Unveiling Complexity: Higher-Order Modelling of Swayed Structure Branches

Project Done by
Deepali Pandey ,302
MSc. Physics-II, Department of Physics, Ramniranjan Jhunjhunwala College, Ghatkopar(W).

Guided By
Dr. Kiran Kolwankar
Department of Physics, Ramniranjan Jhunjhunwala College, Ghatkopar(W).

2023-24

Abstract

In Physics, the dynamics of a system are often described by differential equations. Nonlinear dynamics as expressed through equations introduces a level of complexity that goes beyond the predictability of linear systems.

In the context of the wind-induced swaying of branch structure, the nonlinear differential equation becomes apparent when considering the relation between restoring and driving forces. Employing a systematic approach dynamics of the system are analyzed. starting from simpler systems and then further extending to more complex systems, linear stability analysis is performed. Further using numerical simulation phase portraits are plotted and dynamics of systems are analyzed. Moving forward, we delved into Y-branch system, exploring its behavior through the equation of motion.

Ramniranjan Jhunjhunwala College of Arts, Science Commerce
(Autonomous),Ghatkopar(W),Mumbai.
Department of Physics

CERTIFICATE

This is to certify that Deepali Pandey with
Roll No: 302 has completed her project with the title
"Unveiling Complexity: Higher-Order Modelling of Swayed Structure Branches"
as a part of Project I (semester-IV) under the supervision of
Dr.Kiran Kolwankar in the academic year 2023-24.

Signature of Guide

Dr.Kiran Kolwankar
Department of Physics
Ramniranjan Jhunjhunwala College
Ghatkopar (W), Mumbai

Head

Department of Physics
Ramniranjan Jhunjhunwala College
Ghatkopar (W), Mumbai

Contents

1 Semester III Work Summary **4**

1.1 Introduction 4

1.1.1 Background 5

1.2 Methodology Overview 5

1.2.1 Chimney Model 8

1.2.2 Single Element System, N=1 9

1.2.3 Two Element System,N=2 9

1.3 Conclusion 10

2 Phase Portrait Analysis **11**

2.1 For Single Element system: 11

2.2 For Two Element system: 12

3 Introduction to Y-Shaped System **13**

3.1 Equation of Motion: 13

3.1.1 Dynamics of system : 13

3.1.2 Equation of Motion: 14

3.2 Linear Stability Analysis : 15

3.3 Phase Portrait Analysis : 16

4 Programs used for Numerical Solution **18**

4.1 Using Python : 18

5 Summary, Conclusion and Future Directions **23**

5.1 Summary 23

5.2 Future Direction 23

Chapter 1

Semester III Work Summary

1.1 Introduction

On the dynamical front, Nonlinear dynamics have assumed sizeable proportions. In the last two decades, chaos theory, an offshoot from nonlinear dynamical processes has almost revolutionized the existing paradigms. It has evolved as a new frontier, which has increased in all disciplines where time evolution processes are involved. Although it had a beginning in physics, it has now increased in all branches of science [Lawande and Maiti, 2000].

In contrast to linear systems, nonlinear systems display complex responses, often characterized by irregular and unpredictable behaviors. Understanding these intricate dynamics is crucial for interpreting the behavior of various physical, biological, and ecological systems. The interplay between trees and the wind unfolds as a captivating natural spectacle, influencing both structural integrity and environmental dynamics.

This report delves into the thesis, ”Nonlinear Dynamical Modelling of Swaying of Branched Structures,” aiming to distill crucial insights from the exploration of wind-induced tree motion. Trees, swaying in response to the wind’s forces, exhibit a dynamic behavior that impacts their growth and structural resilience. This thesis builds upon this emerging paradigm, specifically examining irregular tree motion resulting from nonlinear restoring forces and destabilizing gravitational terms.

The objective is to provide a systematic nonlinear analysis of swaying trees, progressing from fundamental structures to more intricate configurations. The incorporation of nonlinear elements, including restoring forces and gravitational terms, allows for a more accurate representation of the true dynamics of tree movement. I am looking forward to presenting my understanding of the thesis and the methodology used in it to understand the tree model but before we hop into the analysis performed, we need to have certain informational resources to get this report to be understood effectively without developing loss of continuity and chasms of knowledge.

Last semester was primarily dedicated to theoretical comprehension and understanding of the dynamics of the systems under study. This semester, my focus shifted to coding, plotting phase portraits, and analyzing system dynamics. While I haven’t gotten into practical applications just yet, I’ve made significant strides in understanding the theoretical concepts and implementing them in computational simulations.

Hence here I have presented the essential literature survey required, as a major part of the introduction. The flow of the matter discussed here is as follows: We first delve into the essentials of non-linear dynamics, its development, and various methods to study non-linear systems. Then having more deeper look at the methodology in context to the dynamics of the Swaying of the Branched Structures model. And then finally after the discussion on non-linear dynamics, we will have a glance at the major motivation of the study relating to the Modelling of Swaying of Branched Structures. For those well-versed in the subject, feel free to skip the introduction.

1.1.1 Background

Nonlinear dynamics, stemming from foundational work by Poincare in celestial mechanics, has undergone significant development since the mid-20th century. Birkhoff's contributions expanded our understanding, coinciding with advancements in computational capabilities. This field encompasses differential equations and iterated maps, revealing complex behaviors unattainable in linear systems[Strogatz, 2018][Kashyap, 2021].

Unlike linear systems, which exhibit predictable behaviors, nonlinear equations introduce complexities that challenge traditional methods of analysis. Nonlinear terms, such as those involving functions raised to powers other than 1 and their derivatives, lead to diverse and sometimes unexpected behaviors. These equations often give rise to periodic oscillations and chaotic trajectories, amplifying the challenges of prediction[Guckenheimer, 1986],[Ott, 2002],[Strogatz, 2018].

In the context of wind-induced tree swaying, nonlinear dynamics play a crucial role in modeling the complex interactions between trees and external forces. Restoring forces, which aim to bring the system back to equilibrium, and gravitational effects contribute to the nonlinear nature of the equations governing tree motion. Understanding these dynamics is essential for predicting tree behavior and enhancing structural resilience.

Given the intricate nature of nonlinear systems, linear stability analysis emerges as a valuable tool for understanding their behavior. By examining equilibrium points and their stability characteristics, linear stability analysis provides insights into how small perturbations influence the overall stability of dynamic systems. This methodology serves as a foundation for analyzing the swaying motion of branched structures, offering valuable insights into their dynamic behavior[Morgan, 2015],[Strogatz, 2018].

1.2 Methodology Overview

The majority of our analysis of systems of ODEs will focus on whether or not the system has stable equilibria. We define the equilibrium point as stable or unstable according to the behavior of solutions whose initial conditions are close to equilibrium, if the solution near a critical point of a system stays close to the critical point as t approaches infinity we think of the critical point as stable, if this condition is not met then the critical point is unstable. [Roussel, 2019]. Additionally if, over time the solutions approach the critical point as opposed to simply staying within a certain radius, we say a stable critical point asymptotically stable[Morgan, 2015].

Linear stability analysis focuses on examining the response of a system to perturbations around an equilibrium point. For swaying branched structures, this analysis aims to assess how small deviations from the equilibrium state influence the overall stability of the system. Linear stability analysis is a powerful method employed to investigate the stability of fixed points within a dynamic system[Strogatz, 2018]. To comprehend this approach, consider a small perturbation introduced away from a fixed point within the system. The differential equation governing the system's motion is then modified to account for this perturbation. A second-order differential equation is converted to a system of two first-order ODEs Utilizing a Taylor series expansion around the fixed point, the system is linearized, focusing solely on the significance of linear terms, while higher-order quadratic terms are deemed negligible[Kashyap, 2021].

The resulting linearized differential equations can be expressed in matrix form, incorporating a matrix of first-order derivatives known as the Jacobian matrix. The eigenvalues of this Jacobian matrix, evaluated at the fixed point, play a crucial role in determining the stability characteristics of the system. Specifically, the real part of the eigenvalues signifies the inflow or outflow of trajectories around the fixed point, while the non-zero imaginary part indicates the rotational behavior of trajectories[Strogatz, 2018],[Kashyap, 2021].

In an n-dimensional system, the Jacobian matrix is an n x n matrix, yielding n eigenvalues. The characteristics of fixed points are dictated by these eigenvalues, providing insights into the system's local behavior around each fixed point. The classification of fixed points based on the sign of real and imaginary parts of eigenvalues is pivotal in understanding the stability of the system[Kashyap, 2021].

