 1, color = 'black', label=r'$\Delta$y')
ax.plot( t,Z, linewidth = 1, color = 'red', label=r'$\Delta$z') #ax.semilog()
ax.set_ylim(pow(10,-9),pow(10,2))
ax.set_xlim(0,8)
ax.legend()
plt.ylabel(r"$\Delta y$,$\Delta z$",fontsize=14)
plt.xlabel("Time (t)",fontsize=14)
plt.savefig('RK4 Master Slave.pdf', format='pdf')
plt.show()
```

A.1.6 For Phase Synchronization

```
# Importing Packages
import math, sys
import numpy as np
import matplotlib.pyplot as plt
import time
```

```
from ipywidgets import interact, interactive
from scipy import integrate
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import cnames
from matplotlib import animation
import random
import pandas as pd
from tqdm.notebook import tqdm

# Compute the time-derivative of a Rossler systems
def x_dot(x1, y1, z1, x2, y2, z2,alp):
    x1_dot=-1*wo1*y1-z1+alp*(x2-x1)
    x2_dot=-1*wo2*y2-z2+alp*(x1-x2)
    return x1_dot,x2_dot

def y_dot(x1, y1, z1, x2, y2, z2):
    y1_dot=wo1*x1+a*y1
    y2_dot=wo2*x2+a*y2
    return y1_dot,y2_dot

def z_dot(x1, y1, z1, x2, y2, z2):
    z1_dot=b+z1*(x1-c)
    z2_dot=b+z2*(x2-c)
    return z1_dot,z2_dot

# 4th order Runge Kutta (RK-4) numerical method
def RungeKutta4(a1,b1,c1,a2,b2,c2, alp, n):

    x1 = np.zeros(n)
    y1 = np.zeros(n)
    z1 = np.zeros(n)

    x2 = np.zeros(n)
    y2 = np.zeros(n)
    z2 = np.zeros(n)
    t = np.zeros(n)

    x1[0] = a1
    y1[0] = b1
    z1[0] = c1

    x2[0] = a2
    y2[0] = b2
    z2[0] = c2
    t[0] = 0

    #Compute the approximate solution at equally spaced times.
```

```
    for k in range (n-1):

        t[k+1] = t[k] + dt

        k1,u1 = x_dot(x1[k], y1[k], z1[k], x2[k], y2[k], z2[k],alp)
        l1,v1 = y_dot(x1[k], y1[k], z1[k], x2[k], y2[k], z2[k])
        m1,w1 = z_dot(x1[k], y1[k], z1[k], x2[k], y2[k], z2[k])

        k2,u2 = x_dot((x1[k] + 0.5*k1*dt), (y1[k] + 0.5*l1*dt), (z1[k] + 0.5*m1*dt),
                      (x2[k] + 0.5*u1*dt), (y2[k] + 0.5*v1*dt), (z2[k] + 0.5*w1*dt),alp)
        l2,v2 = y_dot((x1[k] + 0.5*k1*dt), (y1[k] + 0.5*l1*dt), (z1[k] + 0.5*m1*dt),
                      (x2[k] + 0.5*u1*dt), (y2[k] + 0.5*v1*dt), (z2[k] + 0.5*w1*dt))
        m2,w2 = z_dot((x1[k] + 0.5*k1*dt), (y1[k] + 0.5*l1*dt), (z1[k] + 0.5*m1*dt),
                      (x2[k] + 0.5*u1*dt), (y2[k] + 0.5*v1*dt), (z2[k] + 0.5*w1*dt))

        k3,u3 = x_dot((x1[k] + 0.5*k2*dt), (y1[k] + 0.5*l2*dt), (z1[k] + 0.5*m2*dt),
                      (x2[k] + 0.5*u2*dt), (y2[k] + 0.5*v2*dt), (z2[k] + 0.5*w2*dt),alp)
        l3,v3 = y_dot((x1[k] + 0.5*k2*dt), (y1[k] + 0.5*l2*dt), (z1[k] + 0.5*m2*dt),
                      (x2[k] + 0.5*u2*dt), (y2[k] + 0.5*v2*dt), (z2[k] + 0.5*w2*dt))
        m3,w3 = z_dot((x1[k] + 0.5*k2*dt), (y1[k] + 0.5*l2*dt), (z1[k] + 0.5*m2*dt),
                      (x2[k] + 0.5*u2*dt), (y2[k] + 0.5*v2*dt), (z2[k] + 0.5*w2*dt))