For clarity, let's delve into the characteristics of fixed points in different dimensions:

One-Dimensional System:

- Eigenvalue characteristic:
- λ < 0: Attracting node or stable node.
- λ >0: Repelling node or unstable node.

Two-Dimensional System:

- Eigenvalue characteristic:
- Real and distinct: Attracting node or stable node.
- Real and repeated: Star node or nodal source (unstable) or nodal sink (stable).
- Complex conjugate: Unstable spiral point or repelling focus (if positive) or stable spiral point or attracting focus (if negative) or center (if pure imaginary).

The saddle point represents a point in space where trajectories can approach along one direction but move away along the other, The attractors represent the stable state towards which trajectories tend to converge. extending this principle to two and 4-dimensional systems gives a comprehensive understanding of the stability around fixed points in a given dynamics system. in the further section, we will delve into understanding the dynamics of wind-induced tree swaying systems through the lens of linear stability analysis analysis[Kashyap, 2021].

Characterisation of fixed points:

Eigenvalues	Eigenvalues (symbolic description)	Behavior
Both real negative	$\lambda_1, \lambda_2 \in \mathbb{R}; \quad \lambda_1, \lambda_2 < 0$	Stable node
Both real positive	$\lambda_1, \lambda_2 \in \mathbb{R}; \quad \lambda_1, \lambda_2 > 0$	Unstable node
One real negative, one real positive	$\lambda_1, \lambda_2 \in \mathbb{R};$ $\lambda_1 > 0, \lambda_2 < 0 \;\; or \;\; \lambda_1 < 0, \lambda_2 > 0$	Saddle point (unstable)
Complex conjugate eigenvalues with negative real parts	$\lambda_1 = -a - bi, \lambda_2 = -a + bi \;\; or \;\; \lambda_1 = -a + bi, \lambda_2 = -a - bi$ $a, b \in \mathbb{R};\ a, b > 0$	Stable spiral
Complex conjugate eigenvalues with positive real parts	$\lambda_1 = a - bi, \lambda_2 = a + bi \;\; or \;\; \lambda_1 = a + bi, \lambda_2 = a - bi$ $a, b \in \mathbb{R};\ a, b > 0$	Unstable spiral
Complex conjugate with zero parts	$\lambda_1 = -bi, \lambda_2 = bi \;\; or \;\; \lambda_1 = bi, \lambda_2 = -bi$ $b \in \mathbb{R};\ b > 0$	Stable center

Figure 1.1: Characterisation of fixed points for two-dimensional system. [Collins, 2018]

	Eigenvalue	characteristic of fixed point
Eigenvalues are real	$\lambda_1, \lambda_2, \lambda_3 > 0$ all positive	repellor or source
	$\lambda_1, \lambda_2, \lambda_3 < 0$ all negative	node or sink
	$\lambda_1 > 0$, $\lambda_2, \lambda_3 < 0$ one positive and two negative	saddle or saddle point index-1 or $2:1$ saddle (2-manifold in and 1-manifold out)
	$\lambda_1, \lambda_2 > 0$, $\lambda_3 < 0$ two positive and one negative	saddle or saddle point index-2 or $1:2$ saddle (1-manifold in and 2-manifold out)
One eigenvalue real and one pair of complex conjugate	$Re(\lambda_1) < 0, Im(\lambda_1) = 0, \left.\begin{matrix} Re(\lambda_2) < 0, Im(\lambda_2) \\ Re(\lambda_3) < 0, Im(\lambda_3) \end{matrix}\right\} <> 0$ one eigenvalue is real and negative and other two eigenvalues have negative real parts with nonzero imaginary parts and form complex conjugate pair	spiral node or spiral sink
	$Re(\lambda_1) > 0, Im(\lambda_1) = 0, \left.\begin{matrix} Re(\lambda_2) > 0, Im(\lambda_2) \\ Re(\lambda_3) > 0, Im(\lambda_3) \end{matrix}\right\} <> 0$ one eigenvalue is real and positive and other two eigenvalues have positive real parts with nonzero imaginary parts and form complex conjugate pair	spiral repellor or spiral source
	$Re(\lambda_1) > 0, Im(\lambda_1) = 0, \left.\begin{matrix} Re(\lambda_2) < 0, Im(\lambda_2) \\ Re(\lambda_3) < 0, Im(\lambda_3) \end{matrix}\right\} <> 0$ one eigenvalue real and positive and other two have negative real part with nonzero imaginary parts and form complex conjugate pair	spiral-saddle point index-1 or $s2:1$ spiral saddle
	$Re(\lambda_1) < 0, Im(\lambda_1) = 0, \left.\begin{matrix} Re(\lambda_2) > 0, Im(\lambda_2) \\ Re(\lambda_3) > 0, Im(\lambda_3) \end{matrix}\right\} <> 0$ one eigenvalue real and negative and other two have positive real part with nonzero imaginary parts and form complex conjugate pair	spiral-saddle point index-2 or 1:s2 spiral saddle

Figure 1.2: Characterisation of fixed points for 3-dimensional system. [Kashyap, 2021]

Phase Portrait:

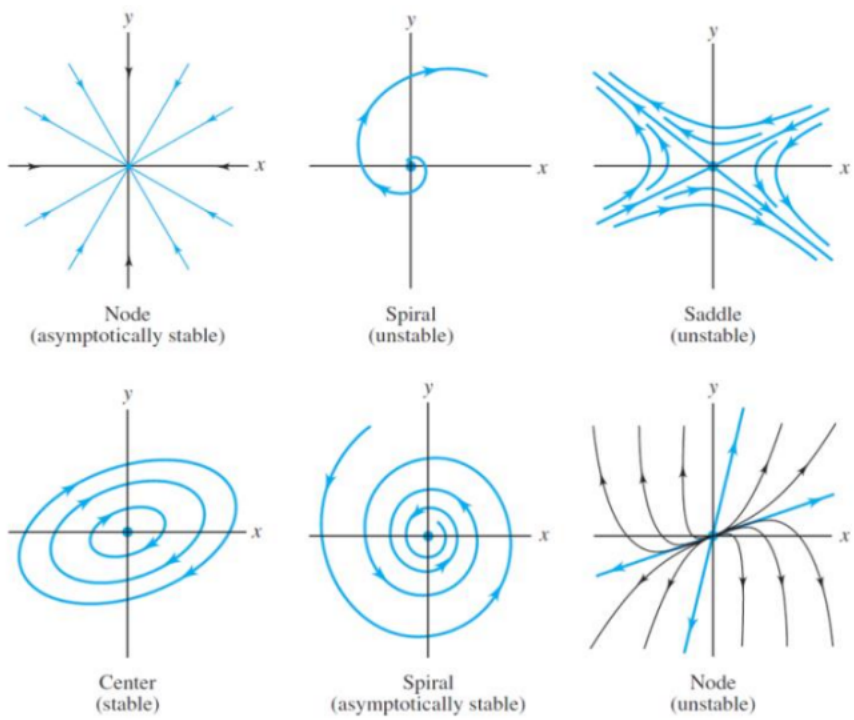


Figure 1.3: Phase portrait for different Eigenvalues. [Collins, 2018]

1.2.1 Chimney Model

The foundational steps involve revisiting the chimney model. This model serves as the starting point for understanding tree dynamics due to wind.

In this method, the tree is assumed to be made up of short sections that are connected end to end, with the biomaterial of these sections acting as the elastic material at the joints. Then the equations are developed to describe the response of each segment, resulting in a system of coupled differential equations. Here in this model, a small angle approximation was made assuming that forces vary linearly with deflection. However, studying tree dynamics with nonlinear stiffness and other nonlinear dependencies will provide a better understanding of the dynamics of trees [Kashyap, 2021].

Lagrangian mechanics emerges as a powerful tool for refining the chimney model. A critical outcome of this approach is the derivation of a generalized equation for an N-element system and further using it to study for N=1,2,3..n System [Kashyap, 2021].

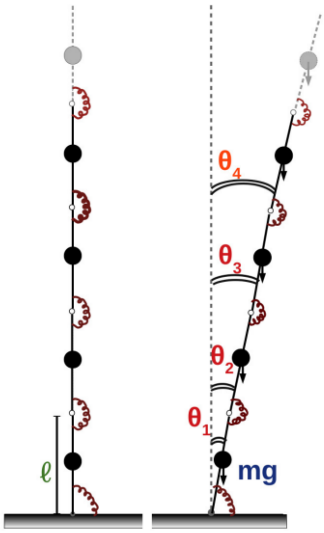


Figure 1.4: Schematic diagram of the Chimney model [Kashyap, 2021]

Generalized Equation for N-element System:

$$
\begin{aligned}
&\ell^2\ddot{\theta}_p M_{p,N} + \ell^2 \sum_{j=p+1}^{N} \left(\ddot{\theta}_j \cos(\theta_j - \theta_p) - \dot{\theta}_j^{\,2} \sin(\theta_j - \theta_p) \right) \widetilde{M}_{j,N} \\
&+\ell^2 \sum_{i=1}^{p-1} \left(\ddot{\theta}_i \cos(\theta_p - \theta_i) + \dot{\theta}_i^{\,2} \sin(\theta_p - \theta_i) \right) \widetilde{M}_{p,N} - g\ell \sin\theta_p \widetilde{M}_{p,N} \\
&+k_p(\theta_p - \theta_{p-1}) \left(1 + \alpha_p(\theta_p - \theta_{p-1})^2 \right) \\
&-k_{p+1}(\theta_{p+1} - \theta_p) \left(1 + \alpha_{p+1}(\theta_{p+1} - \theta_p)^2 \right) \\
&+\frac{1}{2} b\ell^2 \left(\frac{\dot{\theta}_p}{2} + 2 \sum_{i=p+2}^{N} \sum_{k=p+1}^{i-1} \cos(\theta_k - \theta_p)\dot{\theta}_k + 2 \sum_{i=p+1}^{N} \sum_{j=1}^{p-1} \cos(\theta_p - \theta_j)\dot{\theta}_j \right. \\
&\left. + \sum_{j=1}^{p-1} \cos(\theta_j - \theta_p)\dot{\theta}_j + \sum_{i=p+1}^{N} \left(\cos(\theta_p - \theta_i)\dot{\theta}_i + 2\dot{\theta}_p \right) \right) = 0
\end{aligned}
$$

The crux of the model is encapsulated in the derivation of a generalized equation governing the dynamics of the n-element system with nonlinear restoring force and dissipation.

1.2.2 Single Element System, N=1

We try to understand the specific case of the chimney model that is for N=1, a single-element model.

The system consists of a gravitational destabilizing force and a nonlinear restoring force that acts between the segment and the ground. The length of the segment is l and mass m1 is assumed to be concentrated at its centre. Here the angular displacement of the segment is theta 1 has been measured with the vertical direction[Kashyap, 2021].

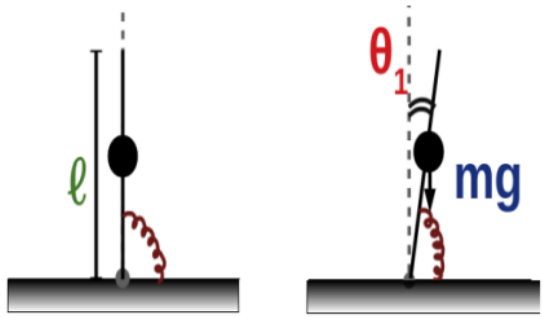


Figure 1.5: Schematic diagram of single element System
No deflection and with deflection[Kashyap, 2021]

System Dynamics Equation

$$\dot{\theta}_1 = 2g\sin(\theta_1) - 4k_1\theta_1(1+\alpha_1\theta_1^2) + 2f\cos(\omega t) - b\dot{\theta}_1$$

Where $2f\cos(\omega t)$ is identified as the driving force, which affects the system dynamics significantly.

Further analysis of potential energy is done to choose the range of parameters for further analysis, and then Linear stability Analysis is performed to determine the stability of the fixed points of this system. Doing it we find eigenvalues to have negative real parts implying that the fixed point will be either stable, a saddle point, or a stable spiral point.

1.2.3 Two Element System,N=2

Further, We delve into the dynamics of the two-element system. The complexity increases as we consider interactions between two elements offering a more comprehensive understanding. And the same way analysis is done for single-element systems and the dynamics are studied.

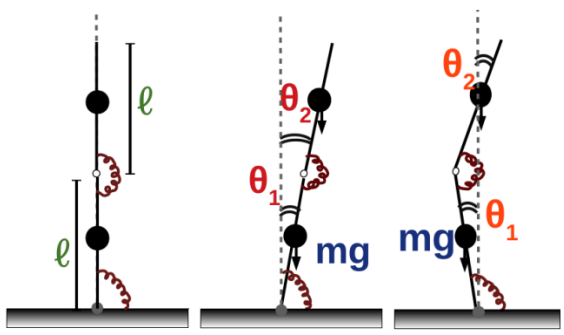


Figure 1.6: Schematic diagram of Two element System
No deflection and with deflection[Kashyap, 2021]

- **System Dynamics Equation**[Kashyap, 2021]

$$
\begin{aligned}
\ddot{\theta}_1 = \frac{1}{1-\frac{4}{5}\cos(\theta_2-\theta_1)^2}\Bigg(&\frac{4}{5}\dot{\theta}_1^2\sin(\theta_2-\theta_1)\cos(\theta_2-\theta_1)\\
&-\frac{4}{5}g\sin(\theta_2)\cos(\theta_2-\theta_1)+\frac{8}{5}k_2(\theta_2-\theta_1)\cos(\theta_2-\theta_1)(1+\alpha_2(\theta_2-\theta_1)^2)\\
&+\frac{4}{5}b\dot{\theta}_1\cos(\theta_2-\theta_1)^2-\frac{4}{5}f\cos(\omega t)\cos(\theta_2-\theta_1)+\frac{2}{5}\dot{\theta}_2^2\sin(\theta_2-\theta_1)\\
&+\frac{6}{5}g\sin(\theta_1)-\frac{4}{5}k_1\theta_1(1+\alpha_1\theta_1^2)+\frac{4}{5}k_2(\theta_2-\theta_1)(1+\alpha_2(\theta_2-\theta_1)^2)\\
&-b\dot{\theta}_1+\frac{2}{5}f\cos(\omega t)\Bigg),
\end{aligned}
$$

$$
\begin{aligned}
\ddot{\theta}_2 = \frac{1}{1-\frac{4}{5}\cos(\theta_2-\theta_1)^2}\Bigg(&-\frac{4}{5}\dot{\theta}_2^2\sin(\theta_2-\theta_1)\cos(\theta_2-\theta_1)\\
&-\frac{12}{5}g\sin(\theta_1)\cos(\theta_2-\theta_1)+\frac{8}{5}k_1\theta_1\cos(\theta_2-\theta_1)(1+\alpha_1\theta_1^2)\\
&-\frac{8}{5}k_2(\theta_2-\theta_1)\cos(\theta_2-\theta_1)(1+\alpha_2(\theta_2-\theta_1)^2)+\frac{4}{5}b\dot{\theta}_2\cos(\theta_2-\theta_1)^2\\
&-\frac{4}{5}f\cos(\omega t)\cos(\theta_2-\theta_1)-2\dot{\theta}_1^2\sin(\theta_2-\theta_1)+2g\sin(\theta_2)\\
&-4k_2(\theta_2-\theta_1)(1+\alpha_2(\theta_2-\theta_1)^2)-b\dot{\theta}_2+2f\cos(\omega t)\Bigg).
\end{aligned}
$$

Linear stability analysis is done. second-order ODEs are converted into a system of four first-order ODEs then those equations are written in matrix form (Jacobian), further the nature of the fixed points can be obtained from the eigenvalues of the Jacobian matrix evaluated at the fixed points. Since the two-segment system is 4-dimensional, the Jacobian is of order 4. so we either get 4 real eigenvalues or 2 real eigenvalues and a complex conjugate pair.

For the two-element system, 3 types of fixed points are observed. The stability of fixed points for various values of parameters is evaluated.

1.3 Conclusion

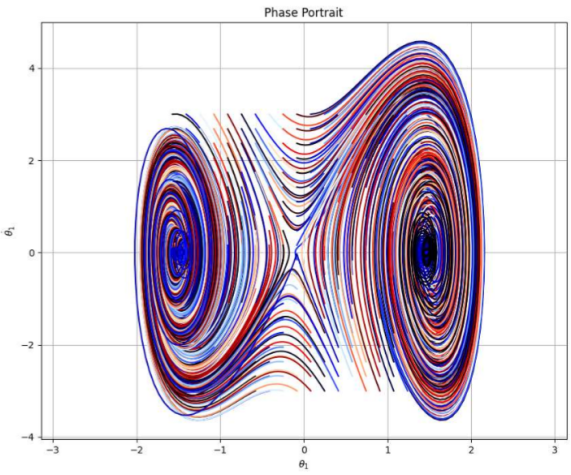
So in semester 3, we focused on a comprehensive understanding of single-element systems and two-element systems and performing linear stability analysis.