        k4,u4 = x_dot((x1[k] + k3*dt), (y1[k] + l3*dt), (z1[k] + m3*dt),
                      (x2[k] + u3*dt), (y2[k] + v3*dt), (z2[k] + w3*dt),alp)
        l4,v4 = y_dot((x1[k] + k3*dt), (y1[k] + l3*dt), (z1[k] + m3*dt),
                      (x2[k] + u3*dt), (y2[k] + v3*dt), (z2[k] + w3*dt))
        m4,w4 = z_dot((x1[k] + k3*dt), (y1[k] + l3*dt), (z1[k] + m3*dt),
                      (x2[k] + u3*dt), (y2[k] + v3*dt), (z2[k] + w3*dt))

        x1[k+1] = x1[k] + (dt*(k1 + 2*k2 + 2*k3 + k4) / 6)
        y1[k+1] = y1[k] + (dt*(l1 + 2*l2 + 2*l3 + l4) / 6)
        z1[k+1] = z1[k] + (dt*(m1 + 2*m2 + 2*m3 + m4) / 6)
        x2[k+1] = x2[k] + (dt*(u1 + 2*u2 + 2*u3 + u4) / 6)
        y2[k+1] = y2[k] + (dt*(v1 + 2*v2 + 2*v3 + v4) / 6)
        z2[k+1] = z2[k] + (dt*(w1 + 2*w2 + 2*w3 + w4) / 6)

    return x1,y1,z1,x2,y2,z2, t

#Function for Average Frequency
def Avg_feq(tmp,alp,N):

  tmp1,tmp2,tmp3=0,0,0
  x_1, y_1, z_1, x_2, y_2, z_2, t=[],[],[],[],[],[],[]
  x_1, y_1, z_1, x_2, y_2, z_2, t = RungeKutta4(tmp[0],tmp[1],tmp[2],tmp[3],tmp[4],tmp[5],alp,N)

  #------------Avg Freq------------
  P1=[]
```

```
  P2=[]
  Omega1,Omega2,D_Omega=[],[],[]
  P10=np.arctan2(y_1[0],x_1[0])
  P20=np.arctan2(y_2[0],x_2[0])
  for i,j,k,l in zip(x_1[1:],y_1[1:],x_2[1:],y_2[1:]):
    Omega1.append((np.arctan2(j,i)-P10)/(N*dt))
    Omega2.append((np.arctan2(l,k)-P20)/(N*dt))
  for m,n in zip(Omega1,Omega2):
    D_Omega.append(n-m)

  return D_Omega #,t,x_1, y_1, z_1, x_2, y_2, z_2

from joblib import Parallel, delayed

# Initial conditions and function call
wo1 = 0.97+0.02
wo2 = 0.97-0.02
a = 0.165
b= 0.2
c= 10
N= int(2000/0.01)
dt=0.01
O=[]

for i in tqdm(range(0,300)):
    tmp=random.sample(range(-10,10), 6)
    o=[]
    res = Parallel(n_jobs=3)(delayed(Avg_feq)(tmp,alp,N)
                             for alp in  np.arange(0.00,0.084,0.004))
    for j in res:
      o.append(np.average(j))

    O.append(o)

A=pd.DataFrame(O)
O_new=A.sum(axis=0)
k=len(O)

# Plot of frequency mismatch and time
plt.figure (figsize = (7.5,5.5))
plt.plot(np.arange(0.00,0.084,0.004), O_new/k, linewidth = 1,
        color = 'black',marker='o', markersize=4)
plt.ylabel(r'$\Delta\Omega$',fontsize=12)
plt.xlabel(r'$\alpha$',fontsize=12)
plt.savefig('Img3_1.pdf', format='pdf')
plt.show()
```

A.1.7 For Secure communication using CS for analog signal

```
# Importing Packages
import math, sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time
from ipywidgets import interact, interactive
from scipy import integrate
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import cnames
from matplotlib import animation
import random
from tqdm.notebook import tqdm
from scipy.integrate import odeint

# Original Message
def message(t):
  org=[]
  for i in t:
    org.append((0.1*np.sin(1.2*np.pi*np.sin(i+0.001)*np.sin(i+0.001)) *
                np.cos(10*np.pi*np.cos(0.9*(i+0.001))))/(np.pi*np.sin(i+0.001)*np.sin(i+0.001)))

  amplitude_noise = 0.2
  noise = 1 + amplitude_noise * np.random.normal(0, 1, len(org))
  m =  org * noise
  return m