Chapter 2

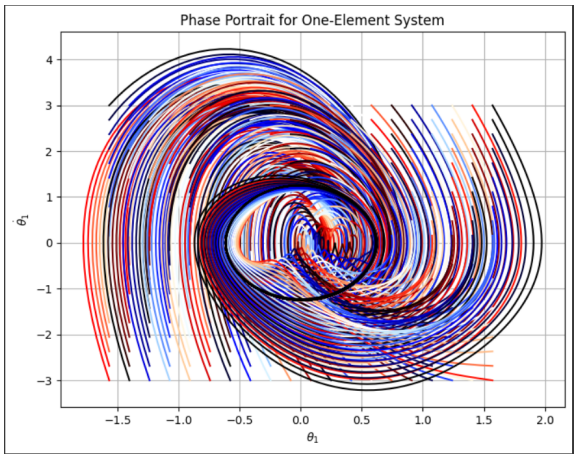
Phase Portrait Analysis

In this chapter, we delve into the analysis of Phase portraits obtained from numerical simulation conducted in Python. These phase portraits serve as a visual representation of the dynamic behavior of our system under varying conditions. The accompanying code used for this analysis is explained further in the chapter, providing a deeper insight into the computational methods employed.

2.1 For Single Element system:



a) for f=1, one saddle and 2 attractors are present



b) for f=5, stable focus.

The phase portrait for the N=1 element system was generated for two different forcing amplitudes f=3 and f=5 (and varying other parameters like b, α_1, k_1, and ω) where f is the amplitude of varying external forces,b is damping coefficient,k_1 is spring constant,ω is the frequency.

- **For f=3:** The phase portrait shows 3 fixed points, including a central saddle point and two stable spirals, attracting nearby trajectories. the central saddle point, where trajectories diverge, denotes the region of instability within the system dynamics.
- **For f=5** With an increased external forcing amplitude, the phase portrait illustrates a noticeable shift in the position of fixed points. The saddle point appears to move further from the center, suggesting the stringer influence of the forcing term. The stable spiral remains but is displaced reflecting the response of the system to high external forces. The shift evolves the spiral curves that approach the equilibrium point and the equilibrium point is called stable focus or attractive focus.

2.2 For Two Element system:

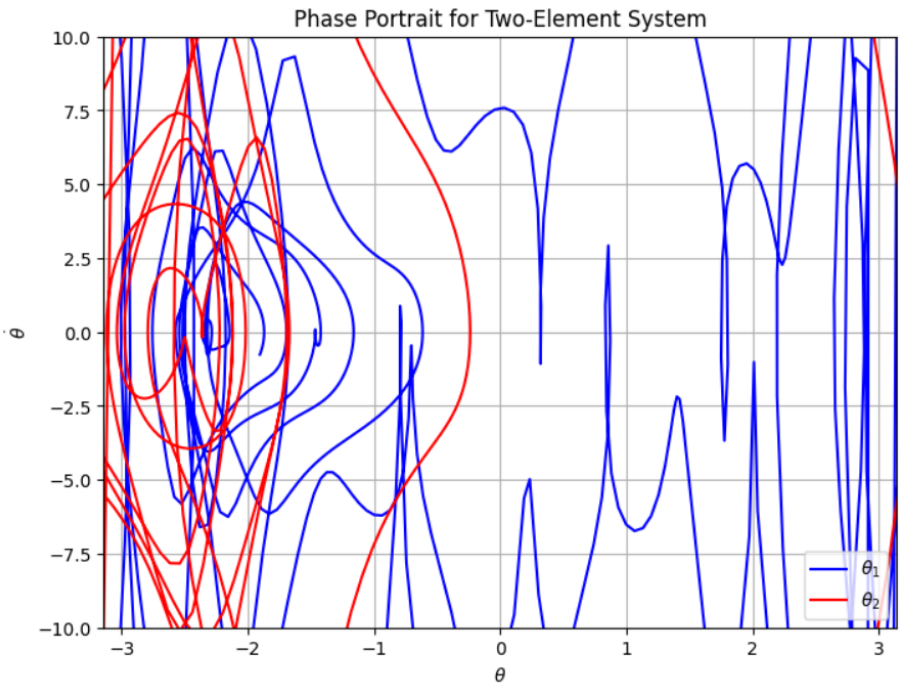


Figure 2.1: a) for f=10

Here the system showed a more dynamic behavior, where the addition of a variable in the equation introduced complexities in the system, leading to a spectrum of behavior from stable to chaotic. As we can see a hint of spiral then further dynamics get chaotic.

Chapter 3

Introduction to Y-Shaped System

The study was further extended to a little more complicated structure of the system, the Y-branched structure. The y-branched structure is an important fundamental tree unit needed to obtain a complete tree structure. considering the chimney model as a reference Y-shape structure with nonlinear restoring force and gravitational force is constructed.

3.1 Equation of Motion:

3.1.1 Dynamics of system :

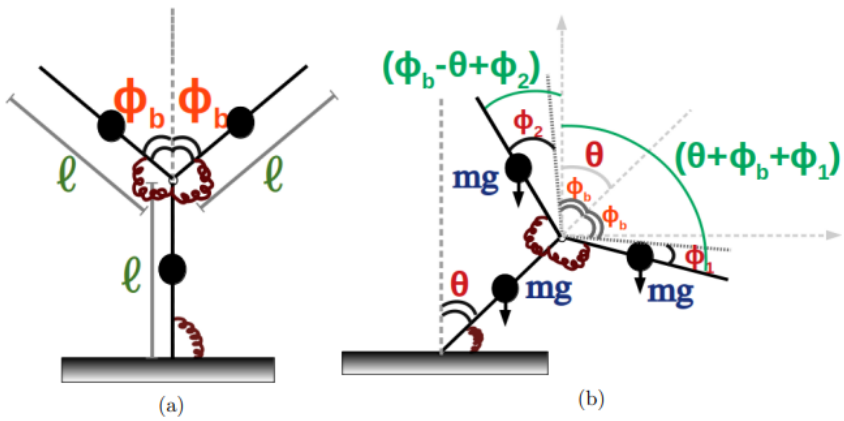


Figure 3.1: Schematic diagram of Y-shaped branch structure [Kashyap, 2021]

It consists of 3 segments the lower segment acts as a trunk and 2 branch segments each making an angle ϕ_b w.r.t lower segment but in opposite direction. The length of the lower segment is l1 and that of the branches is l_2 and the masses of segments are m_1, m_2, and $m_3$respectively. The angular displacement θ of the lowered segment is measured from the vertical direction and ϕ_1,ϕ_2 are the angular displacements of the first and second branches respectively. from the vertical direction, the angular momentum of the first branch and the second branch is [ϕ_b +$\phi_1(t)$)+$\theta(t)$] and [ϕ_b +$\phi_2(t)$)-$\theta(t)$] respectively. Further using the lagrangian mechanism, the equation of motion is Calculated.

3.1.2 Equation of Motion:

$$
\begin{aligned}
\ddot{\theta} = & \left\{ \frac{1}{\frac{9}{4}ml^2 - ml^2\cos^2(\phi_b + \phi_1(t)) - ml^2\cos^2(\phi_b + \phi_2(t))} \right\} \cdot \\
& \Bigg\{ \dot{\theta}(t)\dot{\phi}_1(t)ml^2\sin(\phi_b + \phi_1(t)) + \dot{\theta}(t)\dot{\phi}_2(t)ml^2\sin(\phi_b + \phi_2(t)) + \\
& \frac{1}{2}ml^2\dot{\theta}^2(t)\left(\sin(\phi_b + \phi_1(t)) - \sin(\phi_b + \phi_2(t))\right) + 2\sin(\phi_b + \phi_1(t))\cos(\phi_b + \phi_1(t)) - \\
& 2\sin(\phi_b + \phi_2(t))\cos(\phi_b + \phi_2(t)) + k_2(\phi_1(t) + \phi_2(t)) + k_2\alpha_2(\phi_1^3(t) + \phi_2^3(t)) - \\
& \dot{\theta}(t)\left(\frac{9}{4}bl^2 - bl^2\cos^2(\phi_b + \phi_2(t))\right) + (Q_{\text{drive1}} - Q_{\text{drive2}} + Q_{\text{drive3}}) + \\
& mgl\left(\sin(\phi_b + \phi_2(t)) - \theta(t)\cos(\phi_b + \phi_2(t)) - \sin(\phi_b + \phi_1(t)) - \theta(t)\cos(\phi_b + \phi_1(t))\right) + \\
& 2k_2\left(\phi_1(t)\cos(\phi_b + \phi_1(t)) - \phi_2(t)\cos(\phi_b + \phi_2(t))\right) + \\
& 2k_2\alpha_2\left(\phi_1^3(t)\cos(\phi_b + \phi_1(t)) - \phi_2^3(t)\cos(\phi_b + \phi_2(t))\right) - \\
& 2(Q_{\text{drive2}}\cos(\phi_b + \phi_1(t)) - Q_{\text{drive3}}\cos(\phi_b + \phi_2(t))) + \\
& \frac{1}{2}\dot{\phi}_1^2(t)ml^2\sin(\phi_b + \phi_1(t)) - \frac{1}{2}\dot{\phi}_2^2(t)ml^2\sin(\phi_b + \phi_2(t)) + \\
& \frac{5}{2}mgl\sin(\theta(t)) - k_1\theta(t) - k_1\alpha_1\theta^3(t) \Bigg\}
\end{aligned}
$$