#Compute the time-derivative of a Lorenz systems
def xyz1(a,b,c,k,tmp):
  x=sigma*(b - a)
  y=rho*a - b - a*c
  z=-1*(beta*c) + a*b
  return (x,y,z)

def xyz2(a,b,c,k,tmp):
  x=sigma*(b - a)
  y=(s[k]+tmp)*(rho - c) - b
  z=-1*(beta*c) + (s[k]+tmp)*b
  return (x,y,z)

# 4th order Runge Kutta (RK-4) numerical method
def RungeKutta4(fun,a1,b1,c1,n,dt):

    x1 = np.zeros(n)
    y1 = np.zeros(n)
```

```
    z1 = np.zeros(n)

    t = np.zeros(n)

    x1[0] = a1
    y1[0] = b1
    z1[0] = c1

    t[0] = 0

    #Compute the approximate solution at equally spaced times.
    for k in range (n-1):

        t[k+1] = t[k] + dt
        k1,l1,m1 = fun(x1[k], y1[k], z1[k],k,0)
        k2,l2,m2 = fun((x1[k] + 0.5*k1*dt), (y1[k] + 0.5*l1*dt),
         (z1[k] + 0.5*m1*dt),k,0.5*k1*dt)
        k3,l3,m3 = fun((x1[k] + 0.5*k2*dt), (y1[k] + 0.5*l2*dt),
         (z1[k] + 0.5*m2*dt),k,0.5*k2*dt)
        k4,l4,m4 = fun((x1[k] + k3*dt), (y1[k] + l3*dt), (z1[k] + m3*dt),k,k3*dt)


        x1[k+1] = x1[k] + (dt*(k1 + 2*k2 + 2*k3 + k4) / 6)
        y1[k+1] = y1[k] + (dt*(l1 + 2*l2 + 2*l3 + l4) / 6)
        z1[k+1] = z1[k] + (dt*(m1 + 2*m2 + 2*m3 + m4) / 6)


    return x1,y1,z1,t

#Func for Secure Communication
def sec_com(x1,m):
  s=[]
  for i,l in zip(x1,m):
    s.append(i+l)
  return s

# Initial conditions and function call
sigma = 10 #16
beta = 8/3 #4
rho = 28 #45.2
N=10000
dt=0.01
x_1, y_1, z_1,t =RungeKutta4(xyz1,1,0,0,N,dt)
m=message(t)
s=sec_com(x_1,m)
x_2,y_2, z_2,t = RungeKutta4(xyz2,0.12,-0.01,0.02,N,dt)
```

```
#Plots
plt.figure (figsize = (8.5,8.5))
fig, axs = plt.subplots(3,1)

axs[0].plot ( t, y_1, linewidth = 1, color = 'b' )
axs[0].plot ( t, y_2, linewidth = 1, color = 'r' )
plt.ylabel ( r'$y,\bar{y}$')

axs[1].plot ( t, z_1, linewidth = 1, color = 'b' )
axs[1].plot ( t, z_2, linewidth = 1, color = 'r' )
plt.xlabel ( '<--- Time --->' )
plt.ylabel ( r'$z,\bar{z}$')

axs[2].plot ( t, x_1, linewidth = 1, color = 'b' )
axs[2].plot ( t, x_2, linewidth = 0.7, color = 'r' )
plt.xlabel ( '<--- Time --->' )
plt.ylabel ( r'$x,\bar{x}$')
plt.show ( )

# Restoring Message
m_=[]
for i,j in zip(s,x_2):
    m_.append(i-j)

#Secure communication Plot
plt.figure (figsize = (8.5,7.5))
plt.subplot(3,1,1)
plt.plot ( t, m, linewidth = 1, color = 'b' )
plt.ylim(-0.6,0.6)
plt.ylabel ( r'$m$ (Original signal)')

plt.subplot(3,1,2)
plt.plot ( t, s, linewidth = 1, color = 'b' )
plt.ylabel ( r'$s$ (Masked with chaos)')

plt.subplot(3,1,3)
plt.plot ( t, m_, linewidth = 1, color = 'b' )
plt.ylim(-0.6,0.6)
plt.xlabel ( 'Time (t)' )
plt.ylabel ( r'$m_{r}$ (Restored signal)')
plt.savefig('SC_CS-1.pdf', format='pdf')
plt.show ( )
```

A.1.8 For Secure communication using CS for digital signal

```
# Importing Packages
import math, sys
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time
from ipywidgets import interact, interactive
from scipy import integrate
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import cnames
from matplotlib import animation
import random
from tqdm.notebook import tqdm
from scipy.integrate import odeint