$$
\begin{aligned}
\ddot{\phi}_1 = & \frac{1}{\frac{1}{4}ml^2}\Bigg\{ -\left(\frac{1}{4}ml^2 + \frac{1}{2}ml^2\cos(\phi_b + \phi_2(t))\right) \\
& / \left(\frac{9}{4}ml^2 - ml^2\cos^2(\phi_b + \phi_1(t)) - ml^2\cos^2(\phi_b + \phi_2(t))\right) \\
& \times \Bigg[\dot{\theta}(t)\dot{\phi}_1(t)ml^2\sin(\phi_b + \phi_1(t)) + \dot{\theta}(t)\dot{\phi}_2(t)ml^2\sin(\phi_b + \phi_2(t)) \\
& + \frac{1}{2}ml^2\dot{\theta}^2(t)\left(\sin(\phi_b + \phi_1(t)) - \sin(\phi_b + \phi_2(t))\right) \\
& + 2\sin(\phi_b + \phi_1(t))\cos(\phi_b + \phi_1(t)) - 2\sin(\phi_b + \phi_2(t))\cos(\phi_b + \phi_2(t)) \\
& + k_2(\phi_1(t) - \phi_2(t)) + k_2\alpha_2(\phi_1^3(t) - \phi_2^3(t)) \\
& - \dot{\theta}(t)\left(\frac{9}{4}bl^2 - bl^2\cos^2(\phi_b + \phi_1(t))\right) + (Q_{\text{drive1}} - Q_{\text{drive2}} + Q_{\text{drive3}}) \\
& + mgl(\sin(\phi_b + \phi_2(t)) - \theta(t)\cos(\phi_b + \phi_2(t)) - \sin(\phi_b + \phi_1(t)) - \theta(t)\cos(\phi_b + \phi_1(t))) \\
& + 2k_2(\phi_1(t)\cos(\phi_b + \phi_1(t)) - \phi_2(t)\cos(\phi_b + \phi_2(t))) \\
& + 2k_2\alpha_2(\phi_1^3(t)\cos(\phi_b + \phi_1(t)) - \phi_2^3(t)\cos(\phi_b + \phi_2(t))) \\
& - 2(Q_{\text{drive2}}\cos(\phi_b + \phi_1(t)) - Q_{\text{drive3}}\cos(\phi_b + \phi_2(t))) \\
& + \frac{1}{2}\dot{\phi}_1^2(t)ml^2\sin(\phi_b + \phi_1(t)) - \frac{1}{2}\dot{\phi}_2^2(t)ml^2\sin(\phi_b + \phi_2(t)) \\
& + \frac{5}{2}mgl\sin(\theta(t)) - k_1\theta(t) - k_1\alpha_1\theta^3(t) \Bigg] \Bigg\}
\end{aligned}
$$

$$
\begin{aligned}
\ddot{\phi}_2 =& \frac{1}{\frac{1}{4}ml^2}\Bigg\{ \Bigg[\left(\frac{1}{4}ml^2 + \frac{1}{2}ml^2\cos(\phi_b + \phi_2(t))\right) \\
& / \left(\frac{9}{4}ml^2 - ml^2\cos^2(\phi_b + \phi_1(t)) - ml^2\cos^2(\phi_b + \phi_2(t))\right)\Bigg] \\
& \times \Big[\dot{\theta}(t)\dot{\phi}_1(t)ml^2\sin(\phi_b + \phi_1(t)) + \dot{\theta}(t)\dot{\phi}_2(t)ml^2\sin(\phi_b + \phi_2(t)) \\
& + \frac{1}{2}ml^2\dot{\theta}^2(t)\left(\sin(\phi_b + \phi_1(t)) - \sin(\phi_b + \phi_2(t))\right) \\
& + 2\sin(\phi_b + \phi_1(t))\cos(\phi_b + \phi_1(t)) - 2\sin(\phi_b + \phi_2(t))\cos(\phi_b + \phi_2(t)) \\
& + k_2(\phi_1(t) - \phi_2(t)) + k_2\alpha_2(\phi_1^3(t) - \phi_2^3(t)) \\
& - \dot{\theta}(t)\left(\frac{9}{4}bl^2 - bl^2\cos^2(\phi_b + \phi_1(t))\right) + (Q_{\text{drive1}} - Q_{\text{drive2}} + Q_{\text{drive3}}) \\
& + mgl\left(\sin(\phi_b + \phi_2(t)) - \theta(t)\cos(\phi_b + \phi_2(t)) - \sin(\phi_b + \phi_1(t)) - \theta(t)\cos(\phi_b + \phi_1(t))\right) \\
& + 2k_2\left(\phi_1(t)\cos(\phi_b + \phi_1(t)) - \phi_2(t)\cos(\phi_b + \phi_2(t))\right) \\
& + 2k_2\alpha_2\left(\phi_1^3(t)\cos(\phi_b + \phi_1(t)) - \phi_2^3(t)\cos(\phi_b + \phi_2(t))\right) \\
& - 2\left(Q_{\text{drive2}}\cos(\phi_b + \phi_1(t)) - Q_{\text{drive3}}\cos(\phi_b + \phi_2(t))\right) \\
& + \frac{1}{2}\dot{\phi}_1^2(t)ml^2\sin(\phi_b + \phi_1(t)) - \frac{1}{2}\dot{\phi}_2^2(t)ml^2\sin(\phi_b + \phi_2(t)) \\
& + \frac{5}{2}mgl\sin(\theta(t)) - k_1\theta_1(t) - k_1\alpha_1\theta_1^3(t)\Big]\Bigg\} \\
& - \frac{1}{2}\dot{\theta}^2(t)ml^2\sin(\phi_b + \phi_1(t)) + \frac{1}{2}mgl\sin(\phi_b + \phi_1(t) + \theta(t)). \\
& . - k_2\phi_2(t) - k_2\alpha_2\phi_2(t)^3 - \dot{\phi}_1(t)\left(bl^2/4\right) - \dot{\theta}(t)\left(bl^2/4 + (bl^2/2)\cos(\phi_b + \phi_1(t))\right) + Q_{\text{drive3}}
\end{aligned}
$$

Above is the equation of motion for the lower segment first branch and second branch of the system respectively.

3.2 Linear Stability Analysis :

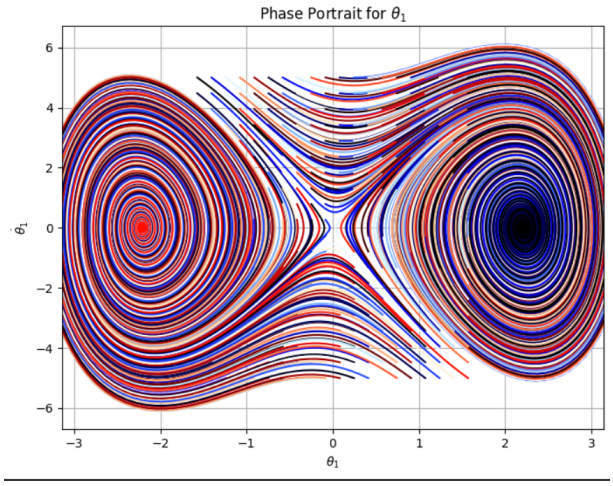
Linear stability analysis focuses on examining the response of a dynamics system to small perturbations around its equilibrium points.

As the method goes on firs, thetion of motion is linearised around the equilibrium points identified in the system. And those are identified by partially derivating potential with respective variables in the system to get the critical points. Further expressing the linearised equations in the Jacobian matrix and finding the eigenvalues of this matrix. eigenvalues of the matrix are evaluated to determine the stability.

Phase portraits are constructed based on linear stability analysis but it's not completely done. It does give a visual representation but not for all the possibilities. Further analysis of the Potential energy landscape is pending which provides additional insights into the stability and equilibrium point of the Y-branch structure. So far we got to Know the Phase portrait from the linearizing equation of motion, we know there exist critical points but not exactly what those points are. the visual representation of the phase portrait will be discussed in a further section.