# Compute the time-derivative of a Lorenz systems
def x_dot(x1, y1, z1,x2, y2, z2, k):
    x1_dot=sigma_1[k]*(y1 - x1)
    x2_dot=sigma*(y2-x2)
    return (x1_dot,x2_dot)

def y_dot(x1, y1, z1, x2, y2, z2, k):
    y1_dot=rho*x1 - y1 - x1*z1
    y2_dot=rho*x1 - y2 - x1*z2
    return(y1_dot,y2_dot)

def z_dot(x1, y1, z1, x2, y2, z2, k):
    z1_dot=-1*(beta*z1) + x1*y1
    z2_dot=-1*(beta*z2) + x1*y2
    return(z1_dot,z2_dot)

# Original Message
def message():
  frequency = 0.05
  duration = 100
  sampling_rate = 100

  # Create a time vector
  t = np.linspace(0, duration, int(sampling_rate * duration), endpoint=False)
  signal = np.where(np.sin(2 * np.pi * frequency * t) >= 0, 0, 1)
  return signal

# 4th order Runge Kutta (RK-4) numerical method
def RungeKutta4(a1,b1,c1,a2,b2,c2,n):

    x1 = np.zeros(n)
    y1 = np.zeros(n)
    z1 = np.zeros(n)
```

```
x2 = np.zeros(n)
y2 = np.zeros(n)
z2 = np.zeros(n)
t = np.zeros(n)

x1[0] = a1
y1[0] = b1
z1[0] = c1

y2[0] = b2
z2[0] = c2
t[0] = 0
dt = 0.01

#Compute the approximate solution at equally spaced times.
for k in range (n-1):

    t[k+1] = t[k] + dt

    k1,u1 = x_dot(x1[k], y1[k], z1[k], x2[k], y2[k], z2[k],k)
    l1,v1 = y_dot(x1[k], y1[k], z1[k], x2[k], y2[k], z2[k],k)
    m1,w1 = z_dot(x1[k], y1[k], z1[k], x2[k], y2[k], z2[k],k)

    k2,u2 = x_dot((x1[k] + 0.5*k1*dt), (y1[k] + 0.5*l1*dt), (z1[k] + 0.5*m1*dt),
                  (x2[k] + 0.5*u1*dt), (y2[k] + 0.5*v1*dt), (z2[k] + 0.5*w1*dt),k)
    l2,v2 = y_dot((x1[k] + 0.5*k1*dt), (y1[k] + 0.5*l1*dt), (z1[k] + 0.5*m1*dt),
                  (x2[k] + 0.5*u1*dt), (y2[k] + 0.5*v1*dt), (z2[k] + 0.5*w1*dt),k)
    m2,w2 = z_dot((x1[k] + 0.5*k1*dt), (y1[k] + 0.5*l1*dt), (z1[k] + 0.5*m1*dt),
                  (x2[k] + 0.5*u1*dt), (y2[k] + 0.5*v1*dt), (z2[k] + 0.5*w1*dt),k)

    k3,u3 = x_dot((x1[k] + 0.5*k2*dt), (y1[k] + 0.5*l2*dt), (z1[k] + 0.5*m2*dt),
                  (x2[k] + 0.5*u2*dt), (y2[k] + 0.5*v2*dt), (z2[k] + 0.5*w2*dt),k)
    l3,v3 = y_dot((x1[k] + 0.5*k2*dt), (y1[k] + 0.5*l2*dt), (z1[k] + 0.5*m2*dt),
                  (x2[k] + 0.5*u2*dt), (y2[k] + 0.5*v2*dt), (z2[k] + 0.5*w2*dt),k)
    m3,w3 = z_dot((x1[k] + 0.5*k2*dt), (y1[k] + 0.5*l2*dt), (z1[k] + 0.5*m2*dt),
                  (x2[k] + 0.5*u2*dt), (y2[k] + 0.5*v2*dt), (z2[k] + 0.5*w2*dt),k)

    k4,u4 = x_dot((x1[k] + k3*dt), (y1[k] + l3*dt), (z1[k] + m3*dt),
                  (x2[k]+ u3*dt), (y2[k] + v3*dt), (z2[k] + w3*dt),k)
    l4,v4 = y_dot((x1[k] + k3*dt), (y1[k] + l3*dt), (z1[k] + m3*dt),
                  (x2[k]+ u3*dt), (y2[k] + v3*dt), (z2[k] + w3*dt),k)
    m4,w4 = z_dot((x1[k] + k3*dt), (y1[k] + l3*dt), (z1[k] + m3*dt),
                  (x2[k]+ u3*dt), (y2[k] + v3*dt), (z2[k] + w3*dt),k)

    x1[k+1] = x1[k] + (dt*(k1 + 2*k2 + 2*k3 + k4) / 6)
    y1[k+1] = y1[k] + (dt*(l1 + 2*l2 + 2*l3 + l4) / 6)
    z1[k+1] = z1[k] + (dt*(m1 + 2*m2 + 2*m3 + m4) / 6)
```