3.3 Phase Portrait Analysis :

Without driving force



With the driving force

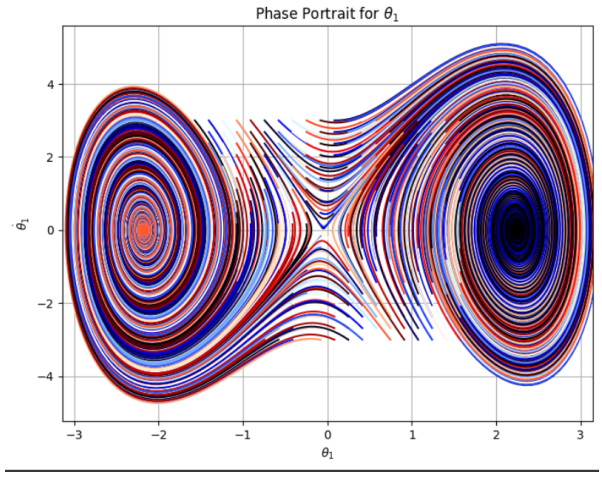
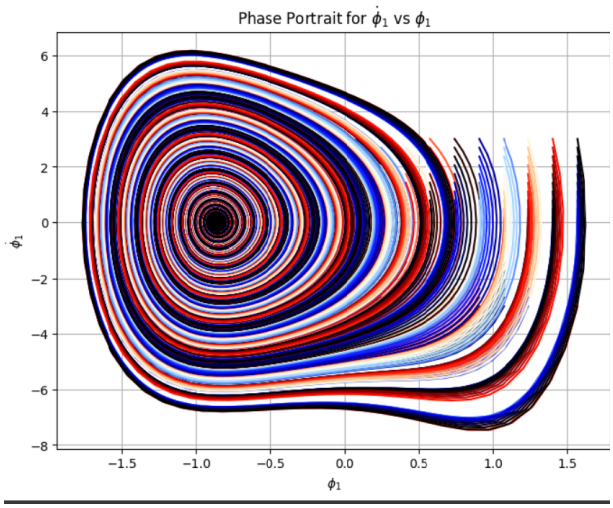


Figure 3.1. The figure is of Y branch structure for θ

Here we see two spirals and a saddle point as the driving force is introduced there is a significant shift in the dynamics of the system the spiral is shifted and repelling each other.

Without driving force



With the driving force

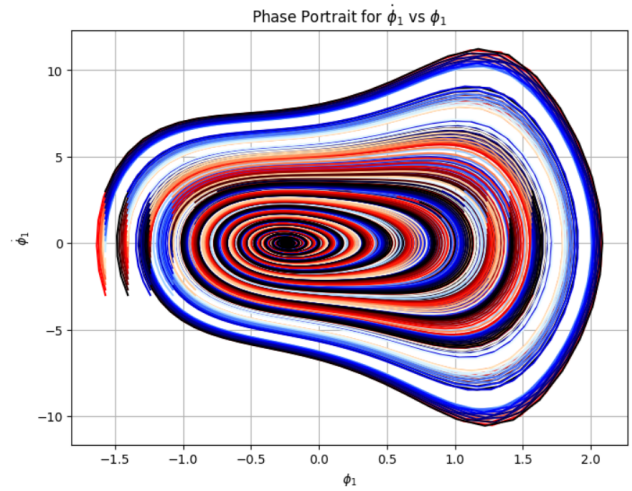
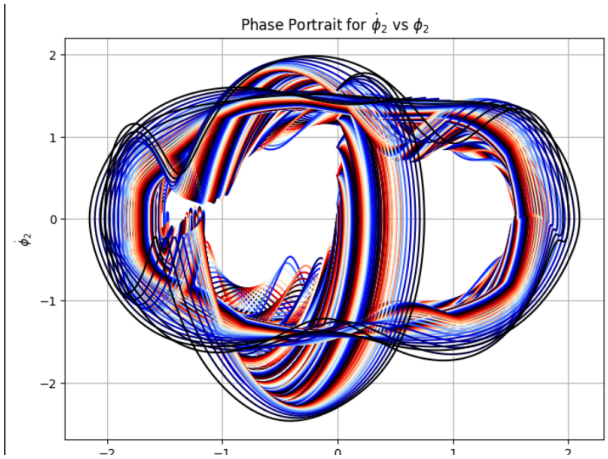


Figure 3.1. The figure is of Y branch structure for Φ_1

Here we see an asymptotic spiral without a driving force, when a driving force is applied there is a shift in dynamics the spiral is elongated horizontally.

Without driving force



With the driving force

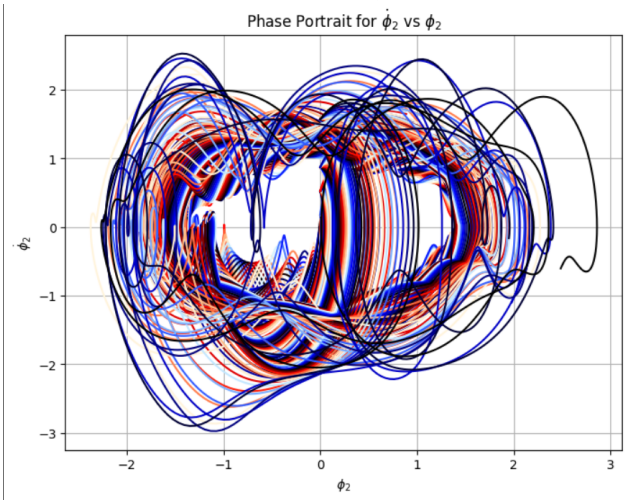


Figure 3.1. The figure is of Y branch structure for Φ_2

Here a huge instability and randomness are observed in the system. It features horizontally aligned spirals suggesting complex chaotic interaction between coupled oscillatory models. As driving force is applied the system gets disturbed and nonlinearity comes into dynamics giving rise to a more chaotic structure.

For this whole phase of portrait generation and analysis, we delve into Python programming. Python coding was done in Google Colab. Python libraries such as ODE integration,numpy, and matplotlib were used. The detailed code is mentioned in the next chapter.

Chapter 4

Programs used for Numerical Solution

4.1 Using Python :

**N=1, SINGLE ELEMENT SYSTEM CODE **

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
from matplotlib.colors import LinearSegmentedColormap

# Define the differential equation
def nldpf_eq(y, t, b, g, k1, alpha1, f, omega):
    theta1, theta1_dot = y
    d_theta1_dt = theta1_dot
    d_theta1_dot_dt = (2 * g * np.sin(theta1) - 4 * k1 * theta1 * (1 + alpha1 * theta1**2) + 2 * f * np.cos(omega * t) - b * theta1_dot)
    return [d_theta1_dt, d_theta1_dot_dt]

# Parameters
b = 1  # Modified value
g = 9.8  # Gravitational acceleration
k1 = 1  # Modified value
alpha1 = 1  # Modified value
f = 1  # Modified value
omega = 1  # No external force

# Initial conditions
theta1_range = np.linspace(-np.pi/2, np.pi/2, 20)  # Range of initial angles for theta1
theta1_dot_range = np.linspace(-3, 3, 20)  # Range of initial angular velocities for theta1_dot

# Time points for integration
t = np.linspace(0, 20, 1000)  # Time points from 0 to 20 with 1000 intervals

# Plot the phase portrait for each combination of initial conditions
plt.figure(figsize=(10, 8))
color_map = plt.cm.flag(np.linspace(0, 1, len(theta1_range) * len(theta1_dot_range)))

idx = 0
for theta1_0 in theta1_range:
    for theta1_dot_0 in theta1_dot_range:
        y0 = [theta1_0, theta1_dot_0]
        solution = odeint(nldpf_eq, y0, t, args=(b, g, k1, alpha1, f, omega))
        theta1, theta1_dot = solution.T
        plt.plot(theta1, theta1_dot, color=color_map[idx], alpha=1)
        idx += 1

plt.xlabel(r'$\theta_1$')
plt.ylabel(r'$\dot{\theta}_1$')
plt.title('Phase Portrait')
plt.grid(True)
plt.xlim(-np.pi, np.pi)  # Set x-axis limits
plt.show()
```

TWO ELEMENT SYSTEM (N=2) CODE

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

# Define the differential equations for theta1'' and theta2''
def nldpf_eq(y, t, b, g, k1, k2, alpha1, alpha2, f, omega):
    theta1, theta1_dot, theta2, theta2_dot = y

    # Equation for theta1''
    theta1_double_dot = (1 / (1 - (4/5) * np.cos(theta2 - theta1)**2))*\
                      (
                             (4/5) * (theta1_dot**2) * np.sin(theta2 - theta1) * np.cos(theta2 - theta1) - (4/5) * g
                             (4/5) * b * theta1_dot * np.cos(theta2 - theta1)**2 - (4/5) * f * np.cos(omega * t) * np
                             (6/5) * g * np.sin(theta1) - (4/5) * k1 * theta1*(1+alpha1*theta1**2)+ (4/5)*k2* (theta2
                        )

    #Equation for theta2''
    theta2_double_dot = (1 / (1 - (4/5) * np.cos(theta2 - theta1)**2)) * \
                        (
                            (-4/5) * (theta2_dot**2) * np.sin(theta2 - theta1) * np.cos(theta2 - theta1) - (12/5) *
                            (8/5) * k1 * theta1 * np.cos(theta2 - theta1) * (1 + alpha1 * (theta1)**2) -(8/5) * k2 *
                            (4/5) * b * theta2_dot * np.cos(theta2 - theta1)**2 - (4/5) * f * np.cos(omega * t) * np
                        )

    return [theta1_dot, theta1_double_dot, theta2_dot, theta2_double_dot]

# Parameters
b = 0.5 # Modified value
g = 9.8  # Gravitational acceleration
k1 = 5  # Modified value
k2 = 0.8  # Modified value
alpha1 =0.1  # Modified value
alpha2 = 1 # Modified value
f = 10 # Modified value
omega = 6  # external force

# Range of initial angles for theta1 and theta2
theta1_range = np.linspace(-np.pi, np.pi, 10)  # Range of initial angles for theta1
theta2_range = np.linspace(-np.pi, np.pi, 10)  # Range of initial angles for theta2

# Range of initial angular velocities for theta1_dot and theta2_dot
theta1_dot_0 = 10 # Initial angular velocity for theta1_dot
theta2_dot_0 = -10  # Initial angular velocity for theta2_dot

# Time points for integration
t = np.linspace(0, 10, 1000)  # Time points from 0 to 20 with 1000 intervals

# Initialize y0 with the initial conditions
y0 = [np.pi, theta1_dot_0, -np.pi, theta2_dot_0]

# Solve the differential equation
solution = odeint(nldpf_eq, y0, t, args=(b, g, k1, k2, alpha1, alpha2, f, omega))
theta1, theta1_dot, theta2, theta2_dot = solution.T

# Plot the phase portrait
plt.figure(figsize=(8, 6))
plt.plot(theta1, theta1_dot, label=r'$\theta_1$', color='b')
plt.plot(theta2, theta2_dot, label=r'$\theta_2$', color='r')
plt.xlabel(r'$\theta$')
plt.ylabel(r'$\dot{\theta}$')
plt.title('Phase Portrait for Two-Element System')
plt.legend()
plt.grid(True)
plt.xlim(-np.pi, np.pi)  # Set x-axis limits
plt.ylim(-10, 10)  # Set y-axis limits
plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
# Define the differential equations for theta1
def nldpf_eq_theta(y, t, b, g, k1, alpha1, k2, alpha2, phib, Qdrive1, Qdrive2, Qdrive3):
    theta1, theta1_dot, _, _, _, _ = y
    d_theta1_dt = theta1_dot
    d_phi1_dt = 0
    d_phi2_dt = 0
 # Equations for theta1_dot
    d_theta1_dot_dt = (
        (1 / (9/4 * m * l**2 - m * l**2 * np.cos(phib) - m * l**2 * np.cos(phib)))) * (
            theta1_dot * 0 * m * l**2 * np.sin(phib) +
            theta1_dot * 0 * m * l**2 * np.sin(phib) +
            (1/2) * m * l**2 * theta1_dot**2 * (np.sin(phib) - np.sin(phib)) +
            2 * np.sin(phib) * np.cos(phib) -
            2 * np.sin(phib) * np.cos(phib) +
            k2 * (0 - 0) +
            k2 * alpha2 * (0**3 - 0**3) -
            theta1_dot * (9 * b * l**2 / 4 - b * l**2 * np.cos(phib)) +
            (Qdrive1 - Qdrive2 + Qdrive3) +
            m * g * l * (np.sin(phib) - theta1 * np.cos(phib) - np.sin(phib) - theta1 * np.cos(phib)) +
            2 * k2 * (0 * np.cos(phib) - 0 * np.cos(phib)) +
            2 * k2 * alpha2 * (0**3 * np.cos(phib) - 0**3 * np.cos(phib)) -
            2 * (Qdrive2 * np.cos(phib) - Qdrive3 * np.cos(phib)) +
            (1/2) * 0**2 * m * l**2 * np.sin(phib) -
            (1/2) * 0**2 * m * l**2 * np.sin(phib) +
            (5/2) * m * g * l * np.sin(theta1) -
            k1 * theta1 -
            k1 * alpha1 * theta1**3
        )

    return [d_theta1_dt, d_theta1_dot_dt, d_phi1_dt, 0, 0, 0]

# Parameters
m = 1  # Mass
g = 9.8  # Gravitational acceleration
l = 1  # Length of the pendulum
k1 = 1  # Spring constant 1
alpha1 = 1  # Nonlinear coefficient 1
k2 = 1  # Spring constant 2
alpha2 = 1  # Nonlinear coefficient 2
phib =30  # Phase shift
Qdrive1 = 0 # External force 1
Qdrive2 = 0  # External force 2
Qdrive3 = 0  # External force 3
b=0.5

# Define the range of initial conditions for theta1 and its derivative
theta1_range = np.linspace(-np.pi/2, np.pi/2, 20)
theta1_dot_range = np.linspace(-5, 5, 20)

# Time points for integration
t = np.linspace(0, 20, 1000)

# Plot the phase portrait for theta1
plt.figure(figsize=(8,6))
color_map = plt.cm.flag(np.linspace(0, 1, len(theta1_range) * len(theta1_dot_range)))

idx = 0
for theta1_0 in theta1_range:
    for theta1_dot_0 in theta1_dot_range:
        y0 = [theta1_0, theta1_dot_0, 0, 0, 0, 0]  # Initial conditions for phi1, phi2 and their derivatives are s
        solution = odeint(nldpf_eq_theta, y0, t, args=(b, g, k1, alpha1, k2, alpha2, phib, Qdrive1, Qdrive2, Qdriv
        theta1, theta1_dot, _, _, _, _ = solution.T
        plt.plot(theta1, theta1_dot, color=color_map[idx], alpha=1)
        idx += 1

plt.xlabel(r'$\theta_1$')
plt.ylabel(r'$\dot{\theta}_1$')
plt.title('Phase Portrait for $\\theta_1$')
plt.grid(True)
plt.xlim(-np.pi, np.pi)  # Set x-axis limits
plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

def phi1_dot_eq(y, t, b, g, k1, alpha1, k2, alpha2, phib, Qdrive1, Qdrive2, Qdrive3, theta1_dot, phi2_dot):
    theta1, theta1_dot, phi1, phi1_dot, phi2, phi2_dot = y

    # Define the equation for phi1_dot
    d_phi1_dot_dt = (
        (1 / ((1/4) * m * l**2)) *
        (
            - ((1/4) * m * l**2 + (1/2) * m * l**2 * np.cos(phib + phi1)) / ((9/4) * m * l**2 - m * l**2 * np.cos(phib +
            (
                theta1_dot * phi1_dot * m * l**2 * np.sin(phib + phi1) +
                theta1_dot * phi2_dot * m * l**2 * np.sin(phib + phi2) +
                (1/2) * m * l**2 * theta1_dot**2 * (np.sin(phib + phi1) - np.sin(phib + phi2)) +
                2 * np.sin(phib + phi1) * np.cos(phib + phi1) -
                2 * np.sin(phib + phi2) * np.cos(phib + phi2) +
                k2 * (phi1 - phi2) +
                k2 * alpha2 * (phi1**3 - phi2**3) -
                theta1_dot * (9 * b * l**2 / 4 - b * l**2 * np.cos(phib + phi1)**2 - b * l**2 * np.cos(phib + phi2)**2) +
                (Qdrive1 - Qdrive2 + Qdrive3) +
                m * g * l * (np.sin(phib + phi2-theta1)*np.cos(phib + phi2) - np.sin(phib + phi1 + theta1)*np.cos(phib +
                2 * k2 * (phi1 * np.cos(phib + phi1) - phi2 * np.cos(phib + phi2)) +
                2 * k2 * alpha2 * (phi1**3 * np.cos(phib + phi1) - phi2**3 * np.cos(phib + phi2)) -
                2 * (Qdrive2 * np.cos(phib + phi1) - Qdrive3 * np.cos(phib + phi2)) +
                (1/2) * phi1_dot**2 * m * l**2 * np.sin(phib + phi1) -
                (1/2) * phi2_dot**2 * m * l**2 * np.sin(phib + phi2) +
                (5/2) * m * g * l * np.sin(theta1) - k1 * theta1 - k1 * alpha1 * theta1**3
            )
         -
        (1/2) * m * l**2 * theta1_dot**2 * np.sin(phib + phi1) +
        (1/2) * m * g * l * np.sin(phib + phi1 - theta1) -
        k2 * phi1 - k2 * alpha2 * phi1**3 -
        phi1_dot * (b * l**2 / 4) +
        theta1_dot * (b * l**2 / 4 + (b * l**2 / 2) * np.cos(phib + phi1)) +
        Qdrive3
    ))

    # Return the derivatives of all variables
    return [theta1_dot, 0, phi1_dot, d_phi1_dot_dt, phi2_dot, 0]
# Parameters
m = 1  # Mass
g = 9.8  # Gravitational acceleration
l = 1  # Length of the pendulum
k1 = 1  # Spring constant 1
alpha1 = 1  # Nonlinear coefficient 1
k2 = 1.5  # Spring constant 2
alpha2 = 1  # Nonlinear coefficient 2
phib = 30  # Phase shift
Qdrive1 = 5 # External force 1
Qdrive2 = 5 # External force 2
Qdrive3 = 5 # External force 3
b = 0.5  # Damping coefficient

# Define the range of initial conditions for phi1 and its derivative
phi1_range = np.linspace(-np.pi/2, np.pi/2, 20)
phi1_dot_range = np.linspace(-3, 3, 20)

# Time points for integration
t = np.linspace(0, 20, 1000)

# Define external parameters theta1_dot and phi2_dot
theta1_dot = 0  # Set to some value
phi2_dot = 0  # Set to some value

# Plot the phase portrait for phi1_dot vs phi1
plt.figure(figsize=(8, 6))
color_map = plt.cm.flag(np.linspace(0, 1, len(phi1_range) * len(phi1_dot_range)))

idx = 0
for phi1_0 in phi1_range:
    for phi1_dot_0 in phi1_dot_range:
        y0 = [0, 0, phi1_0, phi1_dot_0, 0, 0]  # Initial conditions for theta1 and phi2
```