```
        x2[k+1] = x2[k] + (dt*(u1 + 2*u2 + 2*u3 + u4) / 6)
        y2[k+1] = y2[k] + (dt*(v1 + 2*v2 + 2*v3 + v4) / 6)
        z2[k+1] = z2[k] + (dt*(w1 + 2*w2 + 2*w3 + w4) / 6)

    return x1,y1,z1,x2,y2,z2,t

# Initial conditions and function call
sigma = 16
beta = 4
rho = 45.2
N=10000
d_t=0.01

m=message()
sigma_1=sigma+m
x_1, y_1, z_1,x_2, y_2, z_2,t = RungeKutta4(0.2,0.1,1,0.3,0.7,2,N)

plt.plot ( t, sigma_1, linewidth = 1, color = 'b' )

E=[math.sqrt(pow(x1-x2,2)+pow(y1-y2,2)+pow(z1 - z2,2))
            for x1,x2,y1,y2,z1,z2 in zip(x_1,x_2,y_1,y_2,z_1,z_2)]

plt.figure (figsize = (7.5,4.5))
plt.plot ( t[10:], E[10:], linewidth = 0.7, color = 'r' )

#Restored Signal------------
M=[]
for i in E:
  if i> 0.001:
    M.append(1)
  else:
    M.append(0)

#Majority Cleaning Algorithm
np.random.seed(0)
message_length = 100
restored_message = np.array(M)
noisy_message = restored_message.copy()
noise_indices = np.random.choice(message_length, size=10, replace=False)
noisy_message[noise_indices] = 1 - noisy_message[noise_indices]

def clean_binary_message_majority_voting(binary_message, window_size=5):
    cleaned_message = []

    for i in range(len(binary_message)):
        start = max(0, i - window_size // 2)
```

```
        end = min(len(binary_message), i + window_size // 2 + 1)
        window = binary_message[start:end]
        cleaned_bit = 1 if sum(window) > len(window) // 2 else 0
        cleaned_message.append(cleaned_bit)

    return cleaned_message


cleaned_message_majority_voting = clean_binary_message_majority_voting(noisy_message)

# Evaluate the accuracy of the method by comparing it with the restored message
accuracy_majority_voting = np.mean(cleaned_message_majority_voting == restored_message)
print(accuracy_majority_voting*100)

# Plot for secure communictaion for digital signal
plt.figure (figsize = (8.5,7.5))
plt.subplot(4,1,1)
plt.plot ( t[10:], m[10:], linewidth = 1, color = 'b' )
plt.ylabel ( r'$m$')
plt.ylim(-0.5,1.5)

plt.subplot(4,1,2)
plt.plot ( t[10:], E[10:], linewidth = 1, color = 'r' )
plt.ylabel ( r'$E$')
plt.ylim(-0.5,1.5)

plt.subplot(4,1,3)
plt.plot ( t[10:], restored_message[10:], linewidth = 1, color = 'b' )
plt.ylabel ( r'$m^{*}$')
plt.ylim(-0.5,1.5)

plt.subplot(4,1,4)
plt.plot ( t[10:], cleaned_message_majority_voting[10:], linewidth = 1, color = 'b' )
plt.xlabel ( 'Time (t)' )
plt.ylabel ( r'$m^{*}$ (Cleaned)')
plt.ylim(-0.5,1.5)
plt.savefig('SC_CS-2.pdf', format='pdf')
plt.show ( )

plt.plot ( t, [math.sqrt(pow(z1-z2,2)) for z1,z2 in zip(z_1,z_2)],
           linewidth = 0.7, color = 'b' )
```

A.1.9 For Secure communication using Phase Synchronization

```
# Importing Packages
import math, sys
import numpy as np
```

```
import matplotlib.pyplot as plt
import time
from ipywidgets import interact, interactive
from scipy import integrate
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import cnames
from matplotlib import animation
import random
from tqdm.notebook import tqdm