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

def phi1_dot_eq(y, t, b, g, k1, alpha1, k2, alpha2, phib, Qdrive1, Qdrive2, Qdrive3, theta1_dot, phi2_dot):
    theta1, theta1_dot, phi1, phi1_dot, phi2, phi2_dot = y

    # Define the equation for phi1_dot
    d_phi1_dot_dt = (
        (1 / ((1/4) * m * l**2)) *
        (
            - ((1/4) * m * l**2 + (1/2) * m * l**2 * np.cos(phib + phi1)) / ((9/4) * m * l**2 - m * l**2 * np.cos(phib +
            (
                theta1_dot * phi1_dot * m * l**2 * np.sin(phib + phi1) +
                theta1_dot * phi2_dot * m * l**2 * np.sin(phib + phi2) +
                (1/2) * m * l**2 * theta1_dot**2 * (np.sin(phib + phi1) - np.sin(phib + phi2)) +
                2 * np.sin(phib + phi1) * np.cos(phib + phi1) -
                2 * np.sin(phib + phi2) * np.cos(phib + phi2) +
                k2 * (phi1 - phi2) +
                k2 * alpha2 * (phi1**3 - phi2**3) -
                theta1_dot * (9 * b * l**2 / 4 - b * l**2 * np.cos(phib + phi1)**2 - b * l**2 * np.cos(phib + phi2)**2) +
                (Qdrive1 - Qdrive2 + Qdrive3) +
                m * g * l * (np.sin(phib + phi2-theta1)*np.cos(phib + phi2) - np.sin(phib + phi1 + theta1)*np.cos(phib +
                2 * k2 * (phi1 * np.cos(phib + phi1) - phi2 * np.cos(phib + phi2)) +
                2 * k2 * alpha2 * (phi1**3 * np.cos(phib + phi1) - phi2**3 * np.cos(phib + phi2)) -
                2 * (Qdrive2 * np.cos(phib + phi1) - Qdrive3 * np.cos(phib + phi2)) +
                (1/2) * phi1_dot**2 * m * l**2 * np.sin(phib + phi1) -
                (1/2) * phi2_dot**2 * m * l**2 * np.sin(phib + phi2) +
                (5/2) * m * g * l * np.sin(theta1) - k1 * theta1 - k1 * alpha1 * theta1**3
            )
         -
        (1/2) * m * l**2 * theta1_dot**2 * np.sin(phib + phi1) +
        (1/2) * m * g * l * np.sin(phib + phi1 - theta1) -
        k2 * phi1 - k2 * alpha2 * phi1**3 -
        phi1_dot * (b * l**2 / 4) +
        theta1_dot * (b * l**2 / 4 + (b * l**2 / 2) * np.cos(phib + phi1)) +
        Qdrive3
    ))

    # Return the derivatives of all variables
    return [theta1_dot, 0, phi1_dot, d_phi1_dot_dt, phi2_dot, 0]
# Parameters
m = 1  # Mass
g = 9.8  # Gravitational acceleration
l = 1  # Length of the pendulum
k1 = 1  # Spring constant 1
alpha1 = 1  # Nonlinear coefficient 1
k2 = 1.5  # Spring constant 2
alpha2 = 1  # Nonlinear coefficient 2
phib = 30  # Phase shift
Qdrive1 = 5 # External force 1
Qdrive2 = 5 # External force 2
Qdrive3 = 5 # External force 3
b = 0.5  # Damping coefficient

# Define the range of initial conditions for phi1 and its derivative
phi1_range = np.linspace(-np.pi/2, np.pi/2, 20)
phi1_dot_range = np.linspace(-3, 3, 20)

# Time points for integration
t = np.linspace(0, 20, 1000)

# Define external parameters theta1_dot and phi2_dot
theta1_dot = 0  # Set to some value
phi2_dot = 0  # Set to some value

# Plot the phase portrait for phi1_dot vs phi1
plt.figure(figsize=(8, 6))
color_map = plt.cm.flag(np.linspace(0, 1, len(phi1_range) * len(phi1_dot_range)))

idx = 0
for phi1_0 in phi1_range:
    for phi1_dot_0 in phi1_dot_range:
        y0 = [0, 0, phi1_0, phi1_dot_0, 0, 0]  # Initial conditions for theta1 and phi2
```

Chapter 5

Summary, Conclusion and Future Directions

5.1 Summary

In summary, this report has explored the phase portrait of single and two-element systems and studied their dynamics. Further, an attempt to understand the Y-branch system was made, shedding light on their behavior under various conditions and Perturbations. The detailed stability analysis for the Y-branch system is yet to be done. However, we did a stability analysis of the equation of motion using Python programming and did a phase portrait analysis to get a glimpse of the dynamics of the complex system. In conclusion, we highlighted the use of computer programming to understand the complexities of nonlinear systems. Moving forward there remains ample opportunity for further understanding the dynamics of systems.

5.2 Future Direction

The Pending analysis of the Y-branch system is to be done. which would further lead to a better understanding taking into consideration the synchronization that this structure exhibits and further leading to a detailed analysis of its chaotic nature.

Further, the study is to be extended to the N=3 element system and do the same analysis and understand the dynamics and chaotic properties the system exhibits.

Bibliography

[Collins, 2018] Collins, L. (2018). Notes on dynamical systems. Retrieved from LoganThrasher-Collins.com.

[Guckenheimer, 1986] Guckenheimer, J. (1986). Dynamical systems, and bifurcatins of vector fields. *Appl. Math. Sci. Series*, 42.

[Kashyap, 2021] Kashyap, A. R. (2021). Nonlinear dynamical modeling of swaying of branched structures.

[Lawande and Maiti, 2000] Lawande, Q. V. and Maiti, N. (2000). Role of nonlinear dynamics and chaos in applied sciences. Technical report, Bhabha Atomic Research Centre.

[Morgan, 2015] Morgan, R. (2015). Linearization and stability analysis of nonlinear problems. *Rose-Hulman Undergraduate Mathematics Journal*, 16(2):5.

[Ott, 2002] Ott, E. (2002). *Chaos in dynamical systems*. Cambridge University Press.

[Roussel, 2019] Roussel, M. (2019). *Stability analysis for ODEs*.

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