# Rossler System
def x_dot(x1, y1, z1, x2, y2, z2, x3, y3, z3, w, k):
    x1_dot=-1*w[k]*y1-z1+eps*(x2-x1)
    x2_dot=-1*w[k]*y2-z2+eps*(x1-x2)
    Phi_m=np.arctan2(0.5*(y1+y2),0.5*(x1+x2))

    x3_dot=-y3-z3+alp*(np.sqrt(pow(x3,2)+pow(y3,2))*np.cos(Phi_m)-x3)
    return x1_dot,x2_dot,x3_dot

def y_dot(x1, y1, z1, x2, y2, z2, x3, y3, z3, w, k):
    y1_dot=w[k]*x1+a*y1
    y2_dot=w[k]*x2+a*y2
    y3_dot=x3+a*y3
    return y1_dot,y2_dot,y3_dot

def z_dot(x1, y1, z1, x2, y2, z2, x3, y3, z3, w, k):
    z1_dot=b+z1*(x1-c)
    z2_dot=b+z2*(x2-c)
    z3_dot=b+z3*(x3-c)
    return z1_dot,z2_dot,z3_dot

# 4th order Runge Kutta (RK-4) numerical method
def RungeKutta4(a1,b1,c1,a2,b2,c2,a3,b3,c3,n,w):

    x1 = np.zeros(n)
    y1 = np.zeros(n)
    z1 = np.zeros(n)

    x2 = np.zeros(n)
    y2 = np.zeros(n)
    z2 = np.zeros(n)

    x3 = np.zeros(n)
    y3 = np.zeros(n)
    z3 = np.zeros(n)
    t = np.zeros(n)
```

```
x1[0] = a1
y1[0] = b1
z1[0] = c1

x2[0] = a2
y2[0] = b2
z2[0] = c2

x3[0] = a3
y3[0] = b3
z3[0] = c3

t[0] = 0
dt = 0.01

#Compute the approximate solution at equally spaced times.
for k in range(n-1):

    t[k+1] = t[k] + dt

    k1,u1,p1 = x_dot(x1[k], y1[k], z1[k], x2[k], y2[k], z2[k], x3[k], y3[k], z3[k], w,k)
    l1,v1,q1 = y_dot(x1[k], y1[k], z1[k], x2[k], y2[k], z2[k],x3[k], y3[k], z3[k], w,k)
    m1,w1,r1 = z_dot(x1[k], y1[k], z1[k], x2[k], y2[k], z2[k],x3[k], y3[k], z3[k], w,k)

    k2,u2,p2 = x_dot((x1[k] + 0.5*k1*dt), (y1[k] + 0.5*l1*dt), (z1[k] + 0.5*m1*dt),
     (x2[k] + 0.5*u1*dt), (y2[k] + 0.5*v1*dt), (z2[k] + 0.5*w1*dt),
      (x3[k]+ 0.5*p1*dt), (y3[k]+ 0.5*q1*dt), (z3[k]+ 0.5*r1*dt), w,k)
    l2,v2,q2 = y_dot((x1[k] + 0.5*k1*dt), (y1[k] + 0.5*l1*dt), (z1[k] + 0.5*m1*dt),
     (x2[k] + 0.5*u1*dt), (y2[k] + 0.5*v1*dt), (z2[k] + 0.5*w1*dt),
      (x3[k]+ 0.5*p1*dt), (y3[k]+ 0.5*q1*dt), (z3[k]+ 0.5*r1*dt), w,k)
    m2,w2,r2 = z_dot((x1[k] + 0.5*k1*dt), (y1[k] + 0.5*l1*dt), (z1[k] + 0.5*m1*dt),
     (x2[k] + 0.5*u1*dt), (y2[k] + 0.5*v1*dt), (z2[k] + 0.5*w1*dt),
      (x3[k]+ 0.5*p1*dt), (y3[k]+ 0.5*q1*dt), (z3[k]+ 0.5*r1*dt), w,k)

    k3,u3,p3 = x_dot((x1[k] + 0.5*k2*dt), (y1[k] + 0.5*l2*dt), (z1[k] + 0.5*m2*dt),
     (x2[k] + 0.5*u2*dt), (y2[k] + 0.5*v2*dt), (z2[k] + 0.5*w2*dt),
      (x3[k]+ 0.5*p2*dt), (y3[k]+ 0.5*q2*dt), (z3[k]+ 0.5*r2*dt), w,k)
    l3,v3,q3 = y_dot((x1[k] + 0.5*k2*dt), (y1[k] + 0.5*l2*dt), (z1[k] + 0.5*m2*dt),
     (x2[k] + 0.5*u2*dt), (y2[k] + 0.5*v2*dt), (z2[k] + 0.5*w2*dt),
      (x3[k]+ 0.5*p2*dt), (y3[k]+ 0.5*q2*dt), (z3[k]+ 0.5*r2*dt), w,k)
    m3,w3,r3 = z_dot((x1[k] + 0.5*k2*dt), (y1[k] + 0.5*l2*dt), (z1[k] + 0.5*m2*dt),
     (x2[k] + 0.5*u2*dt), (y2[k] + 0.5*v2*dt), (z2[k] + 0.5*w2*dt),
      (x3[k]+ 0.5*p2*dt), (y3[k]+ 0.5*q2*dt), (z3[k]+ 0.5*r2*dt), w,k)

    k4,u4,p4 = x_dot((x1[k] + k3*dt), (y1[k] + l3*dt), (z1[k] + m3*dt),
     (x2[k]+ u3*dt), (y2[k] + v3*dt), (z2[k] + w3*dt),
      (x3[k]+ p3*dt), (y3[k]+ q3*dt), (z3[k]+ r3*dt), w,k)
```

```
        l4,v4,q4 = y_dot((x1[k] + k3*dt), (y1[k] + l3*dt), (z1[k] + m3*dt),
         (x2[k]+ u3*dt), (y2[k] + v3*dt), (z2[k] + w3*dt),
          (x3[k]+ p3*dt), (y3[k]+ q3*dt), (z3[k]+ r3*dt), w,k)
        m4,w4,r4 = z_dot((x1[k] + k3*dt), (y1[k] + l3*dt), (z1[k] + m3*dt),
         (x2[k]+ u3*dt), (y2[k] + v3*dt), (z2[k] + w3*dt),
          (x3[k]+ p3*dt), (y3[k]+ q3*dt), (z3[k]+ r3*dt), w,k)

        x1[k+1] = x1[k] + (dt*(k1 + 2*k2 + 2*k3 + k4) / 6)
        y1[k+1] = y1[k] + (dt*(l1 + 2*l2 + 2*l3 + l4) / 6)
        z1[k+1] = z1[k] + (dt*(m1 + 2*m2 + 2*m3 + m4) / 6)

        x2[k+1] = x2[k] + (dt*(u1 + 2*u2 + 2*u3 + u4) / 6)
        y2[k+1] = y2[k] + (dt*(v1 + 2*v2 + 2*v3 + v4) / 6)
        z2[k+1] = z2[k] + (dt*(w1 + 2*w2 + 2*w3 + w4) / 6)

        x3[k+1] = x3[k] + (dt*(p1 + 2*p2 + 2*p3 + p4) / 6)
        y3[k+1] = y3[k] + (dt*(q1 + 2*q2 + 2*q3 + q4) / 6)
        z3[k+1] = z3[k] + (dt*(r1 + 2*r2 + 2*r3 + r4) / 6)

    return x1,y1,z1,x2,y2,z2,x3,y3,z3,t

# Original Message
def message():
  frequency = 0.05
  duration = 100
  sampling_rate = 100

  # Create a time vector
  t = np.linspace(0, duration, int(sampling_rate * duration), endpoint=False)
  signal = np.where(np.sin(2 * np.pi * frequency * t) >= 0, 0, 1)
  return signal

# Managing Phase
def getPhase(x,y):
  p1 = np.zeros(len(x), dtype=x.dtype)

  if x[0]<0:
    p1[0] = np.arctan(y[0]/x[0]) + np.pi
  else:
    p1[0] = np.arctan(y[0]/x[0])

  for i in range(len(x)-1):
    i1 = i+1

    if x[i1]<0:
      p1[i1] = np.arctan(y[i1]/x[i1]) + np.pi
    else:
```

```
      p1[i1] = np.arctan(y[i1]/x[i1])

    if p1[i1] < p1[i]:
      nlaps = np.ceil( (p1[i]-p1[i1])/(2*np.pi) )
      p1[i1] = p1[i1] + 2 * np.pi * nlaps

  return p1

# Initial conditions and function call
alp=4.3
eps=0.005
a = 0.15
b= 0.2
c= 10
N= 10000
dt=0.01
m=message()
wo=[]
for i in m:
  if i==1:
    wo.append(1+0.01)
  else:
    wo.append(1-0.01)

x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_3, t = RungeKutta4(2, 4.5, 11, 2, 4, 1.1,
                                                             2, 4, 3.2, N, wo)

phi_m=getPhase(0.5*(x_1+x_2),0.5*(y_1+y_2))
phi_3=getPhase(x_3,y_3)

# Calculating synchronization error
E=[(i-j) for i,j in zip(phi_m,phi_3)]
plt.plot ( t, E , linewidth = 0.7, color = 'b' )
plt.ylabel('$\phi_m-\phi_3$')

# Setting Threshold
for i, j in enumerate(E):
  if j>0.04 or j<-0.04:
    E[i]=0
plt.plot ( t, E , linewidth = 0.7, color = 'b' )
plt.ylabel('$\phi_m-\phi_3$')

#Phase sync plot
plt.figure (figsize = (8.5,6.5))
plt.subplot(3,1,1)
plt.plot ( t, x_1, linewidth = 0.7, color = 'b' )
plt.plot ( t, x_2, linewidth = 1, color = 'g' )
```

```
plt.plot ( t, x_3, linewidth = 0.7, color = 'r' )
plt.ylabel ( r'$x_{1,2,3}$',fontsize=12)

plt.subplot(3,1,2)
plt.plot ( t, y_1, linewidth = 1, color = 'b' )
plt.plot ( t, y_2, linewidth = 1, color = 'g' )
plt.plot ( t, y_3, linewidth = 1, color = 'r' )
plt.ylabel ( r'$y_{1,2,3}$',fontsize=12)

plt.subplot(3,1,3)
plt.plot ( t, z_1, linewidth = 1, color = 'b' )
plt.plot ( t, z_2, linewidth = 1, color = 'g' )
plt.plot ( t, z_3, linewidth = 1, color = 'r' )
plt.ylabel ( r'$z_{1,2,3}$',fontsize=12)
plt.xlabel ( 'Time (t)',fontsize=10)

plt.savefig('SC_PS-1.pdf', format='pdf')
plt.show ( )

# Restoring message
m_=[]
for i in range(len(t)):
  if(E[i]<0):
    m_.append(0)
  else:
    m_.append(1)

#Majority Cleaning Algorithm
np.random.seed(0)
message_length = 100
restored_message = np.array(m_)
noisy_message = restored_message.copy()
noise_indices = np.random.choice(message_length, size=10, replace=False)
noisy_message[noise_indices] = 1 - noisy_message[noise_indices]  # Flip 0s to 1s and vice versa

def clean_binary_message_majority_voting(binary_message, window_size=71):
    cleaned_message = []

    for i in range(len(binary_message)):
        start = max(0, i - window_size // 2)
        end = min(len(binary_message), i + window_size // 2 + 1)
        window = binary_message[start:end]
        cleaned_bit = 1 if sum(window) > len(window) // 2 else 0
        cleaned_message.append(cleaned_bit)

    return cleaned_message
```

```
cleaned_message_majority_voting = clean_binary_message_majority_voting(noisy_message)
# Evaluate the accuracy of themethod by comparing with the original message
accuracy_majority_voting = np.mean(cleaned_message_majority_voting == restored_message)
print(accuracy_majority_voting*100)

#Secure communication Plot for Phase sync
plt.figure (figsize = (9,8.5))
plt.subplot(5,1,1)
plt.plot ( t, x_1, linewidth = 1, color = 'b' )
plt.plot ( t, x_2, linewidth = 1, color = 'g' )
plt.plot ( t, x_3, linewidth = 1, color = 'r' )
plt.yticks([-20,-10,0,10,20])
plt.ylabel ( r'$x_{1,2,3}$',fontsize=12)

plt.subplot(5,1,2)
plt.plot ( t, m, linewidth = 1.2, color = 'navy' )
plt.ylim(-0.5,1.5)
plt.ylabel ( r'$m$')

plt.subplot(5,1,3)
plt.plot ( t, E, linewidth = 1.2, color = 'darkcyan' )
plt.yticks([-0.04,0,0.04])
plt.ylabel('$\phi_m-\phi_3$',fontsize=12)

plt.subplot(5,1,4)
plt.plot ( t, restored_message, linewidth = 1.2, color = 'navy' )
plt.ylim(-0.5,1.5)
plt.ylabel ( r'$m^{*}$')

plt.subplot(5,1,5)
plt.plot ( t, cleaned_message_majority_voting, linewidth = 1.2, color = 'navy' )
plt.ylim(-0.5,1.5)
plt.xlabel ( 'Time (t)',fontsize=11)
plt.ylabel ( r'$m^{*}$ (Cleaned)')
plt.savefig('SC_PS-2.pdf', format='pdf')

plt.subplots_adjust(hspace=0.9)
plt.show ( )
